EasternGraphics

visualize your business

Die OFML–Schnittstelle *Property*

Version 2.9

Thomas Gerth, EasternGraphics GmbH (Editor)

2023-09-14

Rechtliche Hinweise

Inhaltsverzeichnis

Literatur

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

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

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

Die Dokumente sind über das Download Center von EasternGraphics

https://download-center.pcon-solutions.com

in der Kategorie *OFML Specifications* verfügbar.

1 Einleitung und Allgemeines

Die Festlegungen in diesem Dokument ersetzen den Abschnitt 4.4[1] im Part III der OFML–Spezifikation [ofml]. Die dort beschriebenen Methoden können jedoch auch weiterhin verwendet werden. Sie werden der Vollständigkeit halber zwar noch im Anhang B aufgelistet, aber nicht mehr spezifiziert. Bei Bedarf ist [ofml] zu konsultieren.

Diese Dokumentversion bezieht sich auf Version 1.43 der OFML–Basisbibliothek OI (Herbst 2023).

1.1 Motivation zur Überarbeitung der Schnittstelle

- Ersatz der Methode *getProperties()* mit ihrer String-basierten Schnittstelle durch separate Methoden zum Abrufen der Property-Defintionen und zum Abrufen der Choicelisten, die jeweils eine definierte Struktur liefern.
  (Damit entfällt das Parsen der String-basierten Property-Definitionen und Probleme mit speziellen Zeichen in Werten und Texten werden vermieden.)
- Analoge Auftrennung von Property-Definition und Choiceliste-Definition beim OFML-seitigen Festlegen einer Property.
- Bereinigung von Unstimmigkeiten der bisherigen Property-Definitionen.
- Support für neue Anforderungen, die sich aus Projekten bzw. der OCD-Weiterentwicklung ergeben haben:
  - mehrere Intervalle
  - Raster
  - Darstellung nicht auswählbarer Werte in Choicelisten
  - Darstellung von Aufpreisen zu Werten in Choicelisten
- Neuer Property-Typ für Choicelisten, deren Werte Strings sind.
  (Die Notwendigkeit der Konvertierung von OCD-Werten in OFML-Symbole würde damit entfallen und damit verbundene Probleme würden vermieden.)
- Verbesserte Sprach–Behandlung.

1.2 Attribute von Properties

Die Attribute von Properties kann man wie folgt unterteilen:

1. Unveränderliche Attribute, die den Typ der Property festlegen: Typ der Werte, Formatierungsangaben etc.
2. Attribute, die von der Konfiguration des Objektes/Artikels abhängen: Wert, Choiceliste, Wertebereich(e), Status, Position in der Liste der Properties
3. sprachabhängige Bezeichner für Property und Choicelisten-Werte

Die Attribute der ersten Gruppe werden in der Schnittstelle immer in Form eines Vectors übergeben, der als *Property-Definition* bezeichnet wird (s. nächster Abschn.).

Für die Property-Definition und die sonstigen Attribute gibt es je eigene Methoden zum Setzen und Abrufen.

[1] in diesem Dokument im Weiteren als *alte* bzw. *bisherige* Property-Schnittstelle referenziert

1.3 Die Property–Definition

Wie im vorangegangenen Abschnitt erklärt, werden unter dem Begriff *Property-Definition* alle unveränderlichen Attribute zusammengefasst, die den Typ der Property festlegen. Diese werden immer zusammen über einen Vector zugewiesen und abgerufen, der folgenden Aufbau besitzt:

```
[<type>, <width>, <dec-places>, <choice-list-type>]
```

Die Bedeutung der Attribute ist wie folgt:

1. **type** (String) – Typ der Property:

   Der Property-Typ legt die grundsätzliche Art und Weise der Eingabe und Darstellung von Werten sowie deren OFML-Datentyp fest.

   Folgende Typen sind definiert (entsprechender OFML-Datentyp in runden Klammern):

   L Länge in Meter (Float)

   Der Wert (in *m*) wird entsprechend den Nutzereinstellungen (insbesondere entsprechend der eingestellten Einheit) formatiert und angezeigt. Bei der Eingabe wird der in *m* umgerechnete Wert vor Übergabe an OFML auf die in `dec-places` angegebene Anzahl von Dezimalstellen gerundet[2].
   Das Attribut `width` wird generell ignoriert.

   A Winkel im Bogenmaß (Float)

   Der Wert (in *rad*) wird entsprechend den Nutzereinstellungen (insbesondere entsprechend der eingestellten Einheit) formatiert und angezeigt. Bei der Eingabe wird der in *rad* umgerechnete Wert vor Übergabe an OFML auf die in `dec-places` angegebene Anzahl von Dezimalstellen gerundet[2]
   Das Attribut `width` wird generell ignoriert.

   N Zahl (Float, Int)

   Abhängig von `dec-places` handelt es sich um eine ganze Zahl oder um eine Zahl mit Nachkommastellen. Das Attribut `dec-places` legt für die Anzeige die Anzahl der darzustellenden Dezimalstellen fest. Bei der Eingabe wird der eingegebene Wert vor Übergabe an OFML auf die angegebene Genauigkeit gerundet. Wenn `dec-places` den Wert `0` hat, dann werden bei der Darstellung keine Tausender-Trennzeichen verwendet, und bei der Eingabe wird die Eingabe von anderen Zeichen als Dezimalziffern und eines Vorzeichens an erster Stelle unterbunden. Ansonsten erfolgt die Darstellung und das Parsen eines eingegebenen Wertes entsprechend den Regeln der aktuellen Locale.
   Das Attribut `width` wird generell ignoriert.

   B Wahrheitswert (Int`{0,1}`)

   Die Attribute `width`, `dec-places` und `choice-list-type` werden ignoriert[3]. Die Darstellung und die Formen der Eingabe sind dem GUI überlassen.

   S Zeichenkette (String)

   Die Attribute `dec-places` und `choice-list-type` werden ignoriert[4]. Der aktuelle Wert wird unabhängig von `width` immer komplett dargestellt. Von den in OFML–Zeichenketten möglichen Escape-Sequenzen sind dabei nur `\"` (Anführungszeichen), `\'` (Hochkomma) und `\\` (Gegenschrägstrich) erlaubt. Enthält der Wert–String andere Escape-Sequenzen (z.B. eine Newline-Escape-Sequenz), so ist das Verhalten undefiniert.
   Bei der Eingabe wird das Hinzufügen von Zeichen unterbunden, wenn `width` erreicht oder überschritten ist. Im Falle von Unicode wird `width` als die Anzahl Code Points nach Konvertierung in NFC verstanden.

[2] Ob und in welcher Form das Attribut `dec-places` auch bei der Anzeige berücksichtigt wird, ist den Applikationen überlassen.
[3] Die entsprechenden Elemente der Property-Definition sollten die Werte `1`, `0` bzw. `@None` haben.
[4] Die entsprechenden Felder in der Property-Definition sollten die Werte `0` bzw. `@None` haben.

T (mehrzeiliger) Text (String)

Die Attribute `width`, `dec-places` und `choice-list-type` werden ignoriert[5]. Eine Newline-Escape-Sequenz im aktuellen Wert führt bei der Anzeige zu einem Zeilenumbruch. Umgekehrt wird bei der Übergabe an OFML ein Zeilenumbruch durch eine Newline-Escape-Sequenz ersetzt.

Y, YS Symbolische Auswahlliste (Symbol, String)

Die Attribute `width` und `dec-places` werden ignoriert[6].

`choice-list-type` muß `@FixedSingleV` oder `@FixedMultiV` sein.

`"Y"` und `"YS"` verhalten sich im Wesentlichen identisch. Der Unterschied besteht darin, dass im Fall von `"Y"` der Wert vom Typ *Symbol* ist, während er im Fall von `"YS"` vom Typ *String* ist.

CT ... Spezial–Typ[7] (Any)

Mit diesem Typ können (noch) nicht standardisierte oder proprietäre Property-Typen abgebildet werden. Dem `"CT "` folgen Angaben zu dem Spezial–Typ. Die Attribute `width`, `dec-places` und `choice-list-type` können mit genutzt werden.

Die Verfügbarkeit eines Spezial-Typs in der aktuell verwendeten Applikation kann/sollte OFML–seitig mit Hilfe der Application Callback *::ofml::app::isPropTypeSupported()* geprüft werden (s. Abschn. 3.4).

2. **width** (Int) – maximale Eingabelänge:

   Ist nur für Properties des Typs `"S"` relevant: legt für diese die maximale Länge der Zeichenkette bei der Eingabe fest[8]. Bei allen anderen Typen ist für `width` in der Property-Definition ein mehr oder weniger sinnvoller Wert anzugeben[9].

3. **dec-places** (Int) – Anzahl der Dezimalstellen:

   Ist nur für Properties der Typen `"L"`, `"A"` und `"N"` relevant[10].

4. **choice-list-type** (Symbol) – Art der Auswahlliste:

   Legt fest, ob eine Auswahlliste existiert und falls ja, von welcher Art.
   Ist nicht für Properties der Typen `"B"`, `"S"` und `"T"` relevant[11].

   @None Keine Choiceliste vorhanden.

   Nicht gültig für Properties der Typen `"Y"` und `"YS"`.

   @FixedSingleV Feste Auswahlliste, nur ein Wert daraus auswählbar.

   Die freie Eingabe von Werten ist nicht möglich.

   @FixedMultiV Feste Auswahlliste, mehrere Werte daraus auswählbar.

   Die freie Eingabe von Werten ist nicht möglich.

   Nur gültig für Properties der Typen `"Y"` und `"YS"`.

   @OpenSingleV Offene Auswahlliste, jeweils nur 1 Wert daraus auswählbar.

   Die freie Eingabe von zusätzlichen Werten ist erlaubt (im Rahmen der vorgegebenen Wertebereiche).

   Nicht gültig für Properties der Typen `"Y"` und `"YS"`.

   Die Definition einer Property mit einer ungültigen Kombination von Property-Typ und Art der Auswahlliste wird zurückgewiesen, d.h., der Aufruf der Methode *setupProperty2()* (s. 2.1) hat dann keinen Effekt.

[5] Die entsprechenden Felder in der Property-Definition sollten die Werte `0`, `0` bzw. `@None` haben.
[6] `width` sollte einen Wert haben, der nicht kleiner ist als die Länge des längsten der möglichen symbolischen Werte. Der Wert für `dec-places` sollte `0` sein.
[7] custom type
[8] Mittels *setPropValue()* kann der Property aber eine Zeichenkette zugewiesen werden, die länger ist als `width`.
[9] Wenn kein sinnvoller Wert ermittelt werden kann, ist `0` zu verwenden.
[10] Für alle anderen Typen ist in der Property-Definition der Wert `0` zu verwenden.
[11] Für diese Typen ist in der Property-Definition der Wert `@None` zu verwenden.

In der folgenden Tabelle sind die möglichen Kombinationen der Attribute mit dem jeweils entsprechenden bisherigen Property–Typ dargestellt.

type	width	dec-places	choice-list-type	bisheriger Property–Typ	OFML-Datentyp
L	-	-	`@None`	`"f"`, Fmt `"@L"`	Float
L	-	-	`@FixedSingleV`	`"u chf"`, Fmt `"@L"`	Float
L	-	-	`@OpenSingleV`	`"u chf.edit"`, Fmt `"@L"`	Float
A	-	-	`@None`	`"f"`, Fmt `"@A"`	Float
A	-	-	`@FixedSingleV`	`"u chf"`, Fmt `"@A"`	Float
A	-	-	`@OpenSingleV`	`"u chf.edit"`, Fmt `"@A"`	Float
N	-	0	`@None`	`"i"`	Int
N	-	0	`@FixedSingleV`	`"u chi"`	Int
N	-	0	`@OpenSingleV`	`"u chi.edit"`	Int
N	-	n	`@None`	`"f"`	Float
N	-	n	`@FixedSingleV`	`"u chf"`	Float
N	-	n	`@OpenSingleV`	`"u chf.edit"`	Float
B	-	-	-	`"b"`	Int
S	n	-	-	`"s"`	String
T	-	-	-	`"t"`	String
Y	-	-	`@FixedSingleV`	`"ch"`, `"chf"`	Symbol
Y	-	-	`@FixedMultiV`	`"mch"`	Symbol[]
YS	-	-	`@FixedSingleV`	(`"ch"`, `"chf"`)[a]	String
YS	-	-	`@FixedMultiV`	-	String[]
CT	*	*	*	`"u ..."`	*

[a] Obwohl die (alte) OFML–Spezifikation eigentlich nur Symbole als Choicelisten-Werte zulässt, unterstützen die Applikationen von EasternGraphics auch Choicelisten, deren Werte Strings sind (ohne sprachspezifischen Anzeigetext). Dies wird z.B. in der Metatyp-basierten Datenanlage ausgenutzt. Derartige Properties verhalten sich gemäß der neuen Schnittstelle im Prinzip wie Properties des Typs "YS", wobei Wert und sprachspezifischer Anzeigetext identisch sind.

1.4 Sprach–Text–Mappings

Sprachspezifische Bezeichnungen von Properties und Property–Werten werden entweder mittels einer Text–Ressource–ID abgebildet oder beziehen sich auf die Sprache, die aktuell laut der Methode *OiPlanning::getPDLanguage()* für Produktdatentexte zu verwenden ist, siehe 3.4.

Einige Clienten erfordern zur effizienten Arbeit jedoch die Texte in allen verfügbaren Sprachen. Die Rückgabestrukturen einiger Methoden im Abschn. 2.7 enthalten dazu sogenannte *Sprach–Text–Mappings*:

Ein Sprache–Text–Mapping besteht aus einem *Vector* aus null oder mehr Elementen, wobei jedes Element wiederum ein *Vector* aus zwei Elementen ist:

1. Sprachcode (*String*) gemäß ISO 639-1 Alpha-2 oder eine leere Zeichenkette (s.u.)
2. Text in der entsprechenden Sprache (*String*)

Der Sprachcode, wenn nicht leer, muss aus zwei Kleinbuchstaben bestehen. Buchstabenkombinationen, die keinem offiziell registriertem Code entsprechen, sind nicht explizit erlaubt, sollten bei der Verarbeitung seitens des Clienten aber nicht zu Fehlern führen.

Eine leere Zeichenkette im 1. Element ist gleichbedeutend mit einer *undefinierten Sprache*. Die zugehörigen Texte können vom Clienten als Default/Fallback verwendet werden für den Fall, dass das Mapping keinen Text für die (oder eine) vom Clienten gewünschte Sprache enthält.

Ein Sprache–Text–Mapping darf keine zwei Einträge mit dem gleichen Sprachcode enthalten!

2 Die Methoden

Die Methoden, die in der bisherigen Schnittstellen–Spezifikation noch nicht enthalten waren, sind mit **new** gekennzeichnet.

Einige Methoden wurden im Sinne einer Bereinigung/Vereinheitlichung bei der Namensgebung umbenannt. Dies betrifft u.a. die Verwendung von „Property“ im Methodennamen: Folgt diesem Wort ein weiteres Wort (z.B. „State“), so wird es mit „Prop“ abgekürzt.

Bei gleichem Zweck aber anderer Spezifikation wird der Name der neuen Methode aus dem Namen der bisherigen Methode durch Anhängen von „2“ gebildet.

2.1 Property–Einrichtung

Die Methoden dieser Gruppe dienen zum Einrichten einer Property inklusive Position und Aktivierungsstatus sowie zum Abrufen der entsprechenden Attribute. Weitere Methoden zum komfortablen, gleichzeitigen Abrufen mehrer Attribute finden sich in der Gruppe “Client–Support“ (s. Abschn. 2.7).

- *setupProperty2(pKey(Symbol), pDef(Any[4]), pName(String), pPos(Int), pState(Int)) → Void* **new**

  Die Methode legt eine Property mit dem angegebenen Schlüssel (Identifikator) an und weist ihr die übergebene Definition, den übergebenen sprachspezifischen Bezeichner, die übergebene Position und den übergebenen Aktivierungsstatus zu.

  Ist bereits eine Property mit dem angegebenen Schlüssel aber einer anderen Definition registriert, so hat der Methodenaufruf keinen Effekt.

  Die Definition einer Property (Parameter *pDef*) ist ein Vector aus vier Elementen. Der Aufbau des Vectors und die Bedeutung der Elemente ist in Abschn. 1.3 ausführlich beschrieben.

  Die sprachspezifische Bezeichnung der Property (Parameter *pName*) kann nachträglich mit der separaten Methode *setPropName()* geändert werden. Details zur Verwendung des Parameters siehe dort/unten.

  Die Position der Property in der Property–Liste (Parameter *pPos*) kann nachträglich mit der separaten Methode *setPropPos()* geändert werden. Details zur Verwendung des Parameters siehe dort/unten.

  Der Aktivierungsstatus der Property (Parameter *pState*) kann nachträglich mit der separaten Methode *setPropState2()* geändert werden. Details zur Verwendung des Parameters siehe Abschn. 2.2.

  Die sprachspezifische Bezeichnung, die Position und der Status können auch nachträglich mittels separater Methoden geändert werden (s.u.).

- *getPropDef2(pKey(Symbol) → Any[4] | Void* **new**[12]

  Die Methode liefert die Definition der Property mit angegebenen Schlüssel. Der Aufbau des zurückgegebenen Vectors entspricht dem Aufbau des als Property–Definition an die Methode *setupProperty2()* übergebenen Parameters *pDef*.

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, ist der Rückgabewert vom Typ *Void*.

- *removeProperty(pKey(Symbol)) → Void*

  Die Methode entfernt die durch den angegebenen Schlüssel spezifizierte Property aus der Property-Liste.

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, hat die Methode keinen Effekt.

[12] Der Name *getPropDef()* wäre für diese neue Methode auch möglich, könnte aber inhaltlich mit der alten Methode *getPropertyDef()* verwechselt werden, welche die alte Form der Property–Definition liefert. Analog zu *setupProperty2()* wird hier also auch noch „2“ angehängt.

- *clearProperties() → Void*

  Die Methode entfernt alle Properties aus der Property-Liste.

- *getPropKeys() → Symbol[]* **new** [13]

  Die Methode liefert eine Liste der Schlüssel aller aktuell für die implizite Instanz definierten Properties.

  Die Properties werden dabei nach ihren expliziten Positionen in aufsteigender Reihenfolge sortiert. Die Properties ohne explizite Position erscheinen am Ende der Liste in undefinierter Reihenfolge.

- *hasProperties() → Int*

  Die Methode liefert `1`, wenn für die implizite Instanz Properties definiert sind, andernfalls `0`.

- *hasProperty(pKey(Symbol)) → Int*

  Die Methode liefert `1`, wenn für die implizite Instanz eine Property mit angegebenen Schlüssel definiert ist, andernfalls `0`.

- *setPropName(pKey(Symbol), pName(String)) → Void* **new**

  Die Methode weist der Property mit dem angegebenen Schlüssel eine sprachspezifische Bezeichnung zur Anzeige im Property–Editor zu.

  Der Parameter *pName* kann eine Text–Ressource–ID sein, die dann durch die OFML–Laufzeitumgebung sprachspezifisch über externe Ressourcen-Dateien aufgelöst wird. Ist der Parameter keine Text–Ressource–ID, so muß die Bezeichnung in der Sprache übergeben werden, die aktuell laut der Funktion *getPDLanguage()* der Schnittstelle *Article* [article] für Produktdatentexte zu verwenden ist (s.a. 3.4).

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, hat die Methode keinen Effekt.

- *getPropName(pKey(Symbol)) → String | Void* **new**

  Die Methode liefert die sprachspezifische Bezeichnung, die aktuell der impliziten Instanz für die Property mit dem angegebenen Schlüssel zugewiesen ist.

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, so ist der Rückgabewert vom Typ *Void*.

- *setPropPos(pKey(Symbol), pPos(Int |Void)) → Int | Void* **new** [14]

  Die Methode legt für die Property mit dem angegebenen Schlüssel die gewünschte Position in der Property-Liste fest.

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, hat die Methode keinen Effekt und der Rückgabewert ist vom Typ *Void*.

  Ist für die implizite Instanz eine Property mit dem angegebenen Schlüssel definiert, wird die alte Positionsangabe überschrieben. Ist *pPos* eine ganze Zahl größer gleich `0` und wurde die gewünschte Position bereits für eine andere Property vergeben, so werden diese und alle in der Positionsliste nachfolgenden Properties um eine Position nach hinten verschoben. Ist *pPos* vom Typ *Void* oder hat es den Wert `-1`, ist für die Property keine spezielle Position gefordert. Sie wird dann in der Property–Liste nach den Properties einsortiert, für die explizit eine Position angefordert wurde. Rückgabewert ist dann die neue Position der Property bzw. `-1`, wenn keine spezielle Position gefordert ist.

[13] ist identisch mit der bisherigen Methode *getPropertyKeys()*
[14] ist identisch mit der bisherigen Methode *setPropPosOnly()*

- *getPropPos(pKey(Symbol))* → *Int* | *Void* **new**[15]

  Die Methode liefert die Position der Property der impliziten Instanz mit dem angegebenen Schlüssel.

  Wurde für die Property via *setupProperty2()* oder *setPropPos()* keine spezielle Position angefordert, ist der Rückgabewert `-1`.

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, ist der Rückgabewert vom Typ *Void*.

- *setExtPropOffset(pOffset(Int))* → *Void*

  Mit dieser Methode wird der impliziten Instanz ein Offset für Positionen von extern definierten Properties zugewiesen, d.h. von Properties, die von einer anderen als der impliziten Instanz für diese definiert werden.

  Der Offset gibt die kleinste Positionsnummer an, die für extern definierte Properties verwendet werden darf.

  Der Default–Offset ist `0`.

- *getExtPropOffset()* → *Int*

  Die Methode liefert den Offset für Positionen von extern definierten Properties, d.h. von Properties, die von einer anderen als der impliziten Instanz für diese definiert werden.

  Der Offset gibt die kleinste Positionsnummer an, die für extern definierte Properties verwendet werden darf. Dieser Offset ist von einer externen Instanz vor der Definition einer Property für die implizite Instanz abzurufen und bei der Vergabe von expliziten Positionen zu berücksichtigen.

2.2 Aktivierungsstatus

- *setPropState2(pKey(Symbol), pState(Int))* → *Void* **new**

  Die Methode setzt für die implizite Instanz den Aktivierungsstatus der Property mit dem angegebenen Schlüssel auf den übergebenen Wert.

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, hat die Methode keinen Effekt.

  Der Status ist eine Kombination (Addition) der folgenden Flags (Bits):

  `1` die Property ist sichtbar

  `2` die Property ist durch den Anwender editierbar

  `4` es ist noch eine Werteingabe durch den Anwender erforderlich, damit die Konfiguration vollständig bzw. gültig ist

  `8` die Property ist aktuell für die implizite Instanz nicht definiert (kann aber in einer anderen Konfiguration vorhanden sein)

  Das Flag mit dem Wert `8` ist den Rückgabestrukturen von bestimmten Methoden aus dem Abschn. 2.7 vorbehalten. Im Parameter *pState* von *setPropState2()* muss dieses Flag immer den Wert `0` haben!
  Die möglichen Kombinationen für *setPropState2()* sind damit:

sichtbar	editierbar	Eingabe erfordert	Property nicht def.	Status-Wert (alt)	Erklärung
0	0	0	0	0 (-1)	nur intern nutzbare Property
1	0	0	0	1 (0)	sichtbar, aber nicht-aktiv
1	1	0	0	3 (1)	sichtbar und editierbar
1	1	1	0	7 (n.a.)	sicht- und editierbar, Eingabe noch erforderlich

  Wird im Parameter *pState* ein ungültiger Wert übergeben, hat die Methode keinen Effekt.

[15] ist identisch mit der bisherigen Methode *getPropertyPos()*

- *getPropState2(pKey(Symbol)) → Int* **new**

  Die Methode liefert den aktuellen Aktivierungsstatus der Property der impliziten Instanz mit dem angegebenen Schlüssel.

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, ist der Rückgabewert `0`.

- *invalidateProperties() → Void*

  Die Methode setzt den Aktivierungsstatus aller aktuell definierten Properties der impliziten Instanz, deren aktueller Status `3` ist (aktiv/sichtbar), auf `1` (inaktiv/sichtbar).

  Diese Methode kann von Clienten verwendet werden, um die Konfiguration von Artikelinstanzen zu verhindern, die nicht aktualisiert werden konnten.

2.3 Wertebereiche und Auswahllisten

- *setPropRanges(pKey(Symbol), pRanges(Any)) → Void* **new**

  Die Methode weist der numerischen Property der impliziten Instanz mit dem angegebenen Schlüssel null, einen oder mehrere[16] Wertebereiche zu.

  Die relevanten Property-Typen sind `"L"`, `"A"` und `"N"`. Der Choicelist-Typ der Property darf dabei nicht `@FixedSingleV` sein[17].

  Ist der Choicelist-Typ der Property `@OpenSingleV`, kann die Choiceliste Werte enthalten, die außerhalb der zugewiesenen Wertebereiche liegen[18]!

  Der Parameter *pRanges* ist entweder vom Typ *Void* oder ein Vector, bestehend aus einem oder mehreren Vectoren mit je 3 Elementen, die je einen Wertebereich definieren:

  1. Minimum
  2. Maximum
  3. Schrittweite (Raster)

  Die angegebenen Werte für Minimum und Maximum gehören dabei zu dem gültigen Wertebereich. Es werden also die Vergleichsoperatoren `>=` bzw. `<=` angewendet[19].

  Einer der beiden Werte für Minimum und Maximum kann vom Typ *Void* sein, um einen Bereich anzugeben, der keine untere bzw. obere Grenze besitzt.

  Ist das 3. Element vom Typ *Void*, so bestehen keine weiteren Beschränkungen hinsichtlich der erlaubten Werte innerhalb des spezifizierten Wertebereiches. Ist ein Minimum angegeben (nicht vom Typ *Void*), so kann im 3. Element eine Schrittweite angeben werden, die, beginnend ab dem Minimum, bei der Eingabe der Werte eingehalten werden muß und somit die erlaubten Werte innerhalb des spezifizierten Wertebereiches einschränkt[20].

  Ungültige Wertebereich–Definitionen im Parameter *pRanges* werden ignoriert. Konkret betrifft das folgende Situationen:

  – Sowohl Minimum als auch Maximum sind vom Typ *Void*.
  – Das Minimum ist vom Typ *Void* und die Schrittweite ist nicht vom Typ *Void*.

[16] Die Eigenschaftseditoren der Applikationen von EasternGraphics unterstützen aktuell nur einen Wertebereich.
[17] Ist der Choicelist-Typ `@FixedSingleV`, hat diese Methode keinen Effekt.
[18] denn die Wertebereiche definieren in diesem Fall die Bereiche, in denen der Anwender *zusätzlich* zu den Werten aus der Auswahlliste noch Werte frei eingeben darf
[19] Auf die Möglichkeit der Angabe der Vergleichsoperatoren `>` bzw. `<` wurde im Interesse einer möglichst schlanken Schnittstelle verzichtet, da durch Einbeziehung von `dec-places` immer ein kleinster bzw. größter möglicher Wert angegeben werden kann.
[20] Die Eigenschaftseditoren der Applikationen von EasternGraphics unterstützen aktuell noch kein Raster in einem Wertebereich. Bei Bedarf muss dann eine Choicelist angegeben werden, in der die Rasterwerte explizit enthalten sind.

- *getPropRanges(pKey(Symbol))* → *Any* **new**

  Die Methode liefert die aktuell für die numerische Property der impliziten Instanz mit dem angegebenen Schlüssel festgelegten Wertebereiche.

  Der Rückgabewert ist vom Typ *Void*, wenn die implizite Instanz keine Property mit dem angegebenen Schlüssel besitzt, oder wenn der Property keine Wertebereiche mittels der Methode *setPropRanges()* zugewiesen wurden. Anderenfalls entspricht der Rückgabewert dem Parameter *pRanges* der Methode *setPropRanges()*.

- *setPropChoiceList(pKey(Symbol), pChoiceList(Any))* → *Void* **new**

  Die Methode weist der Property der impliziten Instanz mit dem angegebenen Schlüssel eine Choiceliste zu.

  Die Methode hat nur dann einen Effekt, wenn der aktuelle Choicelist-Typ der Property *nicht* `@None` ist.

  Die Choiceliste kann explizit als Liste von *Wertspezifikationen* (s.u.) übergeben werden, oder indirekt über die Angabe einer Methode der impliziten Instanz spezifiziert werden, welche die Liste der Wertspezifikationen liefert. Im letzteren Fall ist der Parameter *pChoiceList* vom Typ *String* und enthält den Namen der Methode inklusive Parameter-Klammern und möglicher konstanter Parameter-Werte[21]. Per default wird die Methode beim Aufruf von *getPropChoiceList()* (s.u.) aufgerufen[22].

  Eine **Wertspezifikation** ist ein Vector aus folgenden Elementen:

  1. Wert (*Any*)
  2. sprachspezifische Bezeichnung (*Void* | *String*)
  3. Status (*Int*)
  4. Aufpreis-Betrag (*Void* | *Float*)
  5. Aufpreis-Währung (*Void* | *String*)

  Der Typ des Wertes (1. Element) ergibt sich aus dem Property–Typ der aktuellen Property–Definition. Der Wert kann jedoch auch vom Typ *Void* sein, wenn es dem Anwender erlaubt sein soll, explizit den unbewerteten Zustand der Property einzustellen[23].

  Die sprachspezifische Bezeichnung (2. Element) ist optional. Ist eine sprachspezifische Bezeichnung angegeben (Typ *String*), wird diese im Property–Editor anstelle des Wertes selber angezeigt, mit folgenden Einshränkungen:

  – Bei den Property–Typen `"L"` und `"A"` wird eine evtl. angegebene sprachspezifische Bezeichnung ignoriert.
  – Beim Choicelisten–Typ `@OpenSingleV` (Property–Typ `"N"`) wird eine evtl. angegebene sprachspezifische Bezeichnung nur in der aufgeklappten Choiceliste selber angezeigt.

  Die Bezeichnung sollte in der Sprache übergeben werden, die aktuell laut der Funktion *getPDLanguage()* der Schnittstelle *Article* [article] für Produktdatentexte zu verwenden ist (s.a. 3.4). Ist keine sprachspezifische Bezeichnung angegeben (Typ *Void*), wird bei Properties der Typen `"Y"` und `"YS"` der Wert als eine Text–Ressource–ID interpretiert, die durch die OFML–Laufzeitumgebung sprachspezifisch über externe Ressourcen-Dateien aufgelöst wird. War die Text–Ressource–Auflösung nicht erfolgreich bzw. ist der Property–Typ nicht `"Y"` und `"YS"`, so wird in der Choiceliste des Property–Editors der Wert selber angezeigt.

[21] Der OFML–Programmierer muss hierbei beachten, dass die Choicelisten-Methode nun eine Liste mit Wertspezifikationen liefern muss, und keinen String mit der Choicelisten-Repräsentation entsprechend der alten Property-Schnittstelle (Methode *setupProperty()*).

[22] Im Gegensatz zu den Methoden *getProperties()* und *getPropertyDef()* der alten Schnittstelle wird der Aufruf der Methode nicht dem Clienten überlassen, sondern bereits innerhalb von *getPropChoiceList()* ausgeführt. Dies ist zum einen für die Komfort-Methode *getPropChoiceList2()* (s. Abschn.2.7) erforderlich, zum anderen auch zur Unterstützung der alten Property-Definition.

[23] Dies wird z.B. bei aktuell nicht bewerteten numerischen einschränkbaren OCD–Merkmalen genutzt, s. Abschn. 3.2.

Der *Status* im 3. Element ist eine Kombination (Addition) der folgenden Flags (Bits):

`1` der Wert ist auswählbar

Hat das Flag den Wert `0`, so ist der Wert in der Choiceliste zwar anzuzeigen, er darf aber nicht ausgewählt werden. In den Property–Editoren sind diese Werte auf geeignete Weise von den effektiv auswählbaren Werten abzuheben, z.B. durch eine eingegraute Darstellung.

`2` der Wert ist in der aktuellen Choiceliste der impliziten Instanz nicht enthalten (kann aber in einer anderen Konfiguration enthalten sein)

Dieses Flag ist den Rückgabestrukturen von bestimmten Methoden aus dem Abschn. 2.7 vorbehalten. Bei der Übergabe an *setPropChoiceList()* muss dieses Flag immer den Wert `0` haben!

Die möglichen Werte für das 3. Element bei Übergabe an *setPropChoiceList()* sind damit `0` und `1`. (Ein Eintrag in der Choiceliste mit einem anderen Wert wird ignoriert.)

Die Elemente 4 und 5 sind optional. Sind sie nicht vom Typ *Void*, so spezifizieren sie Betrag und Währung des Aufpreises, der bei Auswahl des Wertes geltend gemacht wird. Von den Property–Editoren ist dieser Aufpreis in der Choiceliste auf geeignete Weise darzustellen, z.B. dem Wert bzw. seiner Bezeichnung in Klammern nachgestellt. Dabei muß ggf. eine Umrechnung in die Währung vorgenommen werden, die aktuell in der Applikation eingestellt ist.

- *getPropChoiceList(pKey(Symbol), ...)* → *Any* **new**

  Die Methode liefert die aktuell für die Property der impliziten Instanz mit dem angegebenen Schlüssel festgelegte Choiceliste.

  Der Rückgabewert ist vom Typ *Void*, wenn die implizite Instanz keine Property mit dem angegebenen Schlüssel besitzt, oder wenn der Property keine Choiceliste mittels der Methode *setPropChoiceList()* zugewiesen wurde.

  Wurde beim letzten Aufruf von *setPropChoiceList()* eine explizite Choiceliste übergeben, so wird diese direkt als Rückgabewert geliefert.

  Anderenfalls, wenn die Choiceliste indirekt durch Angabe des Namens einer Methode spezifiziert ist, hängt das Verhalten vom optionalen Parameter (vom Typ *Int*) ab:

  – Ist kein optionaler Parameter angegeben (default) oder hat dieser den Wert 1, wird zuerst diese Methode aufgerufen und dann deren Ergebnis als Rückgabewert geliefert.
  – Hat der optionale Parameter den Wert 0, wird der Name der Methode geliefert[24].

2.4 Property–Werte

- *getPropValue2(pKey(Symbol))* → *Any* **new**

  Die Methode liefert den aktuell in der impliziten Instanz gespeicherten Wert für die Property mit dem angegebenen Schlüssel.

  Der Rückgabewert kann vom Typ *Void* sein, wenn der Property aktuell kein regulärer Wert gemäß ihres Typs bzw. ihres Wertebereiches zugewiesen ist[25].

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, ist der Rückgabewert vom Typ *Void*.

  Besitzt der Typ der impliziten Instanz eine zur Property passende get–Methode

  get<Key>() → *Any*

  so wird diese von der Standardimplementierung zum Abrufen des aktuellen Wertes benutzt.

[24] d.h. der Rückgabewert entspricht dem beim letzten Aufruf von *setPropChoiceList()* im Parameter *pChoiceList* übergebenen Wert

[25] Die Standardimplementierung der obsoleten Methode *getPropValue()* liefert in dem Fall den Spezialwert `@VOID`, was bei nicht-symbolischen Properties einen Typ-Konflikt darstellt. Aus Gründen der Abwärtskompatibilität kann diese Implementierung aber auch nicht geändert werden, weswegen die neue Methode *getPropValue2()* eingeführt wird.

Besitzt der Typ der impliziten Instanz keine solche Methode, wird der Wert aus der Hash-Tabelle der dynamischen Properties ermittelt (siehe Methode *getDynamicProps()* in der Schnittstelle *Base*).

Siehe auch Methode *forceDynamicProp()* (Abschn. 2.6).

- *setPropValue(pKey(Symbol), pValue(Any)) → Int*

  Die Methode weist der impliziten Instanz einen neuen Wert für die Property mit dem angegebenen Schlüssel zu.

  Die Standardimplementierung führt dabei folgende Aktionen durch:

  1. Überprüfung der Einhaltung der durch `width` bestimmten maximalen Eingabelänge bei Properties des Typs `"S"` sowie der Einhaltung der Wertebereiche und der Schrittweite (Raster) bei numerischen Properties (`"L"`, `"A"`, `"N"`)[26]. Wird eine Verletzung der Beschränkungen festgestellt, wird die Methode ohne weitere Aktionen beendet.
  2. Wenn der zuzuweisende Wert gleich dem aktuellen Wert der Property ist, wird die Methode ohne weitere Aktionen beendet.
  3. Bei Properties mit einer Choiceliste des Typs `@FixedSingleV` bzw. `@FixedMultiV`, die explizit als Liste von Wertspezifikationen zugewiesen wurde (s. 2.3), wird geprüft, ob der zuzuweisende Wert in der Choiceliste enthalten ist.
     Wen dem nicht der Fall ist, wird die Methode ohne weitere Aktionen beendet.
  4. Eigentliche Wertzuweisung.
     Besitzt der Typ der impliziten Instanz eine zur Property passende set–Methode
     set<Key>(pValue(Any)) → Void
     so wird diese zur Zuweisung des Wertes verwendet. In der Regel weist diese Methode lediglich den Wert einer entsprechenden Instanzvariablen zu. Jede weitere Semantik, wie beispielsweise die Neugenerierung der Geometrie oder Kollisionstests, ist der Methode *propsChanged()* vorbehalten (s.u.).
     Besitzt der Typ der impliziten Instanz keine passende set–Methode, wird der Wert unter dem Schlüssel der Property in die Hash-Tabelle der dynamischen Properties geschrieben.
     Siehe auch Methode *forceDynamicProp()* (Abschn. 2.6).
  5. Ist die Property mit einem Merkmal in den Produktdaten (OCD) assoziiert, werden anschließend vom globalen Produktdatenmanager Beziehungen zwischen den Merkmalen ausgewertet (infolge dessen sich andere Properties bzw. deren Werte ändern können).
  6. Danach wird die Methode *propsChanged()* (s.u.) zur Durchführung von Spezialbehandlungen aufgerufen. Für den Parameter *pDoChecks* wird dabei `1` (True) übergeben.
  7. Wurde die Wertzuweisung vom Produktdatenmanager oder von der Methode *propsChanged()* zurückgewiesen, werden alle Properties auf den Zustand zu Beginn der Methode zurückgesetzt und anschließend noch einmal die Methode *propsChanged()* gerufen, wobei für den Parameter *pDoChecks* diesmal `0` (False) übergeben wird.

  Rückgabewert der Methode ist `1`, wenn sich die Definition einer oder mehrerer Properties geändert hat bzw. wenn Properties hinzugekommen oder weggefallen sind, anderenfalls `0`.

  Einige Basistypen realisieren folgendes zusätzliches Verhalten:

  Interface Article (OiPart, OiPlElement)
  : Wenn der Aktualisierungsstatus der Artikelinstanz *@Undefined* ist, wird vor der eigentlichen Wertzuweisung erst versucht, eine Aktualisierung durchzuführen. Nur wenn in derem Ergebnis der Aktualisierungsstatus *@Up2Date* ist, finden die Standard-Aktionen statt (s.o.).

  OiPart, OiPlElement, OiProgInfo, OiPropertyObj
  : Wenn nach den Standard-Aktionen (s.o.) Properties vorliegen, deren Werte sich geändert haben, wird durch den *OiChangeManager* ein Event vom Typ *@PropertyChange* getriggert, wobei die jeweilige Instanz als Publisher und die Liste der geänderten Properties als Argument übergeben wird.

[26] Die Überprüfung muß hier (auf Kosten einer eventuellen Redundanz) stattfinden, da nicht sichergestellt werden kann, daß die Überprüfung bereits vom aufrufenden Clienten vorgenommen wurde.

OiPlElement

Wenn nach den Standard-Aktionen (s.o.) Properties vorliegen, deren Werte sich geändert haben, wird eine evtl. vorhandene Bemaßung aktualisiert (Anpassung an möglicherweise geänderte Abmessungen)[27].

- *checkPropValue(pKey(Symbol), pValue(Any), ...) → Int*

  Die Methode prüft, ob der Property der impliziten Instanz mit dem angegebenen Schlüssel der übergebene Wert zugewiesen werden kann.

  Der Rückgabewert ist `1`, wenn die implizite Instanz die angegebene Property besitzt und wenn alle Prüfungen erfolgreich absolviert wurden, andernfalls `0`.

  Wenn ein zusätzlicher optionaler Parameter übergeben wird, gibt dieser an, ob der übergebene Wert, sofern er sich als gültig erwiesen hat, mit dem aktuellen Wert der Property verglichen werden soll (`1`) oder nicht (`0`, Standard).
  Wenn der Vergleich aktiviert ist, gibt die Methode `0` zurück, wenn der übergebene Wert gleich dem aktuellen Wert der Property ist.

  Die Standardimplementierung führt folgende Prüfungen durch:

  – Passt der Typ des Wertes zum Typ der Property?
  – Bei Properties des Typs `"S"`:
    Wird die maximale Eingabelänge (Attribut `width`) eingehalten?
  – Bei Properties mit einer Auswahlliste vom Typ `@FixedSingleV`:
    Ist der übergebene Wert in der Auswahlliste enthalten?
  – Bei (numerischen) Properties mit Wertebereichen:
    Ist der übergebene Wert in den definierten Wertebereichen enthalten und passt er zu einer ggf. angegebenen Schrittweite?

- *propsChanged(pPKeys(Symbol[]), pDoChecks(Int)) → Int*

  Die Methode führt Spezialbehandlungen und Prüfungen nach der Änderung von Property–Werten durch. Die Properties, deren Werte sich geändert haben, werden durch ihre Schlüssel spezifiziert. Der Parameter *pDoChecks* gibt an, ob Prüfungen durchgeführt werden müssen oder lediglich auf die Änderung der Property–Werte reagiert werden muß, z.B. durch Geometrieanpassungen.
  Der Rückgabewert ist `1`, wenn die neuen Property–Werte gültig sind, andernfalls `0`.

  Hinweis:
  Die Methode wird aus der Standardimplementierung der Methode *setPropValue()* heraus gerufen (siehe oben). In *propsChanged()* werden in der Regel Anpassungen der Geometrie oder der Materialeigenschaften der impliziten Instanz vorgenommen.

- *changedPropList() → Symbol[]*

  Die Methode liefert die Referenz auf die Liste der Properties, deren Werte sich während der Abarbeitung der Methode *setPropValue()* geändert haben. Die Properties werden in der Liste anhand ihrer Schlüssel vermerkt.

  Hinweis:
  Die Methode wird in der Regel nur von Produktdatenmanagern während der Auswertung von Wissen über Produktdatenbeziehungen innerhalb der Methode *setPropValue()* benutzt.

  Die Liste wird zu Beginn jeder Ausführung von *setPropValue()* geleert.

[27] Diese Anspassung könnte theoretisch auch in *propsChanged()* erfolgen, es ist aber nicht sicher, daß die Methode in abgeleiteten Klassen korrekt überschrieben wird (so daß am Ende die geerbte Implementierung gerufen wird).

2.5 Property–Klassen und –Gruppen

Eine *Property–Klasse* ist eine Menge von Properties, die aus logischen, konzeptionellen oder sonstigen Aspekten zusammengefaßt werden.
Das Konzept der Klasse ist technischer Natur. (So besitzt eine Klasse z.B. auch keine sprachspezifische Bezeichnung.)

Property–Gruppen sind Mengen von Properties, die im Hinblick auf den Benutzer zusammengefaßt werden und durch den Eigenschaftseditor einer OFML–Applikation zur Gruppierung verwendet werden (können).

Im Gegensatz zum statischen Charakter der Klassen kann sich die Einteilung der Properties in Gruppen dynamisch ändern, z.B. in Abhängigkeit von der aktuellen Konfiguration oder von der aktuell eingestellten Sprache.

Eine Property kann genau einer Klasse und einer Gruppe zugeordnet werden. (Eine Property kann aber auch keiner Klasse bzw. keiner Gruppe zugeordnet sein.)
Die Zuweisung zu einer Klasse erfolgt mittels der Methode *setPropClass()*, zu einer Gruppe mittels der Methode *getPropGroupDescriptions()* (beide s.u.).

- *setPropClass(pPKey(Symbol), pPClass(String))* → *Void*

  Die Methode ordnet die Property der impliziten Instanz mit dem angegebenen Schlüssel der spezifizierten Klasse zu.

- *delPropClass(pPKey(Symbol))* → *Void*

  Die Methode entfernt die Property der impliziten Instanz mit dem angegebenen Schlüssel aus der spezifizierten Klasse. (Die Methode hat keinen Effekt, wenn die angegebene Property aktuell nicht der spezifizierten Klasse zugeordnet ist.)

- *getPropClass(pPKey(Symbol))* → *String* | *Void*

  Die Methode liefert die Property–Klasse, der die Property der impliziten Instanz mit dem angegebenen Schlüssel aktuell zugeordnet ist, oder einen Wert vom Typ *Void*, wenn die Property aktuell keiner Klasse zugeordnet ist.

- *getPropClasses()* → *String[]*

  Die Methode liefert eine Liste aller Property–Klassen, denen die Properties der impliziten Instanz aktuell zugeordnet sind.

- *getPropGroupDescriptions(pLanguage(String))* → *Any[]*

  Die Methode liefert eine Liste (OFML–Typ *List*) mit Beschreibungen für die aktuell definierten Property–Gruppen der impliziten Instanz.

  Jedes Element der Liste beschreibt eine Property–Gruppe und ist ein *Vector* aus folgenden Elementen:

  1. Name der Gruppe (*String*)
  2. sprachspezifische (Kurz-)Bezeichnung der Gruppe (*String*)
  3. Liste der Schlüssel der (aktuell definierten) Properties, die zur Gruppe gehören (*Symbol[]*)

  Die Reihenfolge der Gruppen in der Liste legt die Reihenfolge fest, in der die Gruppen im Eigenschaftseditor einer OFML–Applikation angezeigt werden sollen.
  Die Reihenfolge der Properties im Element 3 einer Gruppe wiederum definiert die Reihenfolge, in der die Properties innerhalb der Gruppe angezeigt werden sollen.

  Properties, die nicht explizit einer Gruppe zugeordnet sind, werden der Dummy–Gruppe `OI_NONE_PROPCLASS`[28] zugeteilt. Die sprachspezifische Bezeichnung (2. Element) für diese Gruppe ist gleich dem Namen der Gruppe. Von den Clienten (Applikationen) sind entsprechende eigene Textressourcen dafür zu verwenden.

[28] Der Name dieser Gruppe sollte besser `OI_NONE_PROPGROUP` lauten. Aus Gründen der Abwärtskompatibilität bzw. einer einfacheren Umstellung in bestehenden Anwendungen wird/wurde auf eine Umbennung aber verzichtet.

Die *Standardimplementierung* der Methode wendet folgendes Verfahren an:

a) Die Rückgabe spezifischer Gruppen wird unterbunden, wenn die Methode *needPropGroupDescriptions()* des für die implizite Instanz zuständigen *ProgInfo*–Objekts für diese Instanz den Wert `0` liefert. Alle (aktuell definierten) Properties der impliziten Instanz werden in diesem Fall der Gruppe `OI_NONE_PROPCLASS` zugeordnet.
Die Positionen der Properties innerhalb dieser Gruppe ergeben sich aus den ihnen zugewiesenen Positionen in der Property-Liste, siehe Methode *setPropPos()* (Abschn. 2.1).

b) Zur Ermittlung der Gruppen erfolgt zunächst eine Anfrage an den globalen Produktdatenmanager.
Das Ergebnis kann auch Properties enthalten, die keiner expliziten Gruppe zugeordnet sind. Diese werden der Gruppe `OI_NONE_PROPCLASS` zugeteilt.

c) Für die Properties, die nicht durch den Produktdatenmanager behandelt wurden, werden die Gruppen wie folgt bestimmt:

   – Properties, die keiner Klasse zugeordnet sind, werden der Dummy–Gruppe `OI_NONE_PROPCLASS` zugeteilt.
   Anderenfalls wird zu der Klasse der Property eine gleichnamige Gruppe definiert[29] und die Property dieser Gruppe zugeteilt.

   – Die sprachspezifische Bezeichnung (2. Element) einer aus einer Klasse abgeleiteten Gruppe wird, wenn möglich, in der Sprache geliefert, die in dem Parameter der Methode übergeben wurde. Sie wird wie folgt ermittelt:

     1. Aufruf einer Methode auf der Produktdatenbank der Serie der impliziten Instanz zur Abfrage einer sprachspezifischen Bezeichnung für die Klasse.
     Falls diese Methode einen Wert vom Typ *Void* liefert[30]:
     2. Suche einer Textresource im Namensraum der impliziten Instanz, wobei als Resource–Schlüssel der Bezeichner der Klasse verwendet wird.
     Falls keine Textresource gefunden wurde:
     3. Verwendung des Bezeichners der Klasse selber.

     Liefert die Methode der Produktdatenbank einen leeren String (Schritt 1), werden alle Properties der Klasse der Gruppe `OI_NONE_PROPCLASS` zugeordnet (und die Klasse/Gruppe aus der Rückgabeliste entfernt).

d) Gruppen, die vom Produktdatenmanager definiert wurden, folgen den Gruppen, die aus Klassen abgeleitet wurden (s.o.) sowie der Gruppe `OI_NONE_PROPCLASS`! Dabei bleibt die vom Produktdatenmanager definierte Reihenfolge erhalten.

   Die Position in der Rückgabeliste einer nicht vom Produktdatenmanager definierten Gruppe (inkl. `OI_NONE_PROPCLASS`) ergibt sich aus der kleinsten Nummer der Positionen der jeweils enthaltenen Properties, d.h., die Gruppe, welche die Property mit der kleinsten Positionsnummer enthält, steht an erster Stelle.

e) Die Position einer Property in einer vom Produktdatenmanager explizit definierten Gruppe wird durch den Produktdatenmanager vorgegeben.

   Die Positionen der Properties in einer nicht vom Produktdatenmanager definierten Gruppe (inkl. `OI_NONE_PROPCLASS`) ergeben sich aus den ihnen zugewiesenen Positionen in der Property-Liste, siehe Methode *setPropPos()* (Abschn. 2.1).

f) Wenn das für die implizite Instanz zuständige *ProgInfo*–Objekt die Methode *getPropGroupDescriptions4Obj()* implementiert, wird diese abschließend gerufen, wobei die Referenz auf die implizite Instanz, die angeforderte Sprache und die bisher durch die Standardimplementierung ermittelte Rückgabeliste übergeben wird. Dies kann dazu genutzt werden, serien-spezifisches Verhalten zu realisieren.

[29] insofern diese Gruppe noch nicht definiert ist
[30] für Klassen, die nicht in den Produktdaten definiert sind

2.6 Sonstige

- *getPropInfo(pKey(Symbol), pPropValue(Any), pInfoType(Symbol)) → Any*

  Die Methode liefert die Information des angeforderten Typs zu dem spezifizierten Property–Wert.

  Der Rückgabewert ist vom Typ *Void*, wenn die Instanz nicht die spezifizierte Property besitzt oder wenn keine Information des angeforderten Typs verfügbar ist.

  Folgende Standard-Infotypen sind vordefiniert:

  @Picture
  : Name der Bilddatei, die den Property-Wert veranschaulicht (*String*)

  @Text
  : textuelle Beschreibung (*String*, kann eine Text-Resource sein)

  @HTML
  : URL der HTML-Beschreibung (*String*)

  Die Infotypen `@Text` und `@HTML` werden aktuell noch nicht verwendet.

  Format, Bildgröße und Ablageort der Bilddatei (Infotyp `@Picture`) werden durch die jeweilige OFML–Applikation festgelegt[31].

  Die Standardimplementierung delegiert den Aufruf an die Methode *getPropInfo4Obj()* der für die implizite Instanz zuständigen *OiProgInfo*–Instanz (insofern vorhanden).
  Die Implementierung von *OiProgInfo::getPropInfo4Obj()* verwendet dabei für den Infotyp `@Picture` entsprechende Einträge aus der Steuerdatentabelle `proginfo.csv`.

- *updateProperties() → Int*

  Die Methode aktualisiert die Properties der impliziten Instanz.

  Die Methode wird vom Property Editor der Applikation aufgerufen, *bevor* die Properties der impliziten Instanz angezeigt werden. Sie bietet somit die Gelegenheit, sowohl auf Änderungen in den Daten zu reagieren, als auch auf Änderungen in der Laufzeitumgebung (z.B. geänderte Spracheinstellung).

  Der Rückgabewert hat keine spezifische Bedeutung[32].

  Die Standardimplementierung führt keine Aktionen aus und liefert `1`.

  In der Basisklasse *OiPlElement* ist die Methode wie folgt implementiert:
  Wenn sich die laut der Methode *OiPlanning::getPDLanguage()* (s. 3.4) für Produktdatentexte zu verwendende Sprache seit der Erzeugung der impliziten Instanz oder seit dem letzten Aufruf von *updateProperties()* geändert hat, und wenn der Aktualisierungsstatus der impliziten Instanz `@Up2Date` ist, wird der globale Produktdatenmanager aufgerufen, die kaufmännischen Merkmale zu aktualisieren[33].
  Der Rückgabewert ist `1`.

- *getPropTitle() → String*

  Die Methode liefert eine Kurzbeschreibung der Instanz zur Verwendung in der Kopfzeile von Property–Editoren[34].

  Die Standard–Implementierung liefert einen leeren String.

  Die Implementierung des Basistyps *OiPlElement* (Schnittstelle *Article*) verwendet entsprechende Einträge der Steuerdatentabelle `plelement.csv`, die die Generierung der Titelzeile steuern. Per default setzt sich dabei der generierte String aus der Grundartikelnummer und dem Artikelkurztext zusammen.

[31] Für die Applikationen von EasternGraphics ist das einheitlich durch die DSR–Spezifikation geregelt [dsr].
[32] Der Typ *Int* stammt noch von der Verwendung der Methode im Rahmen des alten Update-Konzepts.
[33] Dies erfolgt zwar alleine zu dem Zweck, die sprachspezifischen Bezeichnungen für Merkmale und Werte an die neue Sprache anzupassen, beinhaltet aber auch einen Zyklus zur Auswertung von Beziehungswissen, welcher an dieser Stelle eigentlich unnötig ist.
[34] Ob die Methode tatsächlich verwendet wird, liegt in der Entscheidung der Applikationen. Diese können auch ein anderes, abweichendes Schema bei der Gestaltung der Kopfzeile anwenden.

- *setPropHintText(pKey(Symbol), pHint(String |Void)) → Void* **new**

  Die Methode weist der impliziten Instanz für die Property mit dem angegebenen Schlüssel einen Hinweistext zu. Dieser kann von einer Applikation als Hint angezeigt werden, wenn der Nutzer den Mauszeiger über dem Bezeichner der Property im Eigenschaftseditor bewegt.

  Der Text kann mehrzeilig sein (d.h., Newline-Escape-Sequenzen enthalten). Er muß in der Sprache übergeben werden, die aktuell laut der Funktion *getPDLanguage()* der Schnittstelle *Article* [article] für Produktdatentexte zu verwenden ist (s.a. 3.4).

  Wird als Hint ein Wert vom Typ *Void* übergeben, so wird ein eventuell vorher zugewiesener Hinweistext entfernt.

- *getPropHintText(pKey(Symbol)) → String | Void* **new**

  Die Methode liefert den Hinweistext, der aktuell via *setPropHintText()* an die implizite Instanz für die Property mit dem angegebenen Schlüssel zugewiesen ist.

  Der Rückgabewert ist vom Typ *Void*, wenn für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert ist, oder wenn der Property aktuell kein Hinweistext zugewiesen ist.

- *forceDynamicProp(pPKey) → Int* [35]

  Die Methode gibt `1` (true) zurück, wenn der Wert der Property mit dem angegeben Schlüssel in der Hash–Tabelle der dynamischen Properties[36] gespeichert werden soll. In diesem Fall werden möglicherweise vorhandene set– bzw. get–Methoden ignoriert.

  Auf diese Weise kann die fälschliche Verwendung von set– bzw. get–Methoden verhindert werden, die einem anderen Zweck dienen, als zur Speicherung eines Property–Wertes in einer Membervariablen.

  > Dies ist von besonderer Bedeutung für Klassen, die zur Repräsentation von Artikeln (Schnittstelle *Article*) dienen, da der globale Produktdatenmanager dynamisch Properties definieren kann, die mit kaufmännischen Merkmalen assoziert sind. Da Programmierung der Klasse und Erfassung der kaufmännischen Daten in der Regel getrennt erfolgen, kann es passieren, dass ein kaufmännisches Merkmal mit einem Namen angelegt wird, für den es bereits passende set– bzw. get–Methoden gibt, die aber einem ganz anderen Zweck dienen. Es wird deswegen dringend empfohlen, diese Methode in allen Klassen zu überschreiben, die die Schnittstelle *Article* implementieren, und die neue set– bzw. get–Methoden definieren (die nicht zum Speichern eines Property–Wertes dienen).

  Wenn die Methode überschrieben wird, muß für alle Property-Schlüssel, die für die Klasse nicht relevant sind, die geerbte Implementierung gerufen werden!

- *resetProperties() → Void*

  Die Methode setzt den Zustand (Attribute, Werte) aller Properties der impliziten Instanz auf den initialen Zustand (default).

  Die Definition des Default–Zustandes ist typ-spezifisch.

  Die Standardimplementierung führt keine Aktionen aus.

  Die Implementierung der OFML–Grundtypen *OiPlElement* und *OiPart* (Schnittstelle *Article*) delegiert den Request zunächst an den globalen Produktdatenmanager (Herstellung des initialen Zustands des Artikels) und anschließend an die globale Planungsinstanz (Anwendung applikations- bzw. anwenderspezifischer Defaults, z.B. auf der Basis von Konfigurationsprofilen).
  Der Begriff *Default* bezieht sich dabei auf den Artikel, der aktuell durch die implizite Instanz repräsentiert wird, nicht auf den Artikel, der durch die Instanz bei ihrer Erzeugung repräsentiert wurde[37].

[35] Nach den Namenskonventionen für die neue Property-Schnittstelle müsste die Methode eigentlich in *forceDynamicProperty()* umbenannt werden. Dies würde jedoch Anpassungen in abgeleiteten, nicht-standardisierten Klassen erforderlich machen (was bei der Einführung der neuen Property-Schnittstelle vermieden werden sollte) oder aber in der OFML–Standardimplementierung müssten mittels *hasMember()* umständlich Alternativen eingebaut werden.

[36] siehe Methode *getDynamicProps()* in der Schnittstelle *Base*

[37] Bei *artikel-polymorphen* Klassen kann sich die Artikelnummer einer Instanz während ihrer Lebenszeit ändern.

2.7 Client–Support

Die Methoden dieser Gruppe fassen 2 oder mehr Methoden der vorherigen Gruppen zusammen und dienen somit zur effektiveren Abfrage von Property–Attributen seitens verschiedener Clienten, insbesondere der Property–Editoren.

- *getPropSpec(pKey(Symbol))* → *Any* **new**

  Die Methode liefert die erweiterte Spezifikation der Property der impliziten Instanz mit dem angegebenen Schlüssel.

  Neben der eigentlichen Property–Definition umfaßt die erweiterte Spezifikation u.a. noch den Aktivierungsstatus, die Position, Wertebereiche, den aktuellen Wert (nebst optionaler Wertbezeichnung) sowie den (optionalen) Bezeichner der Bilddatei.
  (Die Methoden, deren Funktionalität in *getPropSpec()* einfließt, werden unten bei den einzelnen Elementen der Rückgabestruktur erwähnt[38].)

  Ist für die implizite Instanz keine Property mit dem angegebenen Schlüssel definiert, ist der Rückgabewert vom Typ *Void*. Anderenfalls wird aktuell ein Vector mit folgendem Aufbau geliefert[39]:

  1. Property-Typ (*String*)
  2. max. Eingabelänge (*Int*)
  3. Anzahl der Dezimalstellen (*Int*)
  4. Art der Choiceliste (*Symbol*)
  5. sprachspezifische Bezeichnung (*String*)
  6. Position der Property (*Int*)
  7. Aktivierungsstatus (*Int*)
  8. Wertebereiche (*Any*)
  9. aktueller Wert (*Any*)
  10. sprachspezifische Bezeichnung für den aktuellen Wert (*Void* | *String*)
  11. Name der Bilddatei, die den aktuellen Wert veranschaulicht (*Void* | *String*)
  12. sprachspezifischer Hinweistext für die Property (*Void* | *String*)
  13. Property-Klasse (*Void* | *String*)

  Die Elemente 1 bis 4 entsprechen der eigentlichen Property–Definition, welche von der Methode `getPropDef2(pKey)` geliefert wird (s. Abschn. 1.3).

  Die Elemente 5 un 6 entsprechen dem Ergebnis der Methodenaufrufe `getPropName(pKey)` bzw. `getPropPos(pKey)` (s. 2.1).

  Das Element 7 entspricht dem Ergebnis des Methodenaufrufs `getPropState2(pKey)` (s. 2.2).

  Das Element 8 entspricht dem Ergebnis des Methodenaufrufs `getPropRanges(pKey)` (s. 2.3).

  Das Element 9 entspricht dem Ergebnis des Methodenaufrufs `getPropValue2(pKey)` (s. 2.4).

  Das Element 10 enthält, wenn es nicht vom Typ *Void* ist, die sprachspezifische Bezeichnung, die anstelle des aktuellen Wertes selber anzuzeigen ist.
  Ist der aktuelle Property–Wert (Element 9) vom Typ *Void*, so muß dieses Element *nicht* zwingend auch vom Typ *Void* sein, sondern es kann einen String enthalten, der den aktuellen Zustand beschreibt[40].

[38] Im Prinzip könnte auch noch die Methode *getPropChoiceList()* integriert werden. Da dies aber nur für Properties mit Choicelisten relevant wäre und die Rückgabestruktur enorm aufblähen kann, wurde (vorerst) darauf verzichtet.
[39] Bei Bedarf kann der Vector in Zukunft weitere Elemente enthalten.
[40] Dies wird z.B. bei aktuell nicht bewerteten numerischen einschränkbaren OCD–Merkmalen genutzt, s. Abschn. 3.2.

Für Properties mit einer Choiceliste vom Typ `@FixedSingleV` wird die Bezeichnung nach folgendem Verfahren ermittelt[41]:

1. Wenn in der Choiceliste[42] für den Wert eine sprachspezifische Bezeichnung (nicht-leerer String) angegeben ist, wird diese verwendet, anderenfalls:
2. Aufruf der (obsoleten) Methode *getChLPropValueText()* (Anhang B).
   Wenn dieser Aufruf einen Wert vom Typ *String* liefert und dieser keine leere Zeichenkette ist, wird diese verwendet, anderenfalls:
3. Ermittlung der Textresource für das Symbol bzw. den String (insofern der übergebene Wert von diesen beiden Typen ist).
   Wenn eine Textresource ermittelt werden konnte, wird diese verwendet, anderenfalls:
4. Verwendung der String-Repräsentation des Wertes.

   In den Schritten 1 und 2 wird die Sprache verwendet, die durch die Methode *OiPlanning::getPDLanguage()* geliefert wird (s. 3.4).

Das Element 11 entspricht dem Ergebnis des Methodenaufrufs `getPropInfo(pKey, <element 9>, @Picture)` (s. 2.6).

Das Element 12 entspricht im Wesentlichen dem Ergebnis des Methodenaufrufs `getPropHintText(pKey)` (s. 2.6). Ggf. kann noch ein zusätzlicher Text generiert generiert werden, der z.B. die aktuell festgelegten Wertebereiche einer numerischen Property beschreibt.

Das Element 13 entspricht dem Ergebnis des Methodenaufrufs `getPropClass(pKey)` (s. 2.5).

- *getPropSpecs()* → *Any[]* **new**

  Die Methode liefert eine Liste mit den erweiterten Spezifikationen aller aktuell für die implizite Instanz definierten Properties.

  Die Elemente der Liste beschreiben je eine Property und bestehen aus einem zweistelligen Vector:

  1. Property-Schlüssel (*Symbol*)
  2. Spezifikation der Property (*Any*), entspricht dem Ergebnis des Methodenaufrufs `getPropSpec(<element 1>)` (s.o.)

  Die Reihenfolge der Properties in der Liste ist bereits bezüglich ihrer Positionen sortiert (entspricht also der Reihenfolge der von *getPropKeys()* gelieferten Liste, s. 2.1).

- *getPropChoiceList2(pKey(Symbol))* → *Any* **new**

  Die Methode liefert ausführliche Informationen über die aktuell für die Property der impliziten Instanz mit dem angegebenen Schlüssel festgelegte Choiceliste.

  Die Methode kombiniert die Funktionalität von *getPropChoiceList()* (s. 2.3) und *getPropInfo()* für den Info-Typ *@Picture* (s. 2.6). Der Client erspart sich damit separate Aufrufe von *getPropInfo()* für jeden einzelnen Wert der Choiceliste.

  Der Rückgabewert ist vom Typ *Void*, wenn die implizite Instanz keine Property mit dem angegebenen Schlüssel besitzt, oder wenn keine Choiceliste mittels der Methode *setPropChoiceList()* zugewiesen wurde.

[41] In Applikationen, die noch die alte Schnittstelle verwenden, wurde dies im Property–Editor selber bewerkstelligt. In Applikationen, die die neue Schnittstelle verwenden, ist das nicht mehr notwendig, da dies durch die Methode *getPropSpec()* gekapselt wird.

[42] s. Methode *getPropChoiceList()* in 2.3

Anderenfalls ist der Rückgabewert eine Liste von Vectoren[43]. Jeder Vector enthält die Information zu einem Wert aus den Choiceliste und besteht aktuell aus folgenden Elementen[44]:

1. Wert (*Any*)
2. sprachspezifische Bezeichnung (*Void* | *String*)
3. Status (*Int*: 0 = ungültig, 1 = gültig)
4. Aufpreis-Betrag (*Void* | *Float*)
5. Aufpreis-Währung (*Void* | *String*)
6. Name der Bilddatei, die den Wert veranschaulicht (*Void* | *String*)

Die Bedeutung der Elemente 1 bis 5 ist in der Spezifikation der Methode *setPropChoiceList()* näher beschrieben (s. 2.3).

Wurde bei der Spezifikation eines Wertes via *setPropChoiceList()* keine sprachspezifische Bezeichnung angegeben (Element ist vom Typ *Void*), und ist der Typ der Property `"Y"` oder `"YS"`, wird der Wert als eine Text–Ressource–ID interpretiert, die durch *getPropChoiceList2()* nun sprachspezifisch über externe Ressourcen-Dateien aufgelöst wird (Element 2). Dabei wird die Sprache verwendet, die durch die Methode *OiPlanning::getPDLanguage()* geliefert wird (s. 3.4). Ist die Text–Ressource–Auflösung nicht erfolgreich, so wird der Wert selber verwendet (wozu bei Properties des Typs `"Y"` der Wert in einen String umgewandelt wird).

Das Element 6 entspricht dem Ergebnis des Methodenaufrufs
`getPropInfo(pKey, <element 1>, @Picture)` (s. 2.6).

- *getAllPropSpecs(pRequiredState(Int))* → *Any[]* **new**

  Die Methode liefert eine Liste mit den erweiterten Spezifikationen aller für die implizite Instanz möglichen Properties (also auch derjenigen, welche die implizite Instanz aktuell nicht besitzt).

  Aktuell berücksichtigt die Methode nur Properties, die vom globalen Produktdatenmanager für kaufmännische Merkmale (OCD) des durch die implizite Instanz repräsentierten Artikels generiert werden.

  Der Rückgabewert entspricht dem von *getPropSpecs()* mit folgenden Änderungen/Erweiterungen:

  – Properties, die bekanntermaßen nie den Status (gemäß *getPropState2()*, Abschn. 2.2) haben, der im Parameter *pRequiredState* gefordert ist, werden nicht in die Rückgabeliste aufgenommen.

  – Das 5. Element einer einzelnen Property-Spezifikation (sprachspezifische Bezeichnung) enthält ein Sprache–Text–Mapping (s. Abschn. 1.4).

  – Bei Properties, die aktuell für die implizite Instanz nicht definiert sind, ist im Aktivierungsstatus das Flag (Bit) mit dem Wert `8` gesetzt (s. *setPropState2()* in Abschn. 2.2).
  Die Elemente 9 bis 11 haben dann einen Wert vom Typ *Void*.

  – Das 10. Element einer einzelnen Property-Spezifikation (sprachspezifische Bezeichnung des aktuellen Wertes) ist entweder ein Wert vom Typ *Void* oder ein Sprache–Text–Mapping.

  – Das 12. Element einer einzelnen Property-Spezifikation (sprachspezifischer Hinweistext) ist ein Sprache–Text–Mapping bzw. ein leerer Vektor, wenn keine Hinweistexte hinterlegt sind.

- *getDefaultPropGroupSpecs()* → *Any[]* **new**

  Die Methode liefert eine Liste mit den Beschreibungen der Standard–Gruppen, d.h. der Gruppen, die unabhängig von der Konfiguration der impliziten Instanz und unabhängig von einer bestimmten Sprache definiert sind[45].

[43] Die Liste kann u.U. auch leer sein (s. 3.3)!
[44] Bei Bedarf kann der Vector in Zukunft weitere Elemente enthalten.
[45] Semantik und Verwendungszweck dieser Methode unterscheiden sich damit wesentlich von der Methode *getPropGroupDescriptions()*, s. 2.5.

Jedes Element der Rückgabe–Liste ist ein Vector aus folgenden Elementen:

1. Name/Bezeichner der Gruppe (*String*)
2. Liste der Schlüssel der Properties, die zur Gruppe gehören (*Symbol[]*)
3. Sprache-Text-Mapping (*Any[]*) mit den sprachspezifischen (Kurz-)Bezeichnungen der Gruppe

Aktuell berücksichtigt die Methode nur Properties, die vom globalen Produktdatenmanager für kaufmännische Merkmale (OCD) des durch die implizite Instanz repräsentierten Artikels generiert werden.

Alle Properties, die durch die Methode *getAllPropSpecs()* (s.o.) geliefert werden, sollten auch in den Gruppen enthalten sein, die von *getDefaultPropGroupSpecs()* geliefert werden.

- *getAllPropValueInfos(pKey(Symbol), pPClass(String))* → *Any* **new**

  Die Methode liefert Informationen über alle möglichen Werte der Choiceliste für die Property der impliziten Instanz mit dem angegebenen Schlüssel und der spezifizierten Klasse.
  (Die Methode liefert also auch Informationen über die Werte, die in der aktuellen Choiceliste der impliziten Instanz nicht enthalten sind).

  Aktuell berücksichtigt die Methode nur Properties, die vom globalen Produktdatenmanager für kaufmännische Merkmale (OCD) des durch die implizite Instanz repräsentierten Artikels generiert werden.

  Der Rückgabewert entspricht dem von *getPropChoistList2()* mit folgenden Änderungen bzw. Erweiterungen:

  – Wird die Methode für eine Property gerufen, die nicht vom globalen Produktdatenmanager generiert wird, ist der Rückgabewert vom Typ *Void*.

  – Die sprachspezifische Bezeichnung des Wertes (2. Element einer einzelnen Wertspezifikation) ist ein Sprache–Text–Mapping (s. Abschn. 1.4).

  – Bei Werten, die in der aktuellen Choiceliste der impliziten Instanz nicht enthalten sind, ist im Status (3. Element) das Flag (Bit) mit dem Wert 2 gesetzt (s. *setPropChoiceList()*, Abschn. 2.3).

3 Sonstige Aspekte

3.1 Optionale Merkmale

Mit Properties der Typen `"Y"` und `"YS"` kann das Konzept einer *Option* umgesetzt werden, d.h., eines Merkmals, das vom Anwender bewertet werden kann, aber nicht muß. Dazu wird in der Choiceliste ein Sonder-Wert, typischerweise `@VOID` bzw. `"VOID"`, mit einer entsprechenden Textresource (deutsch z.B. „nicht gewählt“) spezifiziert.

Für die Clienten (z.B. Property–Editoren) ist dies jedoch transparent, d.h. für sie stellt sich dieser Sonder-Wert als ein ganz „normaler“ Wert aus der Choiceliste dar.

3.2 Einschränkbare OCD–Merkmale

Einschränkbare OCD–Merkmale werden entweder auf Properties der Typen `"Y"` bzw. `"YS"` abgebildet, oder (im Fall von numerischen Merkmalen) auf Properties des Typs `"N"` mit einer Choiceliste vom Typ `@FixedSingleV`.

Diese Merkmale können den Zustand „(noch) nicht bewertet“ besitzen. Bei symbolischen einschränkbaren Merkmalen wird dieser Zustand durch den reservierten Sonder-Wert `@UNSPECIFIED` bzw. `"UNSPECIFIED"` mit einer entsprechenden Textresource repräsentiert (Elememt 9 des Rückgabe-Vectors der Methode *getPropSpec()*, Abschn. 2.7). Bei numerischen einschränkbaren Merkmalen wird dieser Zustand durch einen Wert vom Typ *Void* repräsentiert, im Element 10 des Rückgabe-Vectors der Methode *getPropSpec()* wird jedoch eine sprachspezifische Bezeichnung übergeben, die der Textresource für `@UNSPECIFIED` bei symbolischen Merkmalen entspricht.

Je nach Einstellung in den Produktdaten kann die Choiceliste der einschränkbaren Merkmale über die eigentlichen Werte hinaus auch den Sonder-Wert enthalten, der es dem Anwender ermöglicht, explizit den Zustand „nicht bewertet“ einzustellen. Für die Elemente 1 und 2 des Rückgabe-Vectors der Methoden *getPropChoiceList()* und *getPropChoiceList2()* (s.a. Abschn. 2.7) gelten dabei dieselben Aussagen wie oben zu den Elementen 9 und 10 des Rückgabe-Vectors der Methode *getPropSpec()* (aktueller Property-Wert).

Ähnlich wie bei den optionalen Merkmalen (s. vorheriger Abschn.) ist auch die Thematik der einschränkbaren Merkmale für die Clienten (Property–Editoren) transparent.

3.3 Leere Choicelisten

Aus konzeptionller Sicht darf die Choiceliste einer Property nicht leer sein. Aufgrund von Problemen oder Fehlern im Produktbeziehungswissen kann dies u.U. jedoch vorkommen. In so einem Fall und insofern der Property auch nicht explizit ein Wert zugewiesen wurde[46], wird von der Standardimplementierung der Methode *getPropSpec()* (Abschn. 2.7) im Element 10 ein sprachspezifischer Text geliefert, der diesen Zustand beschreibt (deutsch z.B. „kein Wert vorhanden“).

3.4 Application Callbacks

- *::ofml::app::isPropTypeSupported(pPType(String))* → *String* **extended**

  Die Funktion liefert `"1"` (True), wenn die Applikation den im Parameter übergebenen Property–Typ unterstützt, anderenfalls `"0"` (False).

  Eine Applikation, deren Property–Editor auf die neue Schnittstelle umgestellt wurde, liefert für alle Property–Typen der alten Schnittstelle `"0"`.

[46] *getPropValue2()* liefert einen Wert vom Typ *Void*

- *::ofml::app::getPDLanguage(pPID(Symbol)) → String* **new**

  Die Funktion liefert die Sprache, welche für kaufmännische Texte und Property-bezogene Texte bei Artikelinstanzen verwendet werden soll, die dem durch den Parameter spezifizierten OFML–Programm (Serie) angehören.

  Die Sprache wird mittels Abgleich der in der Applikation aktuell eingestellten Sprachreferenzen des Anwenders und den Sprachen ermittelt, welche durch das aktuell für das OFML–Programm installierte Paket unterstützt werden.
  Entspricht keine der durch das OFML–Paket unterstützten Sprachen den aktuell eingestellten Sprachreferenzen des Anwenders, so wird die erste Sprache aus der Liste der durch das Paket unterstützten Sprachen verwendet.

  Der Aufruf dieser Application Callback wird durch folgende OFML–Methoden gekapselt:

  - Standardimplementierung der Funktion

    getPDLanguage() → String

    der Schnittstelle *Article* [article].

  - *OiPlanning::getPDLanguage(pArg(Any)) → String | Void*
    Die Methode liefert die Sprache, welche für kaufmännische Texte und Property-bezogene Texte bei Artikelinstanzen verwendet werden soll, die dem spezifizierten OFML–Programm angehören oder bei der übergebenen Artikelinstanz.

    Der Parameter *pArg* kann die ID eines OFML–Programms (*Symbol*), ein ProgInfo–Objekt (*OiProgInfo*) oder eine Artikelinstanz sein. (Im Fall eines ungültigen Parameters ist der Rückgabewert vom Typ *Void*.)

    Diese Implementierung nutzt die Application Callback *::ofml::app::getPDLanguage()*, wobei das spezifizierte OFML–Programm, die ID des übergebenen ProgInfo–Objekts[47] bzw. das OFML–Programm der übergebenen Artikelinstanz[48] übergeben wird.

[47] gemäß Methode *OiProgInfo::getID()*
[48] gemäß Methode *getProgram()* der Schnittstell *Article*

A Alphabetischer Index der Methoden

Vorbemerkung:
Die obsoleten Methoden (Abschn. B) sind hier nicht aufgeführt.
Neue Methoden sind mit `(N)` gekennzeichnet.

B Obsolete Methoden

Folgende Methoden (in alphabetischer Reihenfolge) sind in der neuen Spezifikation der Schnittstelle nicht mehr (offiziell) enthalten. Aus Gründen der Abwärtskompatibilität bleiben sie in der Implementierung aber erhalten.

- *getChLPropValueText(pPKey(Symbol), pPValue(Any), pLanguage(String))* → *String | Void*

  Die Methode liefert die sprachspezifische Bezeichnung des angegebenen Wertes der Property der impliziten Instanz mit dem angegebenen Schlüssel in der spezifizierten Sprache.

  Der Rückgabewert ist vom Typ *Void*, wenn keine Bezeichnung geliefert werden kann.

  Die Methode wird während der Ermittlung der sprachspezifischen Bezeichnung des aktuellen Wertes einer Choicelist-basierten Property gerufen, siehe Methode *getPropSpec()*[49].

- *getProperties()* → *String*

  Ersetzt durch *getPropSpecs()*.

- *getPropertyDef(pKey(Symbol))* → *Any[]*

  Ersetzt durch *getPropDef2()*.

- *getPropertyKeys()* → *Symbol[]*

  Umbenannt in *getPropKeys()*.

- *getPropertyPos((pKey(Symbol))* → *Int | Void*

  Umbenannt in *getPropPos()*.

- *getPropClassDescriptions(pLanguage(String))* → *Any*

  Umbenannt in *getPropGroupDescriptions()*.

- *getPropState(pKey(Symbol))* → *Int*

  Ersetzt durch *getPropState2()*.

- *getPropValue(pKey(Symbol))* → *Any*

  Ersetzt durch *getPropValue2()*.

- *getVisiblePropValues()* → *Void*

  Ersetzt durch *getPropChoiceList()*[50].

- *setPropPosOnly(pKey(Symbol), pPos(Int))* → *Int | Void*

  Umbenannt in *setPropPos()*.

- *setPropState(pKey(Symbol), pState(Int))* → *Void*

  Ersetzt durch *setPropState2()*.

- *setupProperty(pKey(Symbol), pDef(Any[5]), pPos(Int))* → *Void*

  Ersetzt durch *setupProperty2()*.

[49] De facto hat diese Methode aktuell keine Bedeutung mehr. Sie wurde seinerzeit maßgeblich eingeführt, um Performanz-Probleme bei Artikeln in einem veralteten und nicht mehr unterstützten Produktdatenformat in den Griff zu bekommen. In den OFML Basisklassen hat die Methode eine leere Implementierung und somit keine Auswirkung.

[50] Die Methode *getVisiblePropValues()* wurde erst in OI 1.29 eingeführt und dort explizit als Übergangslösung bis zum Redesign der Schnittstelle deklariert.

C Änderungshistorie

Version 2.9 (2023-09-14):

- Präzisierung zur Methode *setupProperty2()* (Abschn. 2.1):
  Ist bereits eine Property mit dem angegebenen Schlüssel aber einer anderen Definition registriert, so hat der Methodenaufruf keinen Effekt.
- Die Methode *getPropChoiceList()* (Abschn. 2.3) akzeptiert nun einen zusätzlichen optionalen Parameter, der angibt, ob die Methode aufgerufen werden soll, wenn die Auswahlliste indirekt durch Angabe des Namens einer Methode spezifiziert ist.
  Default (abwärtskompatibles Verhalten) ist 1 (ja).
- Obsoleten Abschnitt zur Migration von alter auf die neue Schnittstelle entfernt.
- Anhang zu den Spezial-Typen von EasternGraphics entfernt (da praktisch nicht mehr relevant).

Version 2.8 (2023-05-04):

- Präzisierungen zur Standardimplementierung der Methode *setPropValue()* (Abschn. 2.4).

Version 2.7 (2023-03-17):

- Präzisierungen in der Spezifikation der Methode *getPropGroupDescriptions()* im Abschn. 2.5.

Version 2.6 (2022-09-14):

- Präzisierung zur Behandlung von ungültigen Wertebereich–Definitionen bei der Übergabe an die Methode *setPropRanges()* (Abschn. 2.3).
- Im Abschn. 2.4 fehlende Spezifikation der Methode *checkPropValue()* ergänzt.

Version 2.5 (2022-05-02)

- Verweis auf die Funktion *getPDLanguage()* der Schnittstelle *Article* im Abschn. 3.4.

Version 2.4 (2021-12-22):

- Die Methode *getPropInfo()* wird nun im Abschn. 2.6 beschrieben.
- Die Methode *getChLPropValueText()* ist nun als obsolet deklariert und wird entsprechend im Anhang B beschrieben.
- Das Verfahren zur Ermittlung der sprachabhängigen Bezeichnung des aktuellen Wertes einer Property mit einer Choiceliste vom Typ `@FixedSingleV` wird nun in Abschn. 2.7) bei der Methode *getPropSpec()* beschrieben (nicht mehr bei der Methode *getChLPropValueText()*).

  Das Verfahren wurde dabei korrigiert: ein in der Choicelisten–Spezifikation für den Wert angegebener Text hat Vorrang vor einem ggf. durch *getChLPropValueText()* oder durch eine Textressource bereitgestellten Text.

Version 2.3 (2021-10-12):

- Kleine Korrektur und Präzisierungen bezüglich der Rückgabestruktur der Methoden *getPropSpec()* und *getPropChoiceList2()*.
- Präzisierung zur Verwendung der Methode *getPropTitle()*.

Version 2.2:

- Änderungen in der Dokumenstruktur sowie einige Korrekturen und Präzisierungen.

Version 2.1:

- Die Methode *setPropPos()* ist nun auch im alphabetischen Index der Methoden enthalten (s. Anh. A).

Version 2.0:

- Einführung des Konzepts der *Sprach–Text–Mappings* (Abschn. 1.4)
- Einführung des Flags mit dem Wert `8` im Property–Status zur Kennzeichnung von Properties, welche die implizite Instanz in der aktuellen Konfiguration nicht besitzt (Abschn. 2.1).
- Überführung des Status-Elements eines Choicelisteneintrags in ein Bitfeld (Flags) und Einführung des Flags mit dem Wert `2` zur Kennzeichnung eines Eintrags, welchen die implizite Instanz in der aktuellen Konfiguration nicht besitzt (Abschn. 2.3).
- Einführung des Konzepts der *Property–Gruppen* in Abgrenzung von Property–Klassen (Abschn. 2.5).
- Neue Methoden *getAllPropSpecs()*, *getAllPropValueInfos()* und *getDefaultPropGroupSpecs()* im Abschn. 2.7.
- Sortierung der obsoleten Methoden in alphabetischer Reihenfolge (Abschn. B).