EasternGraphics
visualize your business

Dokumentation
MT - OFML-Metatypen – Tabellen und Spezifika

Autor	*Ekkehard Beier, Andreas Handschuh, EasternGraphics*	
Referenz	Titel	**Metatype Specification**
	Version	**1.17.3**
	Release Date	**2022-09-26**
	Implementation	**::ofml::go::1.17.31**
	Dokumentversion	**(0)**
Historie	siehe am Ende des Dokuments	
Status	Freigabe für EGR und Vertriebspartner, sowie an Kunden erlaubt. Änderungen im Sinne des technischen Fortschritts, zu Optimierungszwecken oder zur Beseitigung konzeptioneller Mängel sind ausdrücklich vorbehalten.	
Feedback	metatypes@EasternGraphics.com	
Anmerkung	Änderungen gegenüber **MT 1.17.2(-0)** durch Unterstreichen hervorgehoben.	

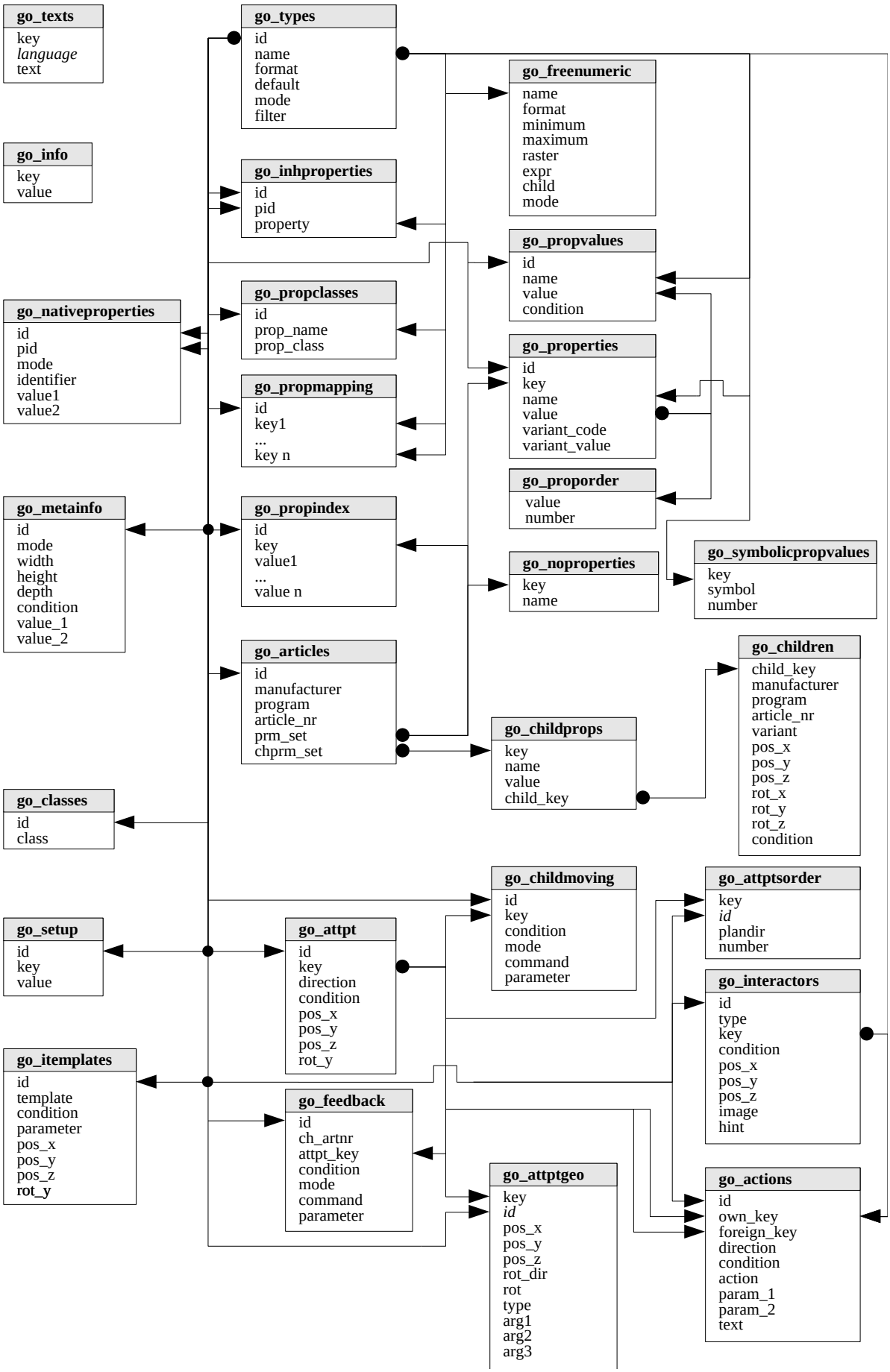
1 Allgemein

Das Konzept der OFML-Metatypen erweitert die traditionelle Grafikdatenanlage um folgende Möglichkeiten:

- grundartikelübergreifende Konfiguration, d.h., für ein gegebenes Objekt können nicht nur die spezifischen Eigenschaften geändert werden (wobei die Grundartikelnummer unverändert bleibt), sondern auch solche, die zu einer Veränderung der Grundartikelnummer führen, z.B. durch Wahl einer anderen Serie
- Verkettungs- und Anbaulogiken, z.B. artikelabhängige Anbauteile mittels tabellarisch erfassbaren Merkmale

2 Tabellen

In diesem Kapitel werden die Tabellen beschrieben, welche für die Metatypen benötigt werden. Die Typen der einzelnen Spalten sind dem Script *mt.inp_descr* bzw. der EBase-Spezifikation zu entnehmen. Es müssen jeweils alle Tabellen vorhanden sein. Wird eine Tabelle nicht benötigt, muss sie als leere Tabelle angelegt sein.



go_info

Die optionale Tabelle *go_info* dient zur Definition MT-Serien-globaler Eigenschaften.

Schlüssel (key)	Der Schlüssel benennt eine Steuervariable. Die Wertemenge der Steuervariablen und die jeweilige zugehörige Wertemenge ist nachfolgend definiert.
Wert (value)	Der Wert setzt den Inhalt der Steuervariablen. Die Wertemenge der Steuervariablen und die jeweilige zugehörige Wertemenge ist nachfolgend definiert.

Die folgenden Steuervariablen und jeweiligen Wertemengen sind definiert. Dabei sind Steuervariablen wie folgt gekennzeichnet:

- [0, 1] – kann maximal einmal auftreten
- [1, 1] – muss genau einmal auftreten
- [0, *] – kann beliebig oft auftreten
- [1, *] – muss mindestens einmal auftreten

Schlüssel *configuration* [0, 1]

Der Schlüssel *configuration* steuert das übergreifende Konfigurationsverhalten der Metatypen dieser Serie.

Die Werte sind wie folgt:

- *consistent* – Die Metatypimplementierung versucht durch Anpassen anderer Merkmale stets wieder einen konsistenten Zustand in Bezug auf die Artikelpolymorphie herzustellen. Dieses Verhalten wird auch angewendet, wenn der Schlüssel nicht angegeben ist.
- *inconsistent* – Die Metatypimplementierung verzichtet auf die Anpassung anderer Merkmale, wodurch es zu inkonsistenten Zuständen kommen kann.
- *serial* – Die Metatypimplementierung fixiert einmal gewählte Merkmale und passt die Freiheitsgrade der noch nicht gewählten dynamisch an.

Schlüssel *pindex* [0, 1]

Der Schlüssel *pindex* ist zu setzen, wenn die merkmalsindizierten Polymorphietabellen *go_propindex* und *go_propmapping* existieren.

Der Wert ist die Anzahl der Merkmalsspalten in diesen Tabellen und muss eine positive Ganzzahl > 1 sein.

Schlüssel *skip_FAN* [0, 1]

Der Schlüssel *skip_FAN* soll gesetzt werden, wenn die Kontrolle (und ggf. Fehlerausgabe) der Endartikelnummer eines entsprechend erzeugten Artikels unterdrückt werden soll.

Der Wert an sich wird ignoriert. Er muss lediglich existieren.

Schlüssel *skipVC2MT* [0, 1]

Der Schlüssel *skipVC2MT* soll gesetzt werden, wenn kein Mapping von Metatyp-Merkmalen auf den Variantcode erfolgen soll. Das kann die Bearbeitungsgeschwindigkeit bei großen Merkmalstabellen beschleunigen.

Der Wert an sich wird ignoriert. Er muss lediglich existieren.

Schlüssel *updateGMode* [0, 1]

Der Schlüssel *updateGMode* soll gesetzt werden, wenn bei einer Änderung der Grundartikelnummer der Wert des Flags *GMode* (Typ der Produktdatenbank) aktualisiert werden soll. Dies ist notwendig, wenn zwei Serien,

die in dem selben Metatypen verwendet werden, unterschiedliche Typen der Produktdatenbanken (z.B. OCD und EPL) verwenden.

Der Wert an sich wird ignoriert. Er muss lediglich existieren.

Schlüssel *utf8* [0, 1]

Der Schlüssel *uft8* muss gesetzt werden, wenn die Tabelle *go_texts* mit UTF-8 Zeichensatz kodiert wird. Am Anfang der Datei kann optional das Byte Order Mark angegeben werden. Als Normalform sollte NFC (Normalization Form Canonical Composition) verwendet werden.

Der Wert an sich wird ignoriert. Er muss lediglich existieren.

go_types

Die Tabelle *go_types* definiert herstellerspezifische Instanzen eines Metatypes. Einer solchen Instanz können dann konkrete Artikelnummern in der Tabelle *go_articles* zugewiesen werden. Nachfolgend werden solche Instanzen als Metatyp bezeichnet.

Bsp.:
Unter Referenzierung des allgemeinen Metatypes ‚Schreibtisch' (*GO_TABLE*, s.u.) definiert der Hersteller HE1 einen Typ, welcher breitenvariable Tische mit konstanter Tiefe (800) und Höhe beschreibt. Er definiert einen zweiten Typ, welcher breitenvariable Tische mit konstanter Tiefe (1200) und Höhe beschreibt.

Dagegen definiert der Hersteller HE2 eine Instanz desselben Metatypes, welche Tische beschreibt, die bzgl. Breite, Höhe und Tiefe variabel sind.

Typ-ID (id)	Der Schlüssel referenziert auf eindeutige Weise im Kontext eines Herstellers einen Metatype. Die Definition eines Metatypes erfolgt über eine oder mehrere Zeilen in der *go_types*. Hierbei definiert jede Zeile genau eine Eigenschaft (Merkmal) des Metatypes. Bsp.: *tisch1*
Property-Name (name)	Dieser Eintrag definiert den Namen des Merkmals, auf welches sich die weiteren Angaben in dieser Zeile der Tabelle beziehen. Merkmalsnamen werden vorzugsweise in Englisch geschrieben. Sie beginnen mit einem ‚G', gefolgt von einem Großbuchstaben. Bei zusammengesetzten Wörtern beginnt auch jedes weitere Wort mit einem Großbuchstaben. Umlaute, Sonder- und Leerzeichen sind nicht erlaubt. Die für den angegebenen GO-Typ vordefinierten Merkmale müssen mindestens beschrieben werden. Weitere Merkmale sind optional. Das Merkmal *GType* ist reserviert. Es kann jedoch in Bedingungen und Aktionen verwendet werden. Das Merkmal *GVarPrefix* ist reserviert. Es kann jedoch für Daten im Modus @EPL in Bedingungen und Aktionen verwendet werden. Es hat entweder den Wert NULL oder den entsprechenden Präfix als Symbol. Bsp.: *GWidth* (Breite), *GHeightAdjust* (Höhenverstellung)
Format (format)	Diese Spalte definiert das Format des Merkmals. Für Merkmale, die durch den GO-Typ vordefiniert sind, muss auch das entsprechend vordefinierte Format verwendet werden. Die unterstützten Formate sind (entsprechend OFML-Standard): • *ch* – Auswahl aus einer Menge von symbolischen Werten • *chf* – Auswahl aus einer Menge von Gleitkommazahlen • *chi* – Auswahl aus einer Menge von Ganzzahlen

	• *f* – Gleitkommazahl
	• *i* – Ganzzahl
	Zusätzlich gibt es die folgenden Formate:
	• *fn* – freies numerisches Merkmal (*free numeric property*). Dabei ist folgendes zu beachten:
	  • Dieses Metatypmerkmal unterscheidet sich von den klassischen Metatypmerkmalen dadurch, dass es nicht zwangsläufig einen diskreten Wertebereich besitzt und außerdem nicht den üblichen Mechanismen (z.B. Filter) unterworfen ist.
	  • fn-Merkmale werden über die separate Tabelle *go_freenumeric* parametrisiert.
	  • fn-Merkmale können an Kindobjekte weitergeleitet werden. Im Fall von interaktiv beweglichen Kindobjekten ist zu beachten, dass keine Rückkopplung bei Bewegung des Kindes hin zum fn-Merkmal erfolgt. Es ist somit auch möglich, dass sich die Bewegungsfreiheitsgrade (Achsen, Minimum, Maximum, Raster, etc.) zwischen fn-Merkmal und zugeordnetem Kind unterscheiden.
	  • Im Fall von fn-Merkmalen kann der Filtereintrag verwendet werden, um eine Liste abhängiger Kindmerkmale anzugeben. Falls nicht anders angegeben (Modus 2048), werden die abhängigen Kinder bei Änderung des Merkmals neu erzeugt. Auf diese Weise kann z.B. eine Positionsanpassung der entsprechenden Kinder implementiert werden.
	• *na* – Wrapper eines nativen Merkmals (*native*). Hier wird ein Merkmal des eingekapselten Kindes auf Ebene des Metatypes zugreifbar gemacht. Dabei ist folgendes zu beachten:
	  • Das Merkmal heißt genau wie das native zzgl. dem Präfix *G*, z.B. *GFrame* als Metatypwrapper von *Frame*.
	  • Das Merkmal ist nur verfügbar, wenn auch das Kindobjekt existiert. Beispielsweise ist es initial nicht verfügbar, da erst das Kindobjekt anhand der Metatypmerkmale gefunden werden muss. Ist das Merkmal nicht verfügbar, so wird der Default-Wert aus dieser Tabelle verwendet, sofern er definiert ist. Analog zu anderen Stellen, wo Werte gesetzt werden, erfolgt die Angabe bei symbolischen Werten ohne führendes ‚@'.
	  • na-Merkmale unterstützen keine weiteren Eigenschaften wie Filter und Sichtbarkeit.
	  • na-Merkmale dürfen nicht in Filtern anderer Merkmale verwendet werden.
	  • na-Merkmale dürfen nur für solche nativen Merkmale verwendet werden, die nicht den Wert NULL annehmen.
	  • Falls bei der Konvertierung nativer Merkmale nach OFML ein Präfix vor die Merkmalswerte vorangestellt wird (um den syntaktischen Vorgaben von OFML zu genügen), so muss dieser auch in den Aktionen verwendet werden. Im Fall von EPL-Daten ist dies häufig der Buchstabe ‚S'. Dies ist auch bei der etwaigen Angabe eines Default-Wertes zu beachten.
	  • Im Fall von na-Merkmalen kann der Filtereintrag verwendet werden, um eine Liste abhängiger Kindmerkmale anzugeben. Falls nicht anders angegeben (Modus 2048), werden die abhängigen Kinder bei Änderung des Merkmals neu erzeugt. Auf diese Weise kann z.B. eine Positionsanpassung der entsprechenden Kinder implementiert werden.
	• *th* – Durchschleusen eines Merkmals (*through*). Hier wird ein Merkmal z.B. eines Vorgänger-Metatyps übernommen und kann von einem Nachfolger-Metatyp ausgewertet oder geerbt werden. Das th-Merkmal ist nicht Bestandteil der Artikelpolymorphie. Weiterhin ist zu beachten:
	  • Als durchgeschleuste Merkmale werden nur reguläre Metatypmerkmale unterstützt.
	  • Der Filter für dieses Merkmal muss den Merkmalstyp des durchgeschleusten Merkmals beinhalten.
	  • Das Merkmal ist nicht anderweitig verfügbar, d.h. keine Anzeige im

	Merkmalseditor, keine Verwendung in Filtern, usw. Die dbzgl. Einstellungen im Merkmalsmodus (Editierbarkeit, Sichtbarkeit) werden ignoriert.
	• cp – Repräsentation der Position (*child position*) von direkten oder indirekten Kindern des Metatyps im Kontext von Bedingungen und Variantenkonstruktion. Hierbei wird durch je ein cp-Merkmal eine Koordinate eines Kindes im Metatyp verfügbar gemacht. Dabei ist zu beachten:
	  • Das Merkmal ist nicht anderweitig verfügbar, d.h. keine Anzeige im Merkmalseditor, keine Verwendung in Filtern, usw.
	  • Der Merkmalswert ist immer in Meter definiert und zwar bezugnehmend auf das lokale Koordinatensystem des Metatyps.
	  • Der Initialwert eines cp-Merkmals wird dann verwendet, wenn das Bezugsobjekt nicht existiert.
	  • Der Modus eines cp-Merkmals bezeichnet die Koordinate, die das Merkmal repräsentiert. Hierfür wird ein zweistelliges Kürzel verwendet. Das erste Zeichen kennzeichnet die Koordinate: 1(x), 2(y) oder 3(z). Das zweite Zeichen beschreibt die Ausrichtung innerhalb der Koordinate. Folgende Zeichen sind erlaubt: 0 – Position, 1 – Maximum der lokalen Bounding-Box, 2 – Minimum der lokalen Bounding-Box, 3 – Zentrum der lokalen Bounding-Box. Bsp.: 21 – Maximum der lokalen Bounding-Box in y-Richtung, also die Oberkante des angegebenen Objekts.
	  • Der Filter eines cp-Merkmals wird als relativer Objektname interpretiert, der das Referenzobjekt angibt. Hierfür ist ein entsprechender Pfadname anzugeben. Bsp.: *e1* oder *e.e1*
	• lb – Ein Merkmal welches als nichteditierbarer Text angezeigt wird. Das Merkmal ist nicht weiter verfügbar, außer im Evaluierungskontext.

Initialwert (default)	Diese Spalte definiert den Initialwert, welcher bei Erzeugung eines Objekts dem in dieser Zeile definierten Merkmal zugewiesen werden soll. Hierbei handelt es sich um einen polymorphen Wert entsprechend dem Format des Merkmals. Für nicht-gewählte Merkmale mit dem Format *ch* wird der Wert *@VOID* verwendet. Soll kein Initialwert explizit vorgegeben werden, bleibt das Feld leer. Bsp.: *1000, H1* Für den Fall, dass das Merkmal eine Anzahl von unsichtbaren identischen Unterpositionen steuert, gilt folgende Verwendung des Initialwertes: *[Start, Minimum, Maximum]* – Es wird ein Wertebereich definiert. Das Minimum muss eine nichtnegative Zahl sein. Das Maximum muss größer als das Minimum sein. Der Startwert muss sich im Bereich Minimum – Maximum bewegen, wobei die Grenzen zulässige Werte sind. Bsp.: *[1, 0, 5]* – Es sind 0 bis 5 Unterpositionen möglich; der Startwert ist 1 Unterposition. Der vordefinierte Wert *MT_UNDEF* kennzeichnet ein nicht gewähltes Merkmal, was insbesondere für den seriellen Konfigurationsmodus relevant ist. *MT_UNDEF* sollte nur für polymorphe Merkmale mit dem Format *ch* verwendet werden.
Modus (mode)	Der Modus steuert die Verwendung des Merkmals. Hierbei erfolgt eine Kombination der nachfolgend definierten Einzelmodi. Achtung: Die Modi 4 und 8 sind **nicht** gleichzeitig erlaubt. • 1 – Das Merkmal ist editierbar. • 2 – Das Merkmal wird bei der globalen Variantenbearbeitung berücksichtigt. • 4 – Das Merkmal hat Einfluss auf den Variantencode. • 8 – Das Merkmal steuert eine Unterposition. Hierbei wird zwischen der regulären Auswahlliste (CH - Format ‚ch') und einer ganzzahlbasierten Ansteuerung unterschieden (INT - Format ‚i'). Im Fall CH steuern verschiedene Merkmalswerte jeweils verschiedene Instanzen von Unterpositionen anhand der entsprechend Kindtabellen (siehe unten). Im Fall INT wird eine Anzahl identischer nichtsichtbarer Unterpositionen erzeugt. Die Angabe erfolgt über den Initialwert des Merkmals welche aus drei Werten besteht:

	dem Initialwert, dem Minimum und dem Maximum (siehe oben). Beim Zugriff auf die Kindertabelle wird das Merkmal als Schlüssel verwendet. Aus der Kindertabelle wird nur der erste Eintrag ausgewertet. Alle weiteren werden ignoriert. • 16 – Das Merkmal wird nicht dargestellt. • 32 – Das Merkmal soll bei der Erzeugung des Objekts vom ggf. vorhandenen Vaterobjekt (sofern dieses auch ein Metaobjekt ist) oder Vorgänger übernommen werden. Falls einem Merkmal per Aktionen während der Erzeugung bereits Werte zugewiesen werden, sollte für dieses nicht Mode 32 gesetzt werden. • 64 – Die Anpassung des Merkmals infolge eines Filterprozesses wird nicht in der Dialogbox angezeigt. Dieser Modus wird nur ausgewertet, wenn in der Tabelle *go_setup* die Eigenschaft *ShowPolyPropFilterMsg* aktiviert wurde. • 128 – Es soll **nicht** die Standardsortierung der Merkmalswerte im Fall von Auswahllisten verwendet werden. Ist dieses Flag für Standard-Metatyp-Merkmale gesetzt, so wird eine explizite Positionssteuerung mittels Tabelle *go_proporder* angewendet. Ist dieses Flag für kinderzeugende Merkmale gesetzt, so wird die Originalsortierung aus der Tabelle *go_childprops* verwendet. • 256 – Nach Änderung des Merkmals soll die 2D- und 3D-Geometrie des Hauptkindes neu erzeugt werden. Anmerkung: Dies ist nur für kindsteuernde Merkmale in Ausnahmefällen notwendig. • 512 – Dieses Flag erzwingt das Löschen und Neuerzeugen der eigenen Kinder bei Änderungen dieses Merkmals. Ist das Flag nicht gesetzt, werden diese Kinder nur repositioniert. Das Löschen und Neuerzeugen kann in speziellen Fällen nützlich sein. Das Verhalten entspricht *GSetup & 16384*. Allerdings gilt letzteres für alle Merkmale des Metatypes. • 1024 – Nach Änderung des Merkmals soll die 2D- und 3D-Geometrie des Hauptkindes neu positioniert werden. Anmerkung: Dies ist bei Polymorphiemerkmalen nicht notwendig. • 2048 – Nach Änderung des Merkmals sollen die eigenen Kinder neu positioniert werden. Anmerkung: Dies ist nur bei na- und fn Merkmalen notwendig. Die Liste der abhängigen Kindmerkmale muss hierbei in der Spalte *Filter* angegeben werden. • 4096 – Nach Änderung des Merkmals soll eine OFML-typische Kollisionsprüfung stattfinden. Schlägt diese fehl, wird die Merkmalsänderung mit eine entsprechenden Hinweismeldung abgebrochen. • 8192 – Die Gültigkeit von Werten dieses Merkmals kann über die Tabelle *go_propvalues* eingeschränkt werden. Bsp.: *3* – Somit sind für das Merkmal Editierbarkeit und globale Bearbeitung aktiviert.
Filter (filter)	Über den Filter kann angegeben werden, welche anderen Merkmale bei der Tabellenabfrage während der Konfiguration dieses Merkmals berücksichtigt werden. Der Filter wird durch eine Liste von Merkmalsnamen, welche voneinander durch Komma getrennt sind, angegeben. Existiert kein Filter, ist dieses Feld leer. Bsp.: *GWidth, GHeight, GDepth, GHeightAdjust* Im seriellen Konfigurationsmodus wird der Filter ignoriert und statt dessen automatisch aus den bereits explizit gesetzten polymorphen Merkmalen gebildet.

Vordefinierte Merkmale sind:

- *GType [-]* – Dieses zwingend erforderliche Merkmal gibt den zu verwendenden Metatyp an.
- *GMode [-]* – Dieses optionale Merkmal gibt die Art der verwendeten Produktdaten an. Die folgenden Arten werden unterstützt:
  - *EPDF (default)* – Die kaufmännischen Daten werden durch eine EPDF-Produktdatenbank bereitgestellt.
  - *XOCD* – Die kaufmännischen Daten werden durch eine XOCD-Produktdatenbank bereitgestellt.
  - *OCD* – Die kaufmännischen Daten werden durch eine OCD-Produktdatenbank bereitgestellt.
  - *EPL* – Die kaufmännischen Daten werden durch die EPL bereitgestellt.
- *GSetup [-]* – Dieses optionale Merkmal kann verwendet werden, um spezielle Eigenschaften des Metatyps zu steuern. Die folgenden Einstellungen werden unterstützt, wobei die einzelnen Einstellungen aufaddiert

werden. Dieses Merkmal ist inzwischen nicht mehr notwendig, stattdessen sollte die Tabelle *go_setup* verwendet werden.

- 1 – Dieses Flag steuert, ob der Metatyp als eigener Knoten in der Bestellliste erscheint. Ist das Flag gesetzt, dann wird keine separate Repräsentation des Metatyps erzeugt.
- 2 – Dieses Flag steuert, ob die weiteren Kinder (d.h. alle außer dem Hauptkind) Unterpositionen des Hauptkindes sein sollen. Ist das Flag **nicht** gesetzt, werden die Kinder auf derselben Ebene wie das Hauptkind in der Bestellliste angezeigt. Andernfalls werden sie zu Unterpositionen des Hauptkindes. Diese Einstellung, welche global für alle Kinder (außer dem Hauptkind) gilt, kann durch GSetup & 8192 für ein konkretes Kind überschrieben werden.
- 4 – Es werden 2D-Symbole generiert. Dies sollte nur angewendet werden, falls keine 2D-Symbole explizit hinterlegt wurden.
- 8 – Es werden die Standardanfügepunkte verwendet. Dies kann benutzt werden, falls keine Anfügepunkte im Rahmen der Metatypen definiert werden sollen.
- 16 – Die per nichteditierbares Merkmal automatisch angezeigte Artikelnummer des Hauptkindes kann über dieses Bit ausgeblendet werden.
- 32 – Falls dieses Bit gesetzt ist, wird die Auflistung der Merkmale, welche sich aufgrund der Filter automatisch angepasst haben, unterdrückt.
- 64 – Dieses Bit steuert die Kollisionserkennung für Kindobjekte während der initialen Platzierung und auch während der interaktiven Bewegung. Ist das Bit nicht gesetzt, erfolgt bei der initialen Platzierung lediglich ein Test, ob sich exakt an der Einfügestelle bereits ein Objekt befindet; während der interaktiven Bewegung erfolgt keine Kollisionsprüfung. Ist das Bit gesetzt, erfolgt in beiden Fällen die OFML-typische Kollisionsprüfung.
- 128 – Metatypobjekte erben native Merkmale von einem Einfügeobjekt. Für den Fall, dass das Metatypobjekt wiederum Kind eines Metatypobjekts ist, kann über dieses Flag gesteuert werden, ob der Vater als Einfügeobjekt verwendet wird oder ein selektiertes Objekt auf gleicher Ebene wie das neu zu erzeugende. Ist das Flag gesetzt und existiert ein solches selektiertes Objekt, dann wird dieses verwendet, um die Eigenschaften zu erben. Andernfalls werden die Eigenschaften vom Vater geerbt.
- 256 – Das automatische Löschen von interaktiv geplanten Anbauteilen, für welche nach einer Konfiguration des Vaterobjekts kein gültiger Anfügepunkt mehr existiert, kann über dieses Flag eingeschaltet werden. Ist das Flag gesetzt, erfolgt im beschriebenen Fall eine Abfrage und – falls diese mit ‚Ja' beantwortet wird – das Löschen des Kindes. Ist dieses Flag nicht gesetzt, erfolgt keine Auswertung dieser Situation. Hierbei wird der *GSetup*-Wert des betreffenden Kindes ausgewertet.
- 512 – Standardmäßig erscheint beim Aneinanderplanen von Objekten, welche zwar planungsrichtungs-kompatible Anfügepunkte besitzen, jedoch ein Anfügepunktpaar nicht durch mindestens 1 gemeinsame Aktion als zusammengehörig kennzeichnen, ein Hinweis, dass keine Anschlussplanung im Sinne der Regelprüfung erfolgt. Setzt man dieses Flag, so wird die Meldung unterdrückt. Ausschlaggebend ist der Wert des Flags am anzufügenden Objekt.
- 1024 – Über dieses Flag kann die vollständige Instanziierung des Hauptkindes während der temporären Erzeugung eingeschaltet werden. Dies ist beispielsweise erforderlich wenn auf präzise Bounding-Boxes oder spezifische Merkmale (z.B. *GVarPrefix*) zugegriffen werden soll.
- 2048 – Ist dieses Flag gesetzt, dann erbt das Objekt keine nativen Merkmale für den Fall, dass sein Vorfahr ein Metatyp ist.
- 4096 – Ist dieses Flag gesetzt, so wird für die initiale Platzierung auf Geschwisterebene während der Überprüfung der Anfügepunkte die Kollisionserkennung für dieses Objekt ausgeschaltet. Dies kann erforderlich sein, wenn erst durch die Anpassung im Rahmen von Aktionen die Kollision beseitigt wird. Es ist dadurch aber auch möglich mehrere Objekte ineinanderzuplatzieren, wodurch fehlerhafte Planungen entstehen können.
- 8192 – Dieses Flag erzwingt für einen Metatyp M1, welcher Kind eines anderen Metatyps M2 ist, das M1 in M2 als Hauptposition geführt wird, auch wenn M2 für seine Kinder mittels GSetup & 2 definiert hat, das diese Unterpositionen sind.
- 16384 – Dieses Flag erzwingt das Löschen und Neuerzeugen der eigenen Kinder bei Merkmalsänderungen. Ist das Flag nicht gesetzt, werden diese Kinder nur repositioniert. Das Löschen und Neuerzeugen kann in speziellen Fällen nützlich sein. Dieses Verhalten war bis einschließlich GO 1.3.9 das Standardverhalten.

  - 32768 – Dieses Flag stellt das bisherige Verhalten wieder her, wonach na-Merkmale im Merkmalseditor der Merkmalsklasse des Metatypes zugeordnet werden. Andernfalls bekommt das na-Merkmal die Merkmalsklasse des zugehörigen nativen Merkmals und wird somit auch in diesem Bereich im Merkmalseditor dargestellt.
- *GXSetup [-]* – Dieses optionale Merkmal kann verwendet werden, um spezielle Eigenschaften des Metatypes zu steuern. Die folgenden Einstellungen werden unterstützt, wobei die einzelnen Einstellungen aufaddiert werden. Dieses Merkmal ist inzwischen nicht mehr notwendig, stattdessen sollte die Tabelle *go_setup* verwendet werden.
  - 1 – Dieses Flag steuert, ob bei Translation eines Metatypes als Kind eines anderen Metatypes M, eine Kollisionserkennung unter den anderen Kindern von M stattfinden soll (1) oder nicht (0).
  - 2 – Dieses Flag steuert, ob bei Rotation eines Metatypes als Kind eines anderen Metatypes M, eine Kollisionserkennung unter den anderen Kindern von M stattfinden soll (1) oder nicht (0).
  - 4 – Sofern das Hauptkind des Metatypes von der Kollisionserkennung in den GXSetup-Modi 1 und 2 ausgeschlossen werden soll, muss dieses Flag gesetzt werden. Andernfalls wird das Hauptkind bei der Kollisionserkennung berücksichtigt.
  - 8 – Der interaktive Feedback-Modus (verfügbar ab Version 1.11) wird freigeschaltet (1) oder nicht bzw. wird nicht unterstützt (0).
- *GAlign* [-] – Dieses optionale Merkmal steuert die geometrische Ausrichtung des Hauptkindes. Diese wird durch 3 Buchstaben angegeben, welche auf die Achsen x, y und z in dieser Reihenfolge abgebildet werden. Für die Buchstaben sind folgende Zeichen erlaubt (welche sich jeweils auf die entsprechende Achse beziehen):
  - *N* – Es findet keine Ausrichtung statt. (<n>o alignment)
  - *I* – Das Objekt wird so ausgerichtet, dass die minimale Abmessung entlang der entsprechenden Achse des lokalen orthogonalen Begrenzungsvolumens des Hauptkindes dem Ursprung des Metatypes entspricht. (m<i>nimum)
  - *C* – Das Objekt wird so ausgerichtet, dass die Mitte entlang der entsprechenden Achse des lokalen orthogonalen Begrenzungsvolumens des Hauptkindes dem Ursprung des Metatypes entspricht. (<c>enter)
  - *A* - Das Objekt wird so ausgerichtet, dass die maximale Abmessung entlang der entsprechenden Achse des lokalen orthogonalen Begrenzungsvolumens des Hauptkindes dem Ursprung des Metatypes entspricht. (m<a>ximum)

  Der Default-Wert ist *III* – somit erfolgt eine Ausrichtung am linken, unteren, hinteren Punkt des lokalen orthogonalen Begrenzungsvolumens des Hauptkindes.
  Bsp.:
  - *NNN* – keinerlei Ausrichtung des Objekts
  - *CIC* – symmetrische Ausrichtung bzgl. x- und z-Achse (z.B. für Stühle)

  Alternativ kann *ATTPT* als Wert für *GAlign* angegeben werden. In diesem Fall spezifiziert der Anfügepunkt *_GO_CHILD* sofern in der Tabelle *go_attpt* definiert, den Offset des Hauptkinds. Auf diese Weise kann außerdem eine Rotation um die y-Achse erreicht werden. Falls eine der oben genannten Ausrichtungen verwendet wird und außerdem ein gültiger _GO_CHILD-Anfügepunkt existiert, dann wird letzterer als Offset bzgl. der durch die Ausrichtung ermittelten Position angewendet.
- *GWidth, GHeight, GDepth* – Diese Merkmale werden intern verwendet – sofern sie definiert sind - um z.B. ein Dummy-Objekt anzuzeigen, falls kein realer Artikel für die aktuellen Parameterwerte gefunden werden konnte. Falls Breite (Höhe, Tiefe) verwendet wird, sollte dafür auch die entsprechende Variable *GWidth* (*GHeight*, *GDepth*) verwendet und mit einem numerischen Wert versehen werden.
- *GMetaLabel* – Dieses optionale Merkmal definiert eine symbolische String-Ressource, welche in der Bestellliste angezeigt wird, falls der Metatyp einen separaten Knoten darstellt (GSetup & 0). Die Angabe erfolgt durch den Default-Wert. Es wird ein Bezeichner angegeben ohne ‚@' und Namensraum, welcher in der oder den spezifischen Ressourcen-Datei(en) definiert sein muss.
- *XHeight* – Dieser Wert, der innerhalb von Evaluierungskontexten zur Verfügung steht, enthält die aktuelle Höhenposition des Objekts in Bezug auf dessen Koordinatensystems (welches entweder das globale oder das des Vorgängers ist).
- *XChildID* – Falls das Objekt reguläres Kind eines Metatypes ist, enthält diese Variable dessen Identifikator. Ansonsten ist der Wert der Variablen NULL.

- *XIsInsObj* – Mit Hilfe dieser Variable kann in der Tabelle *go_actions* definierten INS und REM Aktionen festgestellt werden, ob das in der Aktion behandelte Objekt das gerade neu Eingefügte (XIsInsObj == 1) ist, oder nicht.

go_articles

Die Tabelle *go_articles* ordnet einem in der Tabelle *go_types* definiertem Metatyp konkrete Artikelnummern zu. Zur Laufzeit ist es dann möglich, im Rahmen dieser Artikelnummern zu konfigurieren, ohne dass ein Löschen und eine Neuerzeugung des Objekts notwendig sind.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Hersteller (manufacturer)	Diese Spalte benennt den Hersteller des Artikels. Bsp.: *hex*
Serie (program)	Diese Spalte benennt die Serie des Artikels. Bsp.: *s1*
Grundartikel-nummer (article_nr)	Diese Spalte benennt den Artikel anhand seiner Grundartikelnummer. Bsp.: *S1TA0100*
Merkmale (prm_set)	Diese Spalte referenziert einen Parametersatz (Standardmerkmale) entsprechend den in der Tabelle *go_types* definierten Merkmale für den durch die erste Spalte referenzierten Metatyp. Der Parametersatz befindet sich in der Tabelle *go_properties.* Ebenfalls wird auf die Tabelle *go_noproperties* verweisen. Alternativ können die Tabellen *go_propindex* und *go_propmapping* verwendet werden. Bsp.: *pSITA0100_1*
Kinder-orientierte Merkmale (chprm_set)	Diese Spalte referenziert einen Parametersatz (Kind-orientierte Merkmale) entsprechend den in der Tabelle *go_types* definierten Merkmalen für den in der ersten Spalte referenzierten Metatyp. Der Parametersatz befindet sich in der Tabelle *go_childprops*. Es können mehrere Parametersätze durch Komma getrennt angegeben werden. Bsp.: *chSITA0100_1*

Der Bezeichner *_native_* ist reserviert und soll in Fällen verwendet werden, in denen bestimmte native Artikel von der automatischen Metatypfindung ausgeschlossen werden sollen, auch wenn es für andere Artikel mit derselben Basisartikelnummer Metatyp-Einträge gibt.

go_properties

Die Tabelle *go_properties* gibt für jeden Eintrag in der Tabelle *go_articles* einen vollständigen Parametersatz an. Dabei ist es möglich, dass ansonsten identische Einträge in der *go_articles* auf verschiedene Parametersätze in der *go_properties* verweisen. Dies ist der Fall, wenn unterschiedliche Varianten derselben Grundartikel-nummer verwenden werden. Je Parameter wird eine Zeile definiert; somit besteht ein Parametersatz im Allgemeinen aus mehreren Zeilen.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Schlüssel (key)	Der Schlüssel dient zur eindeutigen Identifikation eines Parametersets aus der Tabelle *go_articles*. Typischerweise werden je Schlüssel mehrere Zeilen eingetragen – jeweils eine je Merkmal. Alternativ können die Tabellen *go_propindex* und *go_propmapping* verwendet werden. Bsp.: *pSITA0100_1*

Name (name)	Diese Spalte benennt den Namen des Merkmals und muss dabei exakt einem der in Tabelle *go_types* definierten Merkmale entsprechen. Bsp.: *GWidth, GHeightAdjust*
Wert (value)	Diese Spalte setzt den Wert für das jeweilige Merkmal im Kontext des jeweiligen Artikeleintrags. Dieser muss dabei ein gültiger Wert entsprechend dem in Tabelle *go_types* definierten Format des Merkmals sein. Bsp.: *800, H1*
Variantencode (variant_code)	Diese Spalte benennt den Teil des Variantencodes, der durch den nachfolgenden Eintrag *Variantenwert* gesetzt wird. Hat das Merkmal keinen Einfluss auf die Variante, wird ein Leerstring verwendet. Bsp.: *ER*
Variantenwert (variant_value)	Diese Spalte beschreibt einen partiellen Variantencode, der im Falle dieses Parametersatzes wirksam wird. Bsp.: *NN*

go_propindex

Die Tabelle *go_propindex* stellt eine optionale, spaltenorientierte Repräsentation einer Untermenge der Tabelle *go_properties* dar. Die Anzahl der Spalten in dieser Tabelle ist serienabhängig und muss der Tabelle *go_propmapping* entsprechen.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Schlüssel (key)	Der Schlüssel dient zur eindeutigen Identifikation eines Parametersets aus der Tabelle *go_articles*. Typischerweise werden je Schlüssel mehrere Zeilen eingetragen – jeweils eine je Merkmal. Bsp.: *pSITA0100_1*
Wert 1 (value1)	Diese Spalte setzt den Wert für das Merkmal entsprechend der Mapping-Tabelle im Kontext des jeweiligen Artikeleintrags. Er muss dabei ein gültiger Wert entsprechend dem in Tabelle *go_types* definierten Format des Merkmals sein. Bsp.: *800*
...	
Wert <n> (value<n>)	Diese Spalte setzt den Wert für das Merkmal entsprechend der Mapping-Tabelle im Kontext des jeweiligen Artikeleintrags. Er muss dabei ein gültiger Wert entsprechend dem in Tabelle *go_types* definierten Format des Merkmals sein. Bsp.: *H1*

go_propmapping

Die Tabelle *go_propmapping* gibt für die polymorphen Merkmale aus der Tabelle *go_propindex* die Reihenfolge an, in welcher diese die Spalten in der Tabelle *go_propindex* belegen. Die Anzahl der Spalten in dieser Tabelle ist serienabhängig und muss der Tabelle *go_propindex* entsprechen.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Name 1 (key1)	Diese Spalte benennt den Namen des Merkmals und muss dabei exakt einem der in Tabelle *go_types* definierten Merkmale entsprechen. Bsp.: *GWidth*
...	
Name <n> (key<n>)	Diese Spalte benennt den Namen des Merkmals und muss dabei exakt einem der in Tabelle *go_types* definierten Merkmale entsprechen. Bsp.: *GHeightAdjust*

go_childprops

Die Tabelle *go_childprops* implementiert das Mapping von Merkmalen auf die Kinderzeugungstabelle *go_children*.

Schlüssel (key)	Der Schlüssel referenziert den entsprechenden Eintrag der Tabelle *go_articles*. Bsp.: *pSITA0100_1*
Name (name)	Der Eintrag benennt das Merkmal, welches für die Erzeugung dieser Unterposition relevant ist. Bsp.: *GCableSet*
Wert (value)	Der Eintrag benennt den Merkmalswert, welcher für die Erzeugung dieser Unterposition relevant ist. Bsp.: *Set2*
Kindschlüssel (child_key)	Diese Spalte referenziert letztlich die konkrete Kinderzeugung aus der Tabelle *go_children*. Es ist möglich, mehrere Kindschlüssel durch Komma (,) getrennt anzugeben. Bsp.: *set2*

go_children

Die Tabelle *go_children* definiert die notwendigen Angaben um eine Unterposition (Kind) zu erzeugen. Sie wird ebenfalls verwendet, um die interaktive Platzierung eines Kindes an einem Anfügepunkt entsprechend *go_attpt* zu erlauben. In diesem Modus ist der Schlüssel ein Anfügepunktschlüssel und die geometrischen Werte werden ignoriert und können somit leer bleiben. Außerdem wird hier nur die Artikelnummer, nicht aber Hersteller und Variantencode berücksichtigt.

Schlüssel (child_key)	Der Schlüssel dient zur eindeutigen Identifizierung aus der Tabelle *go_childprops*. Bsp.: *set2*
Hersteller (manufacturer)	Diese Spalte benennt den Hersteller des Artikels. Bsp.: *hex*
Serie (program)	Diese Spalte benennt die Serie des Artikels. Bsp.: *s1*
Grundartikelnummer (article_nr)	Diese Spalte benennt den Artikel anhand seiner Grundartikelnummer. **Achtung:** Es müssen alle Grundartikelnummern (jeweils 1/Spalte) eingegeben werden, die ein einzufügendes Objekt initial bzw. nach automatischer Anpassung annehmen kann. Bsp.: *S1SET0100*
Variantencode (variant)	Diese Spalte benennt eine optionale zusätzliche Variante des Artikels. Für den Fall, dass das Kind auch ein Metatyp ist, können hierüber auch Metamerkmale gesetzt werden, siehe dazu auch *Parametrisierung*. Bsp.: *[@hex, @anbau, [@GWidth, 1600]]*
X-Position (pos_x); Y-Position (pos_y); Z-Position (pos_z)	Die Einträge geben die X, Y und Z-Position in Metern relativ zum übergeordneten Objekt an. Hierbei kann die Variantenspezifikation eingesetzt werden. Dabei sind alle numerischen Merkmale aus der Tabelle *go_types* erlaubt. Bsp.: *0.2; 0.72; 0.1*
X-Rotation (rot_x); Y-Rotation (rot_y); Z-Rotation (rot_z)	Die Einträge geben die Verdrehung um die X, Y und Z-Achse relativ zum übergeordneten Objekt in Grad an. Die Reihenfolge ist X-Rotation, Y-Rotation, Z-Rotation. Hierbei kann die Variantenspezifikation eingesetzt werden. Dabei sind alle numerischen Merkmale aus der Tabelle *go_types* erlaubt. Bsp.: *0; 90; 0*
Bedingung (condition)	Die Bedingung bezieht sich auf den Zustand eines oder mehrerer Merkmale des Objekts und wird entsprechend OFML-Syntax geschrieben. Die Bedingung ist gültig, wenn ihr Ergebnis 1 ist bzw. sie leer ist. Dabei sind alle Merkmale aus der Tabelle *go_types* erlaubt.

	Achtung: Im Gegensatz zu anderen Tabellen oder Spalten muss symbolischen Werten hier ein ‚@'-Zeichen vorangestellt werden! Bsp.: • *GWidth==1200* • *GConcat != @NONE* • *(GWidth >= 1000 && GConcat != @LEFT)*

go_proporder

Die Tabelle *go_proporder* gibt optional eine positive Ordnungszahl welche bei Standard-Metatyp-Merkmalen zur Sortierung innerhalb einer symbolischen Merkmalsliste verwendet wird. Voraussetzung ist, dass die explizite Sortierung für das jeweilige Merkmal über den Modus 128 in der Tabelle *go_types* aktiviert ist. Die Sortierung erfolgt auf Basis der symbolischen Merkmalswerte und ist damit sprachunabhängig. Die Ordnungszahlen können mehrfach vergeben werden, z.B. kann in jeder Liste der erste Wert die Ordnungszahl 1 bekommen. Werden symbolische Werte in mehreren Listen verwendet, kann über eine geeignete Vergabe von Ordnungszahlen sichergestellt werden, dass der mehrfach verwendete Wert in jeder Liste seinen korrekten Platz erhält.

Ordnungszahlen müssen positive Zahlen kleiner 1000 sein. In einer Werteliste darf es nicht zu mehrfach auftretenden Ordnungszahlen kommen.

Werte innerhalb einer Werteliste, für die keine Ordnungszahl definiert ist, werden in undefinierter Reihenfolge an die Werte, für die eine Ordnungszahl definiert ist, angehangen.

Value (value)	Der Merkmalswert für welchen die Ordnungszahl definiert werden soll. Bsp.: *NONE*
Ordnungszahl (number)	Die Ordnungszahl für diesen Merkmalswert. Bsp.: *1*

go_propvalues

Über die Tabelle *go_propvalues* kann die Gültigkeit kindsteuernder und na-Merkmale eingeschränkt werden. Voraussetzung ist, dass diese Einschränkbarkeit für das jeweilige Merkmal über den Modus 8192 in der Tabelle *go_types* aktiviert ist. Merkmalswerte, die in dieser Tabelle nicht aufgeführt werden, sind generell gültig. Das Setzen der Werte von na-Merkmalen über Beziehungswissen in den kaufmännischen Daten wird hiervon nicht beeinflusst.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Name (name)	Der Eintrag benennt das Merkmal, für das die Gültigkeit eingeschränkt werden soll. Bsp.: *GCableSet*
Wert (value)	Der Eintrag benennt den Merkmalswert, dessen Gültigkeit eingeschränkt werden soll. Bsp.: *Set2*
Bedingung (condition)	Die Bedingung bezieht sich auf den Zustand eines oder mehrerer Merkmale des Objekts und wird entsprechend OFML-Syntax geschrieben. Die Bedingung ist gültig, wenn ihr Ergebnis 1 ist bzw. sie leer ist. Dabei sind alle Merkmale aus der Tabelle *go_types* erlaubt. **Achtung:** Im Gegensatz zu anderen Tabellen oder Spalten muss symbolischen Werten hier ein ‚@'-Zeichen vorangestellt werden! Bsp.: • *GWidth==1200* • *GConcat != @NONE*

go_noproperties

Die Tabelle *go_noproperties* gibt zwingend für einen Eintrag in der Tabelle *go_articles* Parameter an, welche nicht von dem originären Artikel übernommen werden sollen, da sie bereits auf Metatypebene vorhanden sind. Dies betrifft die Anzeige im Merkmalseditor und die Übernahme vom Vater oder Vorfahr beim Einfügen.

Schlüssel (key)	Der Schlüssel dient zur eindeutigen Identifikation eines Parametersets aus der Tabelle *go_articles*. Typischerweise werden je Schlüssel mehrere Zeilen eingetragen – jeweils eine je Merkmal. Bsp.: *pSITA0100_1*
Name (name)	Diese Spalte benennt den Namen des Merkmals und muss dabei einem Merkmal des Artikels entsprechen. Es können mehrere Merkmale durch Kommas getrennt angegeben werden. Bsp.: *ER*

go_inhproperties

Die Tabelle *go_inhproperties* gibt an vom welchem Metatyp welche Metamerkmale im Rahmen der initialen Merkmalsvererbung vom Vorfahr (Vater oder Geschwister) geerbt werden sollen. Existiert für einen Metatyp kein solcher Eintrag, werden alle Merkmale geerbt.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Vorfahr-ID (pid)	Der Schlüssel referenziert den Metatyp aus der Tabelle *go_types*, für den ein zu erbendes Metamerkmal spezifiziert wird. Bsp.: *tisch1*
Merkmal (property)	Es wird der Name des zu erbenden Merkmals angegeben. Bsp.: *GWidth*

go_nativeproperties

Die Tabelle *go_nativeproperties* erlaubt es, die Vererbung der nativen Merkmale beim Einfügen eines Metatyps flexibel zu steuern.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Vorfahr-ID (pid)	Der Schlüssel referenziert den Metatyp aus der Tabelle *go_types*, für den die nachfolgenden Einträge im Fall der Vererbung der nativen Merkmale ausgewertet werden sollen. Ist der Eintrag dagegen *_ANY*, so gelten die Einträge für beliebige Vorfahren, die Metatypen sind. Bsp.: *tisch1*
Modus (mode)	Der Modus bestimmt, ob dieser Eintrag die Vererbung oder den Ausschluss von der Vererbung spezifiziert und hat einen der Werte *INCL* oder *EXCL*.
Bezeichner (identifier)	Der Eintrag benennt entweder das Merkmal, für das dieser Eintrag relevant ist oder einen der beiden Werte *_ALL* oder *_DEFAULT*. Bsp.: *SShape*
Wert1 (value1)	Reserviert für zukünftige Verwendung.
Wert2 (value2)	Reserviert für zukünftige Verwendung.

Die Interpretation der Tabelle erfolgt auf diese Weise:

1. Existiert für eine gegebene Typ-ID mindestens ein Eintrag mit passender Vorfahr-ID, so werden alle weiteren Einträge mit einer anderen Vorfahr-ID bzw. *_ANY* ignoriert.
   Andernfalls werden nur die Einträge mit der Vorfahr-ID *_ANY* ausgewertet.

   Alle nachfolgenden Aussagen beziehen sich auf die sich daraus ergebende Untermenge der Einträge für eine gegebene Typ-ID.
2. Existiert ein Eintrag mit Modus gleich *INCL* und Bezeichner gleich *_ALL,* so werden **alle** nativen Merkmale (von dem angegebenen Vorfahren) geerbt.
3. Falls 2. nicht gilt: Existiert ein Eintrag mit Modus gleich *EXCL* und Bezeichner gleich *_ALL,* so werden **keine** nativen Merkmale (von dem angegebenen Vorfahren) geerbt.
4. Falls 3. nicht gilt: Existiert ein Eintrag mit Modus gleich *INCL* und Bezeichner gleich *_DEFAULT*, so werden **alle** nativen Merkmale (von dem angegebenen Vorfahren) geerbt, für die kein Eintrag mit Modus gleich *EXCL* und passendem Bezeichner (hier: Merkmalsname) existiert.
5. Falls 4. nicht gilt: Existiert ein Eintrag mit Modus gleich *EXCL* und Bezeichner gleich *_DEFAULT*, so werden **alle** nativen Merkmale (von dem angegebenen Vorfahren) von der Vererbung ausgeschlossen, für die kein Eintrag mit Modus gleich *INCL* und passendem Bezeichner (hier: Merkmalsname) existiert.
6. Falls 5. nicht gilt, die Tabelle für die gegebene Typ-ID keine Einträge hat oder die Tabelle gar nicht existiert, dann verhält sich die Anwendung wie unter 2.

Achtung: Native Merkmale im Sinn der Metatypen (na-Merkmale) sind in jedem Fall von der Vererbung auf nativer Ebene ausgeschlossen.

go_resetnativeprops

Über die Tabelle *go_resetnativeprops* kann gesteuert werden, dass bestimmte native Merkmale beim Grundartikeltausch auf den Initialwerte aus den kaufmännischen Daten zurückgesetzt werden. Bei den Merkmalen, für die kein solcher Eintrag existiert, werden die Merkmalswerte beibehalten.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Über den Platzhalter * kann auch für alle Metatypen der Serie ein Verhalten definiert werden. Bsp.: *tisch1*
Merkmal (key)	Es wird der Name des zu behandelnden nativen Merkmals angegeben. Bsp.: *BOARD_WIDTH*
Auslöser (trigger)	Hier kann angegeben werden, dass das native Merkmal nur bei Änderung eines bestimmten Polymorphiemerkmals zurückgesetzt wird. Es ist möglich, mehrere Merkmale durch Komma getrennt anzugeben. Ist dieses Feld leer, wird das entsprechende native Merkmal ausnahmslos bei jedem Grundartikeltausch zurückgesetzt. Bsp.: *GWidth, GDepth*

go_actions

Die Tabelle *go_actions* kann verwendet werden, um bestimmte Aktionen zu definieren, die beim Hinzufügen oder Entfernen eines Artikels oder bei der Konfiguration eines Artikels ausgeführt werden sollen. Die zugrundeliegenden Operationen sind:

- Erzeugung (*CREATE*) – Aufruf nach Erzeugen und Initialisierung eines Objekts
- Initialisierung (*INI*) – explizite Regelprüfung im Fall eines alleinstehenden Objekts
- Hinzufügen (*INS*) – Erzeugen, Einfügen, Bewegen
- Entfernen (*REM*) – Löschen, Ausschneiden, Bewegen
- Konfiguration (*CON*) – Konfiguration eines Objekts
- Anfügepunkt-Paar (*AP*) – Definition eines Anfügepunkt-Paares
- Stellvertreter-Anfügepunkt (*PROXY*) – Definition eines Stellvertreteranfügepunktes
- Kindanfügung (*CH_ADD*) – Anfügen eines Kindes welches auch ein Metatyp ist Etwaige Aktionen werden aufgerufen, nachdem das Kind komplett erzeugt wurde und für dieses auch eventuelle *CREATE*-Aktionen aufgerufen wurden.
- Kindlöschung (*CH_DEL*) – Löschen eines Kindes welches auch ein Metatyp ist
- Selektion eines Interaktors (*INTERACTOR*) – Aufruf nach Klicken auf einen Interaktor

Hinweis: Der Modus *AP* soll dann verwendet werden, wenn eine Paarigkeit von Anfügepunkten ausgedrückt werden soll, aber keine expliziten Aktionen für *REM* und *INS* definiert sind.

Generell werden Aktionen auf Basis zweier Anfügepunkte sowie der Angabe *REM, INS* usw. definiert. Mögliche Aktionen können sein:

- *SET_PROP* - Aktion zum Setzen eines Merkmalszustandes des Artikels
- *ADD_CHILD* - Aktion zum Erzeugen eines Generierteils als Unterposition des Artikels[1]
- *DEL_CHILD* - Aktion zum Entfernen eines Generierteils als Unterposition des Artikels
- *CH_PROP* - Aktion zum Setzen eines Merkmals für eine Unterposition. Hierbei werden alle Unterpositionen (zentrale Unterposition, merkmalsgesteuerte Unterposition und interaktive Unterposition) unterstützt, solange es sich um Instanzen von *GoMetaType* handelt.
- *CON_PROP* - Aktion zum Konfigurieren des Zustands eines Merkmals des Objekts
- *CON_CH_PROP* - Aktion zum Konfigurieren des Zustands eines Merkmals eines Kindobjekts
- *CON_AP* – Aktion zum Aktualisieren eines Anfügepunktes bzw. der Positionen der daran sich befindenden Geschwisterobjekte. Es werden lediglich translative Anpassungen unterstützt. An einem Anfügepunkt wird maximal ein Objekt angepasst. Allerdings kann die Anpassung dieses Objekts zur Anpassung eines weiteren führen, usw.

Die nachfolgende Tabelle zeigt an, welche Aktionen in welchem Modus verfügbar sind:

	ADD_CHILD	DEL_CHILD	SET_PROP	CON_PROP	CON_AP	CH_PROP	CON_CH_PR.
CREATE	x	x	x	x	-	-	-
INI	x	x	x	x	-	-	-
INS	x	x	x	x	-	-	-
REM	x	x	x	x	-	-	-
CON	-	-	x	x	x	x	x
AP	-	-	-	-	-	-	-
PROXY	-	-	-	-	-	-	-
CH_ADD	-	-	x	x	-	x	x
CH_DEL	-	-	x	x	-	x	x
INTERACTOR	x	x	x	x	-	x	x

Für jede Aktion kann die Ausgabe einer Mitteilung erfolgen, um den Nutzer auf Anpassungen aufmerksam zu machen.

[1] Es erfolgt keine Kollisionsprüfung.

Es können bei einem Einfüge-/Entfernungsprozess für einen Artikel mit gültiger Eigen-/Fremdschlüssel-Kombination mehrere Aktionen ausgelöst werden.

Bei Aktionen werden prinzipiell alle Objekte auf der selben Ebene wie das Objekt, welches die Aktionen auslöst, berücksichtigt. Für ein Planungsobjekt (also auf der obersten Ebene) bedeutet dies, dass alle anderen Planungsobjekte einbezogen werden. Für ein Anbauteil an ein Planungsobjekt bedeutet dies, dass alle weiteren Anbauteile desselben Planungsobjekts einbezogen werden.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Eigenschlüssel (own_key)	Der Eigenschlüssel benennt einen Anfügepunkt des Objekts. Bsp.: *vl1* **Abweichungen:** • *CREATE, INI, CH_ADD, CH_DEL* – In diesen Modi ist das Feld nicht belegt. • *CON* – Es wird der Name des Merkmals entsprechend *go_types (name)* angegeben, bei dessen Konfiguration die Aktion ausgelöst werden soll. Es können mehrere Merkmale durch Kommas getrennt angegeben werden. • *INTERACTOR* – Es wird der Schlüssel des Interaktors entsprechend *go_interactors (key)* angegeben.
Fremdschlüssel (foreign_key)	Der Fremdschlüssel benennt einen Anfügepunkt eines Nachbarobjekts. Bsp.: *tr1* **Abweichungen:** • *CREATE, INI, CH_ADD, CH_DEL* – In diesen Modi ist das Feld nicht belegt. • *CON, INTERACTOR* – Es wird hier ein Eintrag aus der Tabelle *go_children (key)* angegeben, der zur Identifizierung des Kindes dient. Es können mehrere Fremdschlüssel durch Kommas getrennt angegeben werden. Soll sich diese Aktion nicht auf Kinder beziehen, muss kein Fremdschlüssel angegeben werden. • *AP* – Wird über das Anfügepunktpaar ein Metaplanning-Workflow verknüpft, bezeichnet der Fremdschlüssel die vollqualifizierte ID dieses Workflows. Bsp.: *::hex::s1::@WorkflowAF*
Ursache (direction[2])	*CREATE, INI, INS, REM, CON, AP, PROXY, CH_ADD, CH_DEL, INTERACTOR* (siehe oben)
Bedingung (condition)	Die Bedingung bezieht sich auf den Zustand eines oder mehrerer Merkmale des Objekts und optional dessen Nachbarobjekts, und wird entsprechend OFML-Syntax geschrieben. Die Bedingung ist gültig, wenn ihr Ergebnis 1 ist bzw. sie leer ist. Für die Bedingungen wird ein Kontext erzeugt, der alle Merkmale des Objekts aus der Tabelle *go_types* mit ihren Werten enthält (Standardkontext). Soweit ein zweites Objekt (Nachbar oder Kind/Vater) involviert ist, wird dessen Zustand ebenfalls bereitgestellt (Nebenkontext). Zur Unterscheidung enthalten die Merkmalsnamen des Nebenkontextes ein vorangestelltes _. **Achtung:** Im Gegensatz zu anderen Tabellen oder Spalten muss symbolischen Werten hier ein ‚@'-Zeichen vorangestellt werden! Bsp.: • *GWidth==1200* • *GConcat != @NONE* • *(GWidth >= 1000 && GConcat != @LEFT)* • *GDepth==_GDepth*

[2] Aus historischen Gründen wird hier der Name *direction* verwendet, da es sich ursprünglich um eine Richtungsangabe handelte (Objekt hin- oder wegschieben, INS bzw. REM).

	Besonderheiten: • *CH_ADD, CH_DEL* – Der Standardkontext ist der des Vaterobjekts. Als Nebenkontext ist der des Kindes verfügbar. Etwaige Bedingungen können das auslösende Kind durch eine Abfrage von *_XChildID* auswerten.
Aktion (action)	Diese Spalte definiert die auszuführende Aktion. Die folgenden Aktionen sind möglich: *SET_PROP*, *ADD_CHILD, DEL_CHILD*, *CH_PROP, CON_PROP, CON_CH_PROP, CON_AP* (siehe oben)
Parameter 1 (param_1)	• *SET_PROP, CON_PROP* - Hier enthält diese Spalte einen Merkmalsnamen. **Achtung:** Es sind nur Merkmale erlaubt, welche im Rahmen von Metatypen definiert wurden. Somit sind native Merkmale hier nicht erlaubt. Bsp.: *GConcat* • *ADD_CHILD, DEL_CHILD* - Im Fall *ADD_CHILD/DEL_CHILD* enthält diese Spalte einen Namen, unter dem die hierdurch beschriebenen Kindobjekte identifiziert werden können. Dieser Name stellt die Verbindung zwischen zusammengehörigen *ADD_CHILD*- und *DEL_CHILD*-Einträgen her. Falls über einen solchen Namen mehrere Kinder zusammengefasst werden, so werden diese simultan behandelt (also erzeugt und gelöscht). **Achtung**: Dieser Name darf nicht mit solchen aus der *go_childprops* identisch sein. Bsp.: *LINK2* • *CON_AP* – Es wird der Anfügepunkt angeben, welcher bezüglich anzupassender Geschwisterobjekte bearbeitet werden soll. Bsp.: *APR*

Parameter 2 (param_2)	• *SET_PROP, CH_PROP* – Hier enthält diese Spalte einen Merkmalswert – entweder ein direkter Wert oder ein Ausdruck welcher einen Wert liefert – z.B. diesen aus einem anderen Merkmal ableitet. **Achtung:** Im Gegensatz zu anderen Tabellen oder Spalten muss symbolischen Werten hier ein ‚@'-Zeichen vorangestellt werden! Bsp.: *both* (*SET_PROP, CH_PROP*)
	• *ADD_CHILD* – Die Spalte enthält eine Referenz auf einen Eintrag in der Tabelle *go_children*. Bsp.: *link2id*
	• *DEL_CHILD* – Eventuelle Einträge werden ignoriert.
	• *CON_*PROP* – Hier wird der Wert durch additive Verknüpfung der folgenden Werte angegeben:
	  • 1 – Editierbarkeit
	  • 2 – Sichtbarkeit
	Bsp.: Der Wert 3 entspricht dem Standardwert. Das Merkmal ist sichtbar und editierbar. Der Wert 2 entspricht einem nicht editierbaren aber sichtbaren Merkmals. Die Werte 0 und 1 verbergen das Merkmal.
	• *CON_AP* – Es wird der korrespondierende Anfügepunkt angeben. Bsp.: *APL*

Text (text)	Diese Spalte enthält (optional) einen Textschlüssel (bezugnehmend auf die externe Ressourcen-Datei(en). Bsp.: *concatLeft*

go_attpt

Die Tabelle *go_attpt* definiert Anfügepunkte für die Implementierung der o.g. Aktionen. **Achtung:** Existieren für eine Typ-ID keine Anfügepunkte, so werden automatisch die des eingekapselten Objekts verwendet.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Schlüssel (key)	Der Schlüssel kennzeichnet den zu beschreibenden Anfügepunkt eindeutig in Bezug auf die Typ-ID. Beim Schlüsselnamen ist ein Herstellerbezug sinnvoll, aber nicht zwangsweise erforderlich. Anfügepunkte werden ebenfalls verwendet, um die Positionierung eines Metatypes als Unterposition an einem anderen Metatyp zu steuern. Dies erfolgt einerseits durch einen entsprechenden *CH*-Anfügepunkt des Vaters. Parallel kann der Ursprung des anzufügenden Kindes definiert werden. Dies erfolgt über den Schlüssel *O*. Falls O nicht existiert, wird nach *OL* oder *OR* gesucht, je nachdem, ob sich der CH-Anfügepunkt in der linken oder rechten Hälfte des Vaterobjekts befindet. Ist weder *O*, noch *OL* bzw. *OR* und somit kein lokaler Ursprung angegeben, wird (0, 0, 0) verwendet. 1. Verschiebung durch den Vater an die gewünschte Position. 2. Verdrehung durch den Vater um das Objekt auszurichten, z.B. an einer Tischkante 3. Verschiebung entsprechend *O*/*OL*/*OR* 4. Verdrehung entsprechend *O*/*OL*/*OR* Weiterhin ist der Schlüssel *_GO_CHILD* vordefiniert. Die Beschreibung hierfür findet sich bei der Variablen *GAlign*, Modus *ATTPT*. Bsp.: *hevl*
Planungsrichtung bzw. Verwendung (direction)	Die Planungsrichtung kann verwendet werden, um den Anfügepunkt als Platzierungshilfe zu nutzen. Hierbei erfolgt bei der initialen Platzierung eine Auswertung der Planungsrichtung und des zugehörigen Anfügepunktes des Objekts, an welches platziert wird. Es sind die folgenden Werte erlaubt: • L – Wird bei Planungsrichtung links->rechts (nach rechts) verwendet. Korrespondierender Anfügepunkt des Nachbars: R • R – Wird bei Planungsrichtung rechts->links (nach links) verwendet. Korrespondierender Anfügepunkt des Nachbars: L • F – Wird bei Planungsrichtung vorn->hinten (nach hinten) verwendet. Korrespondierender Anfügepunkt des Nachbars: B • B – Wird bei Planungsrichtung hinten->vorn (nach vorn) verwendet. Korrespondierender Anfügepunkt des Nachbars: F • T – Wird bei Planungsrichtung unten->oben (nach oben) verwendet. Korrespondierender Anfügepunkt des Nachbars: D • D – Wird bei Planungsrichtung oben->unten (nach unten) verwendet. Korrespondierender Anfügepunkt des Nachbars: T • CH – Es handelt sich um einen interaktiven Anfügepunkt für Kindobjekte. Die hierfür möglichen Kinder werden anhand einer Suche nach dem Anfügepunktschlüssel in der Tabelle *go_children* ermittelt. • CH_REL – Es handelt sich um einen interaktiven Anfügepunkt für Kindobjekte, der sich automatisch beim Bewegen eines verbundenen GO-Typen mitbewegt. Die hierfür möglichen Kinder werden anhand einer Suche nach dem Anfügepunktschlüssel in der Tabelle *go_children* ermittelt. Der GO-Typ muss dabei die Kategorie *GO_AP* bereitstellen und über den Parameter *mAP* den Schlüssel dieses Anfügepunktes liefern. • MP – An diesem Anfügepunkt kann ein Metaplanning-Workflow angeplant werden. Soll der Anfügepunkt nicht wie oben beschrieben verwendet werden, bleibt das Feld leer. Es wird stets der erste gültige Anfügepunkt entsprechend der aktuellen Planungsrichtung verwendet.

	In allen Modi außer CH und CH_REL erfolgt ein Anfügen auf Nachbarschaftsebene. Lediglich im Fall CH / CH_REL erfolgt ein Anfügen auf Kindebene. **Hinweis:** Die Verbindung der Anfügepunkte L-R und F-B stellt lediglich eine Planungshilfe dar. Da diese Punkte serien- und herstellerübergreifend existieren, kann keine Planungsprüfung hinterlegt werden. Mit anderen Worten, an jeden R-Punkt kann jedes beliebige Objekt angefügt werden, solange es einen gültigen L-Punkt besitzt und an der entsprechenden Position nicht kollidiert. Selbiges gilt natürlich für die anderen Kombinationen. Dass Anfügepunkte korrespondieren, wird in der Tabelle *go_actions* über die Modi AP, INS oder REM festgelegt. **Anmerkung:** Die vertikale Planung hat gegenüber den horizontalen eine geringere Priorität, falls mehrere Anfügepunktpaare gültig sind. Im Kontext der Kopieranfügepunkte (s.u.) gibt es außerdem die Richtungen I_L, I_R, I_F, I_B, I_T und I_D.
Bedingung (condition)	Sofern eine Bedingung (s.o.) angegeben wurde, wird sie ausgewertet. Ist sie leer oder ergibt sie 1, existiert der Anfügepunkt; sonst nicht. Hier existiert nur der Standardkontext.
X-Position (pos_x) ; Y-Position (pos_y) ; Z-Position (pos_z)	Diese Spalten enthalten die lokalen Koordinaten der Position des Anfügepunktes in Metern. Hierbei kann die Variantenspezifikation eingesetzt werden. Dabei sind alle numerischen Merkmale aus der Tabelle *go_types* erlaubt. Bsp.: *0; 0.72, 0*
Y-Rotation (rot_y)	Der Eintrag gibt die Verdrehung um die Y-Achse relativ zum übergeordneten Objekt in Grad an. Hierbei kann die Variantenspezifikation eingesetzt werden. Dabei sind alle numerischen Merkmale aus der Tabelle *go_types* erlaubt. Bsp.: *90*

Eine Sonderanwendung der Anfügepunkte implementiert das Kopieren von Objekten. Solche Anfügepunkte werden über die Richtung definiert, welche auf folgende Weise gebildet wird:

C_<Dir><DirMode><HierMode><CloneMode>

mit folgender Bedeutung:

- *C_* – konstanter Präfix
- *<Dir>* – gibt die Richtung an und kann die Werte L, R, F, B, T, D annehmen
- *<DirMode>* – gibt einen speziellen Richtungsmodus an, welcher die folgenden Werte annehmen kann:
  - N – Normal. Die Anfügepunktpaare sind L-R, F-B und T-D.
  - I – Invers. Die Anfügepunktpaare sind L-I_L, R-I_R, F-I_F, B-I_B, T-I_T und D-I_D.
- *<HierMode>* – gibt die Hierarchie des zu erzeugenden Objekts an und kann die Werte S (Geschwister) oder C (Kind) annehmen.
- *<CloneMode>* – beschreibt die Art des Kopierens und kann die folgenden Werte annehmen:
  - B – Clonen auf Basis der Grundartikelnummer
  - F – Clonen auf Basis der Endartikelnummer
  - R – rekursives Clonen incl. Kind(eskinder)

Bsp.: *C_RNSR* – Das Objekt wird ‚normal' zur rechten Seite rekursiv geclont. Der Clone befindet sich auf der selben hierarchischen Ebene wie das Original; die beiden Objekte sind also Geschwister.

Für Kopieranfügepunkte sollte einheitlich und exklusiv die Geometrie ::ofml::go::GoIGeometryCubes verwendet werden.

go_attptgeo

Die Tabelle *go_attptgeo* wird verwendet, um die vordefinierte geometrische Repräsentation von Anfügepunkten zu ersetzen.

Schlüssel (key)	Der Schlüssel referenziert einen Anfügepunkt aus der Tabelle *go_attpt*.
Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Wird dieser Eintrag leer gelassen, dann gilt er für alle Metatypen mit dem entsprechenden Anfügepunkt. Bsp.: *tisch1*
X-Position (pos_x) ; Y-Position (pos_y) ; Z-Position (pos_z)	Diese Spalten enthalten einen lokalen Offset der Geometrie in Bezug die nominalen Koordinaten des Anfügepunktes in Metern. Bsp.: *0; 0.72;0*
Rotationsrichtung (rot_dir)	Die Geometrie kann gegenüber dem nominalen Anfügepunkt um eine Achse verdreht werden. Diese Achse wird hier angegeben. Möglich sind: • N – Es findet keine Verdrehung statt. • X – Verdrehung um die X-Achse. • Y – Verdrehung um die Y-Achse. • Z – Verdrehung um die Z-Achse.
Rotation (rot)	Der Eintrag gibt die Verdrehung um die vorgegebene Achse relativ zum nominalen Anfügepunkt in Grad an. Bsp.: *90*
OFML-Typ (type)	Hier wird ein vollqualifizierter OFML-Typ angegeben, der dann für die Darstellung der Geometrie verwendet wird. Dies kann ein vordefinierter Typ sein (siehe Seite 34) oder ein nutzerdefinierter. Bsp.: *::ofml::go::GoIGeometryHArrow*
Parameter 1 (arg1); Parameter 2 (arg2); Parameter 3 (arg3)	Maximal drei Parameter können für die Parametrisierung der Geometrie angegeben werden. Die Interpretation hängt vom jeweiligen Typ ab.

go_attptsorder

Die Tabelle *go_attptsorder* wird verwendet, um die Arbarbeitungsreihenfolge der Anfügepunkte zu beeinflussen.

Schlüssel (key)	Der Schlüssel referenziert einen Anfügepunkt aus der Tabelle *go_attpt*.
Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Wird dieser Eintrag leer gelassen, dann gilt er für alle Metatypen mit dem entsprechenden Anfügepunkt. Bsp.: *tisch1*
Planungsrichtung (plan_dir)	Diese Spalte enthält die globale Planungsrichtung, für die diese Abarbeitungsreihenfolge gültig ist. Bsp.: *R*
Ordnungszahl (number)	Die Ordnungszahl für diesen Anfügepunkt. Bsp.: *1*

go_childmoving

Die Tabelle *go_childmoving* definiert die Bewegbarkeit eines Kindobjekts in Bezug auf seine Verschiebbarkeit (Translation) und Drehbarkeit (Rotation). Ist für ein Objekt keine Translation angegeben, dann ist es nicht verschiebbar; ist für ein Objekt keine Rotation angegeben, dann ist es nicht drehbar. Das Kindobjekt muss ein Metatyp sein.

Weiterhin kann die Tabelle verwendet werden, um die indirekte Verschiebung von Metatyp-Kindern, hervorgerufen durch Bewegungen anderer Kinder, zu steuern. Dieser Modus wird nachfolgend als **I-Modus** bezeichnet.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. **Achtung:** Hierbei wird der Metatyp angegeben, dem der Vater des Objekts angehört. Bsp.: *tisch1*
Schlüssel (key)	Der Schlüssel kennzeichnet das Kind entsprechend dem für seine Erzeugung verwendeten Anfügepunkt (*go_attpt[key]*) **I-Modus**: Falls das Kind ein Metatyp ist, wird hier die Verkettung (per Unterstrich) von Kind-ID (im Kontext des Vaters) und Metatype-ID des Kindes angegeben. Andernfalls wird entweder im Fall von interaktiven Kindern die Basisartikelnummer oder im Fall von merkmalsgesteuerten Kindern der Merkmalsname zzgl. Kindzähler (beginnend ab 1) angegeben.
Bedingung (condition)	Sofern eine Bedingung (s.o.) angegeben wurde, wird sie ausgewertet. Ist sie leer oder ergibt sie 1, wird die Zeile weiter ausgewertet; sonst nicht. Die Bedingung wird im Kontext des Kindobjektes evaluiert. **I-Modus**: Die Bedingung wird im Kontext des Vaters ausgewertet. Ist das Kind ein Metatyp, so wird dessen Kontext außerdem als sekundärer Kontext bereitgestellt.
Modus (*mode*)	Der Modus gibt die Art der Bewegung an, für die das nachfolgende Kommando gilt. Unterstützt werden: • *XT* – Verschiebung entlang X-Achse • *YT* – Verschiebung entlang Y-Achse • *ZT* – Verschiebung entlang Z-Achse • *XZTLINEAR* – lineare Verschiebung in der X-Z-Ebene • *XZTRANGE* – Flächenverschiebung in der X-Z-Ebene • *XR* – Rotation um die X-Achse • *YR* – Rotation um die Y-Achse • *ZR* – Rotation um die Z-Achse Die elementaren Translationen *XT*, *YT* und *ZT* können frei kombiniert werden. *XZTLIN* und *XZTRANGE* dürfen nicht mit anderen Translationen, die die X- oder Z-Achse betreffen, kombiniert werden. Es darf nur jeweils eine der drei Rotationen *XR*, *YR* und *ZR* verwendet werden. **I-Modus**: Der Modus steuert die Freischaltung der Achsen für die Nachführung der Positionsveränderung. Es wird der erste gültige (im Sinn einer gültigen Bedingung) Eintrag verwendet, alle anderen ignoriert. Falls kein gültiger Eintrag existiert, sind alle Achsen freigeschaltet. • *IN* – Es werden keine Achsen freigeschaltet. • *IX, IY, IZ, IXY, IXZ, IYZ, IXYZ* – Die jeweiligen Achsen werden freigeschaltet (X, Y, Z, XY, XZ, YZ, XYZ).

	Zusätzlich kann über den Modus *GO_AP_POSROT* die Position / Rotation von GO-Typen innerhalb des Objektbaumes abgefragt werden. Der GO-Typ muss dabei die Kategorie GO_AP bereitstellen und über den Parameter *mAP* einen Schlüssel zur Identifizierung liefern. In allen Metatyptabellen steht dann die Methode *GO_AP_POSROT(pAPKey, pMode)* zur Verfügung. *pAPKey* ist dabei der Schlüssel, der dem Parameter *mAP* des GO-Typen enspricht. *pMode* muss einer dieser Werte sein: • *@POS_X* • *@POS_Y* • *@POS_Z* • *@ROT_X* • *@ROT_Y* • *@ROT_Z* `GO_AP_POSROT(@MF_RW, @POS_Y)` liefert z.B. die Y-Position des Knotens mit der ID @MF_RW.
Kommando (command)	Das Kommando dient zur Festlegung von Parametern für die jeweilige Bewegung. Bei den Translationsbewegungen erfolgt die Angabe in Metern; bei den Rotationsbewegungen in Grad. Möglich sind für die Modi: *XT*, *YT* , *ZT* und *YR*: • *MIN* – Minimale Position. Bei Unterschreiten wird auf diesen Wert zurückgesetzt (auch wenn dieser außerhalb eines gewählten Rasters oder einer erlaubten Position liegt). • *MAX* – Maximale Position. Bei Überschreiten wird auf diesen Wert zurückgesetzt (auch wenn dieser außerhalb eines gewählten Rasters oder einer erlaubten Position liegt). • *RASTER* – Rasterisierung der Bewegung. Es wird die nächste Rasterposition im Rahmen des gültigen Bereichs ermittelt (oder eine der Bereichsgrenzen selbst). • *RAS_OFFS* – Offset zur Rasterisierung der Bewegung. • *POS* – Explizite Position. Über diesen Eintrag bzw. mehrere dieser Einträge können eine oder mehrere Positionen angegeben werden. Es wird die nächste Position im Rahmen des gültigen Bereichs ermittelt (oder eine der Bereichsgrenzen selbst). *POS* und *RASTER* dürfen nicht gleichzeitig für eine Achse verwendet werden. Wird kein Kommando angegeben, so wird dadurch die freie Bewegbarkeit bzgl. der angegebenen Achse eingeschaltet, sofern kein weiteres Kommando für diese Achse folgt. Für den Modus *XZTLINEAR* müssen die folgenden Kommandos aufgerufen werden: • *X1* – X-Position des ersten Punktes der Bewegungsgeraden • *Z1* – Z-Position des ersten Punktes der Bewegungsgeraden • *X2* – X-Position des zweiten Punktes der Bewegungsgeraden • *Z2* – Z-Position des zweiten Punktes der Bewegungsgeraden Hierbei dürfen X1 und X2 nicht identisch sein. Gleiches gilt für Z1 und Z2. Für den Modus *XZTRANGE* müssen die folgenden Kommandos aufgerufen werden: • *X1* – X-Position des ersten Punktes des Bewegungsbereichs • *Z1* – Z-Position des ersten Punktes des Bewegungsbereichs • *X2* – X-Position des zweiten Punktes des Bewegungsbereichs • *Z2* – Z-Position des zweiten Punktes der Bewegungsbereichs **I-Modus**: Nicht verwendet.

Parameter (*parameter*)	Der Parameter bezieht sich auf das angegebene Kommando und ist ein Wert in Meter bei Translationen bzw. ein Grad bei Rotationen. Es kann die übliche Variantenspezifikation angewendet werden. Es ist zu beachten, dass die Variantenspezifikation den Kontext des Kindobjektes und den des Vaterobjekts berücksichtigt. Um Namenskonflikte zu vermeiden, erhalten die Parameter des Vaters den Präfix „_”. **I-Modus**: Nicht verwendet.

go_freenumeric

Die Tabelle *go_freenumeric* parametrisiert die freien Merkmale. Um eine optimale Wiederverwendung der freien Merkmale zu ermöglichen, wird auf eine Zuordnung zu den Metatypen in der Tabelle *go_types* verzichtet.

Name (name)	Diese Spalte benennt den Namen des freien Merkmals. Bsp.: *GDeskPos*
Format (format)	Die folgenden Formate aus der Menge der OFML-Merkmalsformate werden unterstützt: • *i* – Integer • *f* – Gleitkomma • *L* – wie *f* jedoch Anzeige als Länge mit variabler Maßeinheit • *A* – wie *f* jedoch Anzeige als Winkel mit variabler Maßeinheit
Minimum (minimum)	Sofern dieses Feld nicht leer ist, definiert es das Minimum des Wertebereiches. Bsp.: *1.0*
Maximum (maximum)	Sofern dieses Feld nicht leer ist, definiert es das Maximum des Wertebereiches. Bsp.: *2.0*
Raster (raster)	Sofern dieses Feld nicht leer ist, definiert es das Raster des Wertebereiches. Bezugspunkt des Rasters ist der Minimumwert – falls definiert – oder ansonsten das lokale Koordinatensystem des Metatypes. Es gilt zu beachten, dass der Minimum- und der Maximum-Wert eine höhere Priorität als das Raster haben. D.h., eine Verletzung des Rasters ist möglich, wenn dies zur Einhaltung der Wertebereichsgrenzen erforderlich ist. Bsp.: *0.1*
Ausdruck (expr)	Sofern dieses Feld nicht leer ist, kann ein Ausdruck angegeben werden, um aus dem aktuellen Wert des Merkmals einen anderen numerischen Wert abzuleiten, z.B. durch Addieren eines Offsets. Ist der Ausdruck leer, wird der Merkmalswert als Ergebnis des Ausdrucks geliefert. Im Rahmen dieses Ausdrucks wird der aktuelle Wert des Merkmals mit *V* referenziert. Für den Ausdruck sind numerische Ausdrücke entsprechend ANSI C anzugeben. In diesem Ausdruck sind im Gegensatz zu den sonst üblichen Bedingungen oder Ausdrücken keine anderen Parameterwerte verfügbar. Eingangswert für den Ausdruck ist der Wert nach etwaiger Rasterisierung und Bereichskorrektur. Bsp.: *0.5*V*
Kind (child)	Sofern dieses Feld nicht leer ist, gibt es ein Kindobjekt an, auf das das Ergebnis des Ausdrucks angewendet werden soll. Bsp.: *e1.goplate.e1*
Modus (mode)	Falls das Feld child nicht leer ist, muss dieses Feld angeben, auf welche Weise das Ergebnis des Ausdrucks angewendet werden soll: • POS_X – Setze die lokale X-Position des angegebenen Kindes • POS_Y – Setze die lokale Y-Position des angegebenen Kindes • POS_Z – Setze die lokale Z-Position des angegebenen Kindes Bsp.: *POS_X*

Achtung: Beim Setzen des initialen Wertes findet keinerlei weitere Funktionalität statt. Es werden dabei weder die Bereichsgrenzen noch das Raster überprüft. Aus technischen Gründen ist es weiterhin nicht möglich, die initiale Wertsetzung an Kindobjekte weiterzuleiten.

go_metainfo

Die Tabelle *go_metainfo* definiert zusätzliche Informationen welche für die Metatypen hinterlegt werden können. Diese Zusatzinformationen dienen zur Anbindung von Algorithmen wie der automatischen Platzierung von Zubehör.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Modus (mode)	Dieser Eintrag gibt den Modus an für den er gültig ist. Die folgenden Modi werden unterstützt: • *AutoDecoration* – automatische Platzierung von Zubehör • *AccCategory*[3] – Zubehörkategorien – Definition oder Unterstützung
Breite (width)	Dieses Feld gibt die Bezugsbreite für den Algorithmus in Metern an. Hierbei sind die üblichen Ausdrücke möglich. Ergebnis muss ein numerischer Wert sein. Bsp.: 1.0 Für *AccCategory* wird dieses Feld ignoriert.
Höhe (height)	Dieses Feld gibt die Bezugshöhe für den Algorithmus in Metern an. Bsp.: 1.0 Für *AccCategory* wird dieses Feld ignoriert.
Tiefe (depth)	Dieses Feld gibt die Bezugstiefe für den Algorithmus in Metern an. Bsp.: 1.0 Für *AccCategory* wird dieses Feld ignoriert.
Bedingung (condition)	Sofern eine Bedingung (s.o.) angegeben wurde, wird sie ausgewertet. Ist sie leer oder ergibt sie 1, wird die Zeile weiter ausgewertet; sonst nicht. Für *AccCategory* wird dieses Feld ignoriert.

Wert 1 (value_1)	Dieses Feld gibt einen Parameter an, dessen Existenz wie auch Interpretation abhängig vom jeweiligen Algorithmus sind. Die folgenden Modi werden unterstützt:
	• *AutoDecoration* – Angabe einer semantischen Domain, z.B. *LivingRoom*, *HomeOffice*
	• *AccCategory:*
	  ○ *child* – Der Metatyp wird dieser Kategorie zugeordnet.
	  ○ *parent* – Der Metatyp akzeptiert Zubehörobjekte die dieser Kategorie angehören.

Wert 2 (value_2)	Dieses Feld gibt einen weiteren Parameter an, dessen Existenz wie auch Interpretation abhängig vom jeweiligen Algorithmus sind. Die folgenden Modi werden unterstützt: • *AutoDecoration* – Angabe einer Template-Identifikation für die oben angegebene semantische Domain, z.B. *living::table::standard* oder *office::desk::std* • *AccCategory:* • *TOP_ELEM* – Platzierung oberhalb eines Objekts. Hinweis: *TOP_ELEM* ist für *parent* vordefiniert.

Die *AutoDecoration*-Einträge können nicht nur für Metatypen verwendet werden, sondern auch für (Kindes-)Kinder, z.B. innerhalb von ODB-Blöcken. In diesem Fall ist zu beachten:

- Die Zuordnung erfolgt über den Typ *::ofml::go::GoAccParameters*, der folgende Argumente hat:
  - Identifikation (String) – Dies entspricht der Metatyp-Indentifikation.
  - Breite, Höhe, Tiefe (numerische Werte in Meter)
- Für die Berechnung parametrischer Werte und Bedingungen wird analog zu den Metatypen ein Kontext generiert, der die lokalen Variablen *W, H* und *D* bereitstellt, welche den o.g. numerischen Werten entsprechen.

[3] Der alte Identifier *acat* wird für eine Übergangszeit aus Gründen der Kompatibilität weiter unterstützt.

- Sofern in der Tabelle *go_metainfo* kein Wert für Breite (Höhe, Tiefe) hinterlegt ist, wird automatisch *W* (*H, D*) angenommen.

go_feedback

Die Tabelle *go_feedback* definiert gültige Positionen eines einzufügenden Kindes zur Verwendung im interaktiven Einfügemodus.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte, wobei hier der Vater des einzufügenden Objekts einzutragen ist. Bsp.: *tisch1*
Kindartikel-nummer (ch_artnr)	Hier ist die Artikelnummer des zu erzeugenden Kindes anzugeben.
Anfügepunkt (attpt_key)	Dieser Schlüssel kennzeichnet das Kind entsprechend dem für seine Erzeugung verwendeten Anfügepunkt (Tabelle *go_attpt[key]*).
Bedingung (condition)	Sofern eine Bedingung (s.o.) angegeben wurde, wird sie ausgewertet. Ergibt diese 1 (oder ist leer), so wird die Zeile weiter ausgewertet; ansonsten nicht. Die Bedingung wird im Kontext des Vaterobjekts evaluiert.
Modus (mode)	Die verfügbaren Modi entsprechen denen der Tabelle *go_childmoving*. Für die Kommandos *POSX, POSY, POSZ* und *ROTY* (s.u.) muss dieser Eintrag leer bleiben.
Kommando (command)	Das Kommando dient zur Festlegung von Parametern für die jeweilige Bewegung. Die verfügbaren Kommandos entsprechen denen der Tabelle *go_childmoving*. Zusätzlich können für jede Kombination aus Typ-ID, Kindartikelnummer und Anfügepunkt die folgenden Kommandos angegeben werden: • *POSX* – X-Position des lokalen Ursprungs • *POSY* – Y-Position des lokalen Ursprungs • *POSZ* – Z-Position des lokalen Ursprungs • *ROTY* – Y-Rotation des lokalen Ursprungs Wird hierfür kein Wert angegeben, wird 0.0 angenommen.
Parameter (parameter)	Der Parameter bezieht sich auf das angegebene Kommando und ist ein Wert in Meter bei Translationen bzw. in Grad bei Rotationen. Es kann die übliche Variantenspezifikation angewendet werden. Es ist zu beachten, dass die Variantenspezifikation den Kontext des Vaterobjekts berücksichtigt. Der Kontext des Kindobjektes ist nicht verfügbar.

go_classes

Die Tabelle *go_classes* ordnet einem in der Tabelle *go_types* definiertem Metatyp eine OFML-Klasse zu. Diese muss von *::ofml::go::GoMetaType* abgeleitet sein. Diese Zuordnung wird nur in speziellen Fällen benötigt und ist daher optional.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Über den Platzhalter * an dieser Stelle wird ggf. die Standardklasse festgelegt. Bsp.: *tisch1*
OFML-Klasse (class)	Hier wird die zu verwendende OFML-Klasse voll referenziert angegeben. Bsp.: *::ofml::go::GoTisch1*

go_propclasses

Die Tabelle *go_propclasses* ordnet einem in der Tabelle *go_types* definiertem Merkmal eine Merkmalsklasse zu. Dadurch ist eine Strukturierung der Merkmale möglich. Diese Zuordnung ist optional. Erfolgt keine Zuordnung eines Merkmals über diese Tabelle, gehört dieses Merkmal zur Klasse *MetaProperties („Meta-Merkmale“)*.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Dieser Schlüssel ist optional, wird hier nichts angegeben, gilt diese Zuordnung für alle Metatypen. Bsp.: *tisch1*
Property-Name (prop_name)	Der Schlüssel referenziert ein Merkmal aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *GWidth*
Merkmalsklasse (prop_class)	Hier wird die zu verwendende Merkmalsklasse angegeben. Es wird ein Bezeichner ohne ‚@‘ angegeben. Der Text dieses Bezeichners muss in der/den spezifischen Ressourcen-Datei(en) definiert sein. Bsp.: *Dimensions*

go_setup

Über die Tabelle *go_setup* können für einen in der Tabelle *go_types* definierten Metatyp spezielle Eigenschaften gesteuert werden. Diese Tabelle ersetzt die speziellen Merkmale *GSetup* und *GXSetup*. Bis auf weiteres können beide Konzepte gleichberechtigt verwendet werden.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Eigenschaft (key)	Hier wird der Name der Eigenschaft angegeben. Eine Liste mit allen Eigenschaften siehe unten. Bsp.: *NoMTOrderRep*
Wert (value)	Hier ist es möglich, die Eigenschaft auf einen bestimmten Wert zu setzen. Dies ist optional; für die meisten Eigenschaften gibt es nur die Werte „gewählt“ (*1*) und „nicht gewählt“ (*0*). Bsp.: *1*

Die folgenden Eigenschaften sind definiert:

- *NoMTOrderRep* – Dieses Flag steuert, ob der Metatyp als eigener Knoten in der Bestellliste erscheint. Ist das Flag gesetzt, dann wird keine separate Repräsentation des Metatypes erzeugt. Entspricht GSetup & 1
- *ChildOrderRep* – Dieses Flag steuert, ob die weiteren Kinder (d.h. alle außer dem Hauptkind) Unterpositionen des Hauptkindes sein sollen. Ist das Flag nicht gesetzt, werden die Kinder auf derselben Ebene wie das Hauptkind in der Bestellliste angezeigt. Andernfalls werden sie zu Unterpositionen des Hauptkindes. Diese Einstellung, welche global für alle Kinder (außer dem Hauptkind) gilt, kann durch *ChildSubPos* für ein konkretes Kind überschrieben werden. Entspricht GSetup & 2
- *Gen2DSymbol* – Es werden 2D-Symbole generiert. Dies sollte nur angewendet werden, falls keine 2D-Symbole explizit hinterlegt wurden. Entspricht GSetup & 4

- *UseStdAttPts* – Es werden die Standardanfügepunkte verwendet. Dies kann benutzt werden, falls keine Anfügepunkte im Rahmen der Metatypen definiert werden sollen. Entspricht GSetup & 8
- *HideOrderNo* – Die per Read-Only-Property automatisch angezeigte Artikelnummer des Hauptkindes kann über dieses Bit ausgeblendet werden. Entspricht GSetup & 16
- *HideFilterMsg* – Falls dieses Bit gesetzt ist, wird die Auflistung der Merkmale, welche sich aufgrund der Filter automatisch angepasst haben, unterdrückt. Entspricht GSetup & 32
- *CollCheck* – Dieses Bit steuert die Kollissionserkennung für Kindobjekte während der initialen Platzierung und auch während der interaktiven Bewegung. Ist das Bit nicht gesetzt, erfolgt bei der initialen Platzierung lediglich ein Test, ob sich exakt an der Einfügestelle bereits ein Objekt befindet; während der interaktiven Bewegung erfolgt keine Kollisionsprüfung. Ist das Bit gesetzt, erfolgt in beiden Fällen die OFML-typische Kollisionsprüfung. Entspricht GSetup & 64
- *InheritSelObj* – Metatypobjekte erben native Merkmale von einem Einfügeobjekt. Für den Fall, dass das Metatypobjekt wiederum Kind eines Metatypobjekts ist, kann über dieses Flag gesteuert werden, ob der Vater als Einfügeobjekt verwendet wird oder ein selektiertes Objekt auf gleicher Ebene wie das neu zu erzeugende. Ist das Flag gesetzt und existiert ein solches selektiertes Objekt, dann wird dieses verwendet, um die Eigenschaften zu erben. Andernfalls werden die Eigenschaften vom Vater geerbt. Entspricht GSetup & 128
- *DelChildrenAP* – Das automatische Löschen von interaktiv geplanten Anbauteilen, für welche nach einer Konfiguration des Vaterobjekts kein gültiger Anfügepunkt mehr existiert, kann über dieses Flag eingeschaltet werden. Ist das Flag gesetzt, erfolgt im beschriebenen Fall eine Abfrage und – falls diese mit 'Ja' beantwortet wird – das Löschen des Kindes. Ist dieses Flag nicht gesetzt, erfolgt keine Auswertung dieser Situation. Hierbei wird der go_setup-Wert des betreffenden Kindes ausgewertet. Entspricht GSetup & 256
- *HideSideBySideMsg* – Standardmäßig erscheint beim Aneinderplanen von Objekten, welche zwar planungsrichtungs-kompatible Anfügepunkte besitzen, jedoch ein Anfügepunktpaar nicht durch mindestens 1 gemeinsame Aktion als zusammengehörig kennzeichnen, ein Hinweis, dass keine Anschlußplanung im Sinne der Regelprüfung erfolgt. Setzt man dieses Flag, so wird die Meldung unterdrückt. Ausschlaggebend ist der Wert des Flags am anzufügenden Objekt. Entspricht GSetup & 512
- *InitTmpChild* – Über dieses Flag kann die vollständige Instanziierung des Hauptkindes während der temporären Erzeugung eingeschaltet werden. Dies ist beispielsweise erforderlich wenn auf präzise Bounding-Boxes oder spezifische Merkmale (z.B. *GVarPrefix*) zugegriffen werden soll. Entspricht GSetup & 1024
- *NoInheritNA* – Ist dieses Flag gesetzt, dann erbt das Objekt keine nativen Merkmale für den Fall, dass sein Vorfahr ein Metatyp ist. Entspricht GSetup & 2048
- *NoInitCollCheck* – Ist dieses Flag gesetzt, so wird für die initiale Platzierung auf Geschwisterebene während der Überprüfung der Anfügepunkte die Kollisionserkennung für dieses Objekt ausgeschaltet. Dies kann erforderlich sein, wenn erst durch die Anpassung im Rahmen von Aktionen die Kollision beseitigt wird. Es ist dadurch aber auch möglich mehrere Objekte ineinanderzuplazieren, wodurch fehlerhafte Planungen entstehen können. Entspricht GSetup & 4096
- *ChildSubPos* – Dieses Flag erzwingt für einen Metatyp M1, welcher Kind eines anderen Metatyps M2 ist, das M1 in M2 als Hauptposition geführt wird, auch wenn M2 für seine Kinder mittels *ChildOrderRep* definiert hat, das diese Unterpositionen sind. Entspricht GSetup & 8192
- *DelChildrenPropsChange* – Dieses Flag erzwingt das Löschen und Neuerzeugen der eigenen Kinder bei Property-Änderungen. Ist das Flag nicht gesetzt, werden diese Kinder nur repositioniert. Das Löschen und Neuerzeugen kann in speziellen Fällen nützlich sein. Dieses Verhalten war bis einschließlich GO 1.3.9 das Standardverhalten. Entspricht GSetup & 16384
- *PropClassNA* – Dieses Flag stellt das bisherige Verhalten wieder her, wonach na-Merkmale im Merkmalseditor der Merkmalsklasse des Metatypes zugeordnet werden. Andernfalls bekommt das na-Merkmal die Merkmalsklasse des zugehörigen nativen Merkmals und wird somit auch in diesem Bereich im Merkmalseditor dargestellt. Entspricht GSetup & 32768

- *CollCheckTranslate* – Dieses Flag steuert, ob bei Translation eines Metatypes als Kind eines anderen Metatyps M, eine Kollisionserkennung unter den anderen Kindern von M stattfinden soll (1) oder nicht (0). Entspricht GXSetup & 1
- *CollCheckRotate* – Dieses Flag steuert, ob bei Rotation eines Metatypes als Kind eines anderen Metatypes M, eine Kollisionserkennung unter den anderen Kindern von M stattfinden soll (1) oder nicht (0). Entspricht GXSetup & 2
- *NoCollCheckMC* – Sofern das Hauptkind des Metatypes von der Kollisionserkennung in den Modi *CollCheckTranslate* und *CollCheckRotate* ausgeschlossen werden soll, muss dieses Flag gesetzt werden. Andernfalls wird das Hauptkind bei der Kollisionserkennung berücksichtigt. Entspricht GXSetup & 4
- *Feedback3D* – Der interaktive Feedback-Modus (verfügbar ab Version 1.11) wird freigeschaltet (1) oder nicht bzw. wird nicht unterstützt (0). Entspricht GXSetup & 8
- *CollCheckMC* – Flag zur Steuerung des Kollisionsverhaltens des gekapselten Objektes
  0 -> normale Kollision
  1 -> gar keine Kollision
  2 -> kein Mittelpunkttest
- *HideDelChildrenAPMsg* – Steuert Abfrage beim automatischen Löschen von interaktiv geplanten Anbauteilen (siehe *DelChildrenAP*):
  0 -> Abfrage bei jedem Kind
  1 -> Zusammenfassung (es wurden n Kinder gelöscht)
  2 -> gar keine Meldung
- *ShowPolyPropFilterMsg* – Die Anpassung des Merkmals infolge eines Filterprozesses wird in einer Dialogbox angezeigt. Ist dieses Flag gesetzt, kann die Ausgabe dieser Meldung für bestimmte Merkmale in der Tabelle *go_types* über den Modus 64 des Merkmals deaktiviert werden.
- *DisableAutoChildReposition* – Steuert, ob Kinder bei Positionsänderung automatisch nachgeführt werden:
  0 -> Kinder werden nachgeführt; eine detaillierte Steuerung ist über die I-Modi der Tabelle *go_childmoving* möglich.
  1 -> Kinder werden nie nachgeführt; eventuell gesetzte I-Modi der Tabelle *go_childmoving* ignoriert.
  2 -> Kinder werden nicht nachgeführt; über I-Modi der Tabelle *go_childmoving* kann die Nachführung für bestimmte Fälle aktiviert werden.
- *SumSubArticlePrices* – Über dieses Flag kann gesteuert werden, ob die Preise der Unterpositionen einzeln dargestellt (SumSubArticlePrices == 0) oder aufsummiert werden (SumSubArticlePrices == 1). Zur Berechnung dieses Wertes kann der Zustand eines oder mehrerer Merkmale des Objekts entsprechend OFML-Syntax verwendet werden.
- *UseMCAxis* – Die Translations- / Rotationsachsen werden vom gekapselten Objekt übernommen.

go_texts

Über die Tabelle *go_texts* können für die im Bereich der Metatypen verwendeten Ressourcen Texte hinterlegt werden.

Ressourcen-Schlüssel (key)	Der Schlüssel referenziert die Ressource, für die der Text verwendet werden soll. In der Regel handelt es sich hierbei um Merkmalsschlüssel. Bsp.: *GWidth*
Sprache (language)	Hier wird der zweistellige ISO–Sprachen–Code (ISO–639) angegeben. Dieses Feld kann für sprachunabhängige Texte leergelassen werden. Sprachabhängige Texte haben jedoch Vorrang. Bsp.: *de*
Text (text)	Dieses Feld enthält den aufgelösten Text. Bsp.: *Breite*

Mit Hilfe des Schlüssels utf8 in der Tabelle *go_info* kann angegeben werden, dass die Werte in der Spalte *Text* im UTF-8 Zeichensatz kodiert werden. Am Anfang der Datei kann optional das Byte Order Mark angegeben werden. Als Normalform sollte NFC (Normalization Form Canonical Composition) verwendet werden.

Ist dieser Schlüssel nicht gesetzt, werden die Texte in der jeweiligen Codepage des Systems kodiert.

go_symbolicpropvalues

Über die Tabelle *go_symbolicpropvalues* kann einem symbolischen Merkmalswert eines in der Tabelle *go_types* definierten Merkmales ein numerischer Wert zugeordnet werden.

Diese Funktionalität wird benötigt, um eine OAP *DimChange* Aktion mit symbolischen Merkmalswerten durchführen zu können.[4]

Merkmal (key)	Der Eintrag benennt das Merkmal, dessen Merkmalswerte zugeordnet werden sollen. Über den Platzhalter * kann auch für alle Metatypen der Serie ein Verhalten definiert werden. Bsp.: *GCableSet*
Symbolischer Merkmalswert (symbol)	Hier wird der symbolischer Wert des Merkmals, wie er in der Tabelle *go_properties* oder *go_childprops* (jeweils Spalte *value*) definiert wurde, angegeben. Bsp.: *H1*
Numerischer Wert (number)	Hier wird die numerische Entsprechung des Merkmalswertes in Metern angegeben. Bsp.: *0.72*

go_itemplates

Über die Tabelle *go_itemplates* kann einem in der Tabelle *go_types* definierten Metatyp ein ITemplate zugeordnet werden.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Über den Platzhalter * an dieser Stelle wird ggf. das Standard-ITemplate festgelegt. Bsp.: *tisch1*
ITemplate (template)	Hier wird ein vollqualifizierter Name des ITemplates angegeben, das dem Metatypen zugeordnet wird. Dies kann ein vordefiniertes Template sein (siehe Seite 38) oder ein nutzerdefiniertes. Bsp.: *::ofml::go::IT_Desk1*
Bedingung (condition)	Sofern eine Bedingung (s.o.) angegeben wurde, wird sie ausgewertet. Ist sie leer oder ergibt sie 1, existiert das ITemplate; sonst nicht. Hier existiert nur der Standardkontext.
Parameter (parameter)	Mit diesem Parametervektor kann das ITemplate parametrisiert werden. Die Interpretation hängt vom jeweiligen Typ ab. Die einzelnen Parameter sind durch Kommas getrennt anzugeben. Bsp.: *@LB, @RB*

[4] Siehe OAP OFML Aided Planning Version 1.4 – Abschnitt 4.8.4

X-Position (pos_x) ; Y-Position (pos_y) ; Z-Position (pos_z)	Diese Spalten enthalten einen lokalen Offset der Geometrie in Bezug die nominalen Koordinaten des ITemplates in Metern. Bsp.: *0; 0.72; 0*
Y-Rotation (rot_y)	Der Eintrag gibt die Verdrehung um die Y-Achse relativ zum nominalen Anfügepunkt in Grad an. Bsp.: *90*

go_interactors

Die Tabelle *go_interactors* wird verwendet, um an einem Metatypen Applikationsinteraktoren[5] zu definieren.

Typ-ID (id)	Der Schlüssel referenziert einen Metatyp aus der Tabelle *go_types* über die gleichnamige Spalte. Bsp.: *tisch1*
Interaktortyp (type)	• *SELECT* – Für einen Interaktor zur Selektion eines Kindartikels • *ACTION* – Für einen Interaktor zur Auslösung von in der Tabelle *go_actions* definierten Aktionen. • *RESIZE*[6] – Für einen Interaktor zur Auslösung einer interaktiven Größenänderung[7]. • *METHOD* – Für einen Interaktor zum Aufruf einer OFML Methode.

Schlüssel (key)	Der Schlüssel kennzeichnet den zu beschreibenden Interaktor eindeutig in Bezug auf die Typ-ID. Beim Schlüsselnamen ist ein Herstellerbezug sinnvoll, aber nicht zwangsweise erforderlich. Die Bedeutung des Schlüssels ist abhängig vom Interaktortyp:
	• *SELECT* – Hier bezeichnet der Schlüssel den zu selektierenden Kindartikel. Es wird hier ein Eintrag aus der Tabelle *go_children (key)* angegeben, der zur Identifizierung des Kindes dient.
	• *ACTION* – Hier verweist der Schlüssel auf die Tabelle *go_actions (own_key)*. Bei Aktivierung des Interaktors werden alle definierten INTERACTOR-Aktionen mit diesem Schlüssel ausgeführt.
	• *RESIZE* – Hier enthält der Schlüssel einen Vektor mit folgendem Inhalt:
	  • 1. vollqualifizierte Metaplanning-Workflow-ID (String)
	  • 2. vollqualifizierte Metaplanningklasse (String)
	  • 3. Flag, ob während der Interaktion die Artikelgeometrie ausgeblendet werden soll (0 | 1)
	• *METHOD* – Hier beinhaltet der Schlüssel einen Vektor aus:
	  • 1. Methodennamen inklusive Argumenten (String)
	  • 2. Ausführungsmodus (Int) Es sind folgende Werte erlaubt:
	  • 0 – Die Methode wird auf dem Top-Szenenelement ausgeführt.
	  • 1 – Die Methode wird auf dem Target-Objekt der Interaktion ausgeführt.
	  • 2 – Die Methode wird auf dem in der Objekthierarchie ausgehend vom Target-Objekt nächsthöheren selektierbaren Objekt ausgeführt.
	Bsp.: *int_add_child*

[5] Siehe Application Note AN-2013-001 „Application Interactors“
[6] Dieser Typ ist ab GO Version 1.17.3 verfügbar
[7] Siehe GO IV (MP) - OFML Metaplanning – Concepts and Tables

Bedingung (condition)	Die Bedingung bezieht sich auf den Zustand eines oder mehrerer Merkmale des Objekts und wird entsprechend OFML-Syntax geschrieben. Die Bedingung ist gültig, wenn ihr Ergebnis 1 ist bzw. sie leer ist. Dabei sind alle Merkmale aus der Tabelle *go_types* erlaubt. **Achtung:** Im Gegensatz zu anderen Tabellen oder Spalten muss symbolischen Werten hier ein ‚@'-Zeichen vorangestellt werden! Bsp.: • *GWidth==1200* • *(GWidth >= 1000 && GConcat != @LEFT)*
X-Position (pos_x) ; Y-Position (pos_y) ; Z-Position (pos_z)	Diese Spalten enthalten einen lokalen Offset der Position des Interaktors in Bezug die nominalen Koordinaten des Referenzobjektes in Metern. Bsp.: *0; 0.72;0*
Interaktorsymbol (image)	Hier kann für den Interaktor eine spezifische Bilddatei hinterlegt werden. Der Verzeichnispfad ist relativ zum Datenverzeichnis des Herstellers anzugeben. Der Dateiname ist ohne Suffix für den Datei-Typ anzugeben. Über ein Präfix ist der Typ des Interaktorsymbols zu bestimmen: • *@IMAGE* – Als Grafikformat ist PNG vorausgesetzt. Das Bild sollte transparent sein (d.h., einen Alphakanal enthalten). Es werden 2 Dateien für das Symbol erwartet: eine für die normale Darstellung, und eine zur Darstellung als aktiver Interaktor. Die beiden Varianten werden anhand der geforderten Suffixe *_normal* bzw. *_hot* unterschieden (die an den hier in diesem Element angegebenen Namen angehängt werden). Bsp.: *@IMAGE:/basics/ANY/1/mat/interactorChild*
Hinweistext (hint)	Diese Spalte enthält (optional) einen Textschlüssel (bezugnehmend auf die externe Ressourcen-Datei(en). Der Text wird in einem Tooltip angezeigt, sobald sich die Maus über dem Interaktor befindet. Bsp.: *concatLeft*

3 Variantenspezifikation

Die Felder der obigen Tabellen, welche die Bemerkung ‚Variantenspezifikation' beinhalten, können der Variantenspezifikation wie folgt unterzogen werden:

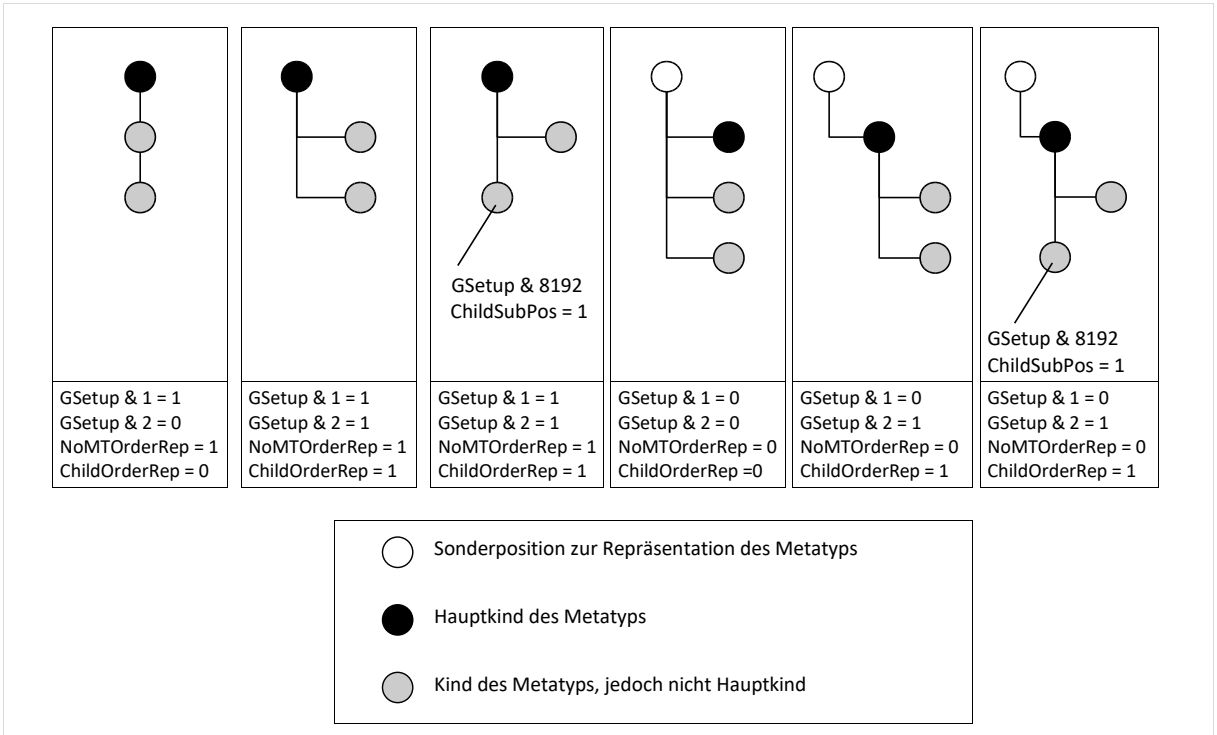
- Es wird ein Ausdruck angegeben, dessen Ergebnis ein numerischer Wert ist.
- Dieser Ausdruck kann die im OFML-Standard definierten mathematischen Konstanten, Funktionen und Operationen, sowie arithmetischen und logischen Operatoren enthalten.
- Die Merkmale aus der Tabelle *go_types*, welche einen numerischen Wert beinhalten, stehen unter ihrem Namen entsprechend der *go_types* als vordefinierte Variablen zur Verfügung.

Das Ergebnis muss dem Format des jeweiligen Tabellenfeldes entsprechen, in dem die Formel benutzt werden soll. Dies sind im Wesentlichen: Meter (m) für Positionen und Grad (deg) für Rotationen. Ist beispielsweise der Parameter *GWidth* in Millimeter angegeben, muss er somit in der Formel mit 0.001 multipliziert werden.

Für die Variantenkonstruktion kann mit der optionalen Datei *go_context.ofml* ein herstellerspezifischer Kontext definiert werden, welcher allen Evaluierungskontexten vorangestellt wird. Dies kann verwendet werden um Konstanten (auf Basis von OFML-Variablen) oder Prozeduren zu definieren – entsprechend OFML-Basissprache (OFML 2.0, Part III) - wodurch sich Lesbarkeit und Effizienz der Variantenspezifikation erhöhen lassen. Da dieser Kontext jeder Variantenspezifikation vorangestellt wird, sollte er nicht zu umfangreich sein.

4 Positionssteuerung im Warenkorb

Die Positionssteuerung im Warenkorb erfolgt weitestgehend über spezielle Einstellungen von *GSetup*. Die folgende Tabelle zeigt hierzu die wichtigsten Szenarien.



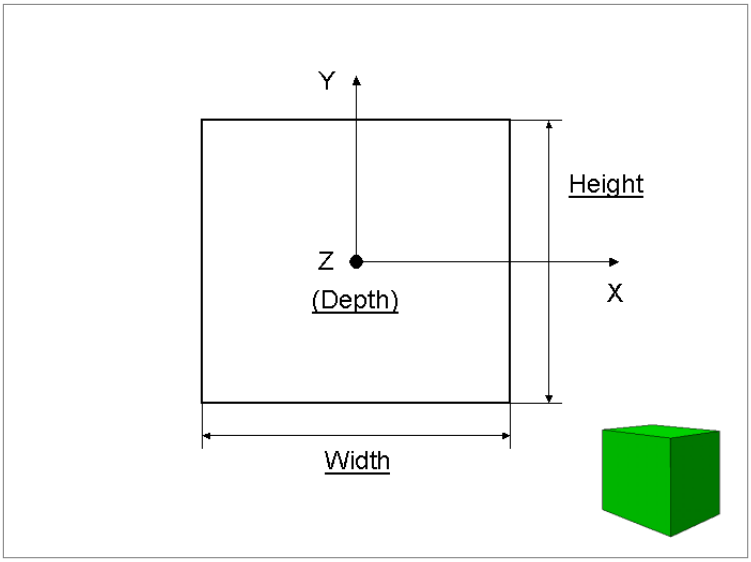
GSetup & 1 = 1 GSetup & 2 = 0 NoMTOrderRep = 1 ChildOrderRep = 0	GSetup & 1 = 1 GSetup & 2 = 1 NoMTOrderRep = 1 ChildOrderRep = 1	GSetup & 1 = 1 GSetup & 2 = 1 NoMTOrderRep = 1 ChildOrderRep = 1	GSetup & 1 = 0 GSetup & 2 = 0 NoMTOrderRep = 0 ChildOrderRep =0	GSetup & 1 = 0 GSetup & 2 = 1 NoMTOrderRep = 0 ChildOrderRep = 1	GSetup & 1 = 0 GSetup & 2 = 1 NoMTOrderRep = 0 ChildOrderRep = 1

5 Vordefinierte Interaktorgeometrien

::ofml::go::GoIGeometryBlock

Diese Geometrie realisiert einen zentrierten Quader. Die Parameter sind:

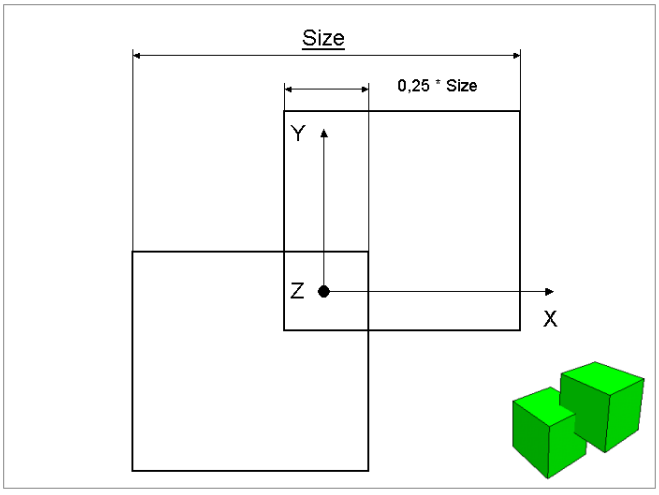
1. *Width* – Die Gesamtbreite des Quaders. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.07 verwendet.
2. *Height* – Die Gesamthöhe des Quaders. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.07 verwendet.
3. *Depth* – Die Gesamttiefe des Quaders. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.07 verwendet.



::ofml::go::GoIGeometryCubes

Diese Geometrie realisiert zwei ineinander geschobene Würfel. Die Parameter sind:

1. *Size* – Die Gesamtgröße des Objekts. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.1 verwendet.
2. *Parameter2* – Es muss ein numerischer Wert (z.B. 0.0) angegeben werden, welcher jedoch keine weitere Anwendung findet.
3. *Parameter3* – Es muss ein numerischer Wert (z.B. 0.0) angegeben werden, welcher jedoch keine weitere Anwendung findet.

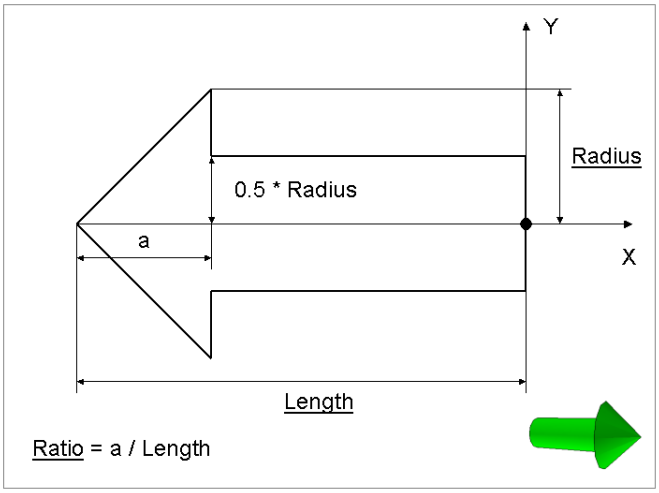


Diese Geometrie ist für die Visualisierung von Kopieranfügepunkten reserviert.

::ofml::go::GoIGeometryHArrow

Diese Geometrie realisiert einen Pfeil um die x-Achse. Die Parameter sind:

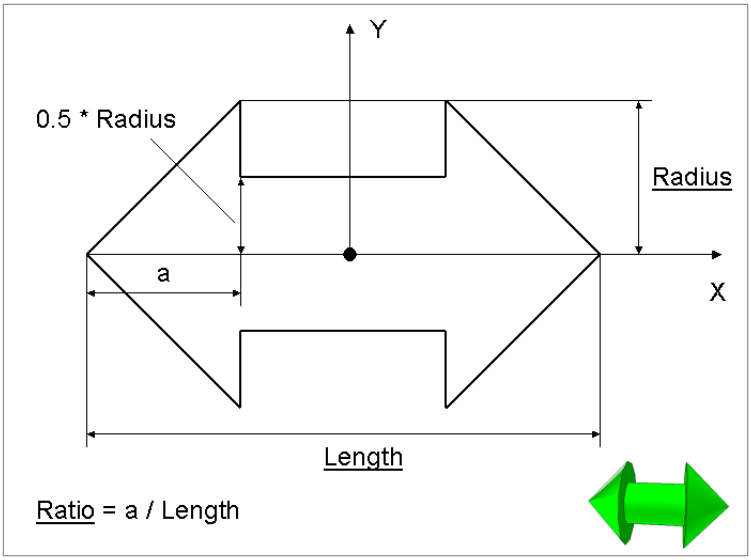
1. *Length* – Die Gesamtlänge des Pfeils. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.1 verwendet.
2. *Radius* – Der maximale Radius des Pfeils. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.03 verwendet.
3. *Ratio* – Das Verhältnis der Pfeilspitze zur Gesamtlänge. Es muss ein positiver numerischer Wert kleiner als 1.0 angegeben werden. Andernfalls wird 0.3 verwendet.



::ofml::go::GoIGeometryHDArrow

Diese Geometrie realisiert einen Doppelpfeil um die x-Achse. Die Parameter sind:

1. *Length* – Die Gesamtlänge des Pfeils. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.1 verwendet.
2. *Radius* – Der maximale Radius des Pfeils. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.03 verwendet.
3. *Ratio* – Das Verhältnis der Pfeilspitze zur Gesamtlänge. Es muss ein positiver numerischer Wert kleiner als 0.5 angegeben werden. Andernfalls wird 0.3 verwendet.



Diese Geometrie ist für die Visualisierung von Skalierungsanfügepunkten reserviert.

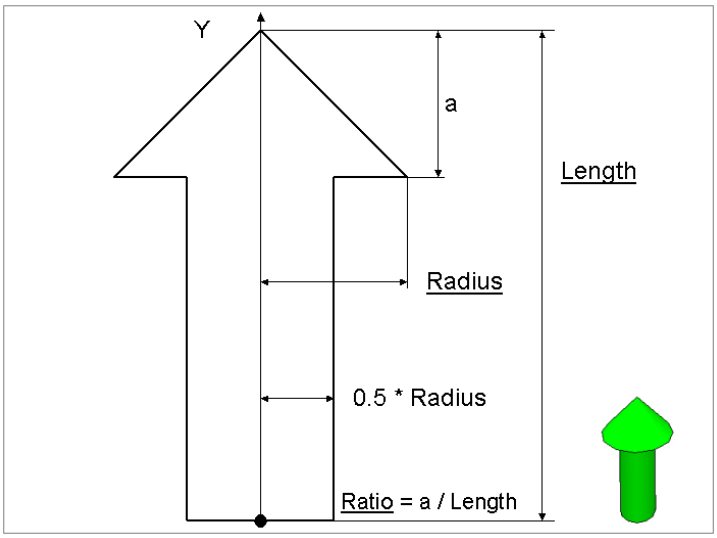
::ofml::go::GoIGeometryInvisible

Diese Geometrie realisiert einen unsichtbaren Anfügepunkt. Es werden keine Parameter unterstützt.

::ofml::go::GoIGeometryVArrow

Diese Geometrie realisiert einen Pfeil um die y-Achse. Die Parameter sind:

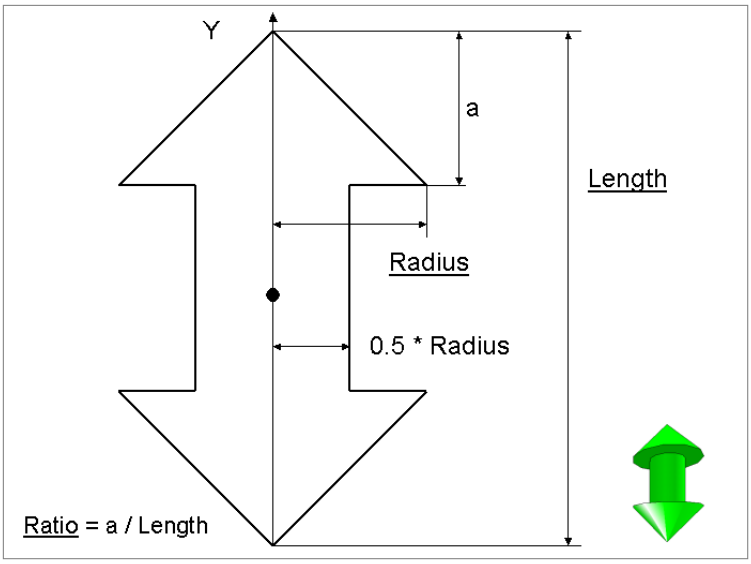
1. *Length* – Die Gesamtlänge des Pfeils. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.1 verwendet.
2. *Radius* – Der maximale Radius des Pfeils. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.03 verwendet.
3. *Ratio* – Das Verhältnis der Pfeilspitze zur Gesamtlänge. Es muss ein positiver numerischer Wert kleiner als 1.0 angegeben werden. Andernfalls wird 0.3 verwendet.



::ofml::go::GoIGeometryVDArrow

Diese Geometrie realisiert einen Doppelpfeil um die y-Achse. Die Parameter sind:

1. *Length* – Die Gesamtlänge des Pfeils. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.1 verwendet.
2. *Radius* – Der maximale Radius des Pfeils. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.03 verwendet.
3. *Ratio* – Das Verhältnis der Pfeilspitze zur Gesamtlänge. Es muss ein positiver numerischer Wert kleiner als 0.5 angegeben werden. Andernfalls wird 0.3 verwendet.



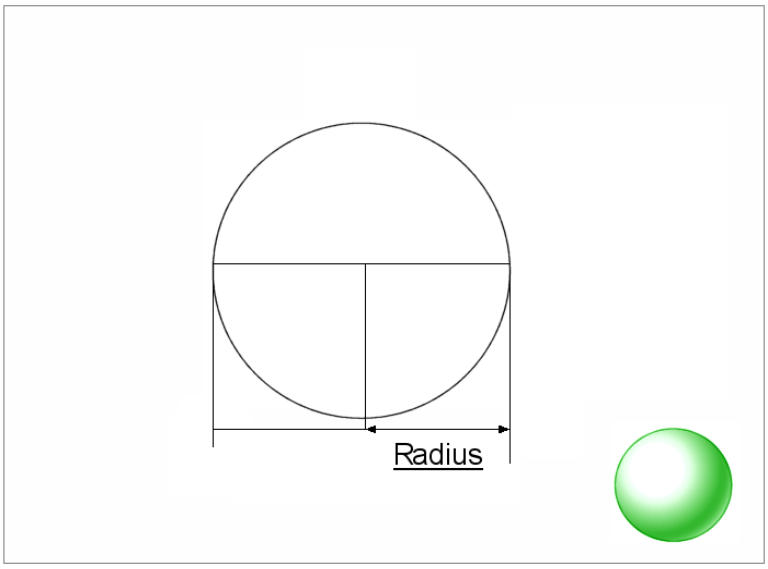
Diese Geometrie ist für die Visualisierung von Skalierungsanfügepunkten reserviert.

::ofml::go::GoIGeometrySphere

Diese Geometrie realisiert eine zentrierte Kugel. Die Parameter sind:

1. *Radius* – Der Radius der Kugel. Es muss ein positiver numerischer Wert angegeben werden. Andernfalls wird 0.07 verwendet.

Die Parameter 2 und 3 werden ignoriert und müssen daher nicht bewertet werden.



6 Vordefinierte ITemplates

::ofml::go::IT_Standard

Mit einer Anschlussplanung an jeder Seite.

::ofml::go::IT_Standard2

Mit zwei Anschlussplanungen an jeder Seite.

::ofml::go::IT_Cabinet1

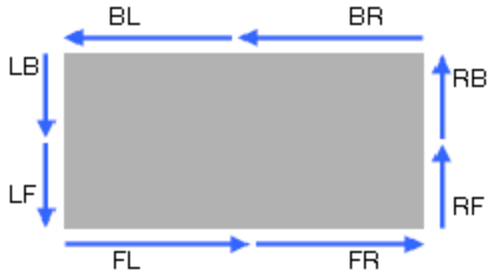
Für Schränke/Schreibtische mit rückseitiger und seitlicher Anschlussplanung.

::ofml::go::IT_Cabinet2

Für Schränke mit rückseitiger und jeweils zwei seitlichen Anschlussplanungen.

::ofml::go::IT_Rectangle

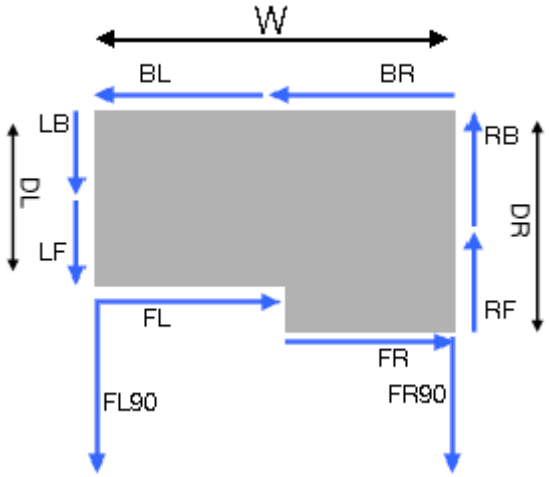
Mit bis zu zwei Anschlussplanungen an jeder Seite. Über den Parametervektor kann jede einzelne der Anschlussplanungen [@BL, @BR, @FL, @FR, @LB, @LF, @RB, @RF] freigeschaltet werden. Wird kein Parameter angegeben, sind alle acht Anschlussplanungen verfügbar.



::ofml::go::IT_Rectangle2

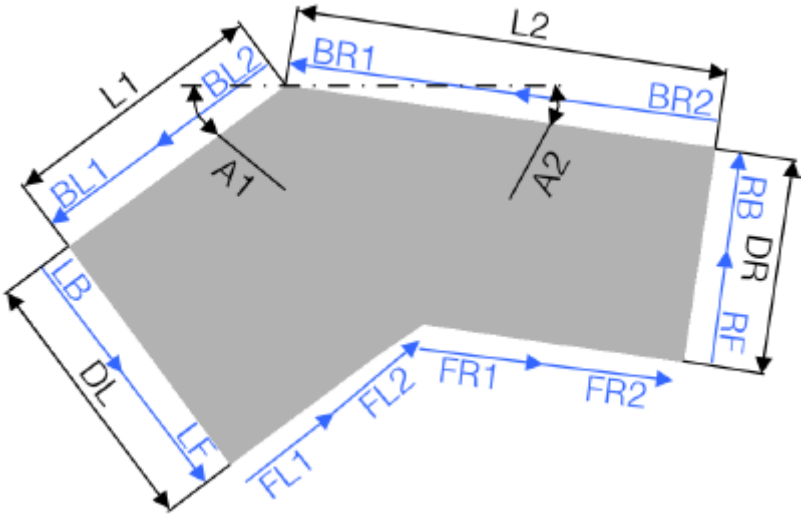
Mit bis zu zwei Anschlussplanungen an jeder Seite. Die Parameter beinhalten die Tiefen DL und DR in Metern sowie einen Vektor für die einzelnen Anschlussplanungen [@BL, @BR, @FL, @FR, @LB, @LF, @RB, @RF, @FL90, @FR90]. Wird dieser Vektor nicht angegeben, sind alle acht Anschlussplanungen verfügbar. Über einen zusätzlichen optionalen Parameter kann die Objektbreite angegeben werden.

Die Reihenfolge der Parameter kann nicht verändert werden.



::ofml::go::IT_Angular

Für Winkelelemente mit bis zu zwei Anschlussplanungen an jeder Seite. Die Parameter beinhalten die Schenkellängen L1 und L2, die Tiefen DL und DR jeweils in Metern, die Winkel A1 und A2 in Grad sowie einen Vektor für die einzelnen Anschlussplanungen [@BL1, @BL2, @BR1, @BR2, @FL1, @FL2, @FR1, @FR2, @LB, @LF, @RB, @RF]. Wird dieser Vektor nicht angegeben, sind alle zwölf Anschlussplanungen verfügbar. Die Reihenfolge der Parameter kann nicht verändert werden.



::ofml::go::IT_Notemplate

Für diesen Fall wird die ITemplate-Funktionalität deaktiviert.

7 Implementierungshinweise

Serie *global*

Die Umsetzung erfolgt über eine neu eingeführte Serie *global*, welche eine Reihe von Tabellen im CSV-Format bzw. kompiliert in das EBASE-Format als *global.ebase* beziehungsweise als *mt.ebase* im Verzeichnis *meta* im jeweiligen Vertriebsgebiet enthält. Die String-Ressourcen werden in den laut OFML-Standard vorgeschriebenen Ressourcen-Dateien angelegt (*global_de.sr*, *global_en.sr*, etc.)

Anmerkung: Anstelle von *global* sind auch andere Bezeichnungen möglich.

Serien-Registrierung

Die Registrierung von Serien, die Metatypen bereitstellen bzw. die in Metatypen verwendet werden, erfolgt laut DSR-Beschreibung ab Version 2.7.0 über die Schlüssel *series_type=go_meta* bzw. *meta_type*.

Zur speziellen Steuerung der Metatypen kann die Funktion *::ofml::go::goSetup()* verwendet werden:

```
meta_type=::ofml::go::GoMetaType;::ofml::go::goGetMetaType();::ofml::go::goSetup(args[2])
```

Dabei können 1 oder mehrere Key-Value-Paare wie folgt angegeben werden:

- *[@FIRST, <series>]* – Gibt eine oder mehrere Serien an, in welchen im Rahmen der automatischen MT-Ermittlung zuerst nach einer entsprechenden Grundartikelnummer gesucht werden soll. Die Serien müssen als OFML-Symbole innerhalb eines Arrays angegeben werden, z.B. *[@s1]*.
- *[@FIND, <mode>]* – Spezifiziert einen Modus für die automatische MT-Ermittlung. Erlaubte Werte sind:
    - *@MATCH* – Nur Rückgabewerte erlaubt, wo sowohl Grundartikelnummer und Serie übereinstimmen.
    - *@ALL* – Rückgabewerte erlaubt, wo mindestens die Grundartikelnummer korrekt ist. (Default-Verhalten)

Basistyp

Alle Metatypen verwenden als OFML-Typ den Typ *::ofml::go::GoMetaType* oder einen davon abgeleiteten Typ, welcher in den entsprechenden Mapping-Dateien (*art2aclass.map*) einzutragen ist. Im Fall eines automatischen Mappings – z.B. bei konvertierten OFML-Daten – entfällt dies.

Parametrisierung

Im Fall einer expliziten Verwendung von Metatypen kann erfolgt deren Parametrisierung entsprechend der XCF-Spezifikation über den Variantencode, welcher im Katalog hinterlegt wird. Hierbei ist zu beachten, dass die Parametrisierung über den XCF-Variantencode (*variant.csv*) erfolgt. Der Variantencode besteht dabei mindestens aus dem Herstellerkürzel und dem Schlüssel der Tabelle *go_types*.

Bsp.: `[@hex, @tisch1]`

Anschließend kann eine beliebige Anzahl von Key-Value-Paaren folgen. Diese überschreiben gegebenenfalls die Initialwerte, die durch die *go_types* definiert wurden. Ihr Format und ihre Schreibweise entsprechen exakt denen der *go_types.* Allerdings muss Namen und symbolischen Werten ein @ vorangestellt werden!

Der Key @MTSeries gefolgt von einem Seriennamen in Symbol-Schreibweise wird dazu verwendet, anstelle der Serie global einen beliebigen anderen Namen der Metaserie anzugeben.
Der Key *@VarCode* gefolgt von einem String wird dazu verwendet, einen Varianten-Code für das Hauptkind anzugeben, welcher nach Erzeugung des Kindes gesetzt wird.

Bsp.: `[@hex, @tisch1, [@GWidth, 1600], [@GDepth, 900]]`

Katalogabhängigkeiten

Metatypen-Serien hängen von allen Serien ab, deren Instanzen sie erzeugen können. Normale Serien hängen von allen Metatypen-Serien ab, aus denen bei Instanziierung von Objekten automatisch Metainstanzen erzeugt werden sollen. Dementsprechend müssen die Kataloge registriert und die Abhängigkeiten gesetzt werden (siehe DSR-Beschreibung verfügbar von EasternGraphics).

Initialisierungsreihenfolge

Es ergibt sich die folgende Initialisierungsreihenfolge für Metatyp-Merkmale:

1. [UNBEDINGT] Wert aus der *go_types*-Tabelle
2. [OPTIONAL] Wert aus dem Katalog (XCF-Variantencode, nur für explizit angelegte Metatypen)
3. [OPTIONAL] Wert der vom Vorgänger geerbt wird
4. [OPTIONAL] Wert der sich durch Metatypaktionen ergibt
5. [OPTIONAL] Wert aus dem Profil
6. [OPTIONAL] Wert über *CH_ADD*-Aktion

Bei automatisch erzeugten Metatypen entfällt Schritt 2.
Native Merkmale werden vom Vorgänger/Vater nach Schritt 4 (Metatypaktionen) geerbt.
Eine weitere Merkmalsbeeinflussung ergibt sich wenn *SET_PROP*-Aktionen im Modus *CON* definiert wurden und die referenzierten Merkmale sich ändern.

Unbedingt zu beachten:

- **Metatypmerkmalsbezeichner dürfen NICHT mit solchen der eingekapselten Objekte identisch sein.**
- **Die unzulässige Anlage der Metatyp-Tabellen kann zu Abstürzen der Anwendungssoftware führen**.

8 Historie

Diese Historie beschreibt nur die Änderungen, welche in Bezug auf diese Spezifikation relevant sind. Alle weiteren sind aus der Historie ersichtlich, welche Bestandteil der Metatypen-Release ist.

MT 1.17.3(-0)

- **Erweiterung:** Neue Tabelle: *go_symbolicpropvalues*.

MT 1.17.2(-0)

- **Erweiterung:** Neuer go_info-Schlüssel: *utf8*.

MT 1.17.1(-0)

- **Erweiterung:** Modus GO_AP_POSROT in Tabelle *go_childmoving*
- **Erweiterung:** Neue Tabelle *go_resetnativeprops*

MT 1.17.0(-0)

- **Erweiterung:** Neue Planungsrichtung MP in der Tabelle go_attpt
- **Erweiterung:** Neue Tabelle *go_interactors*
- **Erweiterung:** Neuer go_setup-Schlüssel: *SumSubArticlePrices*
- **Erweiterung:** Neuer go_setup-Schlüssel: *UseMCAxis*
- **Erweiterung:** Neues go_childmoving-Kommando: *RAS_OFFS*

MT 1.16.1(-0)

- **Erweiterung:** Neuer go_setup-Schlüssel: *DisableAutoChildReposition*

MT 1.16.0(-0)

- **Erweiterung:** Neuer Merkmalsmodus 4096 zur Kollisionsprüfung während der Merkmalsänderung
- **Erweiterung:** Neuer Merkmalsmodus 8192 zur Gültigkeitssteuerung von Werten kindsteuernder und na-Merkmale
- **Erweiterung:** Neue Tabelle *go_propvalues*
- **Erweiterung:** Neue interne Metatyp-Variable *XIsInsObj*
- **Erweiterung:** Neuer go_setup-Schlüssel: *ShowPolyPropFilterMsg*

MT 1.15.0(-0)

- **Erweiterung:** Mehrere Properties in Tabelle *go_noproperties*
- **Erweiterung:** Neue Tabelle: *go_attptsorder*

MT 1.14.3(-0)

- **Erweiterung:** Unterstützung für CH_REL-Anfügepunkt
- **Erweiterung:** Optionaler Parameter für die Objektbreite im ITemplate IT_Rectangle2
- **Erweiterung:** Neue Anfügeseiten @FL90 und @FR90 im ITemplate IT_Rectangle2
- **Änderung:** ITemplate IT_Standard ist nicht mehr per default aktiviert.

MT 1.14.2(-0)

- **Erweiterung:** Neuer Merkmalsmodus 1024 zur Neupositionierung der Basisgeometrie
- **Erweiterung:** Neuer Merkmalsmodus 2048 zur Neupositionierung der eigenen Kinder
- **Erweiterung:** Neue ITemplates IT_Rectangle, IT_Rectangle2 und IT_Angular

- **Erweiterung:** Neue Tabelle: *go_itemplates*
- **Änderung:** go_setup-Schlüssel *ITemplate* entfällt

MT 1.14.1(-0)

- **Erweiterung**: Neuer go_setup-Schlüssel: *ITemplate*.
- **Erweiterung:** Neue ITemplates IT_Standard, IT_Cabinet1, IT_Cabinet2 und IT_NoTemplate
- **Erweiterung:** Mehrere Parametersätze für Kind-orientierte Properties in Tabelle *go_articles*.

MT 1.14.0(-0)

- **Erweiterung:** Interaktorgeometrie GoIGeometrySphere
- **Erweiterung:** Neuer go_info-Schlüssel: *updateGMode*.
- **Erweiterung:** Neue Tabelle: *go_propclasses*
- **Erweiterung:** Mehrere Kindschlüssel in Tabelle go_childprops
- **Erweiterung:** Neue Tabelle go_setup
- **Erweiterung:** Mehrere Eigen- und Fremdschlüssel in Tabelle go_actions bei Ursache CON
- **Erweiterung:** Neue Tabelle go_texts
- **Erweiterung:** Daten können im Vertriebsgebiet im Unterorder *meta* abgelegt werden

MT 1.12.2(-0)

- **Erweiterung:** Neuer go_info-Schlüssel: *skipVC2MT*.

MT 1.12.1(-0)

- **Erweiterung:** Neuer go_info-Schlüssel: *skip_FAN*.

MT 1.12.0(-0)

- **Erweiterung:** Neuer go_info-Schlüssel configuration
- **Erweiterung:** Neue Tabellen go_propindex und go_propmapping.
- **Erweiterung:** Neue Tabelle go_info. Initialer Schlüssel pindex.
- **Erweiterung:** Neue Ursache für Tabelle go_actions: PROXY.
- **Erweiterung:** goSetup-Modus @FIND (@ALL, @MATCH)
- **Erweiterung:** Reservierte MT-ID *_native_* in Tabelle *go_articles*
- **Erweiterung:** Interaktorgeometrie GoIGeometryVDArrow
- **Erweiterung:** Interaktorgeometrie GoIGeometryHDArrow
- **Erweiterung:** Kopieranfügepunkte
- **Erweiterung:** Interaktorgeometrie GoIGeometryCubes

MT 1.11.0-1

- **Erweiterung:** Neue Modi für Tabelle *go_childmoving* (IN, IX, etc.)
- **Erweiterung:** Wildcard in Tabelle *go_classes*

MT 1.11.0(-0)

- **Erweiterung:** Neue Tabelle *go_classes*
- **Erweiterung:** GXSetup, Mode 8 – Interaktiver Feedback-Modus
- **Erweiterung:** Neue Tabelle *go_feedback* zur Unterstützung des interaktiven Feedback-Modus

MT 1.10.1(-0)

- **Erweiterung:** Autodekoration auch für GO I-Objekte
- **Modifikation:** Metakategorie für Zubehör von acat auf AccCategory geändert

MT 1.10.0(-0)

- **Erweiterung:** Interaktorgeometrie GoIGeometryVArrow
- **Erweiterung:** Interaktorgeometrie GoIGeometryHArrow
- **Erweiterung:** Interaktorgeometrie GoIGeometryInvisible
- **Erweiterung:** Interaktorgeometrie GoIGeometryBlock
- **Erweiterung:** neue Tabelle go_attptgeo
- **Erweiterung:** neue Tabelle go_nativeproperties

MT 1.9.3(-0)

- **Erweiterung:** neues Property-Flag 512 (Kind-Neuerzeugung)
- **Modifikation:** Metakategorie für Zubehör von acc auf AutoDecoration geändert
- **Erweiterung:** Filterverwendung bei fn-Merkmalen analog zu na-Merkmalen
- **Erweiterung:** Verknüpfung von Geometrieausrichtung und _GO_CHILD

MT 1.9.0-1

- **Erweiterung:** Kontextdatei go_context.ofml
- **Erweiterung:** neuer Property-Type ‚lb'
- **Erweiterung:** GAlign-Wert ATTPT und vordefinierter Anfügepunktschlüssel _GO_CHILD

MT 1.9.0(-0)

- **Erweiterung:** GXSetup-Modus 4 – Berücksichtigen von Hauptkind in GXSetup Modi 1 und 2
- **Erweiterung:** Neuer Modus für Tabelle go_metainfo: acat.
- **Erweiterung:** GXSetup-Modus 2 – Steuerung der Kollisionserkennung bei Rotation von MT-Kindern
- **Erweiterung:** GXSetup-Modus 1 – Steuerung der Kollisionserkennung bei Translation von MT-Kindern
- **Erweiterung:** Neue Steuervariable GXSetup
- **Erweiterung:** Neuer Merkmalsmodus 256 zum Update der Basisgeometrie

MT 1.8.0(-0)

- **Erweiterung:** Neue Tabelle zur Steuerung der Merkmalsvererbung.
- **Erweiterung:** Neue Aktion *CON_AP* zur Positionsanpassung von Geschwisterobjekten
- **Erweiterung:** Neue Kindbewegungsmodi *XR*, *YR* und *ZR*.
- **Erweiterung:** Der Filtereintrag von na-Merkmalen kann verwendet werden, um Abhängigkeiten mit Kindmerkmalen zu definieren.

MT 1.7.1(-0)

- **Erweiterung:** Neuer GSetup-Modus 32768
- **Erweiterung:** Neue GMode-Werte ‚@XOCD' und ‚@OCD'

MT 1.7.0-1

- **Erweiterung:** Neuer Merkmalstyp ‚th'.

MT 1.7.0(-0)

- **Erweiterung:** Neue Tabelle go_metainfo
- **Erweiterung:** Neue interne Metatyp-Variable XChildID
- **Erweiterung:** Neue Modi für Aktionen: CH_ADD, CH_DEL
- **Erweiterung:** GSetup-Modus 16384, Behandlung interaktiver Kinder bei Merkmalsänderung

MT 1.6.0-2

- **Hinweis:** Merkmalsmodus 32 (Vererbung) sollte nicht verwendet werden, wenn bereits initial Werte über Aktionen gesetzt werden.

MT 1.6.0-1

- **Korrektur:** In der Tabelle zur Warenkorbpositionierung war die Bedeutung von GSetup & 2 invers und somit falsch dargestellt wurden.

MT 1.6.0(-0)

- **Erweiterung:** Planungsrichtungen T und D
- **Erweiterung:** fn-Merkmale und Tabelle go_freenumeric
- **Erweiterung:** lokale Höhe des Objekts über Variable XHeight im lokalen Kontext verfügbar. Analog _XHeight für sekundäres Element in Relationen und Bedingungen.
- **Erweiterung:** Aktionstrigger CREATE
- **Erweiterung:** Tabelle go_proporder
- **Erweiterung:** go_types, mode, Flag 128: Steuerung der Reihenfolge von Werten in Merkmalslisten

MT 1.5.0-1

- **Modifikation:** GSetup, Modus 2, Anpassung an GO 1.4.*

MT 1.5.0(-0)

- **Erweiterung:** GSetup, Modi 1 und 2, Ist Modus 1 gesetzt, wird Hauptkind stets zu HPOS unabh. der Einstellung GSetup & 2
- **Erweiterung:** GSetup, Modus 8192, Erzwingen des HPOS-Status
- **Erweiterung:** Parametrisierung über Katalog: @VarCode
- **Erweiterung:** CON::SET_PROP

MT 1.4.0-3

- **Erweiterung:** GSetup, Modus 2048, Unterbinden des Erbens nativer Merkmale falls Vater ein Metatyp ist

MT 1.4.0-2

- **Erweiterung:** GSetup, Modus 512, Ausblenden des Hinweises ‚Es erfolgt keine Anschlussplanung'
- **Erweiterung:** Wrapping für Positionen von Kindern des Metatyps (Property-Typ „cp")

MT 1.4.0-1

- **Erweiterung:** Aktionsmodus *AP*
- **Erweiterung:** *GSetup*, Modus 1024, vollständige temporäre Erzeugung

MT 1.4.0(-0)

- **Erweiterung:** Merkmal *GVarPrefix*
- **Erweiterung:** Modus 256 für *GSetup* (Löschen interaktiver Kinder)
- **Erweiterung:** Wrapping für native Properties (Property-Typ „na")

MT 1.3.10(-0)

- **Erweiterung:** Modus 64 für go_types (Anzeige von Merkmalen welche durch Filter angepasst wurden)
- **Erweiterung:** Neue Aktionen CON_PROP und CON_CH_PROP.
- **Änderung:** vordefinierter Typ GO_RECTTABLE entfernt

MT 1.1-1.6

- **Erweiterung:** MT-Kinder erben nun native Properties entweder vom Vater oder vom Vorgänger – je nach GSetup Modus 128

MT 1.1-1.5

- **inkompatible Änderung:** bei der Steuerung der interaktiven Bewegung von MT-Kindern gegenüber MT 1.1-1.4 (GO 1.3.3): lokale Parameter ohne Präfix „_", die des Vaters mit Präfix
- **Erweiterung:** Kindbewegungsmodus *XZTLINEAR, XZTRANGE*

MT 1.1-1.4

- **Erweiterung:** *go_types*-Parameter *GSetup* Modus 64 (Einschalten der Kollisionserkennung für Kinder)
- **Erweiterung:** *go_types*-Parameter *GSetup* Modus 32 (Ausblenden der geänderten Merkmalsnamen)
- **Erweiterung:** *go_types*-Parameter *GSetup* Modus 16 (Ausblenden der Artikelnummer)
- **inkompatible Änderung:** Änderung der Einfügepolicy bei interaktiven Kindern die einen Anfügepunkt O, OL oder OR besitzen.
- **Erweiterung:** *go_types*-Parameter *GSetup* Modus 8 (Verwendung der Standardanfügepunkte)

MT 1.1-1.3

- **Erweiterung:** *go_types*-Parameter *GSetup* Modus 4 (2D-Symbol)

MT 1.1-1.2

- **Erweiterung:** *go_types*-Parameter *GAlign*
- **Erweiterung:** *go_types*-Parameter *GSetup* incl. Modi 1und 2 (Art.nr.umlenkung, HPOS/UPOS-Steuerung)

MT 1.1-1.1

- **inkompatible Änderung:** *go_childmoving[parameter]* wird im Kontext des Vaters ausgewertet

MT 1.1-1.0

- **Erweiterung:** Tabelle *go_childmoving* hinzugefügt (XT, YT, ZT x MIN, MAX, POS, RASTER)
- **Erweiterung:** lokale Ursprungsanfügepunkte *OL* und *OR*
- **Erweiterung:** DSR-Registrierung von Metaserien oder Serien, die Metatypen verwenden

MT 1.0-1.0