EasternGraphics
visualize your business

Application Notes (2023-09-18)

AN-2006-01: Steuerdatentabellen

Inhalt

Rechtliche Hinweise

1. Einführung

Das Verhalten der Standardimplementierung von OFML Basistypen, die vom OFML-Laufzeitsystem verwendet werden, kann mittels spezieller Datentabellen gesteuert werden, den so genannten **Steuerdatentabellen**. Durch diesen Ansatz wird vermieden, spezifische Unterklassen ableiten zu müssen, um ein spezielles Verhalten zu erzielen.

Jeder OFML Basistyp, der eine Steuerdatentabelle einführt, implementiert die Methode *openDataTable()*, welche den Namen der Tabelle bestimmt. Abgeleitete Klassen können (zusätzliche) Informationseinheiten (Datenelemente) für die Tabelle definieren und nutzen.

Die Tabelle kann als einzelne CSV-Tabelle vorliegen oder in einer EBase Datenbank namens `ofml.ebase` enthalten sein. (Ist die Tabelle in der `ofml.ebase` enthalten, so hat diese Vorrang gegenüber einer eventuell parallel vorliegenden separaten CSV-Tabelle.)

Alle Steuerdatentabellen besitzen denselben Aufbau:

- Feld 1 legt den **Typ der Information** fest, die in dem Datensatz abgelegt ist. (Der Typbezeichner beginnt immer mit dem `@`-Zeichen.)
- Feld 2, das so genannte **Argumentfeld**, ist optional und kann dazu verwendet werden, den Verwendungszweck bzw. den Anwendungsbereich der Information näher zu bestimmen.
- Feld 3, das so genannte **Wertfeld**, enthält die eigentliche Information.

Die Tabellenbeschreibung für die EBase-Konfigurationstabelle lautet entsprechend:

```
table <name> <name>.csv mscsv
fields 3
field  1 type  vstring delim ; trim hidx link
field  2 args  vstring delim ; trim
field  3 value vstring delim ; trim
```

Die Tabellen bzw. die `ofml.ebase` müssen sich, falls nicht anderweitig spezifiziert, im regulären Datenverzeichnis der betreffenden OFML Bibliothek/Serie befinden. Als Zeichensatz wird ISO-8859-1 (Latin-1) verwendet.

Das Format und die möglichen bzw. geforderten Datentypen der einzelnen Informationstypen werden durch diejenigen Methoden der OFML Basistypen festgelegt, die die Information des jeweiligen Typs verwenden. In den nachfolgenden Abschnitten werden die Informationstypen jedoch entsprechend ihrer Zugehörigkeit zu einer konkreten Steuerdatentabelle zusammengefasst und beschrieben.

Die vorliegende Application Note spiegelt den Entwicklungsstand der OFML Basispakete OI Version 1.43.0 und XOI Version 1.60.0 sowie von EAI Version 1.32.0 wieder (native OCD Implementierung).
Die entsprechend vorausgesetzten Versionen der Applikationen sind pCon.planner >= 8.9, pCon.basket >= 1.13.0 sowie EAIWS >= 4.14 (online Apps)[1].

[1] Releases im November 2023

2. Tabelle *proginfo*

Diese Tabelle dient zur Steuerung des Verhaltens des so genannten *ProgInfo-Objektes*, das für eine gegebene OFML Serie (Programm) registriert wird. Dieses Objekt realisiert allgemeines Verhalten, das die OFML Serie als Ganzes betrifft. (Der konkrete Typ des *ProgInfo*-Objektes wird in der DSR-Registrierungsdatei für die OFML Serie unter dem Schlüssel `proginfo` festgelegt.)

Diese Tabelle wird durch den OFML Basistyp *OiProgInfo* eingeführt und wird auch von der abgeleiteten Klasse *xOiProgInfo* verwendet.

Wenn sich die Tabelle nicht im regulären Datenverzeichnis der betreffenden OFML Serie befindet, wird sie im Verzeichnis gesucht, das in der DSR-Registrierungsdatei der Serie unter dem Schlüssel `proginfodb_path` angegeben ist.

2.1. Allgemeine Optionen

@AutoDecorationPath

- Das Wertfeld gibt den Pfad des Verzeichnisses an, das spezifische Templates enthält, die während der automatischen Dekoration von Artikeln dieser Serie verwendet werden sollen.
- Mehr Informationen zur automatischen Dekoration siehe separate Application Note.
- Verwendet von der Methode *xOiProgInfo::getAutoDecorationPath()*.

@CheckPrice4Consistency

- Enthält das Feld 3 den Wert `1`, werden Artikel dieser Serie als inkonsistent markiert, wenn die Methode *getArticlePrice()* der Schnittstelle *Article* für einen gegebenen Artikel `NULL` liefert.
- Beachte: inkonsistente Artikel können nicht bestellt werden!
- Verwendet von der Methode *xOiProgInfo::checkObjConsistency()*.

@DynamicAttPt

- Wird von der Methode *OiProgInfo::getDynamicAttPts()* verwendet, welche eine Liste von `[key, dir]` Paaren für alle diejenigen Anfügepunkte liefert, die in der Serie vorkommen und die unmittelbar nach der Artikelerzeugung nicht über die Methode *getAttPtsOrder()* (OFML Interface *AttachPts*) abgerufen werden können, da sie abhängig von einem bestimmten Merkmal sind, z.B. einem Höhenraster. Diese Information kann dann von Tools verwendet werden, die Daten für spezielle Feedback-Verfahren generieren, die während der Artikeleinfügung im 3D-Modus einer OFML-Applikation zur Anwendung kommen.
- Das Wertfeld enthält ein oder mehrere `[key, dir]` Paare. (Bei mehreren Paaren erfolgt die Trennung mittels Komma.) Die Paare werden dann der Rückgabeliste von *getDynamicAttPts()* zugefügt.

@MatPackage

- Das Wertfeld (Feld 3) gibt den Namen des OFML Paketes an, das die Materialdefinitionsdateien enthält, die für die Artikel der Serie zu verwenden sind.

- Verwendet von den Methoden *OiProgInfo::getMatPackage()* und *OiProgInfo::getMatName()*.

`@PricesOnRequest`

- Enthält das Feld 3 den Wert `1`, werden Artikel dieser Serie mit dem Etikett „Preis auf Anfrage“ versehen, wenn die Methode *getArticlePrice()* der Schnittstelle *Article* für einen gegebenen Artikel `NULL` oder einen Verkaufspreis mit dem Betrag `0.0` liefert[2].
- Beachte: ein Eintrag dieses Typs hat keinen Effekt, wenn es einen Eintrag vom Typ `@CheckPrice4Consistency` mit dem Wert `1` gibt!
- Verwendet von der Methode *xOiProgInfo::getPricesOnRequest()*.

2.2. Property-bezogene Optionen

`@EPDFPropValPrefix`

- Das Wertfeld gibt denPräfix an, der den Bezeichnern von kaufmännischen Merkmalswerten (EPDF/OCD) vorangestellt werden soll, wenn sie während der Generierung der Choiceliste für die entsprechende OFML Property in ein OFML Symbol umgewandelt werden.
- Hintergrund: OFML Symbole dürfen nicht mit einer Ziffer beginnen.
- Der voreingestellte Präfix ist „S“. Es muss ein anderer Präfix verwendet werden, wenn es Merkmalswerte gibt, deren Bezeichner selber mit einem „S“ beginnt!
- Verwendet von der Methode *xOiProgInfo::getEPDFPropValPrefix()*.

`@ForceDynamicProperties`

- Wenn es einen Eintrag dieses Typs gibt und dieser den Wert `1` (wahr) hat, wird die Methode *forceDynamicProp()* der OFML Schnittstelle *Property* für alle Artikel dieser Serie angewendet.
- Hinweis: wenn die Methode für einen gegebenen Property-Key `1` (wahr) liefert, wird der Wert der Property in der Tabelle der dynamischen Properties gespeichert. Damit werden bewusst eventuell existierende set- und get-Methoden umgangen, um Konflikte mit Properties zu vermeiden, die vom globalen Produktdatenmanager dynamisch für konfigurierbare kaufmännische Merkmale (OCD) generiert werden.
- Beispiel: falls ein OFML Typ, der zur Repräsentation von Artikeln verwendet wird, die Methoden *setFoo()* und *getFoo()* implementiert, besteht eine gewisse Wahrscheinlichkeit, dass ein OCD-Datenanleger einem Artikel, der durch den Typ repräsentiert wird, ein Merkmal namens `Foo` zufügt. Der globale Produktdatenmanager generiert dann eine OFML Property mit dem Key `@Foo`. Wenn die Methode *forceDynamicProp()* dann nicht `1` für den Key `@Foo` liefert, würde das Setzen bzw. das Abfragen des Wertes der Property über die Methoden *setFoo()* und *getFoo()* erfolgen, was unter Umständen nicht dem Verwendungszweck der Methoden entspricht. Es wird deswegen empfohlen, für alle set-/get-Methodenpaare, die nicht für das Setzen bzw. das Abfragen des Wertes einer Property gedacht sind, die Methode *forceDynamicProp()* zu implementieren und für den entsprechenden Property-Key `1` zurückzugeben. Aus Gründen der Abwärtskompatibilität muss dieses Verhalten jedoch mittels eines Eintrags dieses Typs mit dem Wert `1` freigeschaltet werden.

[2] In der aktuellen Implementierung wird das „Etikett“ an den Artikellangtext angehängt.

- Verwendet in den Standardimplementierungen der Methoden *getPropValue()* und *setPropValue()* der OFML Schnittstelle *Property.*

@Property

- Verwendet während der Initialisierung einer Instanz von *OiProgInfo.*
- Definiert eine feste Eigenschaft des Objektes (im Gegensatz zu dynamisch generierten gemeinsamen Eigenschaften, siehe Abschn. 2.4)
- Das Wertfeld besitzt folgendes Format:
  `[prop_key, prop_def, position, init_value, state],`
  wobei die ersten 3 Vektorelemente die Parameter spezifizieren, die an die Methode *setupProperty()* der OFML Schnittstelle *Property* zu übergeben sind.

  Die Elemente `init_value` und `state` sind optional. Wenn `init_value` angegeben ist, wird der Wert der Eigenschaft nach *setupProperty()* als initialer Wert zugewiesen. Wenn `state` angegeben ist, wird der Status an die Methode *setPropState()* übergeben.

  Die folgenden Eigenschaftstypen können in `prop_def` verwendet werden: “s”, “b”, “i”, “u chi”, “f” und “u chf”.

@UseUnspecPropVal4OdbInfo

- Gibt an, ob Properties mit dem Wert `@UNSPECIFIED` in der ODB-Info Hash-Tabelle repräsentiert werden sollen (`1`) oder nicht (`0`). Default: `1`.
- Verwendet von der Methode *OiProgInfo::useUnspecPropVal4OdbInfo()*, die von der Hakenmethode *OiOdbPlElement::useUnspecPropVal4OdbInfo()* in *OiOdbPlElement::getBasicOdbInfo()* gerufen wird.

@UseVoidPropVal4OdbInfo

- Gibt an, ob Properties mit dem Wert `@VOID` in der ODB-Info Hash-Tabelle repräsentiert werden sollen (`1`) oder nicht (`0`). Default: `1`.
- Verwendet von der Methode *OiProgInfo::useVoidPropVal4OdbInfo()*, die von der Hakenmethode *OiOdbPlElement::useVoidPropVal4OdbInfo()* in *OiOdbPlElement::getBasicOdbInfo()* gerufen wird.

@NeedPropGroupDescriptions

- Gibt an, ob die (möglicherweise) für die aktuelle OFML-Instanz definierten Property-Gruppen im Eigenschaftseditor der OFML-Applikationen angezeigt werden sollen (`1`) oder nicht (`0`). Default: `1`.
- Verwendet von der Hakenmethode *OiProgInfo::needPropGroupDescriptions()*, die von der Standard-implementierung der Methode *getPropGroupDescriptions()* der OFML Schnittstelle *Property* gerufen wird.
- Die Property-Gruppen werden angezeigt, wenn die Steuerdatentabelle `proginfo` nicht existiert, oder wenn sie keinen Eintrag dieses Typs enthält, oder wenn der erste wirksame (gültige) Eintrag dieses Typs mit einem gültigen Wert im Wertfeld dort die ganze Zahl `1` enthält.
- Ein Eintrag dieses Typs ist wirksam, wenn das Argumentfeld leer ist oder alle im Argumentfeld angegebenen Bedingungen erfüllt sind.

Eine Bedingung ist als Vektor `[condition_type,condition_value]` anzugeben. Mehrere Bedingungen sind durch Komma zu trennen.

- Aktuell werden folgende **Bedingungstypen** (`condition_type`) unterstützt:

  `@Category`
  : ist wahr, wenn die Instanz einer der im Bedingungswert angegebenen Kategorien angehört (Vektor von Symbolen)

  `@CategoryNOT`
  : ist wahr, wenn die Instanz *nicht* einer der im Bedingungswert angegebenen Kategorien angehört (Vektor von Symbolen)

  `@Article`
  : ist wahr, wenn die Instanz einen Artikel repräsentiert[3] und dessen Grundartikelnummer mit einer der im Bedingungswert angegebenen Zeichenketten beginnt (Vektor von Zeichenketten)

  `@ArticleNOT`
  : ist wahr, wenn die Instanz einen Artikel repräsentiert[4] und dessen Grundartikelnummer *nicht* mit einer der im Bedingungswert angegebenen Zeichenketten beginnt (Vektor von Zeichenketten)

  `@Type`
  : ist wahr, wenn der Typ der Instanz einer der als Bedingungswert angegebenen Typen (Vektor von Zeichenketten) oder ein Untertyp von ihnen ist.

    Typen müssen vollständig qualifiziert sein (d.h., inklusive OFML Paketbezeichner).

  `@TypeNOT`
  : ist wahr, wenn der Typ der Instanz *nicht* einer der als Bedingungswert angegebenen Typen (Vektor von Zeichenketten) und auch kein Untertyp von ihnen ist.

    Typen müssen vollständig qualifiziert sein.

2.3. 2D Layer-Name

Die in diesem Unterabschnitt beschriebenen Informationstypen werden von der Methode *OiProgInfo::get2DLayer()* verwendet, welche den Namen des 2D-Layers liefert, das benutzt werden soll, falls in den 2D-Daten eines gegebenen Artikels kein explizites Layer angegeben ist. Die Methode erzeugt einen Layernamen, der sich aus den folgenden (optionalen) Komponenten zusammensetzt:

1. Präfix
2. OFML Hersteller-ID und/oder kaufmännische Hersteller-ID (normalerweise nur eins von beiden)
3. Trennzeichenkette
4. OFML Serien-ID und/oder kaufmännische Serien-ID (normalerweise nur eins von beiden)
5. Suffix

Welche dieser Komponenten im Layernamen enthalten sind, wird durch das Vorkommen der folgenden Informationstypen gesteuert:

[3] Eine Instanz repräsentiert einen Artikel, wenn die Klasse der Instanz die OFML-Schnittstelle *Article* implementiert und die durch die Methode *getArticleSpec()* gelieferte Grundartikelnummer keine leere Zeichenkette ist.
[4] s.o.

@2DLayerPrefix

- das Wertfeld enthält den Präfix für den Layernamen

@2DLayerSuffix

- das Wertfeld enthält den Suffix für den Layernamen

@2DLayerSeparator

- das Wertfeld enthält die Zeichenkette, die zur Trennung von Hersteller- und Serien-IDs verwendet werden soll

@2DLayerUseManufacturerID

- das Wertfeld enthält `1`, wenn die kaufmännische Hersteller-ID in den Layernamen aufgenommen werden soll

@2DLayerUseSeriesID

- das Wertfeld enthält `1`, wenn die kaufmännische Serien-ID in den Layernamen aufgenommen werden soll

@2DLayerUseManufacturer

- das Wertfeld enthält `1`, wenn die OFML Hersteller-ID in den Layernamen aufgenommen werden soll

@2DLayerUseSeries

- das Wertfeld enthält `1`, wenn die OFML Serien-ID in den Layernamen aufgenommen werden soll

2.4. Merkmalswertbildchen

@PropInfoPicPrefix

- Das Wertfeld gibt einen Präfix für den Bilddateinamen für alle Werte außer `@VOID` an (bzw. für alle Werte inklusive `@VOID`, wenn in der Tabelle kein Eintrag des Typs *@PropInfoPic4Void* enthalten ist, s.u.).
- Der Dateiname wird dann aus dem Präfix und der Zeichenkettendarstellung des Wertes (ohne das führende `@`-Zeichen) gebildet. Wenn die Property mit einem kaufmännischen Merkmal (OCD) verknüpft ist, wird der kaufmännische Wertbezeichner verwendet.
- Verwendet von der Methode *OiProgInfo::getPropInfo4Obj().*

@PropInfoPic4Void

- Das Wertfeld gibt den Bilddateinamen für den Wert `@VOID` an.
- Verwendet von der Method *OiProgInfo::getPropInfo4Obj().*

Es kann mehrere Tabelleneinträge der Typen *@PropInfoPic4Void* und *@PropInfoPicPrefix* geben. Welcher Eintrag zur Anwendung kommt, wird über die Bedingungen gesteuert, die im Argumentfeld (Feld 2) angegeben sind. Wenn das Feld leer ist, wird der Tabelleneintrag ohne Prüfung verwendet. Wenn Bedingungen angegeben sind, müssen alle erfüllt sein, damit der Tabelleneintrag wirksam wird. Es wird der erste passende Eintrag verwendet.

Jede Bedingung wird als Vektorpaar `[condition_type, condition_value]` angegeben. Bedingungen sind untereinander durch Kommata zu trennen. Aktuell werden folgende Bedingungstypen unterstützt:

`@PropKey`

ist wahr, wenn der Key der betreffenden Property mit einer der Zeichenketten beginnt, die als Bedingungswert angegeben sind.

Der Bedingungswert ist eine einzelne Zeichenkette oder ein Vektor von Zeichenketten.

`@PropKeyNOT`

ist wahr, wenn der Key der betreffenden Property NICHT mit einer der Zeichenketten beginnt, die als Bedingungswert angegeben sind.

Der Bedingungswert ist eine einzelne Zeichenkette oder ein Vektor von Zeichenketten.

`@PropName`

ist wahr, wenn die betreffende Property mit einem kaufmännischen Merkmal verknüpft ist und der Bezeichner dieses Merkmals mit einer der Zeichenketten beginnt, die als Bedingungswert angegeben sind.

Der Bedingungswert ist eine einzelne Zeichenkette oder ein Vektor von Zeichenketten.

`@PropNameNOT`

ist wahr, wenn die betreffende Property mit einem kaufmännischen Merkmal verknüpft ist und der Bezeichner dieses Merkmals NICHT mit einer der Zeichenketten beginnt, die als Bedingungswert angegeben sind.

Der Bedingungswert ist eine einzelne Zeichenkette oder ein Vektor von Zeichenketten.

`@PropValue`

ist wahr, wenn die Zeichenkettendarstellung des Merkmalswertes gleich einer der Zeichenketten ist, die als Bedingungswert angegeben sind.

Der Bedingungswert ist eine einzelne Zeichenkette oder ein Vektor von Zeichenketten.

Beispiele

1.

Benutze den Präfix für alle Properties, deren Key mit "Mat" beginnt, aber nicht für die Properties `@MatGroup` und `@MatType`:

```
@PropInfoPicPrefix;[@PropKey,"Mat"],[@PropKeyNOT,["MatGroup","MatType"]];::foo::bar::
```

2.

In einer Serie besitze der Merkmalswert "LG" unterschiedliche Bedeutungen in 2 verschiedenen Merkmale `OPTION1` und `OPTION2`. Deswegen soll für `OPTION1` der normale Präfix verwendet werden, für `OPTION2` aber ein spezieller Präfix:

```
@PropInfoPicPrefix;[@PropName,"OPTION2"];::foo::bar::OPTION2
@PropInfoPicPrefix;;::foo::bar::
```

Hinweis: Da immer der erste passende Eintrag verwendet wird, muss die Reihenfolge der beiden Einträge wie oben angegeben sein, um das gewünschte Ergebnis zu erzielen.

2.5. Gemeinsame Eigenschaften

Siehe auch Application Note zur Globalen Variantenkonfiguration.

@CommonProps4ProgInfo

- Gemeinsame Eigenschaften für das ProgInfo-Objekt selber werden nur dann generiert, wenn es einen Eintrag dieses Typs gibt und sein Wert `1` ist.
- Verwendet von der Methode *xOiProgInfo::getProperties().*

@NonPlElements4CommonProps

- Wenn es einen Tabelleneintrag dieses Typs gibt und sein Wert `1` ist, werden Artikel der Serie auch dann bei der Generierung von gemeinsamen Eigenschaften berücksichtigt, wenn sie sich nicht auf der obersten Planungsebene befinden.
- Hintergrund: Normalerweise werden gemeinsame Eigenschaften für eine Auswahl von Artikeln auf der obersten Planungsebene generiert. Einige Applikationen bzw. Applikationsmodi erlauben jedoch auch die Auswahl von Elementen auf unteren Planungsebenen. In diesen Situationen könnte es sinnvoll sein, auch diese Elemente in die Generierung von gemeinsamen Eigenschaften einzubeziehen.
- Verwendet von der Methode *xOiProgInfo::nonPlElements4CommonProps().*

@IgnorePClass4CommonProps

- Wenn es einen Tabelleneintrag dieses Typs gibt und sein Wert `1` ist, werden Property-Klassen von Artikeln der Serie bei der Generierung von gemeinsamen Eigenschaften nicht herangezogen.
- Hintergrund: Normalerweise erhalten Properties, die zwar denselben Bezeichner besitzen, aber zu verschiedenen Property-Klassen gehören, ein eigenes Pendant im Set der gemeinsamen Eigenschaften, da sie aus konzeptioneller Sicht unterschiedliche Properties darstellen. In bestimmten Datenanlagen besitzen die Property-Klassen jedoch keine spezielle Bedeutung. In diesem Fall kann es angebracht sein, eine gemeinsame Eigenschaft für Properties mit demselben Bezeichner aber verschiedenen Property-Klassen zu generieren.
- Verwendet von der Methode *xOiProgInfo::ignorePClass4CommonProps().*

@CommonProperty

- Das Wertfeld enthält eine Komma-separierte Liste von Property-Bezeichnern. Properties von Artikeln der Serie mit diesen Bezeichnern werden in das Set der gemeinsamen Eigenschaften aufgenommen.
- Der spezielle Bezeichner "ALL_PROPERTIES" deklariert alle Properties als gültig für das Set der gemeinsamen Eigenschaften.
- Verwendet von der Methode *xOiProgInfo::isCommonProperty2().*

@CommonPropPrefix

- Das Wertfeld enthält eine Komma-separierte Liste von Präfixen von Property-Bezeichnern. Properties von Artikeln der Serie, deren Bezeichner mit einem der angeführten Präfixe beginnen, werden in das Set der gemeinsamen Eigenschaften aufgenommen.
- Verwendet von der Methode *xOiProgInfo::isCommonProperty2().*

Es kann mehrere Tabelleneinträge der beiden Typen *@CommonProperty* und *@CommonPropPrefix* geben.

Wenn eine Property mit einem kaufmännischen Merkmal (OCD) verknüpft ist, muss in den Wertfeldern der beiden Informationstypen der Name (Präfix) des kaufmännischen Merkmals angegeben werden (anstelle des vom globalen Produktdatenmanager generierten Keys).

Hinweis:
Wenn eine spezifische, von *xOiProgInfo* abgeleitete Klasse verwendet wird, ist sicherzustellen, dass diese keine Implementierung der Methode *isCommonProperty()* besitzt. Diese Methode hätte keinen Effekt, da die in *xOiProgInfo* implementierte Methode *isCommonProperty2()* Vorrang besitzt.

2.6. Bestelllistengenerierung

@OrderInfo

- Einträge dieses Typs können verwendet werden, um das Erscheinungsbild einer generierten Bestellliste (Artikelliste) zu beeinflussen. (Die Einstellungen beeinflussen dabei aber nur die Artikel, die zu dieser OFML Serie gehören.)
- Verwendet von der Methode *xOiProgInfo::getOrderInfo()*, die während der Bestelllistengenerierung gerufen wird.
- Das Argumentfeld wird zur Festlegung benutzt, für welche Parameter der Methode das Wertfeld des Tabelleneintrags zu verwenden ist.

  Das Format des Argumentfelds ist ein Vektor aus `[parameter, value]` Paaren mit den möglichen Parametern `@OrderMode`, `@OrderDepth`, `@Region` und `@InfoType`:

  - Der Parameter `@OrderMode` spezifiziert den von der Applikation verwendeten Modus der Bestelllistengenerierung (normalerweise `@Standard`), während der Parameter `@OrderDepth` die maximale Tiefe der generierten Bestelllistenstruktur bestimmt (wobei `0` keine Einschränkung der Tiefe bedeutet).

    Diese Parameter sind optional. Wenn kein Wert für `@OrderMode` oder `@OrderDepth` angegeben ist, passt der Tabelleneintrag zu jedem Wert des entsprechenden Parameters der gerade stattfindenden Bestelllistengenerierung. Dies ist die normale Benutzungsform von Einträgen dieses Typs (da spezielle Parameter nur in speziellen Applikationen verwendet werden).

  - Der (optionale) Parameter `@Region` spezifiziert das Vertriebsgebiet und wird mit dem Wert verglichen, welcher in der Registrierungsdatei des OFML Paketes unter dem Schlüssel `distribution_region` hinterlegt ist. Wenn kein `@Region` Parameter angegeben ist, passt der Tabelleneintrag für alle Vertriebsgebiete.

  - Der Parameter `@InfoType` spezifiziert eine konkrete Option, die das Erscheinungsbild der Bestellliste beeinflusst. Die unterstützten Optionen und die entsprechenden möglichen Werte werden unten beschrieben. Die Angabe des `@InfoType` ist zwingend. Wenn es mehrere

Tabelleneinträge für einen gegebenen `@InfoType` mit unterschiedlichen Werten für `@OrderMode`, `@OrderDepth` und `@Region` gibt, wird derjenige Eintrag verwendet, der am besten zu den entsprechenden Parametern der gerade stattfindenden Bestelllistengenerierung bzw. zum Vertriebsgebiet des OFML Paketes passt. Gibt es mehrere, gleich gut passende Tabelleneinträge, so wird der erste gemäß der Reihenfolge in der Tabelle verwendet.

- Die unterstützten Optionen für den Parameter `@InfoType` und die entsprechenden möglichen Werte sind:
  - `@NeedSumOrder`: der ganzzahlige Wert (`0` – *nein*, `1` - *ja*) gibt an, ob identische Artikel (gleiche Konfiguration) zu einer gemeinsamen Position in der Bestellliste zusammengefasst werden sollen.

    Das Standardverhalten entspricht dem Wert `0`, d.h., es findet keine Zusammenfassung statt.

    Hinweis: Diese Option hat nur dann einen Effekt, wenn die Applikation die Zusammenfassung von identischen Artikeln auch tatsächlich erlaubt.

  - `@DeepSumOrder`: der ganzzahlige Wert (`0` – *nein*, `1` - *ja*) gibt an, ob bei der Zusammen-fassung identischer Artikel (s. Option `@NeedSumOrder` oben) Unterartikel berücksichtigt werden sollen: Im Fall von `0` (Standard) werden zwei an sich gleiche Artikel als nicht identisch betrachtet, wenn mindestens einer der beiden Unterartikel besitzt.

    Hinweis: Falls angegeben, muss der Parameter `@OrderMode` den Wert `@Standard` besitzen und der Parameter `@OrderDepth` den Wert `0`.

  - `@ArtSpecMode`: gibt an, welche Art der Artikelnummer in der Bestellliste erscheinen soll. Der Wert ist ein Vektor `[mode, max_length]` mit folgenden möglichen Werten für `mode`:

    `0` – Grundartikelnummer (Standard)
    `1` – Endartikelnummer

    Das Element `max_length` spezifiziert die maximale Anzahl von Stellen, die für die Ausgabe der Artikelnummer verwendet werden darf, wobei `NULL` (Standard) keine Einschränkung bedeutet.

  - `@ArtTextMode`: gibt an, welche Form des Artikeltextes in der Bestellliste verwendet werden soll. Die möglichen Werte sind:

    `0` – Kurztext
    `1` – Langtext (Standard)
    `2` – beide

  - `@NeedsAddPrices`: der ganzzahlige Wert (`0`,`1`) gibt an, ob eine ausführliche Information über die Zusammensetzung des Preises (z.B. inklusive möglicher Aufschläge) ausgegeben werden soll (`1`, Standard), oder ob nur der Endpreis des Artikels ausgewiesen werden soll (`0`).
    Ob die einzelnen Preiskomponenten (beim Wert `1`) tatsächlich auch angezeigt werden, hängt von der jeweiligen Applikation ab!

  - `@MaxOrderDepth`: der ganzzahlige Wert spezifiziert die maximal erlaubte Tiefe der Bestelllistenstruktur, d.h., mit dieser Option kann die von der Applikation vorgegebene maximale Tiefe der gerade stattfindenden Bestelllistengenerierung überschrieben werden.
    Der Wert `0` (Standard) bedeutet keine Einschränkung der Tiefe.

Beispiele:

```
@OrderInfo;[[@InfoType,@ArtSpecMode]];[1,NULL]
```

Gibt an, dass (anstelle der Grundartikelnummer) die Endartikelnummer auszugeben ist, ohne Beschränkung der Länge, und unabhängig von Modus und Strukturtiefe der gerade stattfindenden Bestelllistengenerierung.

```
@OrderInfo;[[@InfoType,@NeedsAddPrices],[@Region,"ANY"]];1
@OrderInfo;[[@InfoType,@NeedsAddPrices]];0
```

Gibt an, dass im Vertriebsgebiet `ANY` die einzelnen Preiskomponenten aufgelistet werden sollen, in den anderen Vertriebsgebieten aber nicht. Achtung: die Reihenfolge der beiden Einträge ist relevant, da der zweite Eintrag auch für das Vertriebsgebiet `ANY` passen würde (und bei umgekehrter Reihenfolge verwendet würde).

```
@OrderInfo;[[@InfoType,@NeedsAddPrices],[@Region,"ANY"]];0
```

Gibt an, dass im Vertriebsgebiet `ANY` die einzelnen Preiskomponenten *nicht* aufgelistet werden sollen. In allen anderen Vertriebsgebieten werden die Preiskomponenten aufgelistet (Standard).

`@OrderRemoveType, @OrderRemoveArticle, @OrderRemoveCategory, @OrderSumArticlesByType, @OrderSumArticlesByArticle, @OrderSumArticlesByCategory`

- Einträge dieser Typen geben an, welche Artikel aus der Bestellliste entfernt werden sollen bzw. für welche Artikel eine Zusammenfassung bei gleicher Konfiguration erfolgen soll.
- Verwendet von der Methode *xOiProgInfo::prepareOrder().*
- Hinweis: Die Informationstypen hinsichtlich Zusammenfassung identischer Artikel sind nur dann wirksam, wenn die Applikation eine derartige Zusammenfassung generell erlaubt.
- Weiterer Hinweis: Wenn die Option `@NeedSumOrder` im Eintrag des Typs *`@OrderInfo`* (s.o.) den Wert `1` besitzt (Standard!), so führt der Algorithmus der Bestelllistengenerierung nach den Aufrufen von *xOiProgInfo::prepareOrder()* noch eine generelle Zusammenfassung durch ohne Beschränkungen hinsichtlich der Artikel der Serie. Bei Verwendung der Informationstypen *`@OrderSumArticles*`* sollte deswegen sichergestellt werden, dass die Option `@NeedSumOrder` im Eintrag des Typs *`@OrderInfo`* den Wert `0` besitzt.
- Die betreffenden Artikel können anhand der sie repräsentierenden OFML Typen, anhand ihrer Grundartikelnummer oder anhand einer Kategorie bestimmt werden.
- Grundartikelnummern können unvollständig angegeben sein. Es werden dann alle Artikel erfasst, deren Grundartikelnummer mit der angegebenen Zeichenkette beginnen.
- Das Wertfeld all dieser Informationstypen kann eine Komma-separierte Liste von voll-qualifizierten OFML-Typbezeichnern, von Artikelnummern (Präfixen) bzw. von Kategorie-Bezeichnern enthalten.
- Es können jeweils mehrere Einträge eines Typs vorkommen.
- Die Anwendbarkeit eines Eintrags wird über die Bedingungen im Argumentfeld gesteuert. Wenn das Argumentfeld leer ist, wird der Tabelleneintrag ohne Prüfung verwendet. Wenn Bedingungen angegeben sind, müssen alle erfüllt sein, damit der Tabelleneintrag wirksam wird.

- Jede Bedingung wird als Vektorpaar `[condition_type,condition_value]` angegeben. Bedingungen sind untereinander durch Kommata zu trennen.
- Aktuell werden folgende Bedingungstypen (`condition_type`) unterstützt:

  `@OrderMode`

  ist wahr, wenn der Modus der gerade stattfindenden Bestelllistengenerierung gleich dem angegebenen Wert ist (siehe Parameter `@OrderMode` oben bei Informationstyp *`@OrderInfo`*).

  `@OrderDepth`

  ist wahr, wenn die maximal erlaubte Hierarchietiefe der gerade stattfindenden Bestelllisten-generierung gleich dem angegebenen Wert ist (siehe Parameter `@OrderDepth` oben bei Informationstyp *`@OrderInfo`*).

  `@LevelGT`

  ist wahr, wenn sich die betreffende Bestellposition auf einer Ebene befindet, die gleich oder höher als der angegebene Wert ist (höchste Ebene ist `1`).

`@OrderPackFolderLabel`

- Spezifiziert die Bezeichnung des Ordners in der Bestellliste, in den generierte Pack-Artikel eingefügt werden sollen. Ausführliche Informationen siehe Application Note zur Generierung von Pack-Artikeln.
- Der Generierungsprozess für die Pack-Artikel sucht die aktuelle Planungshierarchie nach Elementen der OFML Serie ab, die zu so genannten *Pack-Artikeln* zusammengefasst werden sollen.
- Der Generierungsprozess für die Pack-Artikel selber wird über Tabelleneinträge des Typs *`@OrderPackArticle`* gesteuert (s.u.), aber wenn es keinen Eintrag des Typs *`@OrderPackFolderLabel`* gibt, findet die Generierung der Pack-Artikel (für den gegebenen Hersteller) gar nicht statt.
- Alle OFML Serien eines gegebenen Herstellers sollten denselben Bezeichner verwenden, da für jeden Hersteller nur ein Pack-Ordner angelegt wird.
- Das Wertfeld kann eine Text-Ressource-ID enthalten.
- Verwendet von der Methode xOiProgInfo::*prepareOrder().*

`@OrderPackArticle`

- Tabelleneinträge dieses Typs steuern den Generierungsprozess für die Pack-Artikel (s.a. Informationstyp *`@OrderPackFolderLabel`* oben).
- Das Wertfeld enthält einen Vektor, der für jede mögliche Anzahl von Artikeln, die zu einem Pack zusammengefasst werden können, die jeweilige Packartikelnummer angibt. Dabei gibt das erste Vektorelement die Artikelnummer an, die für ein Pack, bestehend aus einem Artikel, verwendet wird, das zweite Element die Artikelnummer für ein Pack, das aus 2 Artikeln besteht, usw.

  Wenn es für eine gegebene Anzahl von Artikeln keinen Pack-Artikel gibt, muss das entsprechende Vektorelement eine leere Zeichenkette enthalten, wobei das letzte Vektorelement *keine* leere Zeichenkette sein darf.

  Es können somit Schemata der Pack-Generierung realisiert werden, die nicht aufeinander folgende Packgrößen-Nummern umfassen, z.B. Packs zu 3 und 7 Artikeln.

- Im Argumentfeld werden OFML Klasse, Artikelparameter und/oder der ODB Type der Artikel angegeben, die Gegenstand der Pack-Generierung sind. Sie müssen in der allgemeinen Form für Bedingungen angegeben werden, wie oben für die Informationstypen `@OrderRemove*` beschrieben. Die entsprechenden Bedingungstypen und zugehörigen Werte sind:

  `@Class`

  ist wahr, wenn der Name der Klasse der gegebenen Artikelinstanz, welcher von der Methode *getClass()* geliefert wird, gleich dem als Bedingungswert angegeben Namen ist.

  `@ArticleParams`

  ist wahr, wenn der Code für die Artikelparameter, der von der Methode *getArticleParams()* der gegebenen Artikelinstanz geliefert wird, gleich dem als Bedingungswert angegeben Code ist.

  `@ODBType`

  ist wahr, wenn der ODB Type der gegebenen Artikelinstanz (falls vorhanden) gleich dem als Bedingungswert angegeben Typ ist.

  Die Angabe der Klasse ist gefordert, die anderen Angaben sind optional.

  Zusätzlich können bei Bedarf die Bedingungen wie für das Entfernen von Bestelllisten-Positionen angegeben werden (s.o. bei `@OrderRemove*`).

- Verwendet von der Methode *xOiProgInfo::prepareOrder().*

3. Tabelle *plelement*

Diese Tabelle wird verwendet, um das Verhalten der Objekte zu steuern, die einen Artikel repräsentieren. Die Tabelle wird vom OFML Basistyp *OiPlElement* eingeführt und wird auch in den abgeleiteten Klassen *OiOdbPlElement*, *xOiPlElement* sowie *xOiBTGPlElement3* verwendet. Einige Informationstypen werden nur von bestimmten der genannten abgeleiteten Klassen verwendet. Damit sind nicht alle der unten beschriebenen Informationstypen auch in allen OFML Serien sinnvoll/anwendbar. Dies hängt von den OFML Typen ab, die den Artikeln der Serie zugeordnet sind (siehe OAM).

@Article4PropTitle

- Legt die Art der Verwendung der Artikelnummer in der Kopfzeile des Eigenschaftseditors fest.
- Verwendet von der Methode *OiPlElement::getPropTitle().*
- Das Wertfeld muss das Format `[spec_type,max_length]` besitzen, wobei `spec_type` entweder

  `@Base` für die Verwendung der Grundartikelnummer, oder

  `@Final` für die Verwendung der Endartikelnummer

  sein kann und `max_length` eine ganze Zahl ist, welche die Anzahl der maximal erlaubten Zeichen angibt (`0` für keine Einschränkung).

  Standard: `[@Base,0]`

@ArtText4PropTitle

- Legt die Art der Verwendung des Artikeltextes in der Kopfzeile des Eigenschaftseditors fest.
- Verwendet von der Methode *OiPlElement::getPropTitle().*
- Das Wertfeld kann folgende Werte enthalten:

  `@s` für die Verwendung des Kurztextes

  `@l` für die Verwendung des Langtextes.

  Standard: `@s`

Die von der Methode *OiPlElement::getPropTitle()* erzeugte Zeichenkette für die Kopfzeile des Eigenschaftseditors setzt sich aus der spezifizierten Artikelnummer (*@Article4PropTitle*) und dem spezifizierten Artikeltext (*@ArtText4PropTitle*) zusammen, getrennt durch ein Leerzeichen.

@NoAddAsSibling

- Die Standardimplementierung des Basistyps *OiOdbPlElement* erlaubt das Einfügen eines neuen Planungselementes neben ein selektiertes Kind-Objekt einer Instanz von *OiOdbPlElement*, z.B. neben ein Accessoire auf einem Tisch. In bestimmten Planungssituationen ist dieses Verhalten jedoch nicht angebracht. Tabelleneinträge des Typs *@NoAddAsSibling* können dann verwendet werden, die Planungssituationen anzugeben, in denen das Einfügen neben ein selektiertes Kind-Objekt einer Instanz von *OiOdbPlElement* **nicht** erlaubt ist.
- Eine Planungssituation wird durch ein Set von Bedingungen im Argumentfeld beschrieben. (Der Wert im Feld 3 muss immer die ganze Zahl `1` sein.)
  Eine Bedingung ist als Vektor `[condition_type,condition_value]` anzugeben. Mehrere Bedingungen sind durch Komma zu trennen. Ein Tabelleneintrag hat Gültigkeit, wenn alle im Argumentfeld angegebenen Bedeutungen erfüllt sind. Wenn keine Bedingungen angegeben sind, ist der

Tabelleneintrag immer wirksam.

- Die folgenden *Begriffe* werden in den nachfolgenden Beschreibungen der möglichen Bedingungstypen verwendet:
  - *Element*: das in die Planung einzufügende Element
  - *Implizite Instanz*: das Objekt (Instanz von *OiOdbPlElement*), das der Vater des neuen Planungselementes sein wird
  - *Referenzobjekt*: das selektierte Kind-Objekt der impliziten Instanz (neben dem das neue Element platziert wird)
- Aktuell werden folgende **Bedingungstypen** (`condition_type`) unterstützt:

  `@ElType`

  ist wahr, wenn der Typ des neuen Elements einer der als Bedingungswert angegebenen Typen (Vektor von Zeichenketten) oder ein Unter-Typ von ihnen ist.

  Typen müssen vollständig qualifiziert sein (d.h., inklusive OFML Paketbezeichner).

  `@SelfType`

  ist wahr, wenn der Typ der impliziten Instanz einer der als Bedingungswert angegebenen Typen (Vektor von Zeichenketten) oder ein Unter-Typ von ihnen ist.

  Typen müssen vollständig qualifiziert sein.

  `@SelfCategory`

  ist wahr, wenn die implizite Instanz einer der im Bedingungswert angegebenen Kategorien angehört (Vektor von Symbolen).

  `@SelfArticle`

  ist wahr, wenn die Klasse der impliziten Instanz die Schnittstelle *Article* implementiert und wenn seine Grundartikelnummer mit einer der im Bedingungswert angegebenen Zeichenketten beginnt (Vektor von Zeichenketten).

  `@RefType`

  ist wahr, wenn der Typ des Referenzobjektes einer der als Bedingungswert angegebenen Typen (Vektor von Zeichenketten) oder ein Unter-Typ von ihnen ist.

  Typen müssen vollständig qualifiziert sein.

  `@RefCategory`

  ist wahr, wenn die Klasse des Referenzobjektes einer der im Bedingungswert angegebenen Kategorien angehört (Vektor von Symbolen).

  `@RefArticle`

  ist wahr, wenn die Klasse des Referenzobjektes die Schnittstelle *Article* implementiert und wenn seine Grundartikelnummer mit einer der im Bedingungswert angegebenen Zeichenketten beginnt (Vektor von Zeichenketten).
- Verwendet von der Methode *OiOdbPlElement::doNotAddAsSibling().*

`@ShowAddAttPts`:

- Steuert, ob zusätzliche (nicht-standardisierte) Anfügepunkte angezeigt werden sollen.
- Verwendet in der *PICK* Regel von *xOiPlElement.*

- Zusätzliche Anfügepunkte werden angezeigt, wenn die Steuerdatentabelle `plelement` nicht existiert, oder wenn sie keinen Eintrag dieses Typs enthält, oder wenn der erste wirksame Eintrag dieses Typs im Wertfeld die ganze Zahl 1 enthält.
- Ein Eintrag dieses Typs ist wirksam, wenn alle im Argumentfeld angegebenen Bedingungen erfüllt sind. Eine Bedingung ist als Vektor `[condition_type,condition_value]` anzugeben. Mehrere Bedingungen sind durch Komma zu trennen.
- Aktuell werden folgende **Bedingungstypen** (`condition_type`) unterstützt:

  `@CountGT`
  : ist wahr, wenn die Anzahl der aktuell definierten zusätzlichen Anfügepunkte größer als die im Bedingungswert angegebene Zahl ist.

  `@Category`
  : ist wahr, wenn die Klasse der Artikelinstanz einer der im Bedingungswert angegebenen Kategorien angehört (Vektor von Symbolen)

  `@Article`
  : ist wahr, wenn die Grundartikelnummer der Artikelinstanz mit einer der im Bedingungswert angegebenen Zeichenketten beginnt (Vektor von Zeichenketten)

  Ein leeres Argumentfeld ist identisch mit der Bedingung `[@CountGT,0]`.

`@FixDimensions`

- Verwendet während der Initialisierung von Instanzen von *xOiBTGPlElement3.*
- Das Wertfeld enthält eine ganze Zahl für die Membervariable *mXoiFixDimensions*, welche spezifiziert, ob die Felder "width", "depth", "height" und "ins_angle" in der Tabelle `btgmlist` als Fließ-kommazahlen angegeben sind. Wenn der Wert `0` ist (nein), dann sind diese Felder als Zeichenkettenfelder definiert und können symbolische Namen enthalten, die durch den jeweils aktuellen Wert eines entsprechenden Merkmals ersetzt wird.

`@UseVarKeys`

- Verwendet während der Initialisierung von Instanzen von *xOiBTGPlElement3.*
- Das Wertfeld enthält eine ganze Zahl für die Membervariable *mXoiUseVarKeys*, welche spezifiziert, ob so genannte *Varkeys* in einer zusätzlichen Tabelle `var` definiert sind und als Variantencode im XCF-Katalog sowie im Feld "variant" der Tabelle `btgmlist` verwendet werden (`1` für *Ja*, `0` für *Nein*).

`@UseOAM`

- Verwendet während der Initialisierung von Instanzen von *xOiBTGPlElement3.*
- Das Wertfeld enthält eine ganze Zahl für die Membervariable *mXoiUseOAM*, welche spezifiziert, ob OAM (anstelle der Tabellen `bgtmlist` und `mat`) verwendet wird, um zusätzliche ODB Parameter anzugeben (`1` für *Ja*, `0` für *Nein*).

@AppInteractorDefs

- Verwendet in der Standardimplementierung der Methode *getAppInteractorDefs().*
- Details zur Verwendung siehe Application Note AN-2013-001.

4. Tabelle *anyarticle*

Die Einträge in dieser Tabelle ermöglichen es, die Erzeugung und das Erscheinungsbild der Geometrie-Attrappe von Instanzen des Basistyps *xOiAnyArticle* zu steuern, der zur Darstellung von Artikeln ohne eine spezifische grafische Repräsentation verwendet wird.

@GeoCreationMode

spezifiziert, ob/wann eine Geometrie-Attrappe erzeugt werden soll:

`1` – erzeuge immer eine Geometrie-Attrappe

`0` – erzeuge eine Geometrie-Attrappe nur, wenn der Vater das Wurzelobjekt der Planung ist.

Standard: `1`

@GeoType

spezifiziert den OFML Typ der Geometrie-Attrappe.

Standard: `OiBlock`

@GeoArgs

spezifiziert die Parameter für die *initialize()* Funktion des OFML Typs im Eintrag des Typs *@GeoType*.

Standard (wenn es keinen Eintrag des Typs *@GeoType* gibt): `[1,1,1]`

@GeoMat

spezifiziert das Materialsymbol für die Geometrie (aufzulösen via *OiProgInfo::getMatName()*).

Standard: weiss

@CreateText

spezifiziert, ob ein Text neben der Geometrie-Attrappe angezeigt werden soll.

Values: `0` (nein), `1` (ja).

Aktuell besteht der Text aus dem Artikelkurztext des repräsentierten Artikels.

Standard: `0`.

@TextPos

spezifiziert die (lokale) Position für die Text-Primitive.

Standard: rechte untere hintere Ecke des Begrenzungsvolumens der Geometrie-Attrappe.

@TextMat

spezifiziert den voll qualifizierten Materialnamen für die Text-Primitive.

Standard: weiss

@TextAlignment

spezifiziert die Art der Textausrichtung.

Werte entsprechend *Mtext::setAlignment()* (siehe OFML Spezifikation).

Standard: `@LEFT`

5. Tabelle *epdfproductdb*

Mit dieser Tabelle können einige Aspekte des Verhaltens der EPDF/OCD Produktdatenbank gesteuert werden, die für die gegebene OFML Serie registriert ist. Der konkrete Typ der Produktdatenbank wird in der DSR Registrierungsdatei der OFML Serie unter dem Schlüssel `productdb` festgelegt und hängt vom konkreten Format ab, das für die kaufmännischen Daten der Serie verwendet wurde, als auch - im Fall von OCD - von der zu verwendenden Implementierung (native vs. nicht-native). Ein Teil der Optionen ist nur in der nativen OCD-Implementierung verfügbar (s. Abschn. 5.3).

Wenn sich die Tabelle nicht im regulären Datenverzeichnis der betreffenden OFML Serie befindet, wird sie im Verzeichnis gesucht, das in der DSR-Registrierungsdatei der Serie unter dem Schlüssel `proginfodb_path` angegeben ist.

Der Name der Tabelle ist vom historisch älteren, proprietären Datenformat EPDF abgeleitet, das den Vorgänger von OCD darstellt, dem OFML Standard für kaufmännische Daten.

5.1. Allgemeine Optionen

`@SafePropertyNames`

Spezifiziert, ob die Bezeichner von OCD-Merkmalen der Serie standard-konform sind, wobei `1` *Ja* bedeutet, und `0` *Nein*. Standard: `0`.

Laut der OCD–Spezifikation dürfen für Bezeichner von OCD–Merkmalen nur alphanumerische Zeichen inklusive dem Unterstrich verwendet werden, wobei das erste Zeichen kein numerisches sein darf. Außerdem dürfen keine reservierten Schlüsselwörter verwendet werden. (Zu den in der OCD–Spezifikation genannten Schlüsselwörtern kommen in OFML–basierten Applikationen noch die reservierten OFML–Schlüsselwörter, s.a. Anhang A.1.)

Wenn die OCD–Daten aus einem externen ERP–System konvertiert werden, kann diese Anforderung u.U. jedoch nicht erfüllt werden. Die OFML–Applikationen versuchen deswegen (im Modus `0`), auch OCD–Merkmale, deren Bezeichner nicht Standard-konform sind, robust zu behandeln. Dazu werden bei der Umwandlung eines Bezeichners eines OCD–Merkmals in die ID der zugehörigen OFML–Property folgende Ersetzungen vorgenommen[5]:

- Ist der Merkmalsbezeichner ein OFML–Schlüsselwort, wird das Zeichen '_' vorangestellt.
- Beginnt der Merkmalsbezeichner mit einer Ziffer, wird der Buchstabe 'S' (für „Symbol“) vorangestellt.
- Nicht-alphanumerische Zeichen werden durch eine Sequenz "_XX" ersetzt, wobei XX die Hexadezimal-Darstellung des Zeichens ist.
- Das Zeichen '_' wird durch "__" ersetzt.

Bei der Bestimmung des zugehörigen OCD–Merkmalsbezeichners zu einer gegebenen OFML–Property-ID werden dann die entsprechenden umgekehrten Modifikationen vorgenommen.

[5] Diese Modifikationen müssen auch bei der ODB—Datenanlage berücksichtigt werden, denn dort müssen OFML-Property-ID's verwendet werden.

Wird bei an sich standard-konformen OCD–Merkmalsbezeichnern für diese Option *nicht* der Wert `1` angegeben, so kann es zu folgenden Problemen kommen[6]:

- Beginnt ein OCD-Merkmalsbezeichner mit dem Buchstaben 'S', gefolgt von einer Ziffer, so wird zwar eine entsprechende OFML-Property-ID verwendet, z.B. `@S2SNA`, rückwärts daraus aber der (nicht-existente) OCD-Merkmalsbezeichner "2SNA" ermittelt (mit der Konsequenz, dass das Merkmal durch den Anwender nicht verändert werden kann).
- Enthält ein OCD-Merkmalsbezeichner Unterstriche, so sind diese in der zugehörigen OFML-Property-ID durch "__" ersetzt (z.B. "_2SNA" => `@__2SNA`). Das muss dann bei der ODB—Datenanlage berücksichtigt werden, d.h., dort kann der OCD-Merkmalsbezeichner *nicht* eins-zu-eins verwendet werden.

@PreselectRestrictable

Spezifiziert das Verhalten in Bezug auf die automatische Wertvorbelegung von einschränkbaren Merkmalen. Ohne eine automatische Wertvorbelegung ist der (initiale) Zustand (Wert) eines einschränkbaren Merkmals „nicht spezifiziert“ („???“).

`1`
: Wertvorbelegung von einschränkbaren Merkmalen mit einem als Standard markierten (und in der aktuellen Konfiguration gültigen) Wert

`2`
: Wertvorbelegung von allen einschränkbaren Merkmalen (es wird der erste gültige Wert verwendet, wenn es keinen als Standard markierten und in der aktuellen Konfiguration gültigen Wert gibt)

`3`
: keine Wertvorbelegung

Das Standardverhalten entspricht `1`.

Hinweis: Ein bewertetes einschränkbares Merkmal behält seinen Wert bei Wertänderungen von anderen Merkmalen solange der Wert in der jeweils aktuellen Konfiguration gültig bleibt und nicht durch Produktbeziehungen überschrieben wird. Wird der aktuelle Wert ungültig, so wird der Wert des Merkmals neu bestimmt gemäß dem mit dieser Option festgelegten Verfahren.

@ObligatoryPropCheck

Gibt an, ob nicht-spezifizierte Pflichtmerkmale den Artikel als inkonsistent markieren, wobei `1` *Ja* bedeutet, und `0` *Nein*. Standard: `0`.

Hinweis:
Ein inkonsistenter Artikel kann/darf nicht bestellt werden. Deshalb sollte der Wert `1` nur nach reiflicher Überlegung verwendet werden. Ein Pflichtmerkmal kann dann nicht-spezifiziert sein, wenn alle hinterlegten Werte in der aktuellen Konfiguration nicht gültig sind. Im Normalfall stellt das einen Fehler in den kaufmännischen Daten dar, da ein Pflichtmerkmal in jeder möglichen Konfiguration mindestens einen gültigen Wert besitzen sollte. Der Wert `1` führt dazu, dass der Anwender den Artikel nicht bestellen kann, und der Hersteller/Datenanleger ist dann darauf angewiesen, dass sich der Anwender wegen der Angelegenheit beim Hersteller meldet, um auf den Fehler aufmerksam zu werden.

Der Wert `1` ist nur in einigen komplexen Konstellationen bei der Datenanlage sinnvoll, wo der Aufwand zu groß wäre, für alle Pflichtmerkmale in allen möglichen Konfigurationen mindestens einen gültigen

[6] was also durch die explizite Angabe des Wertes `1` vermieden werden sollte

Wert zuzusichern. Der Datenanleger nimmt dann bewusst derartige inkonsistente Zwischenzustände in Kauf. Für den Anwender kann es dennoch schwierig sein zu erkennen, dass dies ein Zwischenzustand ist und wie er aus diesem herauskommt. Deswegen sollte der Datenanleger dann die ab OCD 4.0 gegebene Möglichkeit der Ausgabe einer Nachricht an den Anwender in Betracht ziehen.

@MultiplyPricingFactors

Spezifiziert, ob mehrere Preisfaktoren, die während einer Preisbeziehung einer Variantenkondition zugewiesen werden, multipliziert werden sollen, wobei `1` *Ja* bedeutet, und `0` *Nein*. Bei `0` wird nur der zuletzt zugewiesene Faktor verwendet. Standard: `1`.

Hinweis:
Diese Option ist nur für die OCD Versionen 2.0, 2.1 und 3.0 relevant. In den Formatversionen 2.2 und 4.0 entspricht das vorgeschriebene Verhalten dem Wert `0`, da die Zuweisung mehrerer Preisfaktoren an eine Variantenkondition in einer Preisbeziehung wahrscheinlich nicht wirklich vom Datenanleger beabsichtigt ist, und da andererseits der gewünschte Endfaktor genauso gut auch mit einer Zuweisung bewerkstelligt werden kann.
(Der Standardwert für die OCD Versionen 2.0, 2.1 und 3.0 ist `1` aus historischen Gründen und zur Wahrung der Abwärtskompatibilität.)

@InsignificantPropClasses

Gibt an, ob Merkmalsklassen zur Qualifizierung von Merkmalen verwendet werden, wobei `1` *Nein* bedeutet, und `0` *Ja*. Standard: `0`.

Der Wert sollte auf `1` gesetzt werden, wenn die OCD-Daten aus einem ERP-System exportiert werden, das das Konzept der Merkmalsklasse nicht kennt, und wo durch die Export-Routine willkürliche Merkmalsklassen generiert werden. Wird die Option in diesen Fällen nicht auf `1` gesetzt, können gespeicherte Projekte nach der Installation neu exportierter Daten nicht aktualisiert werden, auch wenn sich an den Daten im ERP-System nichts Signifikantes geändert hat.

@GenerateLimits4NumProps

Spezifiziert, ob min/max-Grenzwerte für numerische Merkmale generiert werden sollen, wenn keine Grenzen durch einen Interval-Wert selber definiert sind, wobei `1` *Ja* bedeutet, und `0` *Nein*.
Standard: `0`.

Die Grenzwerte werden basierend auf der für das Merkmal festgelegten Stellenanzahl bestimmt. Wenn für ein ganzzahliges Merkmal z.B. eine Stellenanzahl von `3` festgelegt ist, so sind die entsprechenden Grenzwerte `-99` (min) und `999` (max). Diese Option kann also dazu genutzt werden, die Eingabe eines unzulässigen Wertes durch den Anwender zu verhindern.

@CompleteEAN

Spezifiziert, ob die Endartikelnummer alle Informationen enthält, um die aktuelle Konfiguration wieder korrekt herstellen zu können, wobei `1` *Ja* bedeutet, und `0` *Nein*. Standard: `0`.

Diese Option ist nur bei der Verwendung von nutzerdefinierten Kodierungsschemata (OCD) relevant[7]. Wenn der Wert 1 ist, wird die Endartikelnummer benutzt, um die aktuelle Konfiguration eines Artikels zu beschreiben und zu speichern, anderenfalls (bei Wert `0`) wird dazu ein erweiterter Variantencode generiert und verwendet. Der Wert `1` sollte nur dann verwendet werden, wenn sicher ist, dass die Endartikelnummern aller Artikel der Serie die konfigurierbaren Merkmale jeweils vollständig kodieren.

5.2. Kodierungsschemata

Die folgenden Optionen können zur Angabe eines Kodierungsschemas für die Endartikelnummer (inklusive des Variantencodes) verwendet werden. Im Fall einer OCD Produktdatenbank werden diese Optionen allerdings nur dann herangezogen, wenn für einen gegebenen Artikel in den OCD Daten selber kein Kodierungsschema hinterlegt ist. (Es wird empfohlen, die entsprechende OCD Tabelle zu verwenden, anstatt dieser Steuerdatentabelle.)

Konzeptionell setzt sich die Endartikelnummer aus der Grundartikelnummer, einer Trennzeichenkette und dem Variantencode zusammen, der die aktuelle Konfiguration des Artikels beschreibt.

`@VarCodeType`

Spezifiziert den Typ des Variantencode. Aktuell werden folgende Typen unterstützt:

`@Complete`

Der Variantencode enthält ID von Merkmal **und** Wert für alle aktuell gültigen Merkmale. Jedes Merkmal wird in der Form `prop_class.property=prop_value` repräsentiert. Die einzelnen Merkmalsrepräsentationen werden durch Semikolon getrennt.

Das ist der Standardtyp.
Er entspricht dem vordefinierten OCD Kodierungsschema `KeyValueList`.

`@ValuesOnly`

Der Variantencode enthält nur die IDs der Werte aller aktuell gültigen Merkmale. Die Werte können untereinander durch einen Separator getrennt werden (s.u. bei *`@VarCodeValueSep`*).

Dieser Typ entspricht dem vordefinierten OCD Kodierungsschema `ValueList`.

`@VarCodeMode`

Modifiziert das Verhalten in Bezug auf den festgelegten Typ des Variantencodes.

Für den Typ `@Complete` ist aktuell kein spezieller Modus definiert.

Für den Typ `@ValuesOnly` bestimmt der Modus den Umfang der Merkmale, die im Variantencode repräsentiert werden sollen. Die möglichen Werte sind:

- `0` - nur die aktuell gültigen Merkmale (Standard)
- `1` - alle konfigurierbaren Merkmale

[7] Auch für nutzerdefinierte Kodierungsschemata kann diese Option inzwischen als obsolet betrachtet werden, da ab den Releases der pCon-Applikationen im März 2010 der Hersteller-unabhängige, sogenannte OFML-VarCode zur Speicherung und Wiederherstellung einer Artikelkonfiguration verwendet wird. Für Projekte, die mit älteren Versionen gespeichert wurden, bzw. für Projekte, wo per Spezial-Programmierung Hersteller-spezifische Variantencodes bzw. Endartikelnummern gespeichert und zugewiesen werden, sollte diese Option aber noch weiter gepflegt werden.

@SpecSeparator

Gibt die Zeichenkette an, die zum Trennen von Grundartikelnummer und Variantencode in der Endartikelnummer verwendet werden soll.

Standard: ein Leerzeichen.

@VarCodeValueSep

Gibt die Zeichenkette an, die zum Trennen der Merkmalswerte in Variantencodes des Typs `@ValuesOnly` verwendet werden soll.

Standard: ein Leerzeichen.

@VoidValVarCodeChar

Gibt das Zeichen an, das zur Repräsentation von nicht-spezifizierten optionalen Merkmalen in Variantencodes des Typs `@ValuesOnly` verwendet werden soll.

Im Variantencode werden so viele Instanzen des angegebenen Zeichens platziert, wie durch die für das betreffende Merkmal festgelegte Stellenanzahl bestimmt ist.

Standard: `'X'`.

@InvalidVarCodeChar

Gibt das Zeichen an, das zur Repräsentation von aktuell nicht gültigen Merkmalen in Variantencodes des Typs `@ValuesOnly` im Modus `1` (see *@VarCodeMode*) verwendet werden soll.

Im Variantencode werden so viele Instanzen des angegebenen Zeichens platziert, wie durch die für das betreffende Merkmal festgelegte Stellenanzahl bestimmt ist.

Standard: `'-'`.

@FixVarCodeValLen

Spezifiziert für Variantencodes des Typs `@ValuesOnly`, ob die Repräsentationen der Werte eine feste Länge haben sollen entsprechend der für die Merkmale festgelegten Stellenanzahlen, wobei `1` *Ja* bedeutet, und `0` *Nein*.

Im Fall von `0` werden Leerzeichen weg gekürzt.

Standard: `0`, wenn der Wert-Separator (siehe *@VarCodeValueSep*) keine leere Zeichenkette ist, sonst `1`.

5.3. Spezifische Optionen der nativen OCD-Implementierung

@AllowConsecValsInTrimmedCode

Diese Option bezieht sich auf die Verarbeitung von Endartikelnummern (zur Wiederherstellung der kodierten Artikelvariante), die gemäß eines nutzerdefinierten Schemas kodiert sind und bei denen das Trim-Flag gesetzt ist (d.h., Leerzeichen am Ende eines Merkmalswertes werden entfernt). Derartige Endartikelnummern können im Standardverfahren nicht verarbeitet werden, wenn in der Endartikelnummer zwei Merkmale unmittelbar aufeinander folgen (d.h., nicht durch Trennzeichen separiert sind). Mit dieser Option kann ein erweitertes Verfahren freigeschaltet werden, welches es – auf Kosten der Performanz – ermöglicht, auch derartige Endartikelnummern mit unmittelbar aufeinander folgenden Merkmalen zu verarbeiten.

`0` (oder `false`)
: Das erweiterte Verfahren soll *nicht* verwendet werden.

  Dies ist das Standardverhalten (wenn es keinen Tabelleneintrag für diese Option gibt).

`1` (oder `true`)
: Das erweiterte Verfahren soll verwendet werden.

@DeactivateValuePreconds

Mit dieser Option kann die Auswertung von Werte-Vorbedingungen generell unterbunden werden.

Der Wert der Option (3. Feld) wird dabei wie eine OCD-Bedingung (logischer Ausdruck) behandelt und ausgewertet[8]. Unter anderem ist es also möglich, Merkmale zu referenzieren.

Bsp.: `@DeactivateValuePreconds;;SONDER = '1'`

Die Bedingung muss dabei eindeutig erfüllt sein, damit die Deaktivierung der Vorbedingungen wirksam wird[9].

Die Option zielt auf ein sehr konkretes Szenario ab: Gegeben sei ein Artikel mit vielen Merkmalen, die wiederum zum Teil sehr große Wertemengen besitzen. Durch Einstellung eines speziellen Merkmals (im Beispiel oben `SONDER`) soll der Artikel in einer Sonderausführung bestellt werden können, wofür alle Werte auswählbar sein müssen/sollen. Dazu müssten alle Werte-Vorbedingungen um die Teil-Bedingung `SONDER <> '1'` ergänzt werden. Das ist bei der immensen Wertemenge nicht nur in der Datenanlage aufwändig, sondern würde auch bei der Auswertung zur Laufzeit zu einer schlechten Performanz führen. Mit dieser Option kann das vermieden werden.

@IgnoreUnknownPropInKeyValueVC

Diese sehr spezielle Option wird nur in ganz seltenen Fällen benötigt, und zwar dann, wenn im Rahmen eines Online-Shop-Systems ein über die Schnittstelle des OnlineConfigurators erzeugter Artikel mittels eines aus dem führenden ERP-System generierten Variantencodes initialisiert werden soll, und dieser Variantencode nach dem OCD-Schema *KeyValueList* aufgebaut ist, darin aber die Merkmale nicht mit einer Merkmalsklasse qualifiziert sind. In dem Fall erwartet die OCD-Implementierung, dass ein im Variantencode enthaltenes Merkmal in den (aktuellen) Merkmalsklassen des betreffenden Artikels

[8] Die Bedingung wird vor der Auswertung des Beziehungswissens des bearbeiteten Artikels ausgewertet.
[9] Mehr zu undefinierten logischen Ausdrücken siehe OCD-Spezifikation.

enthalten ist. Sobald dies für ein Merkmal nicht zutrifft, wird die Verarbeitung des Variantencodes abgebrochen und es findet keine Initialisierung des Artikels mit den im Variantencode codierten Merkmalswerten statt.
Mit dieser Option kann nun gesteuert werden, ob das oben beschriebene Standardverhalten verwendet werden soll, oder ob fehlende Merkmale ignoriert werden sollen. Letzteres könnte z.B. notwendig sein, wenn der Variantencode aus einem älteren Auftrag generiert wird und dabei Merkmale verwendet wurden, die es in den aktuellen OCD-Daten nicht mehr gibt.

`0` (oder `false`)

Ein nicht-qualifiziertes Merkmal im Variantencode, das nicht in den Merkmalsklassen des Artikels enthalten ist, führt zum Abbruch der Verarbeitung des Variantencodes.

Dies ist das Standardverhalten (wenn es keinen Tabelleneintrag für diese Option gibt).

`1` (oder `true`)

Ein nicht-qualifiziertes Merkmal im Variantencode, das nicht in den Merkmalsklassen des Artikels enthalten ist, wird bei der Verarbeitung des Variantencodes ignoriert.

`@NeedValuesForRGProps`

Wenn die Option vorhanden ist und den Wert `1` hat, werden alle Werte von Merkmalen des Geltungsbereichs `RG` an die für das Merkmal generierte OFML-Property durchgereicht[10]. Damit wird eine Wertänderung mittels Aufruf der Methode *setPropValue()* (OFML-Schnittstelle *Property*) ermöglicht. Das ist speziell für OAP-Projekte interessant.

`@OCDUserMessageMode`

Mit dieser Option kann das Verhalten der Applikation hinsichtlich von Meldungen an die Anwender gesteuert werden, welche in den OCD-Daten mittels der Funktion *USER_MESSAGE()* ausgegeben werden.

`@MessageOnly`

Die Meldungen sollen immer in einem modalen Message-Dialog ausgegeben werden (der durch die Anwender bestätigt werden muss).

`@HintPreferred`

Die Meldungen sollen bevorzugt in einem nicht-modalen Hinweisfenster angezeigt werden. Nur wenn die Applikation Hints nicht unterstützt, sollen die Meldungen in einem modalen Message-Dialog ausgegeben werden.

Dies ist das Standardverhalten (wenn die Option nicht hinterlegt ist).

`@HintOnly`

Die Meldungen dürfen nur in einem nicht-modalen Hinweisfenster angezeigt werden, d.h., wenn die Applikation Hints nicht unterstützt, erfolgt keine Ausgabe der Meldungen.

[10] Ansonsten enthält die Werte-Auswahlliste nur den aktuellen Wert des Merkmals, womit die Property read-only ist.

@OptPropsWithBaseValues

Diese Option steuert das Verhalten bezüglich von Merkmalen, die als optional deklariert sind, und für die in der Artikelstammtabelle Werte angegeben sind. Die OCD-Spezifikation enthält keine klaren Vorgaben, wie in dieser Situation verfahren werden soll. Es gibt 2 Interpretationsmöglichkeiten, welche mit den entsprechenden Werten dieser Option gewählt werden können:

`0`

Da in der Artikelstammtabelle Angaben zu fixen bzw. erlaubten Werten von ausgewählten Merkmalen gemacht werden, wird das Optional-Kennzeichen der betreffenden Merkmale ignoriert, d.h., diese Merkmale können nicht den Zustand „nicht bewertet“ annehmen, sondern müssen immer mit einem der in der Artikelstammtabelle angegebenen Werte belegt sein.

Dies ist das Standardverhalten (wenn es keinen Tabelleneintrag für diese Option gibt).

`1`

Für konfigurierbare Merkmale dient die Artikelstammtabelle lediglich der Einschränkung der Menge der erlaubten Werte im Vergleich zu den in der Wertetabelle hinterlegten Werten, d.h., ein als optional deklariertes Merkmal kann auch den Zustand „nicht bewertet“ annehmen.

@RelEvalOptimization

Mit dieser Option können bestimmte Verfahren zur Optimierung der Auswertung von Beziehungswissen aktiviert werden. Aktuell wird nur ein Verfahren unterstützt, welches mit dem Wert `2` aktiviert wird:

`2`

Diese Optimierung bezieht sich auf die automatische Wertvorbelegung von einschränkbaren Merkmalen (siehe Option *@PreselectRestrictable*): Dabei wird die Wertemenge eines vorbelegten Merkmals "eingefroren", kann also nicht mehr durch die Vorbelegung nachfolgender Merkmale verändert werden. Die Optimierung setzt also voraus, dass vorbelegte Merkmale keine Abhängigkeiten zu Merkmalen aufweisen, die weiter unten in der Merkmalsliste stehen[11].
Ein bemerkbarer Performanz-Gewinn ergibt sich bei diesem Verfahren dann, wenn sehr viele Merkmale vorbelegt werden, diese große Wertemengen besitzen und viele der Werte mit Vorbedingungen verknüpft sind[12]. Werden die Wertemengen der vorbelegten Merkmale ausschließlich durch Constraints eingeschränkt (was der Normalfall in der OCD-Datenanlage sein sollte), ist der Performanz-Gewinn gering.

[11] Derartige Abhängigkeiten sind aus Anwendersicht eh unschön und sollten bei der Datenanlage sowieso vermieden werden.
[12] De-facto ergibt sich der Performanz-Gewinn dadurch, dass mit dem Verfahren dann deutlich weniger Vorbedingungen ausgewertet werden müssen.

@SetDefaultMode **(obsolete[13])**

Diese Option legt das Verhalten bezüglich der Builtin-Funktion

`$SET_DEFAULT($self,<Merkmal>,<Ausdruck>)`

in SAP—Sprachsets fest[14]:

`@AssignValue`

Der Wert des Ausdrucks wird dem Merkmal zugewiesen, wenn das Merkmal aktuell nicht bewertet ist.

Das Verhalten ist damit identisch mit einer Aktion der Form

`<Merkmal> = <Ausdruck> if <Merkmal> not specified`

Dies ist das Standardverhalten (wenn die Option nicht hinterlegt ist).

`@SetDefault`

Der Wert des Ausdrucks wird als Default-Wert für das Merkmal gesetzt.

Es findet also keine unmittelbare Wertzuweisung an das Merkmal statt, bei einschränkbaren Merkmalen kann der Default-Wert dann aber im Rahmen einer nachfolgenden automatischen Wertvorbelegung wirksam werden (siehe Option *@PreselectRestrictable*).

Für optionale Merkmale hat dieser Modus keine Bedeutung. Das Verhalten ist bei diesen Merkmalen immer wie im Modus `@AssignValue`.

@SetVisibilityMode

Diese Option legt den Geltungsbereich für die Funktion *SET_VISIBILITY()*fest:

`@Standard`

Merkmale, die durch die Funktion als nicht-sichtbar deklariert sind, werden sowohl im Eigenschaftseditor als auch im Variantentext *nicht* angezeigt.

Dies ist das Standardverhalten (wenn die Option nicht hinterlegt ist).

`@PropEditOnly`

Als nicht-sichtbar deklarierte Merkmale werden nur im Eigenschaftseditor *nicht* angezeigt (sind aber im Variantentext enthalten).

@ShowExtraCharge4Value

Mit dieser Option kann bei Merkmalen mit einer Auswahlliste für die erlaubten Werte hinter dem Wert in der Auswahlliste der Aufpreis angezeigt werden, der bei Auswahl des Wertes erhoben wird[15].

Dazu muss im Wertfeld ein Schema für die relevanten Variantenkonditionen in der Preistabelle hinterlegt werden. Dieses Schema besteht aus beliebigen Zeichen und muss den Platzhalter %V enthalten, welcher zur Laufzeit durch den Bezeichner (ID) des Merkmalswertes (Value) ersetzt wird. Optional kann das Schema noch den Platzhalter %P enthalten, welcher zur Laufzeit durch den Bezeichner (ID) des Merkmals (Property) ersetzt wird. Bei der Initialisierung des Artikels wird dann für jeden möglichen Merkmalswert

[13]Wird ab den Releases vom Herbst 2019 nicht mehr benötigt/unterstützt. S.a. Application Note AN-2014-04.
[14]Diese Builtin-Funktion wird im OCD—Standard offiziell *nicht* unterstützt!
[15]Diese Option wird aktuell nur vom Eigenschaftseditor des pCon.planner >= 7 unterstützt.

(aller Merkmale) geprüft, ob in der Preistabelle ein Eintrag des Levels `X` (Aufpreis) mit einem Festbetrag für die Variantenkondition existiert, die sich gemäß des angegebenen Schemas für den Merkmalswert ergibt. Falls ja, wird dieser Aufpreis dann an die Bezeichnung des Wertes in der Auswahlliste angehängt.

Beispiel:

```
@ShowExtraCharge4Value;;%P_%V
```

Angenommen, ein Artikel besitze ein Merkmal `FOO` mit einem Wert `BAR` und in der Preistabelle gibt es einen Eintrag des Levels `X` mit einem Festbetrag für die Variantenkondition `FOO_BAR`, so wird hinter der Bezeichnung für den Wert `BAR` in der Auswahlliste der in dem Preiseintrag angegebene Betrag angezeigt.

@ShowPrecondInvalidValues

Existiert ein Eintrag mit dieser Option und hat diese den Wert `1`, so werden Werte von einschränkbaren Merkmalen, die aktuell wegen nicht erfüllter Vorbedingungen nicht ausgewählt werden können, in der Auswahlliste deaktiviert (ausgegraut) dargestellt.

Im Normalfall (kein Eintrag mit dieser Option bzw. beim Wert `0`) werden aktuell nicht gültige (auswählbare) Werte in der Auswahlliste des Merkmals nicht dargestellt.

Hinweis: Die Option hat keinen Einfluss auf Merkmalswerte, die durch ein Constraint als ungültig deklariert wurden. Diese werden in jedem Fall *nicht* in der Auswahlliste des Merkmals dargestellt.

@UnlockBackwardRestriction

Im OCD-Beziehungswissen können sogenannte rückwärtsgerichtete Abhängigkeiten enthalten sein. Diese haben den Effekt, dass die Wertemenge eines Merkmals durch den Wert eines Merkmals beeinflusst wird, welches in der Merkmalsliste weiter unten steht. Bei der Änderung des Wertes des weiter unten stehenden Merkmals kann dies im Extremfall dazu führen, dass das oben stehende abhängige Merkmal dann nur noch einen einzigen gültigen Wert besitzt und nicht mehr geändert werden kann.

Ein (typisches) Beispiel für eine rückwärtsgerichtete Abhängigkeit ist folgendes *Constraint*:

```
Objects:
    T is_a MAT_PROPS.
Restriction:
    Table FARBE_STAHL_59 (
        FARBE_STAHL_59 = T.FARBE_STAHL_PGR,
        FARBE_STAHL    = T.FARBE_STAHL).
Inferences:
    T.FARBE_STAHL_PGR, T.FARBE_STAHL.
```

Da sowohl das Preisgruppenmerkmal `FARBE_STAHL_PGR` als auch das zugehörige Ausführungsmerkmal `FARBE_STAHL` unter `Inferences` angeführt sind, schränken sich beide Merkmale gegenseitig ein. Wird das in der Merkmalsliste weiter unten stehende Merkmal `FARBE_STAHL` durch den Anwender geändert, kann es zu der oben beschriebenen Blockierung bei dem darüber stehenden Merkmal `FARBE_STAHL_PGR` kommen.

Derartige rückwärtsgerichtete Abhängigkeiten sollten aus Anwendersicht prinzipiell vermieden werden.

In dem Beispiel oben sollte idealerweise nur das Ausführungsmerkmal vom Preisgruppenmerkmal abhängig sein.

Falls dies nicht möglich ist[16] und es sich bei den Merkmalen, die eine rückwärtsgerichtete Abhängigkeit implizieren, um einschränkbare Merkmale handelt, kann mit Hilfe dieser Option (Wert `1`) eine Ausnahme-Behandlung in der OCD-Implementierung aktiviert werden:
Durch eine temporäre Zurücknahme des aktuellen Wertes des geänderten Merkmals und einen zusätzlichen Zyklus der Auswertung des Beziehungswissen wird die Blockierung von Merkmalen weiter oben in der Liste aufgehoben.

Implikationen dieser Spezialbehandlung sind:

1. Evtl. vorhandene Beziehungen des Typs *Reaktion* für das geänderte einschränkbare Merkmal werden u.U. unwirksam!!
2. Im Extremfall kann der gerade gesetzte Wert ungültig werden (und dem Merkmal ein anderer Wert zugewiesen werden).
3. Etwas schlechtere Performanz durch die zusätzliche Auswertung von Beziehungswissen.

Die möglichen Probleme in Folge der Implikationen 1 und 2 erfordern eine ausführliche Testung seitens des Herstellers/Datenanlegers bei Verwendung dieser Option!

Hinweis:
Der zusätzliche Zyklus der Auswertung des Beziehungswissen hat keinen Effekt, wenn die betreffenden Merkmale per Vorbedingung von einem anderen, einschränkbaren Merkmal abhängen und (erst) während der Vorbelegung[17] dieses Merkmals gültig/sichtbar werden!

@UnfixPreselectedChoiceList

Bei der automatischen Wertvorbelegung eines einschränkbaren Merkmals (siehe Option *@PreselectRestrictable*) wird standardmäßig die für den Anwender sichtbare Wertemenge des Merkmals "eingefroren", d.h. entspricht der zu diesem Zeitpunkt definierten Menge gültiger Werte (kann also nicht mehr durch die Vorbelegung nachfolgender Merkmale verändert werden).

Die dem zugrunde liegende Annahme, dass die Merkmale gemäß einer logischen Reihenfolge von oben nach unten in der Merkmalsliste angeordnet sind bzw. angeordnet werden können, ist jedoch nicht immer erfüllt, siehe Beispiel unten. In dem Fall kann diese Option mit dem Wert `1` verwendet werden. Die für den Anwender sichtbare Wertemenge entspricht dann der nach der Vorbelegung aller einschränkbaren Merkmale definierten Menge gültiger Werte.

Hinweis: Wenn die Option *@RelEvalOptimization* den Wert 2 hat, ist diese Option wirkungslos.

Ein Anwendungs**beispiel** für die Option wäre ein Mediaschrank mit einem eingebauten Fernseher. Je nach gewähltem Modell steht eine bestimmte Anzahl von HDMI- und USB-Anschlüssen für Zusatzgeräte zur Verfügung. Jedes eingebaute Zusatzgerät verbraucht eine bestimmte Menge an Anschlüssen. Wenn alle Anschlüsse belegt sind, kann kein weiteres Zusatzgerät einbaut werden. Die Zusatzgeräte werden über entsprechende Merkmale konfiguriert/eingefügt, aber es gibt keine definierte Vorrang-Reihenfolge für die verschiedenen Arten von Geräten.

[16] z.B. aufgrund von Eigenarten des ERP-Systems, aus dem die OCD-Daten exportiert werden
[17] s. Option *@PreselectRestrictable* im Abschn. 5.1

6. Tabellen für Planungsgruppen

Mit den Tabellen `jointplgroup`, `layoutgroup`, `tabularplgroup` und `customplgroup` können einige Aspekte des Verhaltens von Planungsgrupppen gesteuert werden, deren Typ *xOiJointPlGroup*, *xOiLayoutGroup*, *xOiTabularPlGroup* bzw. *xOiCustomPlGroup* ist[18].

Wenn sich die Tabellen nicht im regulären Datenverzeichnis der betreffenden OFML Serie befinden, werden sie im Verzeichnis gesucht, das in der DSR-Registrierungsdatei der Serie unter dem Schlüssel `proginfodb_path` angegeben ist.

Wenn das Argumentfeld (2) leer ist, dann ist der Tabelleneintrag für alle Planungsgruppen gültig. Wenn das Argumentfeld nicht leer ist, muss es eine Bedingung in Form eines Vektors

```
[condition_type, condition_value]
```

enthalten. Wenn mehrere Bedingungen angegeben werden, müssen diese durch ein Komma getrennt werden.

Aktuell werden folgende **Bedingungstypen** (`condition_type`) unterstützt:

`@Article`
: ist erfüllt, wenn die Instanz einen Artikel repräsentiert und dessen Artikelnummer mit einer der im Bedingungswert angegebenen Zeichenketten beginnt.
Die Zeichenketten im Bedingungswert sind durch Komma zu trennen und in eckige Klammern einzuschließen.

`@ArticleX`
: ist erfüllt, wenn die Instanz einen Artikel repräsentiert und dessen Artikelnummer *exakt* mit einer der im Bedingungswert angegebenen Zeichenketten übereinstimmt.
Die Zeichenketten im Bedingungswert sind durch Komma zu trennen und in eckige Klammern einzuschließen.

  Dieser Bedingungstyp sollte/kann anstelle von `@Article` verwendet werden, wenn es mehrere Planungsgruppenartikel mit einem gemeinsamen Artikelnummernstamm gibt, um Abhängigkeiten von der Reihenfolge der Tabelleneinträge zu vermeiden.

`@ArticleNOT`
: ist erfüllt, wenn die Instanz einen Artikel repräsentiert und dessen Artikelnummer mit ***keiner*** der im Bedingungswert angegebenen Zeichenketten beginnt.
Die Zeichenketten im Bedingungswert sind durch Komma zu trennen und in eckige Klammern einzuschließen.

`@Property`
: ist erfüllt, wenn der Schlüssel einer verarbeiteten gemeinsamen Property mit einer der im Bedingungswert angegebenen Zeichenketten beginnt.
Die Zeichenketten im Bedingungswert sind durch Komma zu trennen und in eckige Klammern einzuschließen.

`@PropertyNOT`
: ist erfüllt, wenn der Schlüssel einer verarbeiteten gemeinsamen Property mit ***keiner*** der im Bedingungswert angegebenen Zeichenketten beginnt.
Die Zeichenketten im Bedingungswert sind durch Komma zu trennen und in eckige Klammern einzuschließen.

[18] oder ein davon abgeleiteter Typ

Es darf nur eine Artikel-bezogene Bedingung angegeben werden[19].

Property-bezogene Bedingungen werden derzeit für die folgenden Optionen unterstützt[20]:

- *@CommonPropsChLMode*
- *@SortUnionCommonProps*
- *@NonVisibleProps4Common*
- *@ROPropsEditable4Common*
- *@Classes4CommonProps*
- *@ArticleFeatures4Common*

Wenn die Gruppeninstanz ***keinen*** Artikel repräsentiert, werden Tabelleneinträge mit einem nicht-leeren Argumentfeld ignoriert und für einen gegebenen Informationstyp wird der erste Tabelleneintrag mit leerem Argumentfeld verwendet. (D.h., für Gruppeninstanzen, die keinen Artikel repräsentieren, dürfen weder Artikel- noch Property-bezogene Bedingungen angewendet werden.)

Wenn die Gruppeninstanz einen Artikel repräsentiert, hängt die Verarbeitung der Tabelleneinträge in Bezug auf eine angeforderte Option davon ab, ob die Option Property-bezogene Bedingungen unterstützt (siehe oben):

- Wenn die Option keine Property-bezogenen Bedingungen unterstützt, wird der erste Eintrag mit einer passenden Artikel-bezogenen Bedingung verwendet. Wenn es keinen Eintrag mit einer passenden Artikel-bezogenen Bedingung gibt, wird der erste Eintrag mit einem leeren Argumentfeld verwendet (falls vorhanden).
- Wenn die Option Property-bezogene Bedingungen unterstützt, werden zunächst alle Einträge mit Property-bezogenen Bedingungen *und* einer passenden Artikelbedingung berücksichtigt. Für eine gegebene gemeinsame Eigenschaft wird dann der erste dieser Einträge verwendet, dessen Property-bezogenen Bedingungen erfüllt sind. Gibt es keinen solchen Eintrag, wird der erste Eintrag mit einem leeren Argumentfeld verwendet (falls vorhanden).

6.1. Gemeinsame Optionen

@InsertMode

Der *InsertMode* steuert das Verhalten bzgl. des Einfügens eines neuen Elements zwischen 2 bereits in der Planungsgruppe enthaltene Layout-Elemente[21] sowie bzgl. von Dimensionsänderungen von Elementen.

Für die Klasse *xOiTabularPlGroup* (Tabelle `tabularplgroup`) ist diese Option nicht relevant!

Die folgenden Modi sind definiert:

`0` Sowohl das Einfügen zwischen zwei bereits vorhandene Layout-Elemente als auch Dimensionsänderungen von Elementen sind ***nicht*** erlaubt.
Das ist der Standard-Modus.

`1` Dimensionsänderungen von Elementen sind erlaubt.

`2` Sowohl das Einfügen zwischen zwei vorhandene Elemente als auch Dimensionsänderungen von Elementen sind erlaubt.

[19] Werden mehrere Artikelbedingungen angegeben, werden alle außer der ersten ignoriert.
[20] Für alle anderen Optionen werden ggf. angegebene Property-bezogenen Bedingungen ignoriert.
[21] Bei der Klasse *xOiJointPlGroup* bezieht sich das auf die Elemente, die in der topologischen Liste enthalten sind.

Hinweis:
Die Algorithmen zur Behandlung der Modi `1` und `2` basieren auf der Erkennung von Kollisionen zwischen den Elementen der Planungsgruppe. Deswegen dürfen diese für eine korrekte Funktionsweise nicht mittels der Methode *disableCD()* (OFML Schnittstelle *Base*) von der Kollisionserkennung ausgeschlossen sein!

`@SelectableState4Layout`

Legt fest, ob Layout-Elemente[22] durch den Anwender selektiert werden können, wobei der Wert `1` *Ja* bedeutet, und `0` *Nein*. Standard: `1`.

Der initiale Zustand gemäß OFML-Schnittstelle *Base* ist `1`, gelegentlich ist es aber erwünscht/notwendig, dass die Layout-Elemente nicht selektierbar sind, z.B. wenn die Elemente nur über Methoden der Planungsgruppe geändert werden sollen/dürfen.

`@SelectableState4Other`

Legt fest, ob Nicht-Layout-Elemente[23] durch den Anwender selektiert werden können, wobei der Wert `1` *Ja* bedeutet, und `0` *Nein*. Standard: `1`.

Der initiale Zustand gemäß OFML-Schnittstelle *Base* ist `1`, aber Nicht-Layout-Elemente (wie z.B. Unterbauten) sollen typischerweise nicht selektierbar sein.

`@CutableState4Layout`

Legt den Zustand fest, der mittels der Methode *setCutable()* (OFML-Schnittstelle *Base*) an Layout-Elemente zugewiesen werden soll[24].

Wenn es keinen passenden Tabelleneintrag für diese Option gibt, haben Elemente den initialen Zustand `1`, d.h., sie können mittels *Cut*- bzw. *Delete*-Operation der Applikation aus der Planungsgruppe entfernt werden.

Wenn die Elemente nur im Kontext einer speziellen Interaktion entfernt werden dürfen (z.B. einer OAP-Delete-Aktion), sollte diese Option genutzt werden, um den Zustand `-1` zuzuweisen.

`@CutableState4Other`

Legt den Zustand fest, der mittels der Methode *setCutable()* (OFML- Schnittstelle *Base*) an Nicht-Layout-Elemente zugewiesen werden soll[25].

Diese Option muss nicht angegeben werden, wenn die Option `@SelectableState4Other` (siehe oben) den Wert `0` hat.

Siehe auch die Anmerkungen oben zur Option `@CutableState4Layout`.

`@ShowAsArticle`

Die Option legt fest, ob die Planungsgruppe – falls sie einen Artikel repräsentiert – im Warenkorb als ein Artikel (`1`) anstelle eines Ordners (`0`) angezeigt werden soll.
Default: `0`.

---

[22] Bei der Klasse *xOiJointPlGroup* bezieht sich das auf die Elemente, die in der topologischen Liste enthalten sind.
[23] Bei der Klasse *xOiJointPlGroup* bezieht sich das auf die Elemente, die nicht in der topologischen Liste enthalten sind.
[24] Bei der Klasse *xOiJointPlGroup* bezieht sich das auf die Elemente, die in der topologischen Liste enthalten sind.
[25] Bei der Klasse *xOiJointPlGroup* bezieht sich das auf die Elemente, die nicht in der topologischen Liste enthalten sind.

`@IsPseudoArticle`

Legt fest, ob die Gruppeninstanz einen Pseudoartikel darstellt, d. h. zur Kategorie *@PseudoArticle*[26] gehört (`1`) oder nicht (`0`).

Default ist `1`, wenn die Gruppeninstanz einen Artikel darstellt und die Option `@ShowAsArticle` (siehe oben) den (Default-)Wert `0` hat. Andernfalls ist der Default `0`.

`@IsNonOrderArticle`

Legt fest, ob die Gruppeninstanz zur Kategorie *@NonOrderArticle*[27] gehört (`1`) oder nicht (`0`).
Default: 0

Hinweis: Die Kategorie *@NonOrderArticle* setzt die Kategorie *@PseudoArticle* voraus!

`@CommonProps`

Gibt die gemeinsamen Properties an, die von den Gruppenelementen auf die Ebene der Gruppe hochgezogen werden sollen.

Der Wert ist eine Komma-separierte Auflistung der Schlüssel der betreffenden Properties (OFML-Symbol), eingeschlossen in eckige Klammern[28].

Die Option `@AllObjs4CommonProps` (s.u.) legt fest, ob die Eigenschaften der Properties (Typ, Wertliste etc.) von allen Layout-Elementen oder nur vom ersten Layout-Element der Gruppe übernommen werden sollen. Letzteres ist das Standardverhalten, bei welchem angenommen wird, dass die betreffenden Properties aller Layout-Elemente[29] dieselben Eigenschaften aufweisen.

Zusätzlich kann mittels der Option `@NonLayout4CommonProps` (s.u.) festgelegt werden, dass auch Nicht-Layout-Elemente für gemeinsame Eigenschaften berücksichtigt werden sollen.

Wenn mehrere Elemente berücksichtigt werden und zwei Elemente dieselbe (gemeinsame) Property besitzen, diese aber unterschiedliche Attribute aufweisen, „gewinnt“ das erste Element. Siehe aber Option `@CommonPropsChLMode`, welche die Art und Weise der Definition von Auswahllisten von gemeinsamen Properties festlegt.

Es werden nur diejenigen der aufgezählten Properties für die Planungsgruppe definiert, welche die relevanten Elemente auch tatsächlich besitzen. Dabei wird prinzipiell die in `@CommonProps` spezifizierte Reihenfolge berücksichtigt. Wenn die Option `@AllObjs4CommonProps` den Wert `1` hat, kann es jedoch zu einer abweichenden Reihenfolge kommen, wenn das erste Layout-Element der Gruppe nicht alle in `@CommonProps` spezifizierten Properties besitzt. (Siehe auch Optionen `@FirstPos4CommonProps` und `@CommonPropsPos` unten.)

Nach der Änderung einer gemeinsamen Properties wird das Set der gemeinsamen Properties aktualisiert, um auf mögliche Abhängigkeiten zwischen den Properties zu reagieren.

Diese Option ist nur verfügbar/aktiv, wenn in der Option `@StartLayout` (bzw. `@StartElement` bei *xOiJointPlGroup*) mindestens ein initiales Gruppenelement spezifiziert ist und wenn es erfolgreich erzeugt werden konnte.

[26] siehe Spezifikation der OFML-Schnittstelle *Article*
[27] siehe Spezifikation der OFML-Schnittstelle *Article*
[28] OFML-Vector-Darstellung
[29] Bei der Klasse *xOiJointPlGroup* bezieht sich das auf die Elemente, die in der topologischen Liste enthalten sind.

Hinweis: Für eine bessere Performanz sollten die gemeinsamen Properties auch in der Tabelle `non_pd_properties` angegeben werden (siehe Anhang A.2), wenn der Artikel der Planungsgruppe diese Properties nicht besitzt (was wahrscheinlich in den meisten Fällen der Fall ist).

`@FirstPos4CommonProps`

Legt die Position in der Property-Liste fest, ab der die gemeinsamen Properties platziert werden sollen[30]. Default: `1`

Hinweis: Wenn in abgeleiteten Klasse weitere (fixe) Properties definiert werden, sollten diese entweder vor oder nach den (generierten) gemeinsamen Properties platziert werden[31].

Siehe auch Option *`@CommonPropsPos`* unten.

`@CommonPropsPos`

Mit dieser Option können spezifische Positionen für gemeinsamen Properties festgelegt werden.

Das Wert-Feld enthält die OFML String Notation eines *Vector*s, dessen Elemente entweder vom Typ *Int* oder vom Typ *Void* sind.

Die Reihenfolge in dieser Option entspricht der Reihenfolge der in der Option *`@CommonProps`* (siehe oben) angegebenen Properties, d. h. der erste Eintrag in *`@CommonPropsPos`* gibt die Position für die erste Property in *`@CommonProps`* an etc.

Wenn ein Eintrag im Vektor eine ganze Zahl >= `0` ist, bekommt die entsprechende gemeinsame Property diese Position (gemäß den für die Methode *setPropPos()* der Schnittstelle *Property* festgelegten Regeln). Andernfalls, wenn der Eintrag vom Typ *Void* ist, wird die Position der entsprechenden gemeinsamen Property anhand der Option *`@FirstPos4CommonProps`* (siehe oben) und der Position der Property in der Option *`@CommonProps`* bestimmt.

Wenn es in dieser Option weniger Einträge gibt als in *`@CommonProps`*, werden die fehlenden Einträge in *`@CommonPropsPos`* so behandelt, als hätten sie einen Wert vom Typ *Void*.

`@AllObjs4CommonProps`

Legt fest, ob alle Layout-Elemente bei der Definition der gemeinsamen Properties herangezogen werden sollen (`1`) oder nur das erste Layout-Element (`0`).
Default: `0`

`@NonLayout4CommonProps`

Legt fest, ob auch Nicht-Layout-Elemente bei der Definition der gemeinsamen Properties herangezogen werden sollen (`1`) oder nicht (`0`). Default: `0`

Hinweis: Der Wert `1` hat nur dann einen Effekt, wenn auch die Option *`@AllObjs4CommonProps`* den Wert `1` besitzt!

[30] Bei Instanzen der Klasse *xOiTabularPlGroup* ergibt sich die Position der ersten gemeinsamen Property genau genommen aus dem in dieser Option angegebenen Wert plus 3, da die ersten 3 Positionen der durch diese Klasse generierten Properties für die *Resize*-Properties reserviert sind.
[31] Beachte dabei die Festlegungen zu Property-Positionen gemäß der Spezifikation der Methode *setPropPos()* der Schnittstelle *Property*.

@CommonPropsChLMode

Legt die Art und Weise fest, wie die Auswahllisten der gemeinsamen Properties gebildet werden.

Die folgenden Modi werden unterstützt:

`@FirstEl`
: Das erste Element, das eine bestimmte gemeinsame Property besitzt, hat Vorrang, d. h. die Property-Definition wird von diesem Element übernommen (einschließlich einer möglichen Auswahlliste).

  Dies ist der Default, wenn die Option *@AllObjs4CommonProps* (siehe oben) nicht angegeben ist oder den Wert `0` hat.

  Dieser Modus bietet eine etwas bessere Performanz (als die anderen zwei Modi), sollte aber nur verwendet werden, wenn die Option *@AllObjs4CommonProps* nicht den Wert `1` hat und wenn alle Layout-Elemente tatsächlich die gleichen Attribute in Bezug auf die gemeinsame(n) Eigenschaft(en) besitzen.

`@Intersection`
: Wenn verschiedene Elemente die gleichen Attribute in Bezug auf eine bestimmte gemeinsame Property, aber unterschiedliche Auswahllisten haben, wird die Auswahlliste der gemeinsamen Property durch die Schnittmenge der Auswahllisten der Elemente gebildet.

  Ist die resultierende Auswahlliste leer, wird die Property ignoriert[32]!

  Dies ist der Default, wenn die Option *@AllObjs4CommonProps* den Wert `1` hat.

`@Union`
: Wenn verschiedene Elemente die gleichen Attribute in Bezug auf eine bestimmte gemeinsame Property, aber unterschiedliche Auswahllisten haben, wird die Auswahlliste der gemeinsamen Property durch die Vereinigung der Auswahllisten der Elemente gebildet.

Mit Hilfe der Property-bezogenen Bedingungstypen `@Property` und `@PropertyNOT` kann diese Option für verschiedene gemeinsame Properties unterschiedlich bewertet werden.

@SortUnionCommonProps

Legt fest, ob die Werte in einer Auswahlliste einer gemeinsamen Property, die mit dem Modus `@Union` (siehe Option *@CommonPropsChLMode* oben) erstellt wurde, sortiert werden sollen (`1`) oder nicht (`0`).
Default ist `1` für numerische Eigenschaften, `0` für symbolische Eigenschaften (Typen `Y` und `YS`).

@Classes4CommonProps

Legt fest, ob bei der Definition der gemeinsamen Properties auch die Klassen von den Properties der Gruppenelemente übernommen werden sollen (`1`) oder nicht (`0`).
Default: `0` (d.h., die gemeinsamen Properties sind keiner Klasse zugeordnet)

@Meta4CommonProps

Legt fest, ob die Properties einer (möglicherweise) kapselnden Instanz (z.B. einer Metatyp-Instanz) für die Menge der gemeinsamen Properties herangezogen werden sollen (`1`) anstelle der Properties der eigentlichen Artikelinstanz (`0`).
Default: `0`

[32] d.h., sie wird nicht für die Gruppeninstanz definiert

@CommonPropsDepth

Gibt die (Anzahl der) Ebenen von Unterartikeln unter der Ebene der Layout-Elemente an, die bei der Generierung der gemeinsamen Properties berücksichtigt werden sollen.
Default ist `0`, d.h., es werden nur die Properties der Layout-Elemente berücksichtigt.

@NonVisibleProps4Common

Legt fest, ob nicht-sichtbare Properties in die Menge der gemeinsamen Properties übernommen werden sollen (`1`) oder nicht (`0`).
Default: `0`

Wenn sie übernommen werden, dann sind die Properties (auf der Ebene der Planungsgruppe) durch den Anwender editierbar.

Mit Hilfe der Property-bezogenen Bedingungstypen `@Property` und `@PropertyNOT` kann diese Option für verschiedene gemeinsame Properties unterschiedlich bewertet werden.

@ROPropsEditable4Common

Legt fest, ob Properties, die originär read-only sind, bei der Übernahme in die Menge der gemeinsamen Properties editierbar geschaltet werden sollen (`1`) oder nicht (`0`).
Default: `0`

Mit Hilfe der Property-bezogenen Bedingungstypen `@Property` und `@PropertyNOT` kann diese Option für verschiedene gemeinsame Properties unterschiedlich bewertet werden.

@ArticleFeatures4Common

Legt fest, ob eine sprachabhängige Beschreibung für die gemeinsamen Properties des ersten Layout-Elements im Ergebnis der Methode *getArticleFeatures()*[33] enthalten sein soll (`1`) oder nicht (`0`).
Default: `0`

Hinweis: Damit diese Option wirksam ist, muss die Option *@ShowAsArticle* den Wert 1 haben.

6.2. Spezifische Optionen für *xOiJointPlGroup*

@_2DMode

Legt den Modus bzgl. 2D-Darstellung einer nicht-leeren Planungsgruppe fest:

`@Elements`
: Verwende nur die 2D-Symbole der Elemente selber (empfohlen).

`@Group`
: Generiere die 2D-Darstellungen für die Elemente (d.h., ignoriere die eigenen 2D-Symbole der Elemente).

`@Combined`
: Verwende sowohl die 2D-Symbole der Elemente selber als auch durch die Planungsgruppe generierte 2D-Darstellungen für die Elemente.

  Das ist der Standard-Modus (wegen Abwärtskompatibilität).

[33] OFML-Schnittstelle *Article*

`@Rectangle`

Generiere ein einfaches umgrenzendes Rechteck.

`@IgnorePlanDir4Remove`

Legt fest, ob das erste bzw. letzte Element (aus der topologischen Liste) unabhängig von der aktuellen Planungsrichtung der Applikation entfernt werden kann (`1`) oder nicht (`0`).

Im Fall des Standard-Wertes `0` kann das letzte Element nur entfernt werden, wenn die aktuelle Planungsrichtung „nach rechts“ ist (und analog für die andere Richtung).

Hinweis: Die Standard-Planungsrichtung ist „nach rechts“. Nicht alle Applikationen erlauben es dem Anwender, die Planungsrichtung zu ändern.

`@AllElements4SubArticle`

Legt fest, ob alle Elemente der Planungsgruppe (die einen Artikel repräsentieren) als Unter-Artikel des Gruppen-Artikels behandelt werden sollen.

Im Fall des Standard-Wertes `0` werden nur die Elemente aus der topologischen Liste als Unter-Artikel behandelt.

Wenn Elemente (wie Rückwände, Abdeckplatten u.ä.), die nicht in der topologischen Liste enthalten sind, als Unter-Artikel behandelt werden sollen, muss der Wert `1` angegeben werden.

`@StartElement`

Gibt die Informationen an, die verwendet werden, um unmittelbar nach Erzeugung der Planungsgruppe (und Zuweisung einer Artikelnummer an diese) einen initialen Unter-Artikel zu erzeugen.

Das Wert-Feld enthält die OFML String Notation eines Vectors mit den folgenden Elementen:

1. OFML-Programm (*Symbol*)
2. Grundartikelnummer (*String*)
3. optional: Typ des Variantencodes (*Symbol*: `@VarCode`, `@OFMLVarCode`, `@Final`)
4. optional: Variantencode (*String*), möglicherweise teil-bestimmt
5. optional: Property-Werte (*Vector* von [key, value]-Paaren)
6. optional: Property-Stati (*Vector* von [key, state]-Paaren)
   Aktuell können die folgenden Status-Werte angegeben werden[34]:
   `0` – Die Property ist nicht sichtbar
   `1` – Die Property ist sichtbar, aber nicht editierbar
   `3` – Die Property ist sichtbar und editierbar

Beispiele:

```
@StartElement;;[@foo_bar,"ArtNr1"]
@StartElement;;[@foo_bar,"ArtNr2",@OFMLVarCode,"PROP1=814"]
@StartElement;;[@foo_bar,"ArtNr3",@VarCode,"",[[@Prop1,1],[@Prop2,@abc]]]
```

`@StartLayout`

Gibt die Informationen an, die verwendet werden, um unmittelbar nach Erzeugung der Planungsgruppe (und Zuweisung einer Artikelnummer an diese) initiale Unter-Artikel (als Layout-Elemente) zu erzeugen.

Hinweis: Diese Option hat Vorrang vor einer evtl. auch vorhandenen Option @StartElement (s.o.).

[34] gemäß der Spezifikation der Methode *setPropState2()* der OFML-Schnittstelle *Property*

Das Wert-Feld enthält die OFML String Notation eines Vectors aus Vectoren mit jeweils den folgenden Elementen:

1. Anfügepunkt des Vorgängers, an den dieses Element positioniert werden soll (*Void/Symbol*[35])
2. OFML-Programm (*Symbol*)
3. Grundartikelnummer (*String*)
4. optional: Typ des Variantencodes (*Symbol*: `@VarCode`, `@OFMLVarCode`, `@Final`)
5. optional: Variantencode (*String*), möglicherweise teil-bestimmt
6. optional: Property-Werte (*Vector* von [key, value]-Paaren)
7. optional: Property-Stati (*Vector* von [key, state]-Paaren)
   Aktuell können die folgenden Status-Werte angegeben werden[36]:
   `0` – Die Property ist nicht sichtbar
   `1` – Die Property ist sichtbar, aber nicht editierbar
   `3` – Die Property ist sichtbar und editierbar

Beispiel:

```
@StartLayout;;\
[[NULL,@foo_bar,"ArtNr1"],\
 [@ATTPT_R1,@foo_bar,"ArtNr2",@OFMLVarCode,"PROP1=814"],\
 [NULL,@foo_bar,"ArtNr3",@VarCode,"",[[@Prop1,1],[@Prop2,@abc]]]]
```

6.3. Spezifische Optionen für *xOiLayoutGroup*

@ChildBranchGrowthMode

Legt fest, ob Unterzweige der Layout-Struktur nur in eine Richtung wachsen können (`@OneDir`) oder in beide Richtungen (`@BothDirs`).
Default: `@OneDir`

@ContinueBranch4SingleForkEl

Gibt an, ob der Branch eines Fork-Elements fortgesetzt werden muss, wenn ihm ein Nachbar hinzugefügt wird und das Fork-Element derzeit das einzige Element in der Verzweigung ist.

Aus Gründen der Abwärtskompatibilität ist das Standardverhalten durch den Wert `0` (no/false) definiert, da grundsätzlich ein neuer Branch begonnen wird, wenn ein Nachbar an ein Fork-Element angefügt wird.

In dem gegebenen Szenario ist jedoch das durch den Wert `1` (yes/true) definierte Verhalten wahrscheinlich in den meisten Fällen das gewünschte Verhalten.

@UseAllAttPts

Legt fest, ob alle Anfügepunkte (einschließlich der Standard-Anfügepunkte) zur Verbindung von Layout-Elementen verwendet werden (`1`) oder nur zusätzlich definierte Anfügepunkte (`0`).

Für eine bessere Performanz sollte der Wert `1` nur verwendet werden, wenn tatsächlich auch Standard-Anfügepunkte zur Verbindung von Layout-Elementen verwendet werden.
Default: `0`

[35] Wenn ein Anfügepunkt angegeben ist (Symbol), wird dieser zur Positionierung des Artikels neben dem Vorgängerelement verwendet (wenn vorhanden), anderenfalls wird ein Standardverfahren zur Ermittlung der Position verwendet.
[36] gemäß der Spezifikation der Methode *setPropState2()* der OFML-Schnittstelle *Property*

@StartLayout

Gibt die Informationen an, die verwendet werden, um unmittelbar nach Erzeugung der Planungsgruppe (und Zuweisung einer Artikelnummer an diese) initiale Unter-Artikel (als Layout-Elemente) zu erzeugen.

Das Wert-Feld enthält die OFML String Notation eines Vectors aus Vectoren mit jeweils den folgenden Elementen:

1. Anfügepunkt des Vorgängers, an den dieses Element positioniert werden soll (*Symbol*[37])
2. OFML-Programm (*Symbol*)
3. Grundartikelnummer (*String*)
4. optional: Typ des Variantencodes (*Symbol*: `@VarCode`, `@OFMLVarCode`, `@Final`)
5. optional: Variantencode (*String*) , möglicherweise teil-bestimmt
6. optional: Property-Werte (*Vector* von [key, value]-Paaren)
7. optional: Property-Stati (*Vector* von [key, state]-Paaren)

   Aktuell können die folgenden Status-Werte angegeben werden[38]:

   `0` – Die Property ist nicht sichtbar

   `1` – Die Property ist sichtbar, aber nicht editierbar

   `3` – Die Property ist sichtbar und editierbar

Beispiel:

```
@StartLayout;;\
[[NULL,@foo_bar,"ArtNr1"],\
 [@ATTPT_R1,@foo_bar,"ArtNr2",@OFMLVarCode,"PROP1=814"],\
 [@ATTPT_R2,@foo_bar,"ArtNr3",@VarCode,"",[[@Prop1,1],[@Prop2,@abc]]]]
```

6.4. Spezifische Optionen für *xOiTabularPlGroup*

@MaxColumns

Legt die Anzahl der maximal erlaubten Spalten fest, wobei der Wert `0` keine Einschränkung bedeutet[39].

@MaxRows

Legt die Anzahl der maximal erlaubten Zeilen fest, wobei der Wert `0` keine Einschränkung bedeutet[40].

@UniformColumnWidth

Legt fest, ob alle Spalten dieselbe Breite haben (`1`) oder nicht (`0`).
Default: `1`

Die einheitliche Breite kann dann mittels der Option *@ColumnWidth* festgelegt werden.

@ColumnWidth

Legt die einheitliche Spaltenbreite (in Metern) fest, wenn die Option *@UniformColumnWidth* den Wert `1` hat.
Default: `1.0`

[37] `NULL` für das erste Element
[38] gemäß der Spezifikation der Methode *setPropState2()* der OFML-Schnittstelle *Property*
[39] Der initial durch diese Option festgelegte Wert kann mittels der Methode *setMaxColumns()* überschrieben werden.
[40] Der initial durch diese Option festgelegte Wert kann mittels der Methode *setMaxRows()* überschrieben werden.

@ColumnWidths

Legt die Spaltenbreiten (in Metern) fest, wenn die Option *@UniformColumnWidth* den Wert `0` hat.

Im Wert-Feld wird dazu ein Vector von *Float*-Werten angegeben, wobei der Index des Vector-Elements mit dem Index der betreffenden Spalte korrespondiert.

Wenn eine neue Spalte in das Layout eingefügt wird, für deren Index der Vector im Wert-Feld dieser Option kein Element enthält, wird für diese Spalte der Wert aus dem letzten Vector-Element verwendet.

Default: `[1.0]`

@UniformRowSize
@UniformRowDepth
@UniformRowHeight

Legt fest, ob alle Zeilen dieselbe Tiefe bzw. Höhe haben (`1`) oder nicht (`0`).
Default: `1`

Die Option *@UniformRowSize* kann in beiden möglichen Ausrichtungen der Planungsgruppe verwendet werden, die Option *@UniformRowDepth* nur in der horizontalen Ausrichtung und die Option *@UniformRowHeight* nur in der vertikalen Ausrichtung. Falls zwei Optionen angegeben sind, hat *@UniformRowSize* Vorrang.

@RowSize
@RowDepth
@RowHeight

Legt die einheitliche Tiefe bzw. Höhe der Zeilen (in Metern) fest, wenn die Optionen *@UniformRowSize*, *@UniformRowDepth* bzw. *@UniformRowHeight* den Wert `1` haben. Default: `1.0`

Die Option *@RowSize* kann in beiden möglichen Ausrichtungen der Planungsgruppe verwendet werden, die Option *@RowDepth* nur in der horizontalen Ausrichtung und die Option *@RowHeight* nur in der vertikalen Ausrichtung. Falls zwei Optionen angegeben sind, hat *@RowSize* Vorrang.

@RowSizes
@RowDepths
@RowHeights

Legt die Tiefen bzw. Höhen der Zeilen (in Metern) fest, wenn die Optionen *@UniformRowSize*, *@UniformRowDepth* bzw. *@UniformRowHeight* den Wert `0` haben.

Im Wert-Feld wird dazu ein Vector von *Float*-Werten angegeben, wobei der Index des Vector-Elements mit dem Index der betreffenden Zeile korrespondiert.

Wenn eine neue Zeile in das Layout eingefügt wird, für deren Index der Vector im Wert-Feld dieser Option kein Element enthält, wird für diese Zeile der Wert aus dem letzten Vector-Element verwendet.

Standard: `[1.0]`

Die Option *@RowSizes* kann in beiden möglichen Ausrichtungen der Planungsgruppe verwendet werden, die Option *@RowDepths* nur in der horizontalen Ausrichtung und die Option *@RowHeights* nur in der vertikalen Ausrichtung. Falls zwei Optionen angegeben sind, hat *@RowSizes* Vorrang.

@AlignmentX

Legt die Ausrichtung der Elemente innerhalb der Felder entlang der X-Achse fest.

Die möglichen Werte sind: `@Left` (Standard), `@Right`, `@Centered`

@AlignmentY

Legt die Ausrichtung der Elemente innerhalb der Felder entlang der Y-Achse fest.

Die möglichen Werte sind: `@Bottom` (Standard), `@Up`, `@Centered`

Bei einer Gruppe mit vertikaler Ausrichtung bezieht sich die Ausrichtung auf die Höhe der Felder, andernfalls auf das lokalen Begrenzungsvolumen des Feldelements.

@AlignmentZ

Legt die Ausrichtung der Elemente innerhalb der Felder entlang der Z-Achse fest.

Die möglichen Werte sind: `@Back` (Standard), `@Front`, `@Centered`

Bei einer Gruppe mit horizontaler Ausrichtung bezieht sich die Ausrichtung auf die Tiefe der Felder, andernfalls auf das lokalen Begrenzungsvolumen des Feldelements.

@DefaultArticle

Gibt die Informationen an, die zur Erzeugung von Artikeln verwendet werden, wenn keine anderweitige (spezifische) Information vorliegt.

Das Wert-Feld enthält die OFML String Notation eines Vectors mit den folgenden Elementen:

1. OFML-Programm (*Symbol*)
2. Grundartikelnummer (*String*)
3. optional: Typ des Variantencodes (*Symbol*: `@VarCode`, `@OFMLVarCode`, `@Final`)
4. optional: Variantencode (*String*) , möglicherweise teil-bestimmt
5. optional: Property-Werte (*Vector* von [key, value]-Paaren)
6. optional: Property-Stati (*Vector* von [key, state]-Paaren)
   Aktuell können die folgenden Status-Werte angegeben werden[41]:
   `0` – Die Property ist nicht sichtbar
   `1` – Die Property ist sichtbar, aber nicht editierbar
   `3` – Die Property ist sichtbar und editierbar

Beispiele siehe Option *@StartElement* im Abschn. 6.2.

@EmptyFieldsPlaceholder

Legt fest, ob leere Felder mit transparenten, selektierbaren Platzhalter-Objekten gefüllt werden sollen (`1`) oder nicht (`0`).
Default: `0`

Hat die Option den Wert `1`, können die Optionen *@FieldPlaceholderThickness*, *@FieldPlaceholderMat* und *@FieldPlaceholderArticle* (s.u.) verwendet werden, um bestimmte Aspekte der Platzhalter-Objekte zu beeinflussen.

[41] gemäß der Spezifikation der Methode *setPropState2()* der OFML-Schnittstelle *Property*

@FieldPlaceholderThickness

Legt die Höhe[42] bzw. Tiefe[43] der Platzhalter-Objekte fest.
Default: `0.1`

@FieldPlaceholderMat

Legt das Material fest, welches für die Platzhalter-Objekte verwendet werden soll.

Im Wert-Feld ist dazu ein vollqualifizierter OFML-Materialbezeichner anzugeben.

Default ist ein Material vom Typ *Glass*.

@FieldPlaceholderType

Gibt den voll qualifizierten OFML-Typ an (*String*), der für Platzhalterinstanzen anstelle des Standardtyps *xOiTabularPlGroupPlaceholder* verwendet werden soll.

Der angegebene Typ muss ein unmittelbarer Subtyp von *xOiTabularPlGroupPlaceholder* sein!

@FieldPlaceholderArticle

Legt die Artikel-Information für die Platzhalter-Objekte fest.

Das Wert-Feld enthält die OFML String Notation eines Vectors mit den folgenden Elementen:

1. OFML-Programm (*Symbol*)
2. Grundartikelnummer (*String*)
3. kaufmännisches Herstellerkürzel (*String*)
4. kaufmännisches Serienkürzel (*String*)

Default: keine Artikel-Information, d.h., die Platzhalter-Objekte repräsentieren keine Artikel (und erscheinen somit auch nicht in der Artikel-Liste).

Hinweis:
Die Artikelinformationen werden nicht verwendet, um eine Instanz des Typs zu erstellen, der dem Artikel in den OAM-Daten zugeordnet ist (falls vorhanden), sondern sie werden den Platzhaltern zugewiesen, die durch Instanzen des internen Typs *xOiTabularPlGroupPlaceholder* repräsentiert werden.

@FieldsPlaceholder4SubArticle

Legt fest, ob Platzhalter-Objekte, die Artikel repräsentieren[44], im Warenkorb als Unterartikel dargestellt werden sollen (`1`) oder nicht (`0`).
Default: `1`

Wenn die Option den Wert `0` hat, besteht keine Notwendigkeit, die Artikelnummer der Platzhalter in einer überschriebenen Methode *getInvalidSubArticleSpecs()* anzugeben (siehe Spezifikation der Basisklasse *xOiPlGroup* in der XOI Dokumentation).

[42] bei horizontaler Ausrichtung der Planungsgruppe
[43] bei vertikaler Ausrichtung der Planungsgruppe
[44] siehe Option *@FieldPlaceholderArticle*

@StartLayout

Gibt die Informationen an, die verwendet werden, um unmittelbar nach Erzeugung der Planungsgruppe (und Zuweisung einer Artikelnummer an diese) initiale Unter-Artikel (als Feldelemente) zu erzeugen.

Das Wert-Feld enthält die OFML String Notation eines Vectors mit den folgenden Elementen:

1. Erzeugungsmodus (*Symbol*):

   `@Uniform`
   Es werden so viele Felder erzeugt, wie im 2. Element angegeben, und diese werden mit Artikeln gemäß der Option *@DefaultArticle* gefüllt.

   `@Specific`
   Erzeugt individuelle Layout-Elemente gemäß der Angaben im 2. Element.

2.
   - Im Modus `@Uniform` ein zweistelliger Vector ganzzahliger Werte, welche die Anzahl der zu erzeugenden Spalten (1. Element) und Zeilen (2. Element) festlegen.
   - Im Modus `@Specific` ein Vector aus Vectoren mit jeweils den folgenden Elementen:
     1. Feld-Adresse des Layout-Elements: [Index der Spalte (*Int*), Index der Zeile (*Int*)][45]
     2. Artikel-Information gemäß der Spezifikation für die Option *@DefaultArticle*

6.5. Spezifische Optionen für *xOiCustomPlGroup*

@StartLayout

Gibt die Informationen an, die verwendet werden, um unmittelbar nach Erzeugung der Planungsgruppe (und Zuweisung einer Artikelnummer an diese) initiale Unter-Artikel (als Layout-Elemente) zu erzeugen.

Das Wert-Feld enthält die OFML String Notation eines Vectors aus Vectoren mit jeweils den folgenden Elementen:

1. Anfügepunkt des Vorgängers, an den dieses Element positioniert werden soll (*Symbol*[46])
2. OFML-Programm (*Symbol*)
3. Grundartikelnummer (*String*)
4. optional: Typ des Variantencodes (*Symbol*: `@VarCode`, `@OFMLVarCode`, `@Final`)
5. optional: Variantencode (*String*) , möglicherweise teil-bestimmt
6. optional: Property-Werte (*Vector* von [key, value]-Paaren)
7. optional: Property-Stati (*Vector* von [key, state]-Paaren)
   Aktuell können die folgenden Status-Werte angegeben werden[47]:
   `0` – Die Property ist nicht sichtbar
   `1` – Die Property ist sichtbar, aber nicht editierbar
   `3` – Die Property ist sichtbar und editierbar

Beispiel:

```
@StartLayout;;\
[[NULL,@foo_bar,"ArtNr1"],\
 [@ATTPT_R1,@foo_bar,"ArtNr2",@OFMLVarCode,"PROP1=814"],\
 [@ATTPT_R2,@foo_bar,"ArtNr3",@VarCode,"",[[@Prop1,1],[@Prop2,@abc]]]]
```

[45] Indizes beginnen mit 0
[46] `NULL` für das erste Element
[47] gemäß der Spezifikation der Methode *setPropState2()* der OFML-Schnittstelle *Property*

@StoreNeighborship

Gibt an, ob Nachbarschaftsbeziehungen zwischen Layoutelementen in einer internen Datenstruktur gespeichert werden sollen (`1`) oder nicht (`0`).
Default: `1`

Wenn Nachbarschaftsbeziehungen auf der Basis von OFML-Anfügepunkten in einem bestimmten Projekt nicht relevant sind, sollte der Wert `0` angegeben werden, um Overhead zu vermeiden (Einsparung von Speicher und Verbesserung der Performanz).

Anhang

A.1 OFML-Schlüsselwörter

Die unten angeführten Schlüsselwörter dürfen nicht als Bezeichner für OCD-Merkmale verwendet werden[48]. Die Reservierung bezieht sich dabei nur auf die unten verwendete Schreibweise in Bezug auf Groß- und Kleinschreibung, d.h., andere Schreibweisen sind erlaubt.

```
abstract
break
case
catch
class
continue
default
do
else
final
finally
for
foreach
func
goto
if
import
instanceof
native
ODB_NAME
operator
package
private
protected
public
return
rule
self
static
super
switch
throw
transient
try
var
while
```

[48] s.a. OFML Spezifikation 2.0.3, Abschn. 3.2.4.

A.2 Tabelle `non_pd_properties`

In dieser Tabelle werden die Properties von Artikelinstanzen angegeben, die nicht mit einem Merkmal in den Produktdaten (OCD) assoziiert sind, d.h., die mittels Programmierung in der OFML-Klasse der Artikelinstanzen definiert werden. Diese Information wird vom globalen Produktdatenmanager verwendet, um Zugriffe auf die Produktdatenbank zu vermeiden (z.B. zur Behandlung einer Änderung einer solchen Property) und somit die Performanz zu verbessern.

Die Tabelle besitzt folgende Struktur:

- Feld 1: vollqualifizierter Name der Klasse
- Feld 2: Property-Keys (OFML Symbole), Komma-separiert

Als Zeichensatz wird ISO-8859-1 (Latin-1) verwendet.

Zu einer Klasse können mehrere Tabelleneinträge vorhanden sein, in welchen Fall dann die Listen der Property-Keys zusammengeführt werden.

Die Tabelle muss sich in der Datenbank `pdate.ebase` der betreffenden OFML-Serie befinden (d.h., zusammen mit der OCD-Daten der Artikel der Serie.)

Die Tabellenbeschreibung für die EBase-Konfigurationstabelle lautet:

```
table  non_pd_properties_tbl non_pd_properties.csv variable_width
fields 2
field  1 class vstring delim ; trim hidx link
field  2 pkey  vstring delim ; trim
```

Wie in Abschnitt 6.1 angemerkt, sollten für eine bessere Performanz die Properties, die in der Option *`@CommonProps`* der Steuerdatentabellen für Planungsgruppen genannt sind, auch in der Tabelle `non_pd_properties` angegeben werden (wenn der Artikel der Planungsgruppe diese Properties nicht besitzt).

Beispiel:

jointplgroup.csv:
```
@CommonProps;[@Article,["GroupXY"]];@Foo,@Bar
```

oam_article2ofml.csv:
```
GroupXY;::ofml::xoi::xOiJointPlGroup;;
```

non_pd_properties.csv:
```
::ofml::xoi::xOiJointPlGroup;@Foo,@Bar
```

A.3 Dokumenthistorie

2023-09-18:

- Neue Option *@UnfixPreselectedChoiceList* in der Tabelle epdfproductdb (Abschn. 5.3)
- Die Steuerdatentabellen für Planungsgruppen unterstützen nun die Bedingungstypen @Property und @PropertyNOT auch für die Optionen *@Classes4CommonProps* und *@ArticleFeatures4Common* (Abschn. 6)
- Neue Optionen *@CommonPropsPos* und *@SortUnionCommonProps* bei allen Planungsgruppen (Abschn. 6.1)
- Präzisierung zur Option *@ArticleFeatures4Common* in Abschn. 6.1
- Neue Option *@FieldPlaceholderType* sowie Präzisierungen zu den Optionen *@AlignmentY* und *@AlignmentZ* in der Tabelle tabularplgroup (Abschn. 6.4)
- Neuer Abschnitt 6.5 zur Tabelle customplgroup (Klasse *xOiCustomPlGroup*)

2022-09-30:

- Präzisierung und Korrektur zum Infotyp @NeedsAddPrices bei der Option *@OrderInfo* in der Tabelle proginfo (Abschn. 2.6)
- Die Steuerdatentabellen für Planungsgruppen unterstützen nun den Bedingungstyp @ArticleX als auch die Bedingungstypen @Property und @PropertyNOT für definierte Optionen, die sich auf gemeinsame Properties der Gruppe beziehen (Abschn. 6)
- Neue Optionen *@IsPseudoArticle*, *@IsNonOrderArticle* und *@CommonPropsChLMode* für Planungsgruppen sowie Präzisierungen zur Option *@CommonProps* im Abschn. 6.1.
- Neue Option *@ContinueBranch4SingleForkEl* in der Tabelle layoutgroup (Abschn. 6.3)

2021-12-22:

- Präzisierung zur Option *@PreselectRestrictable* in der Tabelle epdfproductdb (Abschn. 5.1)

2021-10-11:

- Neue Option *@UnlockBackwardRestriction* in Abschn. 5.3
- Neue Optionen *@SelectableState4Layout* und *@NonLayout4CommonProps* in Abschn. 6.1
- Neue Option *@FieldsPlaceholder4SubArticle* in Abschn. 6.4

2021-03-23:

- Der Begriff *Teilplanung* wurde durch den Begriff *Planungsgruppe* ersetzt (Abschn. 6)
- Neue Optionen *@FirstPos4CommonProps*, *@AllObjs4CommonProps* und *@Classes4CommonProps* in Abschn. 6.1
- Korrektur bzgl. Option *@StartLayout* in der Tabelle jointplgroup (Abschn. 6.2)
- Neuer Anhang A.2 mit Beschreibung der Tabelle non_pd_properties (vermeidet Referenz auf die XOI-Dokumentation in Abschn. 6.1)

2020-11-04:

- Neue Option *@NeedPropGroupDescriptions* in der Tabelle proginfo
- Ergänzende Hinweise zu den Optionen *@InsertMode* und *@CommonProp* im Abschn. 6.1 (Gemeinsame Optionen in den Steuerdatentabellen für Planungsgruppen)
- Neue allgemeine Option *@ArticleFeatures4Common* im Abschn. 6.1
- Neue Option *@StartLayout* in der Tabelle jointplgroup (Abschn. 6.2)
- Neuer Abschnitt 6.4 mit den Beschreibungen der Optionen für die Tabelle tabularplgroup

2019-11-29:

- Explizite Beschreibung der Optionen für die Tabellen jointplgroup und layoutgroup (anstatt Verweis auf die XOI-Dokumentation)

2019-10-24:

- Neue Optionen *@DeactivateValuePreconds*, *@NeedValuesForRGProps* und *@RelEvalOptimization* in der Tabelle epdfproductdb
- Option *@SetDefaultMode* (Tabelle epdfproductdb) ist ab den Herbst-Releases 2019 obsolet

2019-07-05:

- Neue Optionen *@IgnoreUnknownPropInKeyValueVC* und *@SetVisibiltyMode* in der Tabelle epdfproductdb
- Präzisierung zur Option *@SafePropertyNames* in der Tabelle epdfproductdb inkl. neuem Anhang A.1 mit den OFML Schlüsselwörtern
- Neuer Abschnitt zu den Tabellen jointplgroup und layoutgroup

2017-01-24:

- In der Tabelle proginfo Beschreibung zum Infotyp @DeepSumOrder in der Option *@OrderInfo* ergänzt
- Hinweis auf die Option *@AppInteractorDefs* in der Tabelle plelement ergänzt
- Neue Optionen *@OCDUserMessageMode* und *@AllowConsecValsInTrimmedCode* in der Tabelle epdfproductdb

2014-07-28:

- Anpassung an neuen CI-Stil
- Leichte Umstrukturierung im Abschnitt 2
- Neue Optionen *@UseUnspecPropVal4OdbInfo* und *@UseVoidPropVal4OdbInfo* in der Tabelle proginfo
- Neue Option *@OptPropsWithBaseValues* in der Tabelle epdfproductdb

2013-10-25:

- Platzhalter %P in der Option *@ShowExtraCharge4Value* (Tabelle epdfproductdb) ist nun optional
- Neue Optionen *@SafePropertyNames* und *@SetDefaultMode* in der Tabelle epdfproductdb

2012-05-23:

- Ergänzende Anmerkung zur Option *@PricesOnRequest* in der Tabelle proginfo.
- Neuer Parameter-Typ @Region für Option *@OrderInfo* in der Tabelle proginfo (inkl. Beispiele).
- Neuer Unterabschnitt für spezifische Optionen der nativen OCD-Implementierung (Tabelle epdfproductdb)

2010-12-17:

- Option *@InsignificantPropClasses* in der Tabelle epdfproductdb ergänzt

2010-09-14:

- Die Dokumentstruktur ist nun um die Tabellen (anstatt um die OFML Typen) zentriert und damit besser für die Datenanleger geeignet.
- Kleinere stilistische Änderungen und Korrekturen.
- Aktualisierung der Abschnitte zu den Tabellen proginfo, plelement und epdfproductdb.
- Entfernung des Abschnittes zur obsoleten Tabelle eplproductdb.

2007-08-24:

- Viele Ergänzungen zur Klasse *xOiProgInfo*

2007-01-08:

- Ergänzungen und Korrekturen

2006-11-09:

- Initiale Version