EasternGraphics
visualize your business

Specification

MT – OFML Metatypes – Tables and Specifics

Author	*Ekkehard Beier, Andreas Handschuh, EasternGraphics*	
Reference	Title	**Metatype Specification**
	Version	**1.17.2**
	Release Date	**2021-04-27**
	Implementation	**::ofml::go::1.17.26**
	Version of the document	**(0)**
History	At the end of the document	
Status	Release to EGR, sales partners and customers. Subject to change due to technical progress, purpose of optimization and removal of conceptual errors.	
Feedback	metatypes@EasternGraphics.com	
Remark	Changes relative to **MT 1.17.1(-0)** are marked by underline.	

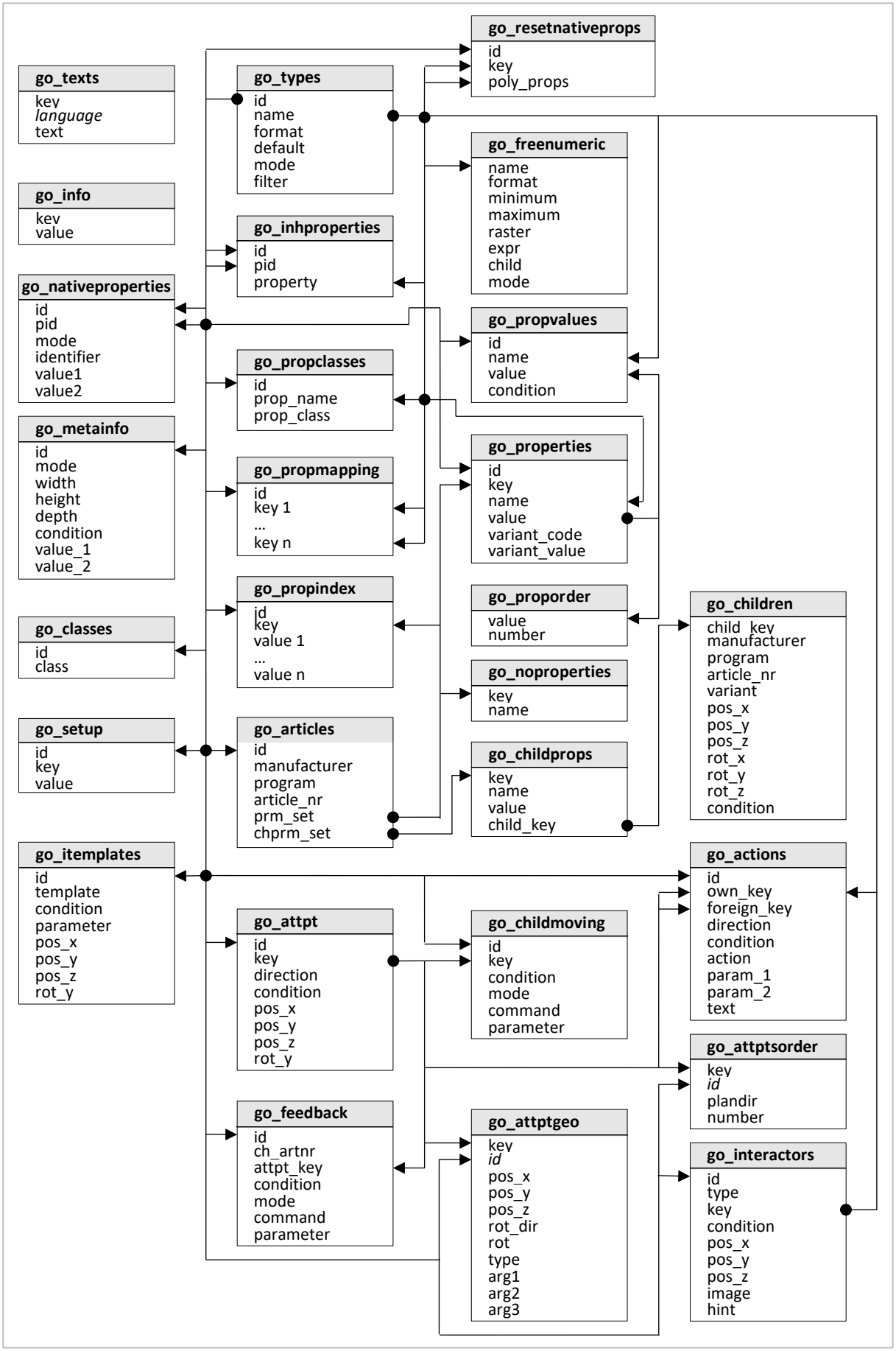
1 General

The concept of the OFML metatypes enhances the traditional way of modeling graphic data by the following opportunities:

- Configuration on inter-product level, i.e., for a given object not only the intra-product properties can be changed (here, the basic article number remains unchanged) but even those changing the basic article number, such as the selection of another program or collection
- Concatenation and attachment rules, e.g. attachment parts can be described by properties described in tables in an article-dependent way

2 Tables

In this chapter we describe the tables needed for the metatypes. The physical types of the entries are described in the script *mt.inp_descr* or in the EBase specification. All tables must be present. Unused tables should be empty.



go_info

The optional table *go_info* is used to define properties that are relevant to the whole MT series.

Key (key)	The key names a control variable. The set of control variables and the associated set of values are defined in the following.
Value (value)	The value sets the contents of the control variable. The set of control variables and the associated set of values are defined in the following.

The following control variables and associated sets of values are defined. The control variables are marked in the following way:

- [0, 1] – can occur but not more than once
- [1, 1] – must occur exactly once
- [0, *] – can occur arbitrarily often
- [1, *] – can occur arbitrarily often, at least once however

Key *configuration* [0, 1]

The key configuration controls the global configuration behavior of the Metatypes of this series. The values are as follows:

- *consistent* – The Metatypes implementation tries to recover to a consistent state with regard to the article polymorphism by adaption of other properties. This behavior is applied also if this key is not specified.
- *inconsistent* – The Metatypes implementation does not modify other properties to get a consistent state and therefore does tolerate inconsistent states.
- *serial* – The Metatypes implementation fixes properties once set and creates the degrees of freedom for the remaining ones dynamically.

Key *pindex* [0, 1]

The key *pindex* must be set if the property-indexed polymorphism tables *go_propindex* and *go_propmapping* exist.

The value is the number of property columns in these tables and must be a positive integer number > 1.

Key *skip_FAN* [0, 1]

The key *skip_FAN* should be set to suppress the verification (and message output) of the Final Article Number after the creation of an accordingly created article.

The value as such is ignored but must exist.

Key *skipVC2MT* [0, 1]

The key *skipVC2MT* should be set to suppress the mapping from metatype properties to the variant code. This can improve the performance with large data tables.

The value as such is ignored but must exist.

Key *updateGMode* [0, 1]

The key *updateGMode* should be set if the value of the flag *GMode* (kind of product data) should be updated when the base article number is changed. This is necessary if two series in the same metatype use different kinds of product data (e.g. OCD and EPL).

The value as such is ignored but must exist.

Key *utf8* [0, 1]

The key *uft8* must be set if the table *go_texts* is encoded in UTF-8 character set. Optionally, the byte order mark can be specified at the beginning of the file. The normal form should be NFC (Normalization Form Canonical Composition).

The value as such is ignored but must exist.

go_types

The table *go_types* defines the metatypes or metatype instances of a manufacturer. Such a metatype represents a set of article numbers defined in the table *go_articles*. In the following, the terms metatype and instance of a metatype are used synonymously.

Example:

By referencing the general metatype 'desk' (*GO_TABLE*, see below) a manufacturer Man1 defines a type that represents width-variable tables with constant depth (800) and height. Man1 defines a second type that represents width-variable tables with constant depth (1200) and height.

A second manufacturer, lets say Man2, uses the same metatype to describe the tables. But in this case the tables are variable in width, height and depth.

Type ID (id)	This key references a metatype instance. The key must be unique in the context of the manufacturer. Primarily, a metatype is defined by at least one entry in *go_types*. Here each entry defines exactly one property of the metatype. Example: *table1*
Property Name (name)	This entry defines the name of the property to which the further values in this entry of the table apply. Property names should be written in English language. They should start with 'G' followed by a capital letter. For composite names each word should begin with a capital letter. No umlauts, special characters or space are allowed. All properties defined for a given GO type must be described. Further properties can be defined. The property *GType* is reserved but can be used in conditions and actions. The property *GVarPrefix* is reserved but can be used in conditions and actions in mode @EPL. It either has the value NULL or the corresponding prefix as a symbol. Example: *GWidth* (width), *GHeightAdjust* (height adjustability)

Format (format)	This column defines the format of the properties. Properties already defined by the GO type have to apply the predefined format. The supported formats are (according to the OFML standard):
	• *ch* – Selection from a set of symbolic values
	• *chf* – Selection from a set of float numbers
	• *chi* – Selection from a set of integer numbers
	• *f* – float number
	• *i* – integer number
	Additionally the following formats are available:
	• *fn* – free numeric property. To use fn properties consider these rules, please:
	  • This metatype property is different from traditional metatype properties because it does not necessarily have a discrete range of values, nor does the regular mechanisms such as filters available here.
	  • The fn properties are parametrized by the separate table *go_freenumeric*.

	• The fn properties may be forwarded to child objects. In the case of interactively moveable children there is no feedback from the child back the fn property. Thus it is possible that the degrees of freedom (axes, minimum, maximum, raster, etc.) may be differing between fn property and associated child property. • For an fn property the filter entry can be used to specify a list of child properties that get affected by changes of this na property, e.g. if the corresponding child objects must be replaced in space. If not defined otherwise (mode 2048), these children will be re-created if this property is changed. • *na* – wrapper of a native property. This way a property of the wrapped child object becomes available on the metatype level. Please note that: • The property must have the same name as the native one plus the prefix *G*. E.g., the native property *Frame* must be wrapped by the na property *GFrame*. • Such properties are only available as long as the child object exists. For instance, it is not available initially because the child object must be created by means of the metatype properties at first. As long as a default value is defined in this table for the property; this value will be used when the native property isn't available. In analogy to other locations where values are set, there is no '@' allowed for symbolic values. • na properties do not support further features like filters or visibility. • na properties must not be used within filters of other properties. • na properties may be used only for such native properties that cannot have the value NULL. • If an additional prefix is added to native properties during OFML conversion (such as an 'S' for EPL data), the same prefix must be used in actions too. This applies also to the default value. • For an na property the filter entry can be used to specify a list of child properties that get affected by changes of this na property, e.g. if the corresponding child objects must be replaced in space. If not defined otherwise (mode 2048), these children will be re-created if this property is changed • *th* – so-called thru properties. Th properties can be used – e.g. in a concatenation of objects – to transfer the value of a metatype property from the predecessor of a given object to the successor of this object. Th properties are not part of the article polymorphism. Furthermore the following rules apply: • Only regular metatype properties can become th properties in another object. • The type of the transferred property must be specified in the entry normally used to specify the filter. • The property is not available anywhere else, e.g. it is not visible in the property editor, nor can it be used in filters. Any settings concerning to this (visibility, read/write mode) are ignored. • *cp* – Representation of the position of immediate or indirect children of the metatype in the context of conditions and variant-based construction. Please consider: • The property is not available anywhere else, e.g. it is not visible in the property editor, nor can it be used in filters. • The property value is defined in meters with regard to the local coordinate system of the metatype. • The default value of the cp property is used if the reference object does not exist. • The mode of the cp property refers to the coordinate represented by the property. A two-digit code is used here as follows: The first digit describes the coordinate: 1(x), 2(y) or 3(z). The second digit describes the alignment within this coordinate. The following codes are possible: 0 – position, 1 – maximum of the local bounding box, 2 – minimum of the local bounding box, 3 – center of the local bounding box. Example: 21 – maximum of the local bounding box in y

	direction, i.e. the upper edge of the specified object. • For a cp property, the filter entry specifies the reference object in terms of a relative object name resp. relative path, e.g. *e1* or *e.e1* • *lb* – A property to be used as read-only label. The property is not available furthermore. However it is available inside the evaluation context.
Default value (default)	This entry defines the default value to be assigned to the associated property after creation of the metatype object. This is a polymorphic value corresponding to the format of the property. To describe non-selection for properties of format ch the value *@VOID* must be used. If you don't want to prescribe an explicit default value, leave this entry empty. Example: *1000, H1* If the property is used to control invisible identical sub-positions, the default value is interpreted as follows: [*Start, Minimum, Maximum*] – The numbers define a range of integer values. The minimum must be a non-negative number. The maximum must be greater than the minimum. The start value must be greater than or equal to the minimum, and less than or equal to the Maximum. Example: *[1, 0, 5]* – Up to 5 sub-positions are possible; initially there is 1 sub-position. The predefined value *MT_UNDEF* marks an undefined or not selected property. This is especially relevant for the Serial configuration mode. Value *MT_UNDEF* should only be used for polymorphic properties of format *ch*.
Mode (mode)	This mode controls the application of the properties. A set of single modes is combined to a composite mode. Note: The modes 4 and 8 are **not** allowed in parallel. • 1 – The property can be edited. • 2 – The property is considered for global modification of properties. • 4 – The property controls the variant code. • 8 – The property controls a sub-item. Here we can use either a regular choice list (CH – format 'ch') or an integer-based concept (INT – format 'i'). In the CH case, different property values create various instances of sub-positions controlled by the corresponding children tables (see below). In the INT case, a number of identical and invisible sub-positions will be created. This is specified by the initial value of the property in terms of three parameters: the initial value, the minimum and the maximum (see above). To access the definition of the child the parameter is used as key. Only the first matching entry from the children table will be considered. All furthers will be ignored. • 16 – The property is invisible. • 32 – The property will be inherited initially from either the parent object or the predecessor object (must be metatypes, too). Do not use mode 32 for properties that get assigned values already during creation actions. • 64 – Do not show the adaptation of the property during filter processes in a dialog window. This mode is only evaluated if the option *ShowPolyPropFilterMsg* is activated in the table *go_setup*. • 128 – Do **not** apply the standard way of sorting of property values in case of choice lists. If this flag is set for standard metatype properties the position of the property values will be specified explicitly by table *go_proporder*. In case of child-controlling properties the original sorting from the table *go_childprops* is used. • 256 – After modification of this property the 2D and 3D geometry of the main child should be re-created. Note. This is only needed for child-creating properties in very special cases. • 512 – This flag enforces the removal and re-creation of the object's own children after modification of this property. Otherwise, if the flag isn't set, the children stay alive but get a new position/orientation. In specific cases, removal and re-creation of

	children can be useful. The behavior is equal to *GSetup & 16384* that operates for all properties of the metatype however. • 1024 – After modification of this property the 2D and 3D geometry of the main child should be re-positioned. Note. This is not needed for polymorphic properties. • 2048 – After modification of this property the object's own children should be re-positioned. Note: This is only needed for na and fn properties. In this case the list of the affected child properties has to be specified in the column *filter*. • 4096 – After modification of this property a typical OFML collision detection is applied. If there is a collision, the change of the property is rejected with a message. • 8192 – The validity of values of this property can be limited via the table *go_propvalues*. Example: *3* – The property can be edited and is enabled for global modification.
Filter (filter)	In this field properties to be considered during the configuration of this property, must be enumerated. This filter has to be specified by property names, separated by comma. If there is no filter the entry must be empty. Example: *GWidth,GHeight,GDepth,GHeightAdjust* In the Serial configuration mode the filter will be ignored. Instead the filter will be generated automatically from the explicitly set polymorphic properties.

The following properties are pre-defined:

- *GType [-]* – This required property defines the metatype class to be used.
- *GMode [-]* – This optional property refers to the kind of product data. The following kinds are supported:
  - *EPDF (default)* – The commercial data are provided in EPDF format.
  - *XOCD* – The commercial data are provided in XOCD format.
  - *OCD* – The commercial data are provided in OCD format.
  - *EPL* – The commercial data are provided in EPL format.
- *GSetup [-]* – This optional property is provided to control specific features of the metatype. The final value is composed by adding the following single values. This property is now replaced by the table *go_setup*.
  - 1 – This flag specifies if the metatype is depicted in the order list as a separate node. If the flag is set no separate representation will be created for the metatype.
  - 2 – The flag controls of all further children (i.e. all children except the main child) should be sub-items of the main child. If the flag is **not** set, the children will be displayed on the same level as the main child in the order list. Otherwise they will be sub-items of the main child. However, this global adjustment can be overwritten locally by GSetup & 8192 for specific children.
  - 4 – If this flag is set, 2D symbols will be created automatically. This should be used only if 2D symbols are not provided by the data explicitly.
  - 8 – If this flag is set, the standard attachment points should be applied. This can be used if there are no attachment points defined on the metatype level.
  - 16 – Usually the article number of the main child is displayed by an automatically created read-only property. By using this flag, the article number property can be hidden.
  - 32 – To suppress the display of properties changed by the filter, this flag should be set.
  - 64 – This flag controls the collision detection of children during initial placement and interactive movement. If the flag is not set there is only a check if there is already an object exactly at the insert position, during initial placement. Furthermore no collision detection is triggered during interactive placement. Otherwise, if the flag is set the typical OFML collision detection is applied in both cases.
  - 128 – Metatype objects inherit native properties from a so-called insertion object. In case that the metatype object will be a child of another metatype object, it can be controlled via this flag if the parent object or the selected object on the same topological level, should be the insertion object. If the flag is set and does such a selected object exist on the same level then this object is used to inherit the properties. Otherwise the properties are inherited from the parent object.
  - 256 – This flag enables the automatic deletion of interactively created child objects for which – due to configuration of the parent object – valid attachment points do not exist anymore. If the flag is set, a question dialog pops up in the specified case. If the dialog is answered with 'Yes' the child will be

deleted. If the flag is not set the situation is not considered at all. Note that the *GSetup* value of the specific child is relevant here.
  - 512 – If objects that have a pair of compatible attachment points, but there is no action defined for this pair, are planned side-by-side, there will be shown a message that this isn't a rule-based concatenation. Set this flag for the object to be concatenated, to suppress this message.
  - 1024 – Normally because of performance issues, only an incomplete creation of the main child is applied during the temporary creation. By setting this flag the complete creation can be enforced. This could be needed of an accurate bounding box or specific properties such as *GVarPrefix* must be accessed during the temporary creation.
  - 2048 – If this flag is set, the object does not inherit native properties. Otherwise and if it's an ancestor is also a metatype, native properties are inherited.
  - 4096 – Set this flag to turn off the collision detection that is normally applied on the level of siblings during the initial placement based on attachment points. This could be necessary if the initial collision will be removed by the object itself due to one or more actions. However, by setting this flag it is possible to place objects inside others and therefore create invalid designs.
  - 8192 – This flag enforces for a metatype M1 which is a child of M2 that M1 will be handled as a major position in the order item structure of M2 – even if M2 handles its children as sub-positions (by GSetup & 2).
  - 16384 – This flag enforces the removal and re-creation of the object's own children after modification of the related property. Otherwise, if the flag isn't set, the children stay alive but get a new position/orientation. In specific cases, removal and re-creation of children can be useful. For GO version 1.3.9 and earlier, this was the standard behavior.
  - 32768 – In former application versions the native properties (na properties) was displayed in the property editor in the context of the metatype. This kind of behavior can be enforced by setting this flag. Otherwise the na property will be displayed in the context of the (real) native properties inside the property editor. Internally this is implemented by the assignment of the property class – either the property class of the metatype or the original (native) property class.
- *GXSetup [-]* – This optional property is provided to control specific features of the metatype. The final value is composed by adding the following single values. This property is now replaced by the table *go_setup*.
  - 1 – This flag controls if during the translation of a metatype object – that is child of another metatype M – a collision detection against the other children of M should take place (1) or not (0).
  - 2 – This flag controls if during the rotation of a metatype object – that is child of another metatype M – a collision detection against the other children of M should take place (1) or not (0).
  - 4 – If you want to exclude the main child from collision detection during the GXSetup modes 1 and 2 set this flag. Otherwise collisions with the main child are considered.
  - 8 – The Interactive Feedback Mode (available with version 1.11) will be enabled (1) or disabled/is not supported (0).
- *GAlign* [-] – This optional property controls the geometric alignment of the main child, and must be specified by 3 letters. The letters will be mapped to the axes X, Y and Z (in this sequence). The following letters are supported (each of them refers to the related axis):
  - *N* – There is no alignment at all for this axis. (<n>o alignment)
  - *I* – For the related axis, the minimal dimension of the bounding volume of the main child along this axis will be mapped to the origin of the metatype. (m<i>nimum)
  - *C* – For the related axis, the center of the bounding volume of the main child along this axis will be mapped to the origin of the metatype. (<c>enter)
  - *A* – For the related axis, the maximal dimension of the bounding volume of the main child along this axis will be mapped to the origin of the metatype. (m<a>ximum)

  The default value is III – thus the object is oriented at the left, lower and back coordinate of the local orthogonal bounding volume of the main child.
  Example:
  - *NNN* – no explicit orientation
  - *CIC* – symmetrical orientation with regard to x and z axis, e.g. for chairs

  Alternatively you can set the value *ATTPT* for *GAlign*. Then the offset for the main child will be taken from the attachment point *_GO_CHILD* if defined in table *go_attpt*. Note. This way an Y rotation can be specified

too. If one of the above-mentioned alignments is used and a valid _GO_CHILD attach point does exist as well, then the attach point will be used as offset with regard to the position detected by the alignment.

- *GWidth, GHeight, GDepth* – These properties are used internally (if defined) to display a dummy object for instance, if there couldn't be created a real article from the current set of parameters. If width (height, depth) are needed as properties for any metatype then use the related variable *GWidth* (*GHeight*, *GDepth*) to represent it. Also apply a numeric value to it.
- *GMetaLabel* – This optional property defines a symbolic text resource to be displayed in the order list if the metatype represents a separate node (GSetup & 0). The value itself is given by the default value; the value must be a name without '@' and name space, defined in the specific resource file(s).
- *XHeight* – This value which is available in conditional contexts provides the current height position of the object relative to its coordinate system - which is either the global one or the one of the predecessor.
- *XChildID* – For objects that are regular children of a metatype this variable refers to the id of the metatype. Otherwise the value of the variable is NULL.
- *XIsInsObj* – With this variable INS and REM actions defined in the table *go_actions* can check if the object which is handled in the action has just been inserted (XIsInsObj == 1) or not.

go_articles

The table *go_articles* maps metatypes defined in table *go_types* to specific article numbers. Then, at run-time, it is possible to change the article number without deletion and re-creation of the object.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Manufacturer (manufacturer)	This column contains the manufacturer of the article. Example: *man*
Series (program)	This column contains the series of the article. Example: *s1*
Basic article number (article_nr)	This column identifies the article by means of its basic article number. Example: *S1TA0100*
Properties (prm_set)	This column references a parameter set (standard properties) according to the properties defined in table *go_types* for the metatype referenced by the first column. The parameter set is located in the table *go_noproperties*. Alternatively the tables *go_propindex* and *go_propmapping* can be used. Example: *pSITA0100_1*
Child-oriented properties (chprm_set)	This column references to a parameter set (child-oriented properties) according to the properties defined in table *go_types* for the metatype referenced by the first column. The parameter set is located in the *go_childprops*. It is possible to enter several keys here, separated by commas (,). Example: *chSITA0100_1*

The id *_native_* (in field *id*) is reserved and should be used if specific native articles should be excluded from the automatic Metatype detection, even if for other articles with the same basic article number, Metatype entries do exist.

go_properties

The table *go_properties* defines for each entry in table *go_articles* a complete set of parameters. Here identical entries in *go_articles* are supported as long as they refer to different sets of parameters in *go_properties*. This happens if various variants use the same basic article number. For each parameter one line is defined. Consequently a set of parameters is typically made of multiple lines.

Type ID **(id)**	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Key **(key)**	The key is used for the unique identification of a set of parameters in table *go_articles*. Typically multiple lines are provided for each key – one of them for each property. Example: *pSITA0100_1*
Name **(name)**	This entry names the property and must match exactly to a property defined in table *go_types*. Example: *GWidth, GHeightAdjust*
Value **(value)**	This column defines the value for the related property in the context of the corresponding article entry. It must be a valid value with regard to the property format defined in table *go_types*. Example: *800, @H1*
Variant code **(variant_code)**	This entry specifies that part of the variant code that is set by the following *variant value*. If this property does not affect the variant use an empty string here. Example: *ER*
Variant vaule **(variant_value)**	This column describes the partial variant code that is applied in the case of this set of parameters. Example: *NN*

go_propindex

The table *go_propindex* is an optional column-oriented representation of a subset of table *go_properties*. The number of columns in this table is specific for a given series and must correspond to the table *go_propmapping*.

Type ID **(id)**	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Key **(key)**	The key is used for the unique identification of a set of parameters in table *go_articles*. Typically multiple lines are provided for each key – one of them for each property. Example: *pSITA0100_1*
Value 1 **(value1)**	This column defines the value for the property according to the mapping table in the context of the corresponding article entry. It must be a valid value with regard to the property format defined in table *go_types*. Example: *800*
…	
Value <n> **(value<n>)**	This column defines the value for the property according to the mapping table in the context of the corresponding article entry. It must be a valid value with regard to the property format defined in table *go_types*. Example: *@H1*

go_propmapping

The table *go_propmapping* specifies for the polymorphic properties in table *go_propindex* the sequence in which these properties are assigned to the columns of table *go_propindex*. The number of columns in this table is specific for a given series and must correspond to the table *go_propindex*.

Type ID **(id)**	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Key 1 **(key1)**	This entry names the property and must match exactly to a property defined in table *go_types*. Example: *GWidth*
…	
Key <n> **(key<n>)**	This entry names the property and must match exactly to a property defined in table *go_types*. Example: *GHeightAdjust*

go_childprops

The table *go_childprops* implements the mapping of properties to the child-creation table *go_children*.

Key (key)	The key references the according entry in table *go_articles*. Example: *pSITA0100_1*
Name (name)	This entry names the property that is relevant for the creation of this sub-position. Example: *GCableSet*
Value (value)	This entry gives the property value that is relevant for the creation of this sub-position. Example: *Set2*
Child key (child_key)	This column identifies the specific child creation in table *go_children*. It is possible to enter several keys here, separated by commas (,). Example: *set2*

go_children

The table *go_children* fines the required entries to create a sub-position (child). It is also used for the interactive placement of a child at an attachment point, according to *go_attpt*. In the later mode the key is used to specify the attachment point. The geometrical values however are ignored and therefore can remain empty. Furthermore only the basic article number is considered here. Manufacturer and variant code are ignored.

Key (child_key)	This key is used for the unique identification starting at table *go_childprops*. Example: *set2*
Manufacturer (manufacturer)	This column gives the manufacturer of the article. Example: *man*
Program (program)	This column gives the program of the article. Example: *s1*
Basic article number (article_nr)	This column references to the article by means of its basic article number. **Note.** All basic article numbers that can be assigned to an article – either initially or after automatic modification – must be specified here. Example: *S1SET0100*
Variant code (variant)	Optionally, this column defines an additional variant code of the article. If the child is a metatype as well, the meta-properties of this child can also be modified this way, see *Parametrization*. Example: *[@man, @addon, [@GWidth, 1600]]*
X Position (pos_x); Y Position (pos_y); Z Position (pos_z)	These entries define the X, Y and Z position in Meters, relative to the higher-ranking object. In this context parametric values are supported in which all numeric values from table *go_types* are available. Example: *0.2; 0.72; 0.1*
X Rotation (rot_x); Y Rotation (rot_y); Z Rotation (rot_z)	These entries define the rotation around the X, Y and Z axises, in Degrees, relative to the higher-ranking object. In this context parametric values are supported in which all numeric values from table *go_types* are available. Example: *0; 90; 0*

Condition (condition)	The condition refers to the state of one or more properties of the object, and must be written in OFML syntax. The condition is considered as valid if its result is 1, or it is empty. All properties from table *go_types* can be used here. **Note.** In contradiction to other tables or columns, a '@' sign must be used for symbolic values. Examples: • *GWidth==1200* • *GConcat != @NONE* • *(GWidth >= 1000 && GConcat != @LEFT)*

go_proporder

The table *go_proporder* defines an optional position number to be used inside symbolic property lists for standard metatype properties. To use this, the explicit sorting must be enabled for the corresponding property by mode 128 in table *go_types*. The sorting is done by means of symbolic property values and that way is language-independent. The position numbers can multiply be assigned, for example, in each list the first value can get the position number 1. Even if one and the same symbolic values are used in various lists, one can ensure that the multiply be used value gets its correct location in each of the lists. To achieve this, a suitable schema of positions numbers is needed.

Position numbers must be positive numbers less than 1000. One and the same value must not exist twice in one list of values.

Values inside a list of values, for which there is no position number assigned, are placed behind values that have a position number, in undefined sequence.

Value (value)	For this value the position number is going to be defined. Example: *NONE*
Position number (number)	The position number for this property value. Example: *1*

go_propvalues

Via the table *go_propvalues* the validity of child controlling and na properties can be limited. This feature has to be enabled by using the mode 8192 of the corresponding property in table *go_types*. Property values that are not listed in this table are always valid. The assignment of values of na properties via the relations in the commercial data is not influenced by this table.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Name (name)	This entry names the property whose values have to be limited. Example: *GCableSet*
Value (value)	This entry gives the property value that has to be limited. Example: *Set2*
Condition (condition)	The condition refers to the state of one or more properties of the object, and must be written in OFML syntax. The condition is considered as valid if its result is 1, or it is empty. All properties from table *go_types* can be used here. **Note.** In contradiction to other tables or columns, a '@' sign must be used for symbolic values. Examples: • *GWidth==1200* • *GConcat != @NONE*

go_noproperties

The table *go_noproperties* defines for each entry in *go_articles* parameters that should not be taken over from the native article. This applies to native properties that already exist as metatype properties. The table will be evaluated for the display in the property editor and the inheritance from parent or ancestor during insertion.

Key (key)	The key identifies the corresponding parameter set in table *go_articles*. Typically a number of lines are defined for each key – each of them for one property. Example: *pSITA0100_1*
Name (name)	This column must refer to a property of the article. It is possible to enter several properties here, separated by commas (,). Example: *ER*

go_inhproperties

The table *go_inhproperties* defines – for the initial property inheritance - which meta-properties should be inherited from the ancestor (parent or sibling). If there are no entries for the metatype of the ancestor, all properties will be inherited.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Ancestor ID (pid)	The key references a metatype from table *go_types* (id-column), from which the subsequently following property should be inherited. Example: *desk1*
Property (property)	The name of the property to be inherited. Example: *GWidth*

go_nativeproperties

The table *go_nativeproperties* should be used to control the inheritance of native properties during the creation of the metatype in a flexible manner.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Ancestor ID (pid)	This key references a metatype from table *go_types*, or that the following entries should be evaluated in the case of inheriting the native properties. If the value is however *_ANY* then the entries apply to any ancestors as long as they are metatypes. Example: *desk1*
Mode (mode)	The mode defines if this entry specifies inclusion to or exclusion from inheritance and must have one these values: *INCL* or *EXCL*.
Identifier (identifier)	This value identifies either the property referenced by this entry, or one of the two values: *_ALL* or *_DEFAULT*. Example: *SShape*
Value 1 (value1)	Reserved for future use.
Value 2 (value2)	Reserved for future use.

The table will be interpreted as follows:

1. If for a given Type ID at least one entry with a matching Ancestor ID does exist, then all further entries with a different Ancestor ID or *_ANY* will be ignored.
   Otherwise, only entries with the Ancestor ID *_ANY* will be considered.

   All following statements refer to the resulting sub set of entries for a given Type ID
2. If one entry with Mode equal to *INCL* and Identifier equal to *_ALL* does exist, all native properties will be inherited (from the specified ancestor).
3. If 2. does not hold: If one entry with Mode equal to *EXCL* and Identifier equal to *_ALL* does exist, **no** native properties will be inherited (from the specified ancestor).
4. If 3. does not hold: If one entry with Mode equal to *INCL* and Identifier equal to *_DEFAULT* does exist, **all** native properties will be inherited (from the specified ancestor) that do not provide an entry with Mode equal to *EXCL* and matching Identifier (i.e. property name)rt.
5. If 4. does not hold: If one entry with Mode equal to *EXCL* and Identifier equal to *_DEFAULT* does exist, **all** native properties will be **excluded** from inheritance (in case of the specified ancestor) that do not provide an entry with Mode equal to *INCL* and matching Identifier (i.e. property name).
6. If 5. does not hold or there are no entries for the given TypeID or this table does not exist at all, then the behavior is as defined in 2.

Note. Native properties in the sense of the metatypes (na properties) are excluded from the inheritance on native level in any case.

go_resetnativeprops

Via the table *go_resetnativeprops* can be specified if certain native properties are reset to their default value from the commercial data when the base article code is changed. Properties that are not listed in this table keep their value.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Using '*' as id sets a default behavior for all metatypes of this series. Example: *desk1*
Property (key)	This column contains the name of the property. Example: *BOARD_WIDTH*
Trigger (trigger)	The trigger specifies that the native property only is reset if certain polymorphic properties are changed. It is possible to enter several properties here, separated by commas (,). If this entry is empty, the native property is reset by every change of the base article code. Example: *GWidth, GDepth*

go_actions

The table *go_actions* can be used to define specific actions to be called after the insertion or removal of an article, or after the configuration of an article. The basic operations are:

- Creation (*CREATE*) – after the creation and initialization of an object
- Initialization (*INI*) – explicit check of rules in case of stand-alone objects
- Insertion (*INS*) – creation, insertion, movement of an object
- Removal (*REM*) – removing, cut, movement of an object
- Configuration (*CON*) – configuration of an object
- Pair of attachment points (*AP*) – definition of a pair of attachment points
- Proxy attachment point (*PROXY*) – Definition of a proxy attachment point
- Child addition (*CH_ADD*) – Adding of a child that is also a metatype. Related actions will be called after the complete creation of the child (this includes the invocation of potential *CREATE* actions).
- Child removal (*CH_DEL*) Deletion of a child that is also a metatype
- Selection of an interactor (*INTERACTOR*) – Called when the interactor is clicked on

Note: Use mode *AP* to define a pair of attachment points if there are no explicit actions defined for *REM* and *INS*.

Generally, actions are defined for two attachment points together with modes like *REM* or *INS*. Possible actions can be:

- *SET_PROP* - Action to set a property of the article
- *ADD_CHILD* - Action to create a sub-position of the article[1]
- *DEL_CHILD* - Action to remove a sub-position of the article
- *CH_PROP* - Action to set a property of a sub-position. This is supported for all kinds of sub-positions like main sub-position, property-based sub-position and interactive sub-position – as long as the sub-position is an instance of *GoMetaType*.
- *CON_PROP* - Action to configure the state of a property of the object
- *CON_CH_PROP* Action to configure the state of a property of a sub-position
- *CON_AP* – Action to adapt an attachment point i.e. the positions of the sibling objects at this point. Only translational adaptations are supported at the moment. At one attachment point only one object can be adapted. However, the adaptation of an object can trigger the adaptation of a further object and so on.

The following table shows which actions are available for a given mode:

	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

For each action a message can be displayed to inform the user about the actions and adaptations.

[1] There is no collision detection.

During an insertion/removal process multiple actions can be triggered for an article with valid combination of own and foreign key.

For the actions all objects on the same topological level as the object that triggers the action, are considered. For a planning object, i.e. at the topmost level, all other planning objects will be involved. For a sub-position of a planning object all further sub-positions of this planning object will be considered.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Onw key (own_key)	The own key contains the attachment point of the object. Example: *vl1* **Differences:** • *CREATE, INI, CH_ADD, CH_DEL* – Here the entry is ignored. • *CON* – The name of that property – according to table *go_types (name)* - must be specified that should invoke the action after configuration. It is possible to enter several properties here, separated by commas (,). • *INTERACTOR* – The key of the interactor – according to table *go_interactors (key)*.
Foreign key (foreign_key)	The foreign key contains the attachment point of a neighbor object. Example: *tr1* **Differences:** • *CREATE, INI, CH_ADD, CH_DEL* – Here the entry is ignored. • *CON, INTERACTOR* – Here an entry from table *go_children (key)* is provided that identifies the child. It is possible to enter several keys here, separated by commas (,). If this action does not influence children, the foreign key can be empty. • *AP* – If a Metaplanning Workflow is linked via the attachment point pair, the foreign key contains the fully-qualified ID of the Workflow. Example: *::man::s1::@WorkflowAF*
Reason (direction[2])	*CREATE, INI, INS, REM, CON, AP, PROXY, CH_ADD, CH_DEL, INTERACTOR* (see above)
Condition (condition)	The condition refers to the state of one or more properties of the object itself and its neighbor object. The condition must be written in OFML syntax. The condition is true if it is empty or the evaluation returns 1. To evaluate the conditions a context is created that contains all properties of the object according to table *go_types* (this is the so-called standard context). As far as there is a second object involved (neighbor or parent/child) its state is also available (in the so-called secondary context). To distinguish standard and secondary context, the property names of the secondary context have an underscore '_' as prefix. **Note:** In contradiction to the other tables and fields the prefix '@' must be used for symbolic values. Examples: • *GWidth==1200* • *GConcat != @NONE* • *(GWidth >= 1000 && GConcat != @LEFT)* • *GDepth==_GDepth* **Specifics:** • *CH_ADD, CH_DEL* – The parent object provides the standard context. The child is represented by the secondary context. The special property *_XChildID* can be used to identify the related child.

[2] The name *direction* is used because of historical reasons. Initially it specified a directional value: moving an object away or back, resp. INS or REM).

Action **(action)**	This field defines the action to be executed. The following actions are supported: *SET_PROP*, *ADD_CHILD, DEL_CHILD*, *CH_PROP, CON_PROP, CON_CH_PROP, CON_AP* (see above)
Parameter 1 **(param_1)**	• *SET_PROP, CON_PROP* - Here the entry contains a property name. **Note:** Only properties defined in the metatype context are supported. Thus native properties are not possible. Example: *GConcat* • *ADD_CHILD, DEL_CHILD* - In this case the entry contains a name for the identification of the associated child objects. This name provides a link between related ADD_CHILD and DEL_CHILD entries. If such a name links to more than one child, these children are treated simultaneously, i.e. created and deleted. **Note:** This name must not match to those in *go_childprops*. Example: *LINK2* • *CON_AP* – Provide an attachment point used for the adaptation of sibling objects. Example: *APR*

Parameter 2
(param_2)
	• *SET_PROP, CH_PROP* – The entry must contain a property value – either an immediate value or an expression that provides a value, e.g. by deriving it from another property.
	Note: In contradiction to other tables or fields use '@' as prefix to symbolic values.
	Example: *both* (*SET_PROP, CH_PROP*)
	• *ADD_CHILD* – This field must contain a reference to an entry in table *go_children*.
	Example: *link2id*
	• *DEL_CHILD* – Further entries will be ignored.
	• *CON_*PROP* –*T he value is defined by adding the following opportunities:
	  • 1 – Changeability
	  • 2 – Visibility
	Example: The value 3 is the standard value. The property is visible and can be edited. The value 2 corresponds to a non-editable but visible property. The values 0 and 1 hide the property.
	• *CON_AP* – The corresponding attachment point must be specified here.
	Example: *APL*

Text **(text)**	This optional entry provides a text key referring to the external resource file(s). Example: *concatLeft*

go_attpt

The table *go_attpt* defines attachment points for the implementation of the actions mentioned above. **Note**. If there are no attachment points for a given type id, automatically, the attachment points of the wrapped object are used.

Type ID **(id)**	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Key **(key)**	This key marks the attachment point to be described by the following entries in the context of the *type id*. It might be useful to use the manufacturer name in the key name, however this is not required. Attachment points are used also to control the position of a metatype as sub-position at another metatype. On the one hand this is defined by a *CH* attachment point of the parent. On the other hand the origin of the child to be added can be defined. This must be done by key *O*. If *O* is not defined, *OL* or *OR* are checked depending on the location (left or right side) of the *CH* attachment point at the parent's geometry. If neither *O* nor *OL*(*OR*) are defined, (0, 0, 0) is used as local origin. 1. Translation by parent to the required position. 2. Rotation by parent in order to orientate the object, e.g. at an edge of a table 3. Translation according to *O*/*OL*/*OR*

	4. Rotation according to *O*/*OL*/*OR* Furthermore the key *_GO_CHILD* is predefined. See variable *GAlign*, value *ATTPT* for description. Example: *manvl*
Planning direction or purpose (direction)	The planning direction can be applied to use the attachment point for the placement of the objects. For this, during the initial placement, the planning direction as well as the related attachment point of the object to be placed are evaluated. The following values are supported: • L – Planning direction left->right (i.e. to the right side). The corresponding attachment point of the neighbor: R. • R – Planning direction right->left (i.e. to the left side). The corresponding attachment point of the neighbor: L. • F – Planning direction front->back (i.e. to the back). The corresponding attachment point of the neighbor: B. • B – Planning direction back->front (i.e. to the front). The corresponding attachment point of the neighbor: F. • T – Planning direction top->down (i.e. down). The corresponding attachment point of the neighbor: D. • D – Planning direction bottom->up (i.e. to the top). The corresponding attachment point of the neighbor: T. • CH – Interactive attachment point for child objects. The potential children are detected by searching for the attachment point key in table *go_children*. • CH_REL – Interactive attachment point for child objects. The attachment point moves automatically with a connected GO-type. The potential children are detected by searching for the attachment point key in table *go_children*. The GO-type has to support the category *GO_AP* and supply the key of this attachment point via the parameter *mAP*. • MP – This attachment point is linked to a Metaplanning Workflow. If none of the above-mentioned purposes applies the field remains empty. The first valid attachment point with regard to the current planning direction is used. In all modes except CH and CH_REL the new object is created on the level of siblings. Only in mode CH / CH_REL it is created on child level. **Note.** The connection of the attachment points L-R and F-B is just a tool to improve planning convenience. Because of the fact that these points exist for different series and even manufacturers there is no way to define consistency checks based on them. That means on any R point (in analogy for any other of the above-mentioned combinations) each arbitrary object can be placed as long as it has a valid L point and it does not collide. The correspondence of attachment points is defined in the table *go_actions* via the modes AP, INS or REM. **Note.** The vertical placement has lower priority compared to the horizontal placement if there are multiple pairs of matching attachment points. In the context of the Clone Attachment Points (see below) additional directions exist: I_L, I_R, I_F, I_B, I_T and I_D.
Condition (condition)	As far as a condition was specified (see above) it will be evaluated. If it is empty or its evaluation returns 1 then the attachment point exists. Otherwise not. Only the standard context is available here.
X position (pos_x) ; **Y position (pos_y) ;** **Z position (pos_z)**	These columns contain the local coordinates of the position of the attachment point, in Meters. Here parametric values are supported. All numeric properties of table *go_types* are available. Example: *0; 0.72, 0*

Y rotation (rot_y)	This entry defines the rotation around the Y axis in Degrees relative to the parent object. Here parametric values are supported. All numeric properties of table *go_types* are available. Example: *90*

A very special application of the attachment points can be used to clone objects. Such so-called Copy Attachment Points are defined by the Direction value which must be built in the following way:

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

with the following meaning:

- *C_* – constant prefix
- *<Dir>* – defines the direction and should be one of these values: L, R, F, B, T, D
- *<DirMode>* – defines a special direction mode and must be one of the following values:
  - N – Normal. The attachment point pairs are: L-R, F-B and T-D.
  - I – Inverse. The attachment point pairs are: L-I_L, R-I_R, F-I_F, B-I_B, T-I_T and D-I_D.
- *<HierMode>* – defines the hierarchy for the object to be created and can be S (Sibling) or C (Child).
- *<CloneMode>* – describes the specific cloning mode and should be one of the following:
  - B – Cloning by Basic Article Number
  - F – Cloning Cloning by Final Article Number
  - R – Recursive cloning incl. (grand) children

Example: *C_RNSR* – The object will be cloned recursively to the right side. The clone will be created at the same hierarchical level as the original; so both objects are siblings.

All copy attachment points should use the geometry ::ofml::go::GoIGeometryCubes.

go_attptgeo

The table *go_attptgeo* is used to modify the predefined geometrical representation of attachment points.

Key (key)	The key identifies an attachment point in the table *go_attpt*.
Type ID (id)	The key references a metatype from the table *go_types.* If this entry is empty then it applies to all metatypes with a corresponding attachment point. Example: *desk1*
X position (pos_x) ; Y position (pos_y) ; Z position (pos_z)	These columns define a local offset for the geometry with regard to the nominal coordinates of the attachment point, in Meters. Example: *0; 0.72;0*
Rotation direction (rot_dir)	The geometry can be rotated around one axis with regard to the nominal attachment point. The axis can be specified as follows: • N – No rotation at all. • X – Rotation around X axis. • Y – Rotation around Y axis. • Z – Rotation around Z axis.
Rotation (rot)	This entry specifies the rotation around the given axis relatively to the nominal attachment point, in Degrees. Example: *90*
OFML-Typ (type)	Here you should enter a fully-qualified OFML type for the representation of the geometry. This can be a predefined type (see page 31), or a user-defined one. Example: *::ofml::go::GoIGeometryHArrow*

Parameter 1 (arg1); Parameter 2 (arg2); Parameter 3 (arg3)	Up to three parameters can be used to parametrize the geometry. The interpretation of them depends on the specific type.

go_attptsorder

The table *go_attptsorder* is used to specify the order of processing of attachment points.

Key (key)	The key identifies an attachment point in the table *go_attpt*.
Type ID (id)	The key references a metatype from the table *go_types*. If this entry is empty then it applies to all metatypes with a corresponding attachment point. Example: *desk1*
Planning direction (plan_dir)	This entry contains the global planning direction, for which this order should be used. Example: *R*
Position number (number)	The position number for the attachment point. Example: *1*

go_childmoving

The table *go_childmoving* defines the movability of a child object in terms of translation and rotation behavior. If for an object no translation entries have been defined, the object cannot be translated. In analogy, if for an object no rotation entries have been defined, the object cannot be rotated. The child object has to be a Metatype.

Furthermore the table can be used to control the indirect movement of Metatype children caused by movement of other children. This mode will be called **I-Mode** in the following.

Type ID (id)	This key references a metatype from table *go_types* (id-column). **Note.** This must refer to the metatype of the object's parent. Example: *desk1*
Key (key)	This key marks the child with respect to the attachment point used for its creation (*go_attpt[key]*) **I-Mode**: In case the child is a metatype, the concatenation (by underscore) of the child ID (in the context of the parent) and the Metatype ID of the child is entered here. Otherwise, either the basic article number in case of interactive children, or the property name plus child counter (starting with 1) in case of property-based children must be entered.
Condition (condition)	The condition is true if it is empty or the evaluation returns 1. The condition will be evaluated in the context of the child object. **I-Mode**: The condition will be evaluated in the context of the parent. If the child is a Metatype, the child's context will be additionally provided as secondary context.
Mode (*mode*)	This mode specified the kind of moving for the subsequently following command. These modes are supported: • *XT* – Translation along X axis • *YT* – Translation along Y axis • *ZT* – Translation along Z axis • *XZTLINEAR* – lineare translation in X-Z plane

	• *XZTRANGE* – area translation in X-Z plane • *XR* – Rotation around X axis • *YR* – Rotation around Y axis • *ZR* – Rotation around Z axis The elementary translations *XT*, *YT* and *ZT* can be combined without any restriction. *XZTLIN* and *XZTRANGE* must not be combined with other translations affecting the X or Z axis. Only one of the three rotations *XR*, *YR* and *ZR* can be used. **I-Mode**: The mode controls the enable of the axes for the position modification. The first valid (in terms of a true condition) entry will be applied, all others ignored. If there isn't a valid entry, all axes are enabled. • *IN* – All axes are disabled. • *IX, IY, IZ, IXY, IXZ, IYZ, IXYZ* – The corresponding axes are enabled (X, Y, Z, XY, XZ, YZ, XYZ). Additionally the position / rotation of GO-Types inside the object tree can be accessed via the mode *GO_AP_POSROT*. The GO-Type has to support the category GO_AP and deliver a key for identification by the parameter *mAP*. In all Metatype tables the method *GO_AP_POSROT(pAPKey, pMode)* is available. *pAPKey* contains the key corresponding with the parameter *mAP* of the GO-Type. *pMode* has to be one of these values: • *@POS_X* • *@POS_Y* • *@POS_Z* • *@ROT_X* • *@ROT_Y* • *@ROT_Z* Example: `GO_AP_POSROT(@MF_RW, @POS_Y)` delivers the Y position of the GO-Type with the ID @MF_RW.
Command (command)	The command entry defines the parameters for the related movement. For translations the values must be specified in Meters; for rotations in Degrees. These modes are available: *XT*, *YT* , *ZT* and *YR*: • *MIN* – Minimal position. If the position falls below this value it will be reset to this value, even if it doesn't fit the raster or an explicit position. • *MAX* – Maximal position. If the position exceeds this value it will be reset to this value, even if it doesn't fit the raster or an explicit position). • *RASTER* – Rasterization of the movement. The closest position within the valid range or one of the range boundaries itself will be selected. • *RAS_OFFS* – Offset for rasterization of the movement. • *POS* – Explicit position. Via this entry (or a multitude of such entries) explicit positions can be specified. The closest position within the valid range of one of the range boundaries itself will be selected. *POS* and *RASTER* must not be used in parallel for a given axis. If there is no command and no further command follows for the specified axis, free movability is enabled.

	For mode *XZTLINEAR* the following commands must be called: • *X1* – X position of the first point of the straight line • *Z1* – Z position of the first point of the straight line • *X2* – X position of the second point of the straight line • *Z2* – Z position of the second point of the straight line Here X1 must not be equal to X2. The same applies to Z1 and Z2. For mode *XZTRANGE* the following commands must be called: • *X1* – X position of the first point of the movement range • *Z1* – Z position of the first point of the movement range • *X2* – X position of the second point of the movement range • *Z2* – Z position of the second point of the movement range **I-Mode**: Unused.
Parameter (*parameter*)	This parameter refers to the specified command and is measured in Meter for translations and Degree for rotations. Parametric values are possible. Here both the child and the parent context are available. To avoid naming conflicts the parameters of the parent get the prefix '_'. **I-Mode**: Unused.

go_freenumeric

The table *go_freenumeric* specifies the parameters for the so-called free properties. In order to support re-usability, the free properties are not bound to the metatypes in table *go_types*.

Name (name)	This entry provides the name of the free property. Example: *GDeskPos*
Format (format)	The following subset of the set of OFML property formats is available: • *i* – Integer • *f* – Floating point • *L* – as *f* but display as length value with variable measuring unit • *A* – as *f* but display as angle with variable measuring unit
Minimum (minimum)	If this field is not empty it defines the minimum of the range of values. Example: *1.0*
Maximum (maximum)	If this field is not empty it defines the maximum of the range of values. Example: *2.0*
Raster (raster)	If this field is not empty it defines the raster for the range of values. The reference for the raster is the minimum value if defined, otherwise the local origin of the metatype. Please note that minimum and maximum have higher priority than the raster. Thus the raster can be violated to fit the range boundaries. Example: *0.1*
Expression (expr)	If this field is not empty it should contain an expression that derives a numeric value from the current value of the related value (coordinate or orientation), e.g. by adding an offset. If it is empty the value itself is returned as result. Inside the expression the value is available as variable V. Numeric expressions must be specified according to ANSI-C. In contradiction to other expressions no parametric values are supported here. Before the value is passed to the evaluation, raster and range correction are applied if defined. Example: *0.5*V*
Child (child)	If this field is not empty it must provide the child object to which the result of the expression should be applied. Example: *e1.goplate.e1*

Mode **(mode)**	If entry child is not empty this entry must specify in what way the result of the expression should be used: • POS_X – Set the local X position of the named child. • POS_Y – Set the local Y position of the named child. • POS_Z – Set the local Z position of the named child. Example: *POS_X*

Note: During the assignment of the default value there is no further functionality at all. For example neither the geometrical range nor the raster will be checked. Also, because of technical reasons it is not possible to contribute the initial assignment to child objects.

go_metainfo

The table *go_metainfo* defines additional information to be provided for metatypes. This kind of information is used for the implementation of further functionality and algorithms such as the automatic placement of accessories.

Type ID **(id)**	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Mode **(mode)**	This entry specifies the associated mode. The following modes are supported: • *AutoDecoration* – automatic placement of accessory • *AccCategory*[3] – categories of accessories, definition or support
Width **(width)**	Here the reference width to be used by the algorithm must be specified (in Meter). The usual variable or numeric terms are supported. The result must be a numeric value. Example: *1.0* For *AccCategory* this entry is ignored.
Height **(height)**	Here the reference height to be used by the algorithm must be specified (in Meter). Example: *1.0* For *AccCategory* this entry is ignored.
Depth **(depth)**	Here the reference depth to be used by the algorithm must be specified (in Meter). Example: *1.0* For *AccCategory* this entry is ignored.
Condition **(condition)**	As long as a condition (see above) was specified it will be evaluated. If it is empty or the result is 1 this raw will be evaluated further; otherwise not. For *AccCategory* this entry is ignored.

Value 1
(value_1)
	This entry defines a parameter that will be interpreted depending on the specific algorithm. The following modes are currently supported:
	• *AutoDecoration* – Definition of a semantic domain, e.g. *LivingRoom*, *HomeOffice*
	• *AccCategory:*
	  ○ *child* – The metatype belongs to this category.
	  ○ *parent* – The metatype accepts accessory objects belonging to this category.

Value 2 **(value_2)**	This entry defines a parameter that will be interpreted depending on the specific algorithm. The following modes are currently supported: • *AutoDecoration* – Definition of a template identifier for the semantic domain as written above, e.g. *living::table::standard* or *office::desk::std* • *AccCategory:* • *TOP_ELEM* – Placement on top of objects. Note. *TOP_ELEM* is predefined for *parent*.

[3] The old tag *acat* is supported for a certain time period for compatibility.

The *AutoDecoration* entries are not only provided for Metatypes, but can be used also for (children of) children, e.g. inside ODB blocks. In this case consider the following:

- The mapping is done by type *::ofml::go::GoAccParameters*, that requires the following arguments:
  - Identification (String) – Same as the Metatype identification.
  - Width, Height, Depth (numerical values in Meter)
- For the calculation of parametric values and conditions a context is generated (similar to the Metatypes) that provides the numerical values mentioned above in terms of the local variables *W, H* and *D*.
- If the entries for Width (Height, Depth) in table *go_metainfo* are empty, *W* (*H, D*) is used automatically.

go_feedback

The table *go_feedback* defines valid positions for the child to be inserted for use in the interactive feedback mode.

Type ID (id)	This key references a metatype from table *go_types* (id-column). **Note.** This must refer to the metatype of the object's parent. Example: *desk1*
Child Article Number (ch_artnr)	Here the article number of the child to be created must be inserted.
Attach Point (attpt_key)	This key marks the child with respect to the attachment point used for its creation (*go_attpt[key]*)
Condition (condition)	If a condition (see above) is specified here, it will be evaluated. For result 1 (or if the condition is empty) this column is considered further; otherwise not. The condition is evaluated in the context of the parent object.
Mode (mode)	The available modes are the same as documented in table *go_childmoving*. For the commands *POSX, POSY, POSZ* and *ROTY* (see below) this entry must be empty.
Command (command)	By using command entry, parameters can be specified for the related movement. The available commands are described in table *go_childmoving*. Additionally for each combination of type id, child article number and attach point the following commands can be set: • *POSX* – X position of the local origin • *POSY* – Y position of the local origin • *POSZ* – Z position of the local origin • *ROTY* – Y rotation of the local origin 0.0 is used if no value is entered here.
Parameter (parameter)	The parameter refers to the command and must be specified in Meter for translations or in Degree in case of rotations. Parametric values are possible. Only the parent context is available, i.e. the child context is not available.

go_classes

The table *go_classes* maps a metatype defined in table *go_types* to a specific OFML class that must be derived from *::ofml::go::GoMetaType.* As this mapping is only needed in specific cases it is optional.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Using '*' as id sets a class as default type. Example: *desk1*
OFML class (class)	This entry must contain the corresponding fully qualified OFML class. Example: *::ofml::go::GoDesk1*

go_propclasses

The table *go_propclasses* maps a property defined in table *go_types* to a specific property class. This way the properties can be structured. This mapping is optional. If a property is not mapped here, it belongs to the property class MetaProperties ("Meta-Properties").

Type ID (id)	This key references a metatype from table *go_types* (id-column). This key is optional. If it is left empty, this assignment is valid for all metatypes. Example: *desk1*
Property Name (prop_name)	This key references a property in table *go_types*, column *name*. Example: *GWidth*
Property Class (prop_class)	This entry must contain the corresponding property class. The value must be a name without '@'. The corresponding text is defined in the specific resource file(s). Example: *Dimensions*

go_setup

Via the table *go_setup* it is possible to control special properties of a metatype defined in the table *go_types*. This table replaces the special properties *GSetup* and *GXSetup*. For the time being both concepts can coexist equally.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Property (key)	This column contains the name of the property. For a list of possible properties see below. Example: *NoMTOrderRep*
Value (value)	The value of the special property can be set here. This is optional since most properties can only have the values "selected" (1) or "not selected" (0). Example: *1*

The following properties are defined:

- *NoMTOrderRep* – This flag specifies if the metatype is depicted in the order list as a separate node. If the flag is set no separate representation will be created for the metatype. Equivalent to GSetup & 1
- *ChildOrderRep* – The flag controls of all further children (i.e. all children except the main child) should be sub-items of the main child. If the flag is not set, the children will be displayed on the same level as the main child in the order list. Otherwise they will be sub-items of the main child. However, this global adjustment can be overwritten locally by *ChildSubPos* for specific children. Equivalent to GSetup & 2
- *Gen2DSymbol* – If this flag is set, 2D symbols will be created automatically. This should be used only if 2D symbols are not provided by the data explicitly. Equivalent to GSetup & 4
- *UseStdAttPts* – If this flag is set, the standard attachment points should be applied. This can be used if there are no attachment points defined on the metatype level. Equivalent to GSetup & 8
- *HideOrderNo* – Usually the article number of the main child is displayed by an automatically created read-only property. By using this flag, the article number property can be hidden. Equivalent to GSetup & 16
- *HideFilterMsg* – To suppress the display of properties changed by the filter, this flag should be set. Equivalent to GSetup & 32

- *CollCheck* – This flag controls the collision detection of children during initial placement and interactive movement. If the flag is not set there is only a check if there is already an object exactly at the insert position, during initial placement. Furthermore no collision detection is triggered during interactive placement. Otherwise, if the flag is set the typical OFML collision detection is applied in both cases. Equivalent to GSetup & 64
- *InheritSelObj* – Metatype objects inherit native properties from a so-called insertion object. In case that the metatype object will be a child of another metatype object, it can be controlled via this flag if the parent object or the selected object on the same topological level, should be the insertion object. If the flag is set and does such a selected object exist on the same level then this object is used to inherit the properties. Otherwise the properties are inherited from the parent object. Equivalent to GSetup & 128
- *DelChildrenAP* – This flag enables the automatic deletion of interactively created child objects for which – due to configuration of the parent object – valid attachment points do not exist anymore. If the flag is set, a question dialog pops up in the specified case. If the dialog is answered with 'Yes' the child will be deleted. If the flag is not set the situation is not considered at all. Note that the GSetup value of the specific child is relevant here. Equivalent to GSetup & 256
- *HideSideBySideMsg* – If objects that have a pair of compatible attachment points, but there is no action defined for this pair, are planned side-by-side, there will be shown a message that this is no rule-based concatenation. Set this flag for the object to be concatenated, to suppress this message. Equivalent to GSetup & 512
- *InitTmpChild* – Normally because of performance issues, only an incomplete creation of the main child is applied during the temporary creation. By setting this flag the complete creation can be enforced. This could be needed of an accurate bounding box or specific properties such as *GVarPrefix* must be accessed during the temporary creation. Equivalent to GSetup & 1024
- *NoInheritNA* – If this flag is set, the object does not inherit native properties. Otherwise and if it's an ancestor is also a metatype, native properties are inherited. Equivalent to GSetup & 2048
- *NoInitCollCheck* – Set this flag to turn off the collision detection that is normally applied on the level of siblings during the initial placement based on attachment points. This could be necessary if the initial collision will be removed by the object itself due to one or more actions. However, by setting this flag it is possible to place objects inside others and therefore create invalid designs. Equivalent to GSetup & 4096
- *ChildSubPos* – This flag enforces for a metatype M1 which is a child of M2 that M1 will be handled as a major position in the order item structure of M2 - even if M2 handles its children as sub-positions by *ChildOrderRep*. Equivalent to GSetup & 8192
- *DelChildrenPropsChange* – This flag enforces the removal and re-creation of the object's own children after modification of the related property. Otherwise, if the flag is not set, the children stay alive but get a new position/orientation. In specific cases, removal and re-creation of children can be useful. For GO version 1.3.9 and earlier, this was the standard behavior. Equivalent to GSetup & 16384
- *PropClassNA* – In former application versions the native properties (na properties) was displayed in the property editor in the context of the metatype. This kind of behavior can be enforced by setting this flag. Otherwise the na property will be displayed in the context of the (real) native properties inside the property editor. Internally this is implemented by the assignment of the property class – either the property class of the metatype or the original (native) property class. Equivalent to GSetup & 32768
- *CollCheckTranslate* – This flag controls if during the translation of a metatype object – that is child of another metatype M – a collision detection against the other children of M should take place (1) or not (0). Equivalent to GXSetup & 1
- *CollCheckRotate* – This flag controls if during the rotation of a metatype object – that is child of another metatype M – a collision detection against the other children of M should take place (1) or not (0). Equivalent to GXSetup & 2
- *NoCollCheckMC* – If you want to exclude the main child from collision detection using *CollCheckTranslate* or *CollCheckRotate* set this flag. Otherwise collisions with the main child are considered. Equivalent to GXSetup & 4

- *Feedback3D* – The Interactive Feedback Mode (available with version 1.11) will be enabled (1) or disabled/is not supported (0). Equivalent to GXSetup & 8
- *CollCheckMC* – Flag to control the collision behavior of the main child
  0 -> normal collision detection
  1 -> no collision detection
  2 -> no center point test
- *HideDelChildrenAPMsg* – Controls the user dialog when during an automatic deletion of interactively places add-ons (see *DelChildrenAP*):
  0 -> Dialog before deleting each add-on
  1 -> Summary (n add-ons have been deleted)
  2 -> No dialog
- *ShowPolyPropFilterMsg* – The adaptation of the property during filter processes is shown in a dialog window. If this flag is set, the output of this message can be suppressed for a property via the property mode 64 in the table *go_types*.
- *DisableAutoChildReposition* – Controls the repositioning of children after position modifications:
  0 -> Children are repositioned; a detailed control can be done via the I-Modes in the table *go_childmoving*.
  1 -> Children are never repositioned; any I-Mode set in the table *go_childmoving* is ignored.
  2 -> Children are not repositioned in general; via I-Modes in the table *go_childmoving* the repositioning can be activated for specific children.
- *SumSubArticlePrices* – This flag controls if the prices of sub positions are displayed individually (SumSubArticlePrices == 0) or summarized (SumSubArticlePrices == 1). The value can be calculated using the state of one or more properties of the object, and must be written in OFML syntax.
- *UseMCAxis* – Use the translation / rotation axis of the main child.

go_texts

The table *go_texts* contains texts for the resources used with the metatypes.

Resource key (key)	The key references the resource the text should be used for. Usually this is a property key. Example: *GWidth*
Language (language)	This column contains the double-digit ISO language code (ISO–639). This column can be left empty for language-independent texts. Language-dependent texts are prioritized. Example: *en*
Text (text)	This column contains the actual text. Example: *Width*

If the key *uft8* is set in the table *go_info*, the values in column *Text* are encoded in UTF-8 character set. Optionally, the byte order mark can be specified at the beginning of the file. The normal form should be NFC (Normalization Form Canonical Composition).

If this key is not set, the texts are encoded in the character set of the system's codepage.

go_itemplates

The table *go_itemplates* is used to specify which ITemplate has to be applied to the Metatype defined in table *go_types*.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Using '*' as id sets a default ITemplate. Example: *desk1*
ITemplate (template)	Here the fully-qualified name of the ITemplate has to be supplied. This can be a predefined type (see page 35), or a user-defined one. Example: *::ofml::go::IT_Desk1*
Condition (condition)	As far as a condition was specified (see above) it will be evaluated. If it is empty or its evaluation returns 1 then the attachment point exists. Otherwise not. Only the standard context is available here.
Parameter (parameter)	This vector of parameters can be used to parametrize the ITemplate. The interpretation of them depends on the specific type. Each parameter has to be separated by a comma. Example: *@LB, @RB*
X position (pos_x) ; Y position (pos_y) ; Z position (pos_z)	These columns define a local offset for the geometry with regard to the nominal coordinates of the ITemplate, in meters. Example: *0; 0.72; 0*
Y rotation (rot_y)	This entry specifies the rotation around the y-axis relatively to the nominal attachment point, in Degrees. Example: *90*

go_interactors

The table *go_interactors* is used to define application interactors[4] on a Metatype.

Type ID (id)	This key references a metatype from table *go_types* (id-column). Example: *desk1*
Interactor type (type)	• *SELECT* – To select a child article • *ACTION* – To trigger an action defined in the table *go_actions*. • *RESIZE*[5] – To start an interactive resize operation[6]. • *METHOD* – To call an OFML method.

Key (key)	This key marks the attachment point to be described by the following entries in the context of the type id. It might be useful to use the manufacturer name in the key name, however this is not required. The meaning of the key depends on the interactor type:
	• *SELECT* – The key described the child article to be selected. It references to the table *go_children (key)*.
	• *ACTION* – The key references to the table *go_actions (own_key)*. When the interactor is activated, all INTERACTOR actions defined with this key are triggered.
	• *RESIZE* – The key represents a vector consisting of:
	  ◦ 1. fully qualified Metaplanning-Workflow-ID (String)
	  ◦ 2. fully qualified Metaplanning class (String)

[4] See application note AN-2013-001: "Application Interactors"
[5] This type is available on GO version 1.17.3 and later
[6] See GO IV (MP) - OFML Metaplanning – Concepts and Tables

	• 3. Flag indication whether the article graphics should be hidden during the interaction (0 | 1) • *METHOD* – The key is a vector consisting of: • 1. Method name including arguments (String) • 2. mode (Int) The following values are supported: • 0 – Method is called on top scene element. • 1 – Method is called on target object of the interaction. • 2 – Method is called on next selectable object to the object hierarchy starting with the target object. Example: *int_add_child*
Condition (condition)	The condition refers to the state of one or more properties of the object, and must be written in OFML syntax. The condition is considered as valid if its result is 1, or it is empty. All properties from table *go_types* can be used here. Note. In contradiction to other tables or columns, a '@' sign must be used for symbolic values. Examples: • *GWidth==1200* • *(GWidth >= 1000 && GConcat != @LEFT)*
X position (pos_x) ; Y position (pos_y) ; Z position (pos_z)	These columns define a local offset for the interactor with regard to the nominal coordinates of the reference object, in meters. Example: *0; 0.72;0*
Interactor symbol (image)	Via this column a specific image for the interactor symbol can be specified. The directory path has to be relative to the data directory of the manufacturer. The file name must be specified without a suffix for the file type. Via a prefix the type of the interactor symbol has to be specified: • *@IMAGE* – PNG as a graphics format is presupposed. The image should be transparent (i.e. contain an alpha channel). 2 files are expected for the symbol: one for the normal display, and one for display as active interactor. Both variants are distinguished on the basis of the required suffixes *_normal* resp. *_hot* (which are appended to the name specified here in this element). Example: *@IMAGE:/basics/ANY/1/mat/interactorChild*
Hint (hint)	This optional entry provides a text key referring to the external resource file(s). The text is shown as a tooltip as soon as the mouse pointer is over the interactor. Example: *concatLeft*

3 Parametric values

The fields of the above tables that contain the term 'parametric values' may use a parametric specification as described in the following:

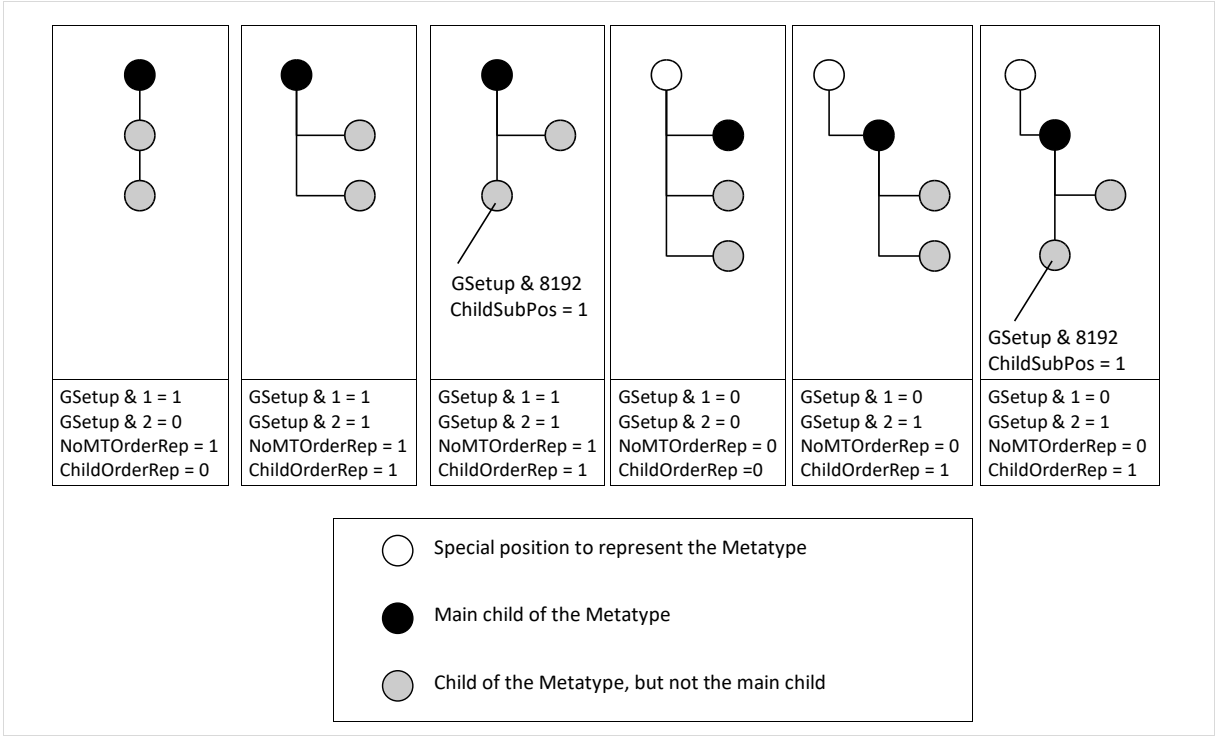
- The result of the expression must be a numeric value.
- The expression can contain all mathematical constants, functions and operations, as well as arithmetic and logical operators, as defined in the OFML standard.
- Those properties defined in table *go_types* that represent a numeric value are available as predefined variables under their name defined in *go_types*.

The result of the expression must match the format of the corresponding table entry. In most cases this is Meter (m) for positions and Degree (deg) for rotations. If for instance the parameter GWidth is given in Millimeter it must be multiplied by 0.001.

By using the optional file *go_context.ofml* a specific context for parametrization can be provided that will be the initial part of any evaluation context. This may be used to define constants (based on OFML variables) and procedures according to OFML basic language (OFML 2.0, Part III). Therefore readability and efficiency of parametric expressions can be increased. Note that this context is integrated into any evaluation context. Thus limit as much as possible.

4 Basket order position control

The property *GSetup* provides the relevant features to control the positions of articles and sub-articles in the basket. The following table shows the most-important scenarios.



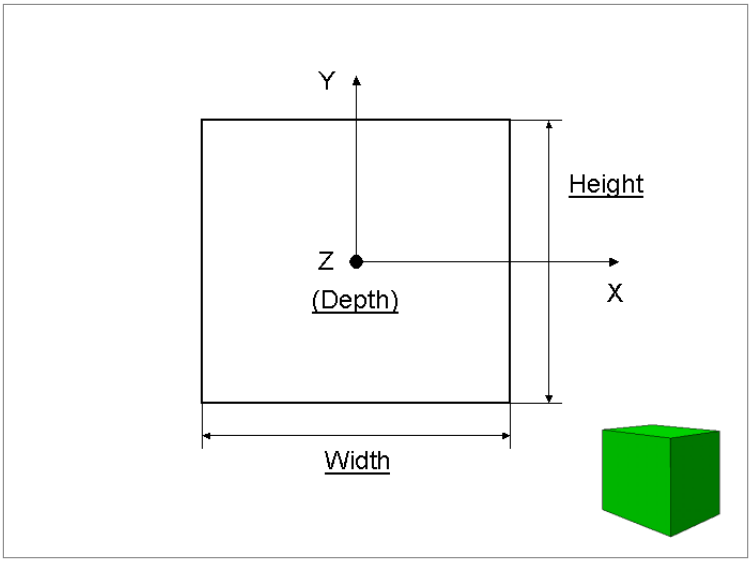
GSetup & 1 = 1 GSetup & 2 = 0 NoMTOrderRep = 1 ChildOrderRep = 0	GSetup & 1 = 1 GSetup & 2 = 1 NoMTOrderRep = 1 ChildOrderRep = 1	GSetup & 1 = 1 GSetup & 2 = 1 NoMTOrderRep = 1 ChildOrderRep = 1	GSetup & 1 = 0 GSetup & 2 = 0 NoMTOrderRep = 0 ChildOrderRep =0	GSetup & 1 = 0 GSetup & 2 = 1 NoMTOrderRep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

5 Predefined Interactor geometries

::ofml::go::GoIGeometryBlock

This geometry implements a centered block. The parameters are:

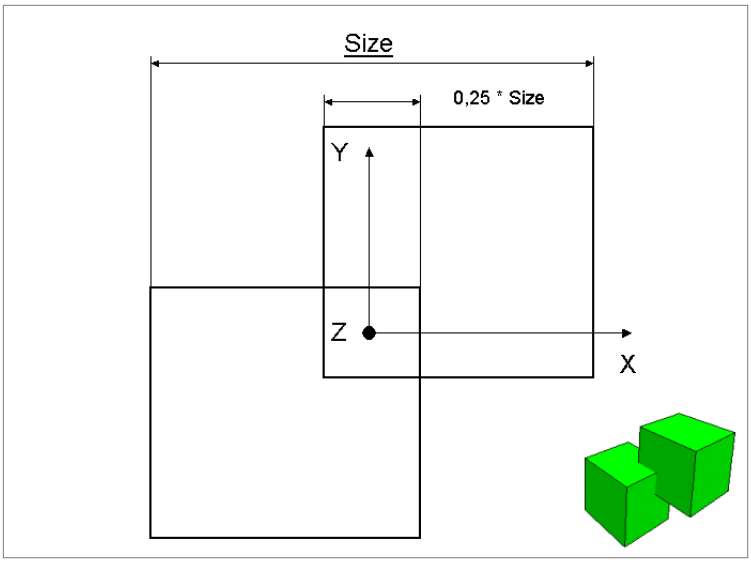
1. *Width* – The overall width of the block. A positive numeric value must be assigned. Otherwise 0.7 is used.
2. *Height* – The overall height of the block. A positive numeric value must be assigned. Otherwise 0.07 is used.
3. *Depth* – The overall depth of the block. A positive numeric value must be assigned. Otherwise 0.07 is used.



::ofml::go::GoIGeometryCubes

This geometry implements two telescoped cubes. The parameters are:

1. *Size* – The overall size of the object. A positive numeric value must be assigned. Otherwise 0.1 is used.
2. *Parameter2* – A numerical value must be entered e.g. 0.0, that will not be used any further however.
3. *Parameter3* – A numerical value must be entered e.g. 0.0, that will not be used any further however.

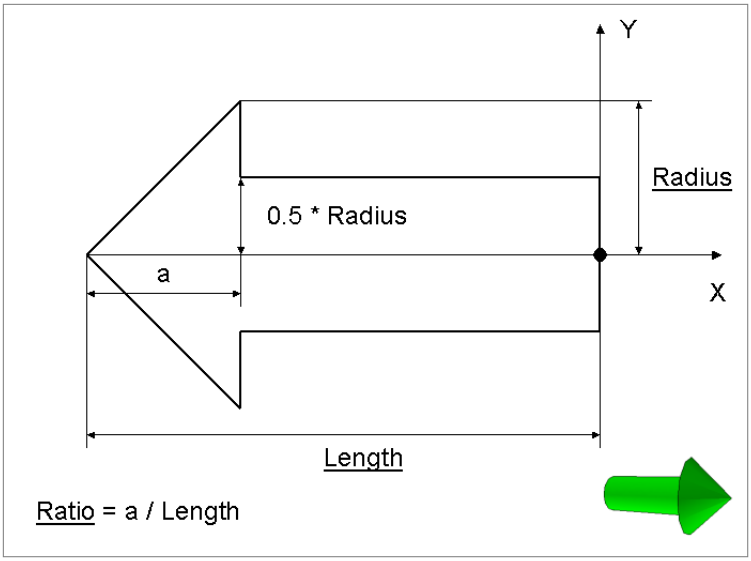


This geometry is reserved for Copy Attachment Points.

::ofml::go::GoIGeometryHArrow

This geometry implements an arrow around the x axis. The parameters are:

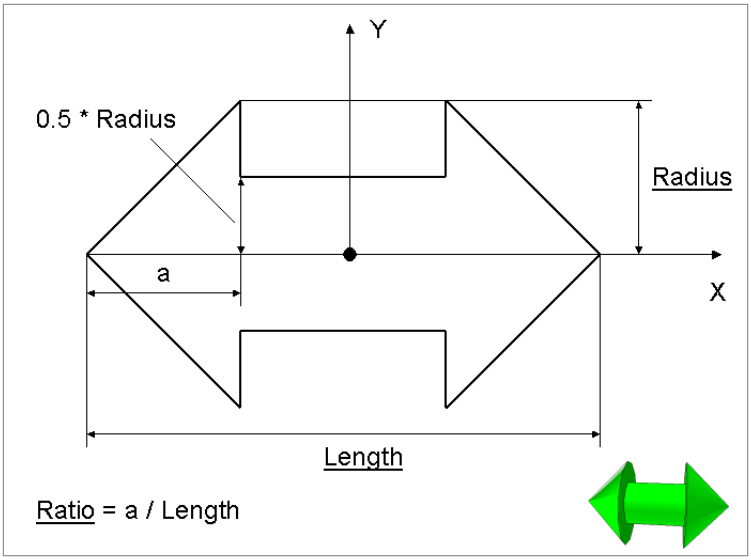
1. *Length* – The overall length of the arrow. A positive numeric value must be assigned. Otherwise 0.1 is used.
2. *Radius* – The maximal radius of the arrow. A positive numeric value must be assigned. Otherwise 0.03 is used.
3. *Ratio* – The ratio between arrow head and overall length. A positive numeric value less than 1.0 must be assigned. Otherwise 0.3 is used.



::ofml::go::GoIGeometryHDArrow

This geometry implements a double arrow around the x axis. The parameters are:

1. *Length* – The overall length of the arrow. A positive numeric value must be assigned. Otherwise 0.1 is used.
2. *Radius* – The maximal radius of the arrow. A positive numeric value must be assigned. Otherwise 0.03 is used.
3. *Ratio* – The ratio between arrow head and overall length. A positive numeric value less than 0.5 must be assigned. Otherwise 0.3 is used.



This geometry is reserved for Scaling Attachment Points.

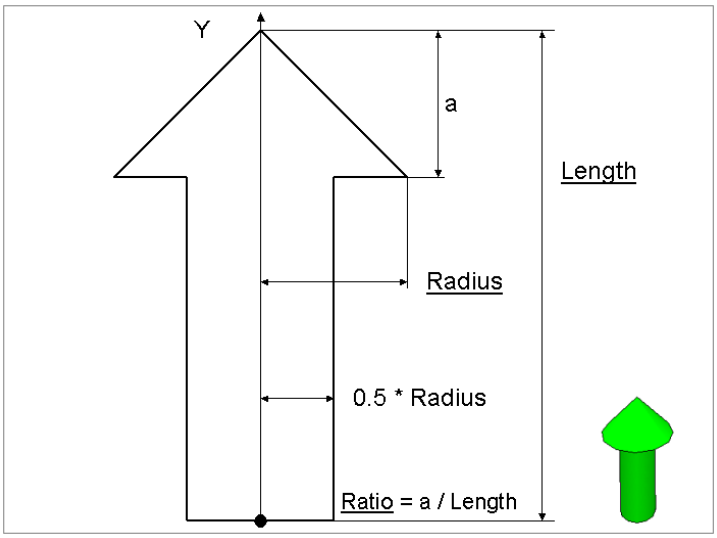
::ofml::go::GoIGeometryInvisible

This geometry implements an invisible attachment point. There are no parameters.

::ofml::go::GoIGeometryVArrow

This geometry implements an arrow around the y axis. The parameters are:

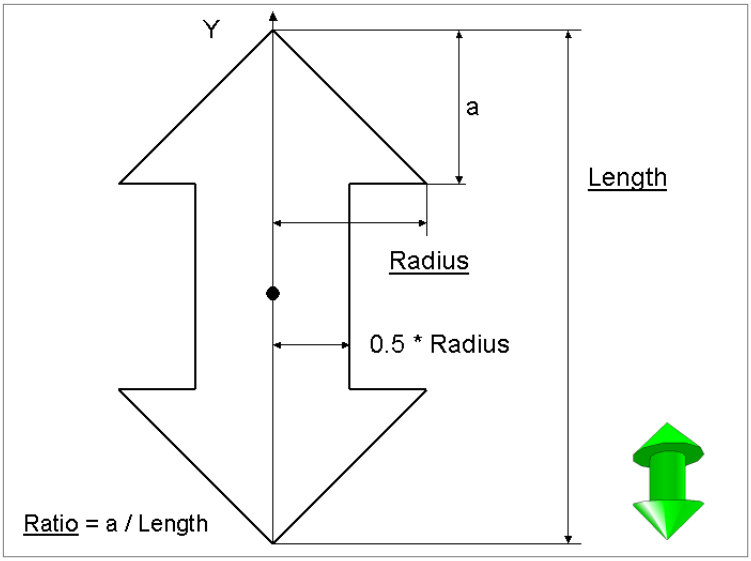
1. *Length* – The overall length of the arrow. A positive numeric value must be assigned. Otherwise 0.1 is used.
2. *Radius* – The maximal radius of the arrow. A positive numeric value must be assigned. Otherwise 0.03 is used.
3. *Ratio* – The ratio between arrow head and overall length. A positive numeric value less than 1.0 must be assigned. Otherwise 0.3 is used.



::ofml::go::GoIGeometryVDArrow

This geometry implements a double arrow around the y axis. The parameters are:

1. *Length* – The overall length of the arrow. A positive numeric value must be assigned. Otherwise 0.1 is used.
2. *Radius* – The maximal radius of the arrow. A positive numeric value must be assigned. Otherwise 0.03 is used.
3. *Ratio* – The ratio between arrow head and overall length. A positive numeric value less than 0.5 must be assigned. Otherwise 0.3 is used.



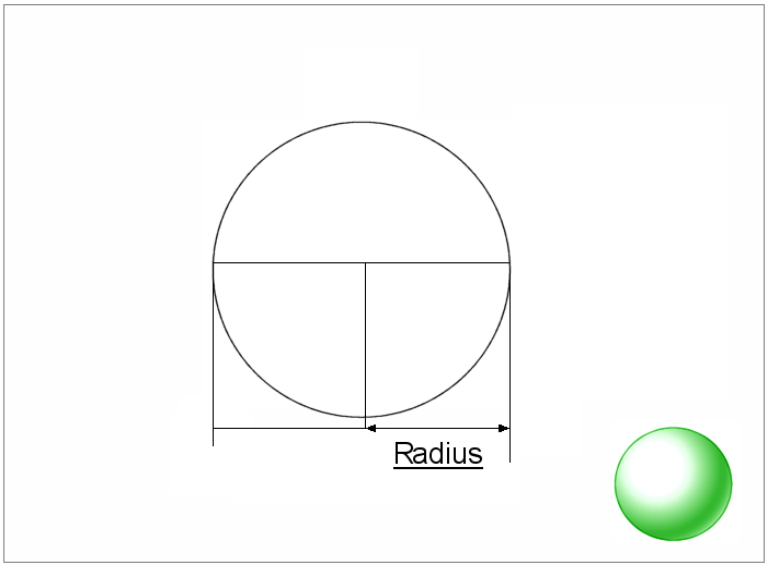
This geometry is reserved for Scaling Attachment Points.

::ofml::go::GoIGeometrySphere

This geometry implements a centered sphere. The parameters are:

1. *Radius* – The radius of the sphere. A positive numeric value must be assigned. Otherwise 0.07 is used.

The parameters 2 and 3 are ignored and can be left empty.



6 Predefined ITemplates

::ofml::go::IT_Standard

This template is used when no template is assigned explicitly.

::ofml::go::IT_Standard2

For two aligned plannings on each side.

::ofml::go::IT_Cabinet1

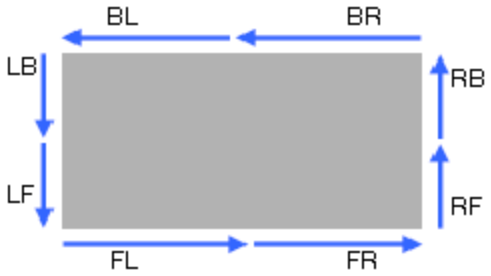
For cabinets/desks with back side-aligned planning.

::ofml::go::IT_Cabinet2

For cabinets with both front and back side-aligned planning.

::ofml::go::IT_Rectangle

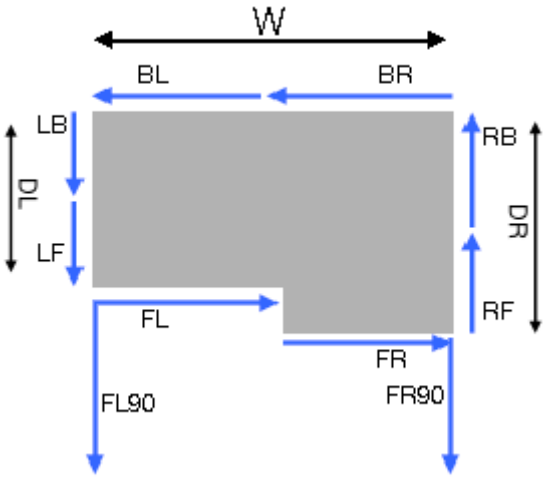
For up to two aligned plannings on each side. Via the parameter vector each of the possible aligned plannings [@BL, @BR, @FL, @FR, @LB, @LF, @RB, @RF] can be enabled. If this parameter is left empty, all eight aligned plannings are available.



::ofml::go::IT_Rectangle2

For up to two aligned plannings on each side. The parameters include the depths DL and DR in meters as well as a vector containing all possible aligned plannings [@BL, @BR, @FL, @FR, @LB, @LF, @RB, @RF, @FL90, @FR90]. If this vector is left empty, all eight aligned plannings are available. An additional optional parameter specifies the width of the object.

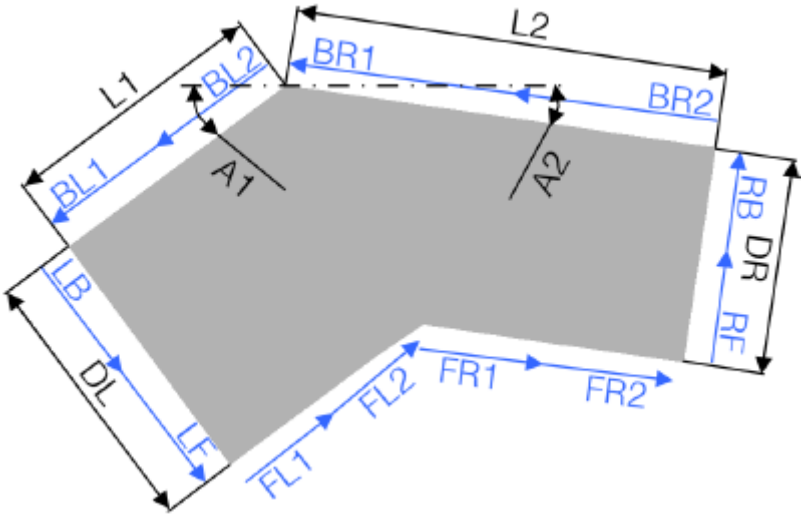
The order of the parameters must not be changed.



::ofml::go::IT_Angular

For angular elements with up to two aligned plannings on each side. The parameters include the leg lengths L1 and L2, the depths DL and DR in meters, the angles A1 and A2 in degrees as well as a vector containing all possible aligned plannings [@BL1, @BL2, @BR1, @BR2, @FL1, @FL2, @FR1, @FR2, @LB, @LF, @RB, @RF]. If this vector is left empty, all twelve aligned plannings are available.

The order of the parameters must not be changed.



::ofml::go::IT_Notemplate

ITemplate functionality is deactivated here.

7 Implementation issues

Series *global*

The metatypes are created in the new series *global*. The subfolder *meta* inside the sales region contains the relevant tables in CSV format or file *mt.ebase* if compiled to EBASE format. The text resources must be resolved in the resource files according to the OFML standard (*global_de.sr*, *global_en.sr*, etc.).

Note. Instead of *global*, alternative names are possible.

Registration of series

The registration of series that provide metatypes or are used in metatypes, is defined in the DSR specification 2.7.0 (or higher) via the keys *series_type=go_meta* and *meta_type*.

To control advanced behavior of the Metatypes use the function *::ofml::go::goSetup()*:

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

You may enter 1 or more key-value pairs as follows:

- *[@FIRST, <series>]* – Defines one or more series in which–during automatic detection of MT–the search for the specified Basic Article Number should start. The series must be given as OFML symbols inside an array, e.g. *[@s1]*.
- *[@FIND, <mode>]* – Defines a mode for the automatic MT detection. Legal values are:
    - *@MATCH* – Only return values possible that match to both Basic Article Number and Series.
    - *@ALL* – Return values match at least the passed Basic Article Number (Default behavior)

Base type

All metatypes must use the OFML type *::ofml::go::GoMetaType* or a derived type that has to be entered into the relevant mapping files (*art2aclass.map*). For automatic mappings as for converted OFML data, this is not needed.

Parametrization

For explicitly assigned metatypes there can be defined a variant code in the XCF catalog (file *variant.csv*). In any case, the variant code must with the manufacturer id followed by the metatype id from table *go_types*.

Example: `[@man, @desk1]`

Then an arbitrary number of key-value pairs of metatype property assignments can be appended. These assignments overwrite the default values defined in table *go_types*. The format and naming of the assignments must exactly correspond to the definitions in *go_types.* However for names and symbolic values a '@' must be used as prefix.
Use the key *@MTSeries* followed by a series name written as symbol, to use another metatype series than global.
Use the key *@VarCode* followed by a text string if you want to set a variant code for the main child after its creation.

Example: `[@man, @desk1, [@GWidth, 1600], [@GDepth, 900]]`

Catalog dependencies

Metatype series depend on all series from which they can create objects. Standard series depend on all metatype series from which metatype objects should be created during automatic metatype mapping. Therefore, the catalogs must be registered accordingly. Also the dependencies must be set properly (see DSR specification available from EasternGraphics).

Sequence of initialization

Metatype properties are initialized in the following sequence:

1. [REQUIRED] Value from table *go_types*
2. [OPTIONAL] Value from the catalog (XCF variant code but only for explicitly defined metatypes)
3. [OPTIONAL] Value inherited by ancestor
4. [OPTIONAL] Value created by metatype actions
5. [OPTIONAL] Value from profile
6. [OPTIONAL] Value created by *CH_ADD* action

For automatically created metatypes step 2 does not exist.
Native properties are inherited from predecessor/parent after step 4 (metatype actions).
A further modification of properties is possible if *SET_PROP* actions defined in mode *CON* to change the according properties.

Please note:

- **Names of metatype properties must NOT match property names of the wrapped objects.**
- **Invalid metatype data can cause crashes of the application software**.

8 History

This history describes all modifications relevant to this specification. All further changes are documented in the history provided with the metatype implementation.

MT 1.17.2(-0)

- **Extension:** New go_info key: *utf8*

MT 1.17.1(-0)

- **Extension:** Mode GO_AP_POSROT in table *go_childmoving*
- **Extension:** New table *go_resetnativeprops*

MT 1.17.0(-0)

- **Extension:** New planning direction MP in table go_attpt
- **Extension:** New table *go_interactors*
- **Extension:** New go_setup key: *SumSubArticlePrices*
- **Extension:** New go_setup key: *UseMCAxis*
- **Extension:** New go_childmoving command: *RAS_OFFS*

MT 1.16.1(-0)

- **Extension:** New go_setup key: *DisableAutoChildReposition*

MT 1.16.0(-0)

- **Extension:** New property mode 4096 – collision detection during change of properties
- **Extension:** New property mode 8192 – limitation of values of child controlling and na properties
- **Extension:** New table *go_propvalues*
- **Extension:** New internal metatype property *XIsInsObj*
- **Extension:** New go_setup key: *ShowPolyPropFilterMsg*

MT 1.15.0(-0)

- **Extension:** Multiple properties in table *go_noproperties*
- **Extension:** New table: *go_attptsorder*

MT 1.14.3(-0)

- **Extension:** Support for CH_REL attachment points
- **Extension:** Possibility to specify object width in IT_Rectangle2
- **Extension:** New attach sides @FL90 and @FR90 in IT_Rectangle2
- **Modification:** ITemplate IT_Standard not activated by default anymore.

MT 1.14.2(-0)

- **Extension:** New property mode 1024 – re-positioning of main geometry
- **Extension:** New property mode 2048 – re-positioning of own children
- **Extension:** New ITemplates IT_Rectangle, IT_Rectangle2 und IT_Angular
- **Extension:** New table: *go_itemplates*
- **Modification:** go_setup key *ITemplate* removed

MT 1.14.1(-0)

- **Extension**: New go_setup key: *ITemplate*.
- **Extension:** New ITemplates IT_Standard, IT_Cabinet1, IT_Cabinet2 and IT_NoTemplate
- **Extension:** Multiple parameter sets for child-oriented properties in table *go_articles*

MT 1.14.0(-0)

- **Extension:** Interactor geometry *GoIGeometrySphere*
- **Extension:** New go_info key: *updateGMode*
- **Extension:** New table: *go_propclasses*
- **Extension:** Multiple child keys in table *go_childprops*
- **Extension:** New table go_setup
- **Extension:** Multiple own and foreign keys in table go_actions if the reason is CON
- **Extension:** New table go_texts
- **Extension:** Data can be stored in the sales region in the subfolder *meta*

MT 1.12.2(-0)

- **Extension:** New go_info key: *skipVC2MT*

MT 1.12.1(-0)

- **Extension:** New go_info key: *skip_FAN*

MT 1.12.0(-0)

- **Extension:** New go_info key configuration
- **Extension:** New tables go_propindex and go_propmapping
- **Extension:** New table go_info. Initial key pindex
- **Extension:** New go_actions direction PROXY
- **Extension:** goSetup-Mode @FIND (@ALL, @MATCH)
- **Extension:** Reserved MT-ID *_native_* in table *go_articles*
- **Extension:** Interactor geometry GoIGeometryVDArrow
- **Extension:** Interactor geometry GoIGeometryHDArrow
- **Extension:** Clone attachment points
- **Extension:** Interactor geometry GoIGeometryCubes

MT 1.11.0-1

- **Extension:** New modes for table *go_childmoving* (IN, IX, etc.)
- **Extension:** Wildcard mode in table *go_classes*

MT 1.11.0(-0)

- **Extension:** New table *go_classes*
- **Extension:** GXSetup, Mode 8 – Interactive Feedback Mode
- **Extension:** New table *go_feedback* to support the Interactive Feedback Mode

MT 1.10.1(-0)

- **Extension:** AutoDecoration for GO I-Objects
- **Modification:** Meta category for accessories changed from acat to AccCategory

MT 1.10.0(-0)

- **Extension:** Interactor geometry GoIGeometryVArrow

- **Extension:** Interactor geometry GoIGeometryHArrow
- **Extension:** Interactor geometry GoIGeometryInvisible
- **Extension:** Interactor geometry GoIGeometryBlock
- **Extension:** New table go_attptgeo
- **Extension:** New table go_nativeproperties

MT 1.9.3(-0)

- **Extension:** New property control flag 512 (child re-creation)
- **Modification:** Meta category or accessories changed from acc to AutoDecoration
- **Extension:** Usage of fn filters similar to the na filters.
- **Extension:** Concatenation of geometry alignment and _GO_CHILD

MT 1.9.0-1

- **Extension:** Context file go_context.ofml
- **Extension:** New property type 'lb'
- **Extension:** GAlign value ATTPT and predefined attachment key _GO_CHILD

MT 1.9.0(-0)

- **Extension:** GXSetup-Modus 4 – considering main child in GXSetup modes 1 and 2
- **Extension:** New mode for table go_metainfo: acat.
- **Extension:** GXSetup-Modus 2 – controlling the collision detection for rotated MT children
- **Extension:** GXSetup-Modus 1 – controlling the collision detection for translated MT children
- **Extension:** New control variable GXSetup
- **Extension:** New property mode 256 – update of main geometry

MT 1.8.0(-0)

- **Extension:** New table zur Steuerung der Merkmalsvererbung.
- **Extension:** New action *CON_AP* for adaptation of positions of neighbor objects
- **Extension:** New child moving modes *XR*, *YR* and *ZR*.
- **Extension:** Filter entry for na properties can now be used to define dependencies to child properties.

MT 1.7.1(-0)

- **Extension:** New GSetup mode 32768
- **Extension:** New GMode values '@XOCD' und '@OCD'

MT 1.7.0-1

- **Extension:** New property type 'th'.

MT 1.7.0(-0)

- **Extension:** New table go_metainfo
- **Extension:** New internal metatype property XChildID
- **Extension:** New modes for actions: CH_ADD, CH_DEL
- **Extension:** GSetup mode 16384, handling of interactive children after change of properties

MT 1.6.0-2

- **Note:** Property mode 32 (inheritance) shouldn't be used if there are already initial values set by actions.

MT 1.6.0-1

- **Correction:** Wrong (inverse) description of GSetup & 2 in the table that depicts the controlling of the basket positions.

MT 1.6.0(-0)

- **Extension:** Planning directions T and D
- **Extension:** fn properties and table go_freenumeric
- **Extension:** local object height via variable XHeight available in the local context. In analogy, _XHeight for the secondary object in relations and conditions.
- **Extension:** Action trigger CREATE
- **Extension:** Table go_proporder
- **Extension:** go_types, mode Flag 128 – controlling the sequence of values in choice lists

MT 1.5.0-1

- **Modification:** GSetup, mode 2, adaptation to GO 1.4.*

MT 1.5.0(-0)

- **Extension:** GSetup, modes 1 and 2, if mode 1 is set, the main child will be a major position in basket, regardless the setting of GSetup & 2
- **Extension:** GSetup, mode 8192, enforce a major position in basket
- **Extension:** Parametrization via catalog @VarCode.
- **Extension:** CON::SET_PROP

MT 1.4.0-3

- **Extension:** GSetup, mode 2048, avoid the inheritance of native properties if parent is a metatype.

MT 1.4.0-2

- **Extension:** GSetup, modus 512, hiding of the hint 'This is not a concatenation.'
- **Extension:** Wrapping of positions of metatype children (property type "cp")

MT 1.4.0-1

- **Extension:** Action mode *AP*
- **Extension:** *GSetup, mode 1024, complete temporary creation*

MT 1.4.0(-0)

- **Extension:** Property *GVarPrefix*
- **Extension:** Modus 256 for *GSetup* (removal of interactive children)
- **Extension:** Wrapping of native Properties (property type "na")

MT 1.3.10(-0)

- **Extension:** Mode 64 for go_types (display of properties adapted by filter)
- **Extension:** New actions CON_PROP and CON_CH_PROP.
- **Modification:** Predefined type GO_RECTTABLE removed

MT 1.1-1.6

- **Extension:** MT children inherit native properties either from the parent or the predecessor depending on GSetup mode 128

MT 1.1-1.5

- **Incompatible Modification:** In the context of the control of the interactive movement of MT children with regard to MT 1.1-1.4 (GO 1.3.3): local parameters without prefix "_", those of the parent with prefix
- **Extension:** Child movement modes *XZTLINEAR, XZTRANGE*

MT 1.1-1.4

- **Extension:** *go_types* parameter GSetup mode 64 (collision detection for children)
- **Extension:** *go_types* parameter *GSetup* mode 32 (hiding of changed property names)
- **Extension:** *go_types* parameter *GSetup* mode 16 (hiding the article number)
- **Incompatible modification:** Modification of the insertion policy for interactive children that define an attachment point *O*, *OL* or *OR*.
- **Extension:** *go_types* parameter *GSetup* mode 8 (usage of standard attach points)

MT 1.1-1.3

- **Extension:** *go_types* parameter *GSetup* mode 4 (2D symbol)

MT 1.1-1.2

- **Extension:** *go_types* parameter *GAlign*
- **Extension:** *go_types* parameter *GSetup* incl. modes 1 and 2 (article number redirection, item/sub-item management)

MT 1.1-1.1

- **Incompatible Modification:** *go_childmoving[parameter]* to be evaluated in the context of the parent

MT 1.1-1.0

- **Extension:** Table *go_childmoving* added (XT, YT, ZT x MIN, MAX, POS, RASTER)
- **Extension:** Local origin attach points *OL* and *OR*
- **Extension:** DSR registration of metatype series and series using metatypes

MT 1.0-1.0