Spezifikation

ODB
OFML Database*
(OFML Part I)

Version 2.1

1. überarbeitete Fassung

Jochen Pohl, Ekkehard Beier, Sebastian Schmidt (EasternGraphics GmbH)†

4. November 2015

†ODB wurde im Auftrag des Verbandes Büro-, Sitz- und Objektmöbel e.V. (BSO) durch die EasternGraphics GmbH entwickelt.

Inhaltsverzeichnis

1 Einleitung

Mit Hilfe der ODB können die geometrischen und in begrenztem Maße die logischen Eigenschaften von Planungsobjekten beschrieben werden. Ziel der ODB ist es, eine Beschreibungsform zu haben, die programmtechnisch zum einen auf verhältnismäßig einfache Art und Weise geschrieben und gelesen und zum anderen auf Konsistenz geprüft werden kann. Zu diesem Zweck werden in der ODB die Daten in Tabellenform abgelegt, wobei die folgenden Tabellen existieren:

- Geometrie–Tabellen
  Es existieren zwei getrennte Tabellen für die 2D– und 3D–Geometrien. Diese sind in den Abschnitten 2 und 3 beschrieben.
- Tabellen für Anfügepunkte
  Die Plazierung von Planungsobjekten relativ zu anderen Planungsobjekten erfolgt über Anfügepunkte. Die Anfügepunkte werden in der ODB mittels dreier in Abschnitt 4 beschriebener Tabellen definiert.
- Funktions–Tabelle
  In den Tabellen–Spalten können oftmals arithmetische und logische Ausdrücke angegeben werden, die in Umgekehrter Polnischer Notation formuliert werden. Diese Ausdrücke können sowohl vordefinierte wie auch nutzerdefinierte Funktionen referenzieren. Die nutzerdefinierten Funktionen werden in der in Abschnitt 5 beschriebenen Funktionstabelle definiert.

Die Tabellen müssen in einer Form erfaßt werden, die mittels `ebmkdb` in eine binäre EBase–Datenbank[1] überführt werden kann.

Dazu wird das CSV–Format (comma separated values) empfohlen. Die Tabellen werden dabei als Textdateien mit der Endung `.csv` angelegt, wobei die Spalten durch Semikolon voneinander zu trennen sind[2]. Eine Konfigurationsdatei für `ebmkdb`, die verwendet werden kann, um die CSV–Tabellen in eine EBase–Datenbank zu konvertieren, ist in Anhang A abgedruckt.

[1] Jochen Pohl: *EBase – Datenbank mit ausschließlich lesendem Zugriff*
[2] Als Folge dessen können in den Feldern der Tabelle keine Semikola verwendet werden.

2 2D–Geometrien

Die 2D–Geometrie eines OFML–Objekts wird durch einen oder mehrere aufeinanderfolgende Einträge in der 2D–Tabelle beschrieben. Jeder Eintrag dient zur Erzeugung einer grafischen Primitive[3] und enthält deren Skalierung, Rotation, Verschiebung und ggf. zusätzliche Attribute wie Farbe, Linienstärke und andere.

Der Aufbau der Tabelle mit den 2D–Geometrien ist in Tabelle 1 zusammengefaßt und anschließend genauer beschrieben.

Feld-nummer	Feld-name	Beschreibung
1	`odb_name`	ODB–Name
2	`level`	Hierarchietiefe
3	`visible`	Steuerung der Sichtbarkeit
4	`x_offs`	Verschiebung in x–Richtung
5	`y_offs`	Verschiebung in y–Richtung
6	`rot`	Rotation (um die z–Achse)
7	`x_scale`	Skalierung in x–Richtung
8	`y_scale`	Skalierung in y–Richtung
9	`ctor`	Erzeugung des 2D–Objekts
10	`attrib`	Setzen von grafischen Attributen

Tabelle 1: 2D–Geometrien

2.1 ODB–Name

Objekte, für die mittels der ODB eine 2D–Geometrie erzeugt werden soll, liefern einen voll qualifizierten ODB–Namen. Dieser besteht aus dem Namen des Paketes, in dem die zu verwendende ODB liegt, plus dem ODB–Grundnamen, der zur Ermittlung der zu verwendenden Einträge in der 2D– und der 3D–Tabelle dienen. Ein Beispiel für einen voll qualifizierten ODB–Namen ist `::foo::bar::BAZ`, wobei `::foo::bar` der Paketname und `BAZ` der ODB–Grundname sind.

Die 2D–Tabelle besteht aus einer Folge von ODB–Blöcken. Ein ODB–Block besteht aus mehreren aufeinanderfolgenden Einträgen, von denen der erste in der Spalte `odb_name` den ODB–Grundnamen enthält, wärend bei allen folgenden Einträgen dieses Blockes die Spalte `odb_name` leer ist.

2.2 Hierarchietiefe

Die Einträge in der 2D–Tabelle können innerhalb eines ODB–Namens hierarchisch angelegt werden. Dadurch ist es möglich, eine Menge von Elementen zu gruppieren und gemeinsam zu transformieren (zu skalieren, zu drehen und zu verschieben).

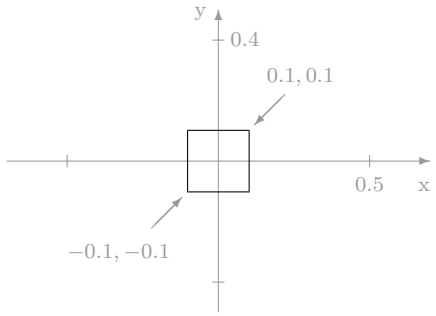
[3] Beim Referenzieren externer Geometrien kann ein Eintrag auch komplexe 2D–Geometrien erzeugen, die jedoch immer als ein zusammenhängendes Objekt behandelt werden.

Der erste Eintag eines ODB–Namens hat immer die Hierarchiestufe 0. Sollen mehrere Elemente zu einer Gruppe zusammengefaßt werden, so sind sie in der 2D–Tabelle in aufeinanderfolgenden Einträgen aufzuführen und ist ihnen die gleiche Hierarchiestufe zu geben, die um eins höher sein muß als die Hierarchiestufe des Eintrags, der die Transformation der Gruppe bestimmt. Der die Transformation einer Gruppe bestimmende Eintrag ist immer der letzte Eintrag vor der Gruppe, dessen Hierarchiestufe kleiner ist als die der Gruppe.

Im folgenden Beispiel werden in seiner endgültigen Form vier Linien, die ein Quadrat bilden, zu einer Gruppe zusammengefaßt, wobei die ganze Gruppe in x–Richtung um 0.6 und in y–Richtung um 0.4 verschoben ist. Der Ursprung des lokalen Koordinatensystems der Gruppe befindet sich im Mittelpunkt des Quadrats, das nicht gedreht ist. Des weiteren wird das Quadrat durch ein Rechteck mit den Abmessungen 1.2×0.8 so umschlossen, daß die Mittelpunkte des Quadrats und des Rechtecks auf den gleichen Punkt fallen.

Im ersten Schritt wird das Quadrat aus vier Linien zusammengesetzt, so daß der Mittelpunkt des Quadrats mit dem Ursprung des Koordinatensystems des OFML–Objekts übereinstimmt:

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
BAZ	0		−0.1	−0.1	0.0	0.2	1.0	hline	
	0		−0.1	0.1	0.0	0.2	1.0	hline	
	0		−0.1	−0.1	0.0	1.0	0.2	vline	
	0		0.1	−0.1	0.0	1.0	0.2	vline	



In obigem Beispiel werden von oben nach unten die folgenden vier Linien erzeugt: von $-0.1, -0.1$ nach $0.1, -0.1$, von $-0.1, 0.1$ nach $0.1, 0.1$, von $-0.1, -0.1$ nach $-0.1, 0.1$ und von $0.1, -0.1$ nach $0.1, 0.1$. Eine detaillierte Beschreibung, wie mit Hilfe der Funktionen `hline` und `vline` Linien erzeugt werden können, befindet sich in Abschnitt 2.7.1.

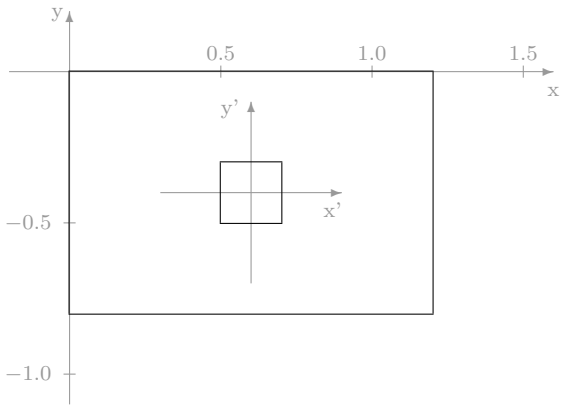
Im nächsten Schritt werden die Linien zu einer Gruppe zusammengefaßt, wobei die Gruppe noch nicht verschoben wird. Die Linien sind also noch immer an der gleichen Stelle wie in obigem Bild.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
BAZ	0		0.0	0.0	0.0	1.0	1.0		
	1		−0.1	−0.1	0.0	0.2	1.0	hline	
	1		−0.1	0.1	0.0	0.2	1.0	hline	
	1		−0.1	−0.1	0.0	1.0	0.2	vline	
	1		0.1	−0.1	0.0	1.0	0.2	vline	

Wie zu sehen ist, wurde den Linien lediglich ein leeres Objekt[4] vorangestellt, dessen Hierarchiestufe 0 ist und die Hierarchiestufe der Linien folglich auf 1 erhöht. Damit werden die Linien relativ zu dem Objekt in der ersten Zeile der Tabelle erzeugt.

Im letzten Schritt wird die Gruppe um 0.6 in x–Richtung und um −0.4 in y–Richtung verschoben und gleichzeitig ein 1.2 breites und 0.8 hohes Rechteck eingefügt, dessen linke obere Ecke sich im Ursprung des Koordinatensystems des OFML–Objekts befindet. Das nun verschobene lokale Koordinatensystem der Gruppe ist im folgenden Bild mit den Achsenbezeichnungen x' und y' dargestellt.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
BAZ	0		0.6	−0.4	0.0	1.0	1.0		
	1		−0.1	−0.1	0.0	0.2	1.0	hline	
	1		−0.1	0.1	0.0	0.2	1.0	hline	
	1		−0.1	−0.1	0.0	1.0	0.2	vline	
	1		0.1	−0.1	0.0	1.0	0.2	vline	
	0		0.0	0.0	0.0	1.2	−0.8	quadrat	



Wie in der Tabelle zu sehen ist, muß die Verschiebung einer Gruppe in dem der Gruppe übergeordneten Objekt, in diesem Beispiel in der ersten Zeile der Tabelle, angegeben werden. Das Rechteck wird in der letzten Tabellenzeile erzeugt, indem ein standardmäßig mit dem Abmessungen 1.0 × 1.0 erzeugtes Quadrat in x–Richtung um 1.2 und in y–Richtung um −0.8 skaliert wird. Eine detaillierte Anleitung zum Erzeugen von Rechtecken findet sich in Abschnitt 2.7.2.

2.3 Sichtbarkeit

Mitunter kommt es vor, daß Teile eines 2D–Symbols nur bei bestimmten Konfigurationen des zugrundeliegenden OFML–Objekts angezeigt werden sollen. Die Sichtbarkeit kann durch einen Eintrag in der Spalte `visible` gesteuert werden. Dabei wird ein Eintrag angezeigt, wenn die Spalte `visible` entweder leer ist oder einen Wert enthält, der von 0 verschieden ist. Ist der Wert in der Spalte `visible` 0, werden weder der Eintrag noch eventuell vorhandene in der Hierarchie unter ihm liegende Einträge dargestellt.

[4] Objekte, bei denen die Spalte `ctor` leer ist, haben keine grafische Repräsentation.

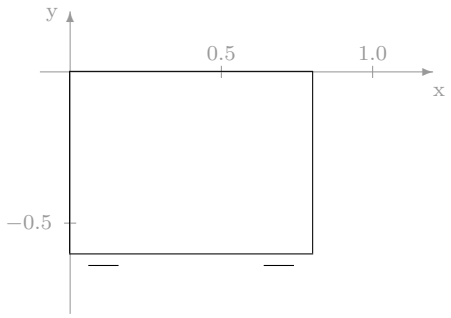
Im folgenden Beispiel soll eine kurze, einen Türgriff symbolisierende Linie entweder links oder rechts vor einem Schrank, der durch ein Rechteck symbolisiert wird, dargestellt werden. Die Entscheidung wird abhängig vom Wert des Parameters `$GRIFF`, der vom zugrundeliegenden OFML–Objekt bereitgestellt wird und entweder `"L"` für links oder `"R"` für rechts ist, getroffen.

In der ersten Tabelle wird ein 0.8 breiter und 0.6 tiefer Schrank mit zwei Griffen, einem links und einem rechts, dargestellt. Die Linien, die die Griffe symbolisieren, haben eine Länge von 0.1, beginnen bzw. enden 0.05 vor der linken oder rechten Kante des Schrankes und befinden sich im Abstand von 0.3 vor dem Schrank.

Das den Schrank repräsentierende Rechteck wird diesmal anders als in Abschnitt 2.2 erzeugt. Während in Abschnitt 2.2 das Rechteck nicht verschoben wurde und somit in y–Richtung negativ skaliert werden mußte, um es nach unten zu klappen, wird in diesem Beispiel der Ursprung des Rechtecks in seine linke untere Ecke verschoben, so daß es in seine y–Richtung positiv skaliert werden kann.

Die zweite Zeile in der Tabelle stellt den linken Griff dar und die dritte Zeile den rechten Griff. Da als Ursprung (und somit Anfangspunkt) der Linie für den rechten Griff ihr rechter Endpunkt angegeben wurde, muß die Linie in x–Richtung negativ skaliert werden, so daß ihr Endpunkt links von ihrem Anfangspunkt liegt.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
SCHRANK	0		0.0	−0.6	0.0	0.8	0.6	quadrat	
	0		0.05	−0.63	0.0	0.1	1.0	hline	
	0		0.75	−0.63	0.0	−0.1	1.0	hline	



Um die Griffe selektiv darstellen zu können, muß für die letzten beiden Zeilen die Spalte `visible` gefüllt werden. Allerdings ist es nicht möglich, wie bisher einen konstanten Wert in sie einzutragen. Des weiteren kann der Parameter `$GRIFF` nicht direkt verwendet werden, da sein Wert eine Zeichenkette ist, als Ergebnis in der Spalte `visible` aber eine Zahl erwartet wird. Folglich wird jeweils für den linken Griff und den rechten Griff ein Ausdruck benötigt, der 1.0 ergibt, wenn der Griff dargestellt werden soll und andernfalls 0.0. Für den linken Griff lautet dieser Ausdruck in Umgekehrter Polnischer Notation „`$GRIFF "L" ==`“, für den rechten Griff analog „`$GRIFF "R" ==`“.

Da die Spalte `visible` eine begrenzte maximale Feldbreite hat, ist es üblich, die Ausdrücke nicht direkt in der Spalte anzugeben, sondern als eine Funktion zu schreiben. Die Funktion wird in einer gesonderten, in Abschnitt 5 beschriebenen, Tabelle abgelegt. Die beiden folgenden Tabellen zeigen die relevanten Einträge in der 2D–Tabelle und in der Funktionstabelle.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
SCHRANK	0		0.0	−0.6	0.0	0.8	0.6	quadrat	
	0	GL	0.05	−0.63	0.0	0.1	1.0	hline	
	0	GR	0.75	−0.63	0.0	−0.1	1.0	hline	

name	body
GL	$GRIFF "L" ==
GR	$GRIFF "R" ==

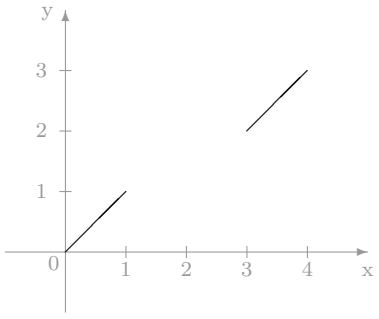
2.4 Verschiebung

Jedes einzufügende Objekt hat einen Einfügepunkt, mit dem es normalerweise im Ursprung des Koordinatensystems plaziert wird. Dieser Einfügepunkt ist immer der Ursprung des Koordinatensystems, in dem die Koordinaten des Objekts erfaßt wurden. Durch den Verschiebeparameter kann der Einfügepunkt in x–Richtung und um y–Richtung verschoben werden. Die Verschiebung findet nach einer eventuellen Skalierung und Drehung des eingefügten Objekts statt.

Die Verschiebung sei am Beispiel einer diagonalen Linie demonstriert. Eine mit dem Kommando `dline` in der Spalte `ctor` eingefügte diagonale Linie verläuft vom Punkt $0.0, 0.0$ (dem Einfügepunkt) zum Punkt $1.0, 1.0$. Wird diese diagonale Linie nicht verschoben, so verläuft sie auch im Koordinatensystem des OFML–Objekts von $0.0, 0.0$ nach $1.0, 1.0$. Soll sie aber von $3.0, 2.0$ nach $4.0, 3.0$ verlaufen, so ist ihr Einfügepunkt um 3.0 in x–Richtung und um 2.0 in y–Richtung zu verschieben.

In der folgenden Tabelle und dem dazugehörenden Bild werden zwei diagonale Linien eingefügt, wobei die erste nicht verschoben und die zweite, wie oben angegeben, um $3.0, 2.0$ verschoben ist.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.0	0.0	0.0	1.0	1.0	dline	
	0		3.0	2.0	0.0	1.0	1.0	dline	



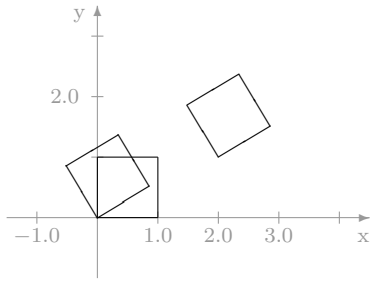
2.5 Rotation

Mit dem Rotationsparameter können eingefügte Objekte um ihren Einfügepunkt, d.h. um den Ursprung ihres lokalen Koordinatensystems, gedreht werden. Die Angabe des Drehwinkels

erfolgt mathematisch positiv (entgegen dem Uhrzeigersinn) in Grad. Die Drehung des Objekts erfolgt nach einer eventuellen Skalierung und vor einer eventuellen Verschiebung.

Im folgenden Beispiel werden drei Quadrate dargestellt. Jedes mit `quadrat` in der Spalte `ctor` erzeugte Quadrat hat ohne Skalierung eine Kantenlänge von 1.0. Das erste Quadrat wird ohne Angabe von Drehung oder Verschiebung eingefügt. Das zweite Quadrat wird um 30 Grad gedreht. Das dritte Quadrat wird erst um 30 Grad gedreht und anschließend um 2.0 in x–Richtung und 1.0 in y–Richtung verschoben.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.0	0.0	0.0	1.0	1.0	quadrat	
	0		0.0	0.0	30.0	1.0	1.0	quadrat	
	0		2.0	1.0	30.0	1.0	1.0	quadrat	



2.6 Skalierung

Die Skalierung ist neben der Verschiebung die wichtigste Transformation bei der Erzeugung von 2D–Geometrien mittels der ODB. Dies liegt in der Tatsache begründet, daß viele grafische Primitiven in Form von „Einheitsprimitiven“ erzeugt werden. Das wesentliche Merkmal dieser Einheitsprimitiven ist, daß für alle ihre Eck- bzw. Endpunkte x_i, y_i gilt, daß sowohl x_i wie auch y_i entweder 0.0 oder 1.0 sind. Durch Skalierung in x– und y–Richtung werden diese Einheitsprimitiven auf die gewünschte Größe gebracht. Da ihre Koordinatenwerte entweder 0.0 oder 1.0 sind, kann in der Regel ihre Abmessung in x– oder y–Richtung direkt als Skalierungsfaktor verwendet werden.

Des weiteren können die Skalierungsparameter verwendet werden, um ein Objekt zu spiegeln. Wird der Wert in der Spalte `x_scale` z.B. auf -1.0 gesetzt, so wird das Objekt an der y–Achse seines lokalen Koordinatensystems gespiegelt. Analog spiegelt eine -1.0 in der Spalte `y_scale` das Objekt an der x–Achse seines lokalen Koordinatensystems.

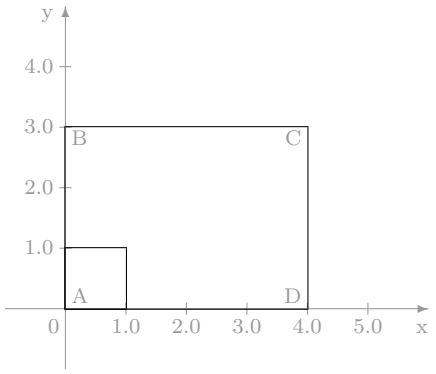
Schließlich können Skalierung und Spiegelung in einem Skalierungsfaktor zusammengefaßt werden. Um z.B. ein Objekt in x–Richtung um 2.5 zu vergrößern und gleichzeitig an der y–Achse zu spiegeln, ist in die Spalte `x_scale` eine -2.5 einzutragen.

Die Skalierung wird vor einer eventuellen Rotation oder Verschiebung auf die Objekt angewendet.

Wenn ein Objekt in eine Richtung nicht skaliert werden soll, so ist als Skalierungsfaktor 1.0 anzugeben. 0.0 als Skalierungsfaktor ist nicht erlaubt.

Im folgenden Beispiel wird in der ersten Zeile ein „Einheitsquadrat“ erzeugt. In der zweiten Zeile wird das gleiche Quadrat erzeugt, diesmal aber in x–Richtung um 4.0 und in y–Richtung um 3.0 skaliert. Beide Quadrate sind in dem nachfolgenden Bild dargestellt.

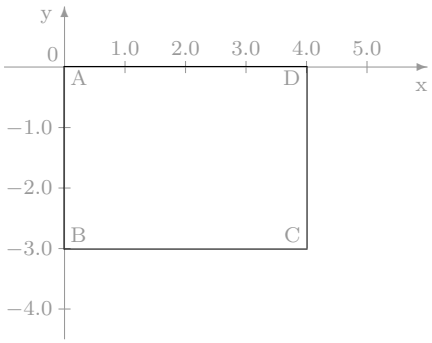
odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.0	0.0	0.0	1.0	1.0	quadrat	
	0		0.0	0.0	0.0	4.0	3.0	quadrat	



Im nächsten Beispiel wird das gleiche Rechteck wie in der zweiten Zeile des vorherigen Beispiels erzeugt, nur mit dem Unterschied, daß es diesmal durch Negation des Skalierungsfaktors in y–Richtung an der x–Achse gespiegelt ist. Die Spiegelung läßt sich an den Buchstaben erkennen, mit denen die Eckpunkte des Rechtecks gekennzeichnet sind.

In diesem Fall hätte sich der gleiche Effekt ohne Spiegelung erzielen lassen, indem das Rechteck um -3.0 in y–Richtung verschoben worden wäre. Einen Unterschied macht es allerdings dann, wenn Objekte gespiegelt werden, die nicht zur Spiegelungsachse symmetrisch sind.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.0	0.0	0.0	4.0	−3.0	quadrat	



2.7 Objekterzeugung

Die Spalte `ctor` dient zur Erzeugung von 2D–Objekten. Dabei kann prinzipiell zwischen drei verschiedenen Fällen unterschieden werden:

- **Die Spalte ist leer.** In diesem Fall hat das 2D–Objekt keine grafische Repräsentation. Dies kann sinnvoll sein, wenn, wie in Abschnitt 2.2 beschrieben, eine Gruppe von Objekten gebildet werden soll, deren Transformation durch das ihnen in der Hierarchie übergeordnete Objekt bestimmt werden soll.
- **Die Spalte erzeugt direkt ein Objekt.** Die zur Verfügung stehenden Objekte sind vertikale, horizontale und diagonale Linien, Quadrate, Kreise und Kreisbögen, Ellipsen, Punkte und Text.
- **Die Spalte referenziert eine externe Geometrie.** In diesem Fall verweist sie auf eine Datei, die möglicherweise eine komplexe 2D–Geometrie enthält.

In jedem Fall enthält die Spalte `ctor`, wenn sie nicht leer ist, einen Funktionsaufruf in Umgekehrter Polnischer Notation, welcher die Erzeugung eines 2D-Objekts veranlaßt.

Die Spalte `ctor` kann auch eine Funktion enthalten, die in der Funktionstabelle definiert ist. Diese Funktion, wie auch Funktionen, die direkt oder indirekt durch sie aufgerufen werden, können alle im folgenden beschriebenen Funktionen zur Objekterzeugung aufrufen.

2.7.1 Linien

Es gibt drei verschiedene Funktionen, die in der Spalte `ctor` verwendet werden können, um „Einheitslinien“ zu erzeugen. Diese Funktionen sind in Tabelle 2 aufgelistet.

Funktion	Anfangspunkt	Endpunkt
`hline`	$x = 0.0;\quad y = 0.0$	$x = 1.0;\quad y = 0.0.$
`vline`	$x = 0.0;\quad y = 0.0$	$x = 0.0;\quad y = 1.0.$
`dline`	$x = 0.0;\quad y = 0.0$	$x = 1.0;\quad y = 1.0.$

Tabelle 2: Funktionen zur Linienerzeugung

Mit der Funktion `dline` zum Erzeugen diagonaler Linien können theoretisch durch entsprechende Skalierung Linien beliebiger Länge und Neigung erzeugt werden. Eine Ausnahme bilden horizontale und vertikale Linien, da zu ihrer Erzeugung ein Skalierungsfaktor auf 0.0 gesetzt werden müßte, dies jedoch nicht erlaubt ist. Horizontale und vertikale Linien müssen somit mit den Funktionen `hline` und `vline` erzeugt werden.

Um eine Linie mit dem Anfangspunkt x_0, y_0 und dem Endpunkt x_1, y_1 darzustellen, können die Parameter x_{offs}, y_{offs} für die Verschiebung und x_{scale}, y_{scale} für die Skalierung, wie in Tabelle 3 dargestellt, berechnet werden.

Bedingung	Funktion	x_{offs}	y_{offs}	x_{scale}	y_{scale}
$y_0 = y_1$	`hline`	x_0	y_0	$x_1 - x_0$	1.0
$x_0 = x_1$	`vline`	x_0	y_0	1.0	$y_1 - y_0$
ansonsten	`dline`	x_0	y_0	$x_1 - x_0$	$y_1 - y_0$

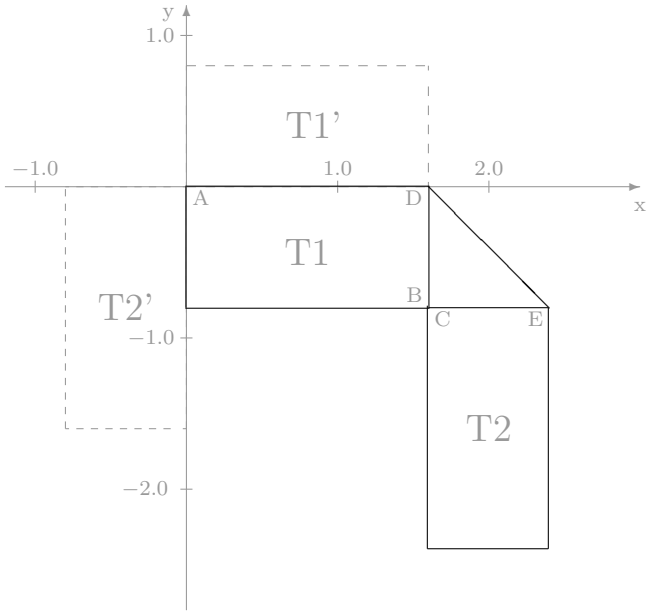
Tabelle 3: Berechnung von Verschiebung und Skalierung für Linien

2.7.2 Quadrate und Rechtecke

Quadrate und Rechtecke können mit der Funktion `quadrat` in der Spalte `ctor` erzeugt werden. Die Funktion `quadrat` erzeugt ein Quadrat mit der Seitenlänge 1.0, dessen linke untere Ecke im Ursprung des lokalen Koordinatensystems liegt. Um ein beliebiges Rechteck mit der Breite w und der Höhe h zu erzeugen, ist der Skalierungsfaktor in x–Richtung auf w und der Skalierungsfaktor in y–Richtung auf h zu setzen. Anschließend kann das so erzeugte Rechteck beliebig rotiert und verschoben werden.

Das folgende Beispiel enthält zwei durch Rechtecke repräsentierte Tische. Diese Tische stehen in einem Winkel von 90 Grad zueinander und werden durch ein Verbindungsstück aneinander gekoppelt. Beide Tische sind 1.6 breit und 0.8 tief. Der erste Tisch steht waagerecht mit seiner linken oberen Ecke (A) im Ursprung des Koordinatensystems des OFML–Objekts. An seiner rechten unteren Ecke (B) befindet sich die linke untere Ecke (C) des um 90 Grad im Uhrzeigersinn gedrehten zweiten Tisches. Zwischen beiden Tischen ist eine Verbindungsplatte, die durch eine Linie zwischen der rechten oberen Ecke (D) des ersten Tisches und der linken oberen Ecke (E) des zweiten Tisches symbolisiert wird[5].

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.0	0.0	0.0	1.6	−0.8	quadrat	
	0		2.4	−0.8	−90.0	1.6	−0.8	quadrat	
	0		1.6	0.0	0.0	0.8	−0.8	dline	



In der ersten Zeile der Tabelle wird der waagerecht stehende Tisch (T1) erzeugt. Dies ist trivial, da dazu nur das Einheitsquadrat in x– und y–Richtung entsprechend skaliert werden muß, wobei der Skalierungsfaktor in y–Richtung negativ ist, um den Tisch an der x–Achse zu spiegeln[6]. In der zweiten Zeile wird der senkrecht stehende Tisch T2 erzeugt. Dazu wird er

[5] Dieses Beispiel ist nicht sonderlich praxisrelevant, da in der Praxis die Tische vermutlich eher einzeln erfaßt würden.

[6] Der nicht gespiegelte Tisch ist im folgenden Bild gestrichelt als Tisch T1' dargestellt.

in der gleichen Weise wie der waagerecht stehende Tisch skaliert, anschließend um 90 Grad im Uhrzeigersinn gedreht[7] und zuletzt an seine Position geschoben. In der dritten Zeile wird die Linie zwischen den beiden Tischen gezeichnet.

2.7.3 Kreise, Kreisbögen und Ellipsen

Zur Darstellung von Kreisen, Kreisbögen und Ellipsen existieren die drei in Tabelle 4 aufgeführten Funktionen.

Syntax	x–Radius	y–Radius	Start–Winkel	End–Winkel
`circle`	1.0	1.0	0.0	360.0
α_{start} α_{end} `arc`	1.0	1.0	α_{start}	α_{end}
x_{radius} y_{radius} `ellipse`	x_{radius}	y_{radius}	0.0	360.0

Tabelle 4: Befehle für Kreise, Kreisbögen und Ellipsen

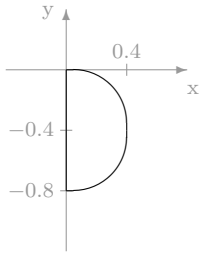
Für alle drei Funktionen ist der Mittelpunkt immer im Ursprung des lokalen Koordinatensystems. Im Falle des Kreisbogens (`arc`) verläuft der Kreisbogen mathematisch positiv (entgegen dem Uhrzeigersinn) vom Start- zum Endwinkel.

Im folgenden einfachen Beispiel ist ein runder Tisch mit einem Durchmesser von 1.2 dargestellt, dessen Mittelpunkt sich bei 0.6, −0.6 befindet. Auf die entsprechende Darstellung wird verzichtet.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.6	−0.6	0.0	0.6	0.6	circle	

Im folgenden Beispiel wird die Verwendung von Kreisbögen demonstriert. Es soll ein halbrunder Tisch mit einem Radius von 0.4 erzeugt werden. Seine gerade Seite verläuft durch den Mittelpunkt des Halbkreises und liegt auf der negativen y–Achse.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.0	−0.4	0.0	0.4	0.4	-90.0 90.0 arc	
	0		0.0	0.0	0.0	1.0	−0.8	vline	



Da derzeit keine Ellipsenbögen unterstützt werden, ist es notwendig, diese durch entsprechend skalierte Kreisbögen darzustellen[8]. Es ist jedoch zu beachten, daß sich Anfangs– und

[7] Der gedrehte Tisch ist im folgenden Bild gestrichelt als Tisch T2' dargestellt.

[8] Es wird davon abgeraten, Ellipsen durch skalierte Kreise darzustellen, da andernfalls verschiedene Fangmodi, insbesondere das Lot, durch die in x– und y–Richtung unterschiedliche Skalierung nicht mehr wie erwartet funktionieren.

Endwinkel in der Regel durch die in x– und y–Richtung verschiedene Skalierung verändern. Die Berechnung der Winkel für den Kreisbogen kann mit Hilfe der Gleichung

$$\alpha_{kreis} = \arctan\left(\frac{y_{scale}}{x_{scale}} \tan \alpha_{ellipse}\right)$$

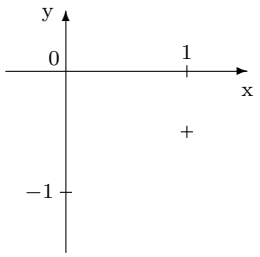
erfolgen, wobei allerdings noch auf den richtigen Quadranten zu achten ist.

2.7.4 Punkte

Mit der Funktion `point` wird ein einzelner Punkt im Ursprung des lokalen Koordinatensystems erzeugt. Rotation und Skalierung werden für ein Punktobjekt nicht berücksichtigt[9].

Mit dem folgenden Beispiel wird ein Punkt auf der Position $1.0, -0.5$ erzeugt. Im Bild ist er, um besser sichtbar zu sein, als kleines Kreuz dargestellt. In der Realität erscheint ein Punkt als ein gesetztes Pixel.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		1.0	−0.5	0.0	1.0	1.0	point	



2.7.5 Text

Mit der Funktion `text` wird ein Textobjekt erzeugt, dessen Bezugspunkt im Ursprung des lokalen Koordinatensystems liegt. Als Parameter müssen für das Textobjekt die Ausrichtung des nicht gedrehten Textes relativ zum Bezugspunkt und der Text selbst angegeben werden. Die Höhe eines Großbuchstaben des Textes ohne Unterlänge beträgt standardmäßig 0.1.

Die horizontale Ausrichtung des nicht gedrehten Textes wird durch den ersten Parameter der Funktion `text` bestimmt. Er kann die in Tabelle 5 aufgeführten Werte annehmen[10]. Vertikal wird der Text so ausgerichtet, daß die Grundlinie[11] des Textes durch den Bezugspunkt führt.

Es wird nicht empfohlen, die Höhe und Breite des Textes über die Skalierungsfaktoren zu steuern. Obwohl es bei der derzeitigen auf Vektorfonts basierenden Implementierung von Text

[9] Werden Rotation oder Skalierung für ein Punktobjekt angegeben, und existieren Objekte, die in der Hierarchie unterhalb des Punktobjekts liegen, so werden diese durch die für das Punktobjekt angegebene Rotation oder Skalierung beeinflußt.

[10] Tatsächlich sind beliebige Gleitkommawerte für die Ausrichtung erlaubt. Für positive Werte befindet sich der Mittelpunkt des Textes immer links vom Bezugspunkt, wobei sein Abstand vom Bezugspunkt mit größer werdenden Absolutwerten wächst. Für negative Werte verhält es sich äquivalent, nur daß sich der Mittelpunkt des Textes dann rechts vom Bezugspunkt befindet.

[11] Die Grundlinie ist die Linie, auf der Großbuchstaben ohne Unterlänge stehen.

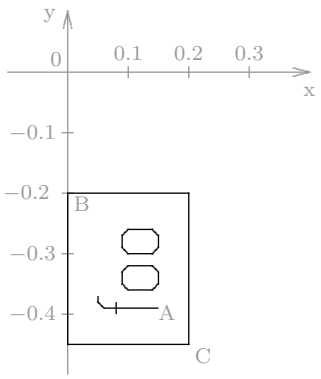
Wert	Bedeutung
−1.0	Der Text befindet sich unmittelbar rechts neben dem Bezugspunkt.
0.0	Der Text ist horizontal zum Bezugspunkt zentriert.
1.0	Der Text befindet sich unmittelbar links neben dem Bezugspunkt.

Tabelle 5: Ausrichtung von Textobjekten

problemlos funktioniert, kann es bei einer eventuellen zukünftigen Verwendung von Bitmapfonts zu Problemen führen. Statt dessen sollten die Attributfunktionen `fheight` und `faspect` verwendet werden, die die Höhe und die Streckung des Textes in die Breite beeinflussen. Diese Funktionen sind in Abschnitt 2.8 beschrieben.

Im folgenden Beispiel wird ein vetikal stehender Text, der von rechts lesbar ist, dargestellt. Der Text wird linksbündig zum Bezugspunkt (A) mit den Koordinaten 0.15, −0.4 erzeugt. Da die Ausrichtung für den nicht gedrehten Text angegeben wird, befindet sich der Text oberhalb des Bezugspunktes. Der Text wird durch ein Rechteck eingerahmt, dessen linke obere Ecke (B) sich bei 0.0, −0.2 und dessen rechte untere Ecke (C) sich bei 0.2, −0.45 befindet.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.15	−0.4	90.0	1.0	1.0	-1.0 "foo" text	
	0		0.0	−0.45	0.0	0.2	0.25	quadrat	



2.7.6 Externe Geometrien

Sollen komplexere Geometrien, die nicht oder als Ganzes skaliert werden sollen, dargestellt werden, so kann es sinnvoll sein, diese als EGM–Symbol Datei zu erfassen und in der ODB als externe Geometrie einzubinden[12].

Die Funktion zum Einbinden heißt `egms`. Sie erwartet als Argument eine Zeichenkette, die den Namen der externen Geometrie enthält. Dieser kann entweder voll qualifiziert oder nicht

[12] Dies ist das Standardverfahren zum Erzeugen von Geometrien in einer ODB, die bei der Konvertierung aus dem FOS–Format entstanden ist. Bei auf der Basis der ODB neu erfaßten Geometrien wird die Verwendung externer Geometrien nicht empfohlen, da die Verwendung von direkt durch die ODB unterstützten grafischen Primitiven effizienter ist.

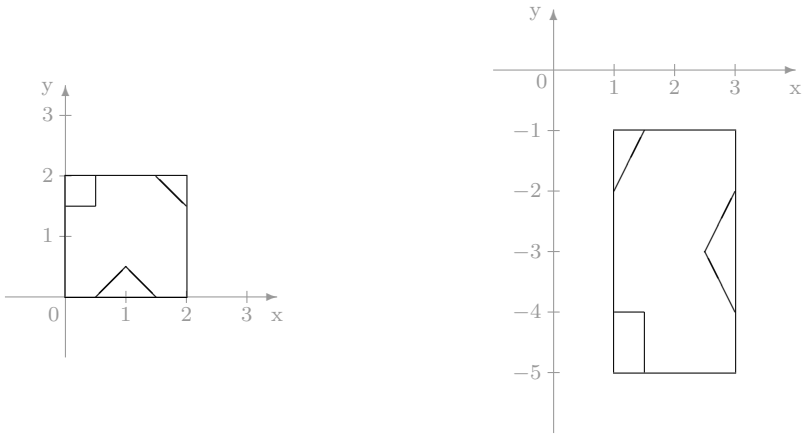
qualifiziert sein. Im zweiten Fall wird die Geometrie in dem gleichen Paket gesucht, in dem sich auch die ODB befindet.

Die Koordinaten in der externen Geometrie werden in dem lokalen Koordinatensystem des ODB–Eintrags, der die externe Geometrie referenziert, interpretiert. Sie können anschließend durch Skalierung, Rotation und Verschiebung transformiert werden.

Gegenwärtig ist es nicht möglich, für externe Geometrien Attribute zu setzen.

Im folgenden Beispiel wird ein EGM–Symbol referenziert. Das EGM–Symbol in seinem lokalen Koordinatensystem ist im linken Bild zu sehen. Die resultierende Geometrie im Koordinatensystem des OFML–Objekts ist im rechten Bild dargestellt. Der Name des EGM–Symbols ist `bar`. Da der Name nicht qualifiziert ist, muß sich in dem Paket, in dem auch die ODB liegt, eine Datei mit dem Namen `bar.egms` befinden.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		3.0	−5.0	90.0	2.0	1.0	"bar" egms	



2.8 Attribute

In der Spalte `attrib` der 2D–Tabelle können verschiedene für das in dem jeweiligen Eintrag erzeugte 2D–Objekt gültige Attribute gesetzt werden.

Die Spalte `attrib` kann auch eine Funktion enthalten, die in der Funktionstabelle definiert ist. Diese Funktion, wie auch Funktionen, die direkt oder indirekt durch sie aufgerufen werden, können alle im folgenden beschriebenen Funktionen zum Setzen von Attributen aufrufen.

2.8.1 Farbe

Die Funktion zum Setzen der Farbe heißt `col`. Sie erwartet drei Argument im Bereich von einschließlich 0.0 bis einschließlich 1.0, die den roten, grünen und blauen Anteil der Farbe spezifizieren. Wird für ein Objekt keine Farbe gesetzt, so wird das Objekt schwarz dargestellt.

Die Farbe kann für alle grafischen Primitiven gesetzt werden. Im einzelnen sind dies `hline`, `vline`, `dline`, `quadrat`, `circle`, `arc`, `ellipse`, `point` und `text`.

Im folgenden Beispiel wird ein rotes 2.0×1.0 großes Rechteck mit zwei blauen Diagonalen erzeugt. Die linke obere Ecke des Rechtecks liegt im Ursprung des Koordinatensystems des OFML–Objekts, und die rechte untere Ecke liegt bei $2.0, -1.0$.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.0	−1.0	0.0	2.0	1.0		
	1		0.0	0.0	0.0	1.0	1.0	quadrat	1.0 0.0 0.0 col
	1		0.0	0.0	0.0	1.0	1.0	dline	0.0 0.0 1.0 col
	1		0.0	1.0	0.0	1.0	−1.0	dline	0.0 0.0 1.0 col

2.8.2 Linienstärke

Die Funktion zum Setzen der Linienstärke heißt `lwidth`. Sie erwartet als Argument eine positive Zahl, die die Stärke der Linie in Pixel spezifiziert. Die normale Stärke einer Linie ist 1 Pixel.

Die Linienstärke kann für die Primitiven `hline`, `vline`, `dline`, `quadrat`, `circle`, `arc` und `ellipse` gesetzt werden.

Das folgende Beispiel erzeugt eine Ellipse, deren Mittelpunkt bei $1.0, -0.5$ liegt und die in x–Richtung einen Radius von 1.0 und in y–Richtung einen Radius von 0.5 hat. Die Linienstärke ist 2 Pixel.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		1.0	−0.5	0.0	1.0	1.0	1.0 0.5 ellipse	2 lwidth

2.8.3 Linienart

Die Funktion zum Setzen der Linienart heißt `lstyle`. Sie erwartet als Argumente zwei Zahlen, die das Muster der Linie und den Streckungsfaktor des Musters angeben. Standardmäßig werden Linien durchgezogen dargestellt, oder gestrichelt, wenn das Objekt, zu dem sie gehören, selektiert wurde.

Die Linienart kann für die Primitiven `hline`, `vline`, `dline`, `quadrat`, `circle`, `arc` und `ellipse` gesetzt werden.

Das erste Argument ist das Linienmuster und kann die in Tabelle 6 aufgeführten Werte annehmen. Das zweite Argument ist der Faktor, dessen Wert als Pixelanzahl interpretiert wird. Die genaue Bedeutung des Faktors in Abhängigkeit von dem verwendeten Muster ist ebenfalls in Tabelle 6 beschrieben[13].

Bei der Verwendung von Linienmustern ist zu beachten, daß die Anzeige von selektierten Objekten darauf beruht, daß Linien gestrichelt dargestellt werden. Dies geschieht allerdings nur für Linien, denen nicht explizit ein anderes Linienmuster als -1 zugewiesen wurde. Bei der Erstellung von 2D–Symbolen ist folglich darauf zu achten, daß nicht allen Linien des Symbols ein Linienmuster zugewiesen wird.

[13] Die tatsächliche Länge der Liniensegmente kann von den in Tabelle 6 gemachten Angaben je nach Faktor und verwendetem Treiber für die 2D–Ausgabe abweichen. So ist insbesondere der OpenGL–Treiber in seinen Möglichkeiten stark eingeschränkt, während der X11–Treiber im Rahmen der Möglichkeiten eines Pixel–Displays relativ exakte Ergebnisse liefert.

Wert	Beschreibung
−1	Es wird die voreingestellte Linienart verwendet.
0	Es wird eine durchgehende Linie gezeichnet.
1	Es wird eine gestrichelte Linie gezeichnet. Der Faktor bestimmt die Länge der dargestellten und nicht dargestellten Liniensegmente.
2	Es wird eine gepunktete Linie gezeichnet. Der Faktor bestimmt den Abstand zwischen den Mittelpunkten zweier benachbarter Punkte.
3	Es wird eine Strich–Punkt Linie gezeichnet. Der Faktor bestimmt die Länge des dargestellten Liniensegments und die halbe Länge der nicht dargestellten Segmente.
4	Es wird eine Strich–2–Punkt Linie gezeichnet. Der Faktor bestimmt die Länge des dargestellten Liniensegments und ein Drittel der Länge der nicht dargestellten Segmente.
5	Es wird eine Strich–3–Punkt Linie gezeichnet. Der Faktor bestimmt die Länge des dargestellten Liniensegments und ein Viertel der Länge der nicht dargestellten Segmente.

Tabelle 6: Linienmuster

Im folgenden Beispiel wird ein gestricheltes Rechteck erzeugt, das die gleichen Abmessungen wie das Rechteck im Beispiel von Abschnitt 2.8.1 hat, und dessen Diagonalen gepunktet dargestellt werden. Der Streckungsfaktor ist in jedem Fall 4, womit normalerweise vernünftige visuelle Ergebnisse erzielt werden.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		0.0	0.0	0.0	2.0	−1.0	quadrat	1 4 lstyle
	0		0.0	0.0	0.0	2.0	−1.0	dline	2 4 lstyle
	0		0.0	−1.0	0.0	2.0	1.0	dline	2 4 lstyle

2.8.4 Punktgröße

Der Durchmesser eines Punktes wird mit der Funktion `psize` gesetzt. Sie erwartet ein Argument, daß die Größe des Punktes in Pixeln angibt. Die normale Größe eines Punktes ist 1 Pixel.

Die Punktgröße kann für Objekte gesetzt werden, die mit der Funktion `point` erzeugt worden sind.

Das folgende Beispiel ist identisch mit dem aus Abschnitt 2.7.4, mit der Ausnahme, daß der Punkt mit einem Durchmesser von 5 Pixeln dargestellt wird.

odb_name	level	visible	offs		rot	scale		ctor	attrib
			x	y		x	y		
FOO	0		1.0	−0.5	0.0	1.0	1.0	point	5 psize

2.8.5 Fontgröße

Für Objekte, die mit der Funktion `text` erzeugt worden sind, kann mit der Funktion `fheight` die Höhe des verwendeten Fonts gesetzt werden.

Die Funktion `fheight` erwartet ein Gleitkommaargument, welches die Höhe des Fonts in Einheiten des lokalen Koordinatensystems angibt. Unter der Höhe des Fonts wird die Höhe eines Großbuchstabens ohne Unterlänge verstanden. Sie ist standardmäßig 0.1.

2.8.6 Fontstreckung

Mit der Funktion `faspect` wird für Objekte, die mit der Funktion `text` erzeugt worden sind, die Streckung des verwendeten Fonts festgelegt.

Die Funktion `faspect` erwartet ein Gleitkommaargument, das die Streckung des Fonts angibt. Der Standardwert für die Streckung ist 1.0. Ein Wert zwischen 0.0 und 1.0 führt dazu, daß der Font in x–Richtung gestaucht dargestellt wird, während ein Wert größer 1.0 zu einer gestreckten Darstellung des Fonts in x–Richtung führt. Eine Streckung von 0.0 ist nicht erlaubt. Für negative Werte ist das Verhalten undefiniert.

2.8.7 Ebene

Mit der Funktion `layer` kann jedes Objekt einer Ebene (s. Abschnitt 6) zugeordnet werden. Die Funktion erwartet als Argument eine Zeichenkette, die den Namen der Ebene enthält.

3 3D–Geometrien

Die 3D–Geometrie eines OFML–Objekts wird durch einen oder mehrere aufeinanderfolgende Einträge in der 3D–Tabelle beschrieben. Jeder Eintrag dient zur Erzeugung einer grafischen Primitive[14] und enthält deren Position und Rotation sowie weitere Attribute wie Materialien, Selektierbarkeit und andere.

Der Aufbau der Tabelle mit den 3D–Geometrien ist in Tabelle 7 zusammengefaßt und anschließend genauer beschrieben.

Feld-nummer	Feld-name	Beschreibung
1	`odb_name`	ODB–Name
2	`obj_name`	Name des Objekts
3	`exist`	Steuerung der Erzeugung
4	`x_offs`	Verschiebung in x–Richtung
5	`y_offs`	Verschiebung in y–Richtung
6	`z_offs`	Verschiebung in z–Richtung
7	`x_rot`	Rotation um die x–Achse
8	`y_rot`	Rotation um die y–Achse
9	`z_rot`	Rotation um die z–Achse
10	`ctor`	Erzeugung des 3D–Objekts
11	`mat`	Zuweisung von Material(ien)
12	`attrib`	Setzen von grafischen Attributen
13	`link`	reserviert für zukünftige Verwendung

Tabelle 7: 3D–Geometrien

3.1 ODB–Name

Objekte, für die mittels der ODB eine 3D–Geometrie erzeugt werden soll, liefern einen voll qualifizierten ODB–Namen. Dieser besteht aus dem Namen des Paketes, in dem die zu verwendende ODB liegt, plus dem ODB–Grundnamen, der zur Ermittlung der zu verwendenden Einträge in der 2D– und der 3D–Tabelle dienen. Ein Beispiel für einen voll qualifizierten ODB–Namen ist `::foo::bar::BAZ`, wobei `::foo::bar` der Paketname und `BAZ` der ODB–Grundname sind.

Die 2D–Tabelle besteht aus einer Folge von ODB–Blöcken. Ein ODB–Block besteht aus mehreren aufeinanderfolgenden Einträgen, von denen der erste in der Spalte `odb_name` den ODB–Grundnamen enthält, wärend bei allen folgenden Einträgen dieses Blockes die Spalte `odb_name` leer ist.

3.2 Objekt–Name

Für ein zu erzeugendes Objekt muß ein relativer Name angegeben werden, der sich auf das übergeordnete OFML–Objekt bezieht. Folgende Regeln gelten für Objekt–Namen:

[14] Beim Referenzieren externer Geometrien kann ein Eintrag auch komplexe 3D–Geometrien erzeugen, die jedoch immer als ein zusammenhängendes Objekt behandelt werden.

- Ein Name darf innerhalb eines ODB–Blocks[15] nur einmal vergeben werden.
- In einem Namen kann eine hierarchische Zuordnung vorgenommen werden. Insofern ist ein Name im allgemeinen Fall eine Verkettung von elementaren Namen unter Anwendung des Punktes (`.`) als Verkettungsoperator.
- Sofern durch den Namen eines Objekts die Existenz eines hierarchischen Vorgängers impliziert wird, muß dieser in der Tabelle vor dem Nachfolgerobjekt definiert werden.
- Per Konvention besteht ein elementarer Name aus dem Präfix `o`, gefolgt von einer Ganzzahl. Diese Zahl beginnt für einen gegebenen Vorgänger jeweils bei 1 für den ersten Nachfolger und wird bei weiteren Nachfolgern desselben Vorgängers entsprechend inkrementiert.

Im folgenden — unvollständigen — Beispiel wird die Erzeugung von vier Objekten illustriert. Auf oberster Ebene — diese entspricht dem Level 0 der 2D–ODB — werden zunächst zwei Objekte erzeugt, welche aufgrund obiger Konvention die Namen `o1` und `o2` bekommen. Die restlichen beiden Objekte sollen bezüglich verschiedener Merkmale relativ zum Objekt `o2` definiert werden. Aus diesem Grund wird eine neue Hierarchieebene bezüglich `o2` eingeführt; dementsprechend sind die zu vergebenden Namen `o2.o1` und `o2.o2`.

odb_ name	obj_ name	ex ist	offs			rot			ctor	mat	attrib	link
			x	y	z	x	y	z				
BAZ	o1											
	o2											
	o2.o1											
	o2.o2											

3.3 Steuerung der Erzeugung

Über einen optionalen Eintrag in der Spalte `exist` kann die Erzeugung eines Objekts gesteuert werden. Das in der Zeile beschriebene Objekt wird erzeugt, wenn die Spalte `exist` entweder leer ist oder der darin enthaltene Ausdruck, in der Regel in Form einer Funktion, einen numerischen Wert liefert, der von 0 verschieden ist. Andernfalls wird das Objekt nicht erzeugt.

Ein Objekt wird auch dann nicht erzeugt, wenn einer seiner hierarchischen Vorgänger auf Grund des Eintrags in der Spalte `exist` nicht erzeugt wird.

3.4 Verschiebung

Jedes einzufügende Objekt hat einen Einfügepunkt, der durch den Typ des Objekts eindeutig festgelegt ist und sich immer im Ursprung des lokalen Koordinatensystems des Objektes befindet. Im Fall eines Quaders ist dies die hintere, untere, linke Ecke, im Fall einer Kugel deren Mittelpunkt. Bei der Erzeugung eines Objekts liegt dieser Einfügepunkt im Ursprung des Koordinatensystems des Vorgängers. Durch den Verschiebeparameter kann der Einfügepunkt relativ zum Koordinatensystem des Vorgängers in allen drei Richtungen verschoben werden.

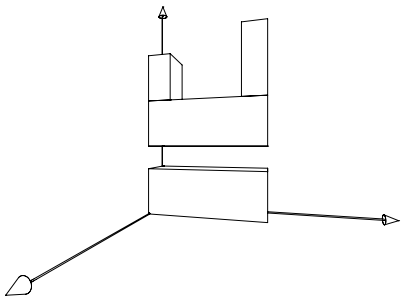
[15] Ein ODB–Block sind alle Einträge, die unter einem ODB–Namen zusammengefaßt sind.

Durch eine derartige Verschiebung des Objekts verschieben sich etwaige Nachfolger dementsprechend.

Eine Verschiebung ist stets unabhängig von einer eventuellen Rotation des Objekts.

Im nachfolgenden Beispiel ist `o1` nicht gegenüber dem Koordinatensystem des OFML–Objekts verschoben. Der Ursprung von `o2` in Bezug auf das Koordinatensystem des OFML–Objekts ergibt sich zu $(0.0, 3.0, 0.0)$. Die Ursprünge von `o2.o1` und `o2.o2` ergeben sich in Bezug auf das Koordinatensystem des OFML–Objekts zu $(0.0, 0.5, 0.0)$ und $(0.4, 0.5, 0.0)$.

odb_ name	obj_ name	ex ist	offs x	offs y	offs z	rot x	rot y	rot z	ctor	mat	attrib	link
BAZ	o1		0.0	0.0	0.0	0.0	0.0	0.0	0.5 0.2 0.2 block			
	o2		0.0	0.3	0.0	0.0	0.0	0.0	0.5 0.2 0.2 block			
	o2.o1		0.0	0.2	0.0	0.0	0.0	0.0	0.1 0.2 0.2 block			
	o2.o2		0.4	0.2	0.0	0.0	0.0	0.0	0.1 0.3 0.2 block			



3.5 Rotation

Durch Angabe von Rotationswinkeln ungleich 0.0 kann ein Objekt aus der durch dessen Typ vorgegebenen Orientierung bezüglich des OFML–Objekts oder seines Vorgängers in eine andere Ausrichtung gedreht werden.

Eine Rotation um die Rotationswinkel (x, y, z) wird auf folgende Weise auf Elementarrotationen abgebildet:

1. Rotation um x bezüglich der initialen x–Achse
2. Rotation um y bezüglich der y–Achse des Koordinatensystems nach Schritt 1
3. Rotation um z bezüglich der z–Achse des Koordinatensystems nach Schritt 2

Dabei ist zu beachten, daß es bei Angabe mehrerer Rotationswinkel ungleich 0.0 zu Wechselwirkungen zwischen den jeweils elementaren Rotationen kommt.

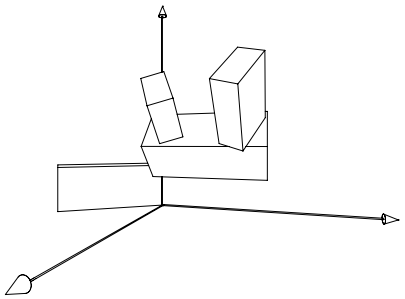
Anmerkung. Durch Verwendung von Hierarchieebenen, wobei jeder Hierarchieebene genau eine Elementarrotation zugewiesen wird, kann die Abarbeitungsreihenfolge der Elementarrotationen selbst bestimmt werden.

Die Angabe von Rotationswinkeln erfolgt im Gradmaß und im mathematisch positiven Sinn, d.h. entgegen dem Uhrzeiger.

Durch eine derartige Verdrehung des Objekts verdrehen sich etwaige Nachfolger dementsprechend.

Eine Verdrehung ist stets unabhängig von einer eventuellen Verschiebung des Objekts.

odb_ name	obj_ name	ex ist	offs x	offs y	offs z	rot x	rot y	rot z	ctor	mat	attrib	link
BAZ	o1		0.0	0.0	0.0	0.0	−143.2	0.0	0.5 0.2 0.2 block			
	o2		0.0	0.3	0.0	0.8	0.0	0.0	0.5 0.2 0.2 block			
	o2.o1		0.0	0.2	0.0	0.0	22.9	0.0	0.1 0.2 0.2 block			
	o2.o2		0.4	0.2	0.0	0.0	−22.9	0.0	0.1 0.3 0.2 block			



3.6 Objekterzeugung

Die Spalte `ctor` dient zur Erzeugung von 3D–Objekten. Dies erfolgt über den Aufruf genau einer der folgenden Funktionen einschließlich der Bereitstellung der entsprechenen Parameter.

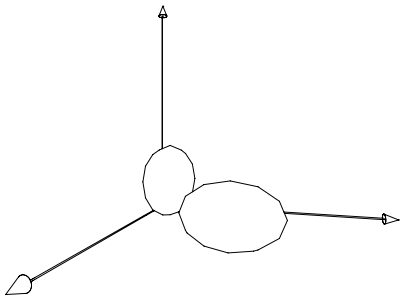
Anmerkung. Bei allen nachfolgend vorgestellten Primitiven ist der Ursprung des lokalen Koordinatensystems Bestandteil des lokalen Begrenzungsvolumens.

3.6.1 Ellipsoid

Die Funktion `ellipsoid` erzeugt einen homogenen Ellipsoid, welcher im Ursprung des lokalen Koordinatensystems beginnt und sich entsprechend den drei Radii nach allen Seiten ausdehnt.

Die Parametrisierung erfolgt durch Angabe dreier Radii in der Reihenfolge x, y und z.

odb_ name	obj_ name	ex ist	offs x	offs y	offs z	rot x	rot y	rot z	ctor	mat	attrib	link
BAZ	o1		0.1	0.15	0.2	0.0	0.0	0.0	0.1 0.15 0.2 ellipsoid			
	o2		0.4	0.1	0.6	0.0	0.0	0.0	0.4 0.1 0.6 ellipsoid			



3.6.2 Import

Die Funktion `imp` importiert einen externen 3D–Datensatz und erzeugt daraus eine entsprechende Primitive. Es werden die folgenden Formate unterstützt:

1. 3DS

   3DS ist ein binäres Format, das Geometrien auf Basis von Dreieckslisten beschreibt. Weitere Bestandteile von 3DS–Dateien sind Material-, Beleuchtungs- und Kameradaten, wobei die beiden letzteren beim Einlesen ignoriert werden.

   Ein 3DS–Datensatz wird durch die ODB so eingelesen, daß die minimale Koordinate seines orthogonalen Begrenzungsvolumens mit dem Ursprung des lokalen Koordinatensystems in Übereinstimmung gebracht wird.

   Die Dateinamenserweiterung von 3DS–Dateien ist `.3ds`.

2. OFF

   OFF ist ein einfaches ASCII–Format zur Beschreibung von indizierten polygonalen Objekten. Zusätzlich zur Geometriedatei mit der Erweiterung `.geo` werden von der ODB–Laufzeitumgebung folgende optionale Dateien unterstützt:

   - Polygon–Farben
     Über eine gleichnamige Datei mit der Erweiterung `.ipc` können Farben im RGB–Format per Polygon zugewiesen werden.
   - Vertex–Normalen
     Über eine gleichnamige Datei mit der Erweiterung `.vnm` können Normalenvektoren per Vertex zugewiesen werden.

Allgemein gilt: Die von der ODB einzulesenden Datensätze sollen einseitig sein und geschlossene Körper beschreiben. Die Elementarflächen (Dreiecke bzw. Polygone) müssen einfach, planar, convex und mit dem Uhrzeigersinn orientiert sein.

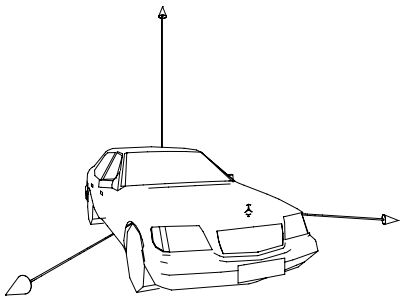
Falls ein Datensatz keine oder keine korrekten Materialien beinhaltet, können die Materialien auf ODB–Ebene zugewiesen werden. Andernfalls bleibt die Materialsetzung auf ODB–Ebene ohne sichtbare Auswirkungen.

Je Datensatz wird optional eine auflösungsreduzierte Variante unterstützt. Die Dateien dieses Datensatzes unterscheiden sich vom primären Datensatz durch einen vorangestellten Unterstrich.

Der erste Funktionsparameter ist der optional voll qualifizierte Name des Datensatzes ohne Dateinamenserweiterung. Ist der Name nicht oder nicht voll qualifiziert, wird ihm automatisch der Qualifikator des voll qualifizierten ODB–Names vorangestellt. Der Datensatz muß dann folglich im gleichen Paket wie die ODB zu finden sein.

Über die drei folgenden Parameter erfolgt die Skalierung der Geometrie in der Reihenfolge x, y und z.

odb_ name	obj_ name	ex ist	offs			rot			ctor	mat	attrib	link
			x	y	z	x	y	z				
BAZ	o1		0.0	0.0	0.0	0.0	−50.1	0.0	"w140" 0.2 0.2 0.2 imp			

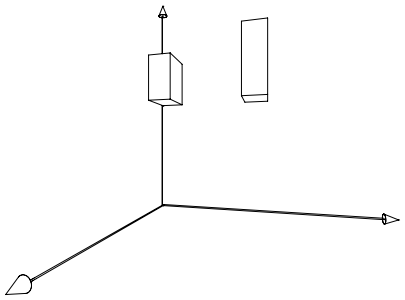


3.6.3 Knoten

Die Funktion `top` erzeugt ein unsichtbares Objekt, das üblicherweise zur Zusammenfassung von Objekten, z.B. mit dem Ziel einer gemeinsamen Plazierung, verwendet wird.

Es ist zu beachten, daß weder das Top–Objekt noch einer seiner direkten oder indirekten Nachfolger selektierbar sein darf.

odb_ name	obj_ name	ex ist	offs			rot			ctor	mat	attrib	link
			x	y	z	x	y	z				
BAZ	o1		0.0	0.3	0.0	0.0	0.0	0.0	top			
	o1.o1		0.0	0.2	0.0	0.0	0.0	0.0	0.1 0.2 0.2 block			
	o1.o2		0.4	0.2	0.0	0.0	0.0	0.0	0.1 0.3 0.2 block			

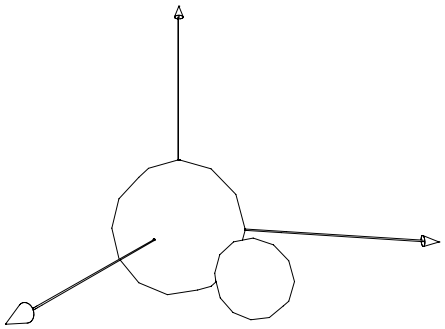


3.6.4 Kugel

Die Funktion `sphere` erzeugt eine homogene Kugel, welche im Ursprung des lokalen Koordinatensystems beginnt und sich gleichmäßig nach allen Seiten ausdehnt.

Die Parametrisierung erfolgt durch Angabe eines Radius.

odb_ name	obj_ name	ex ist	offs x	offs y	offs z	rot x	rot y	rot z	ctor	mat	attrib	link
BAZ	o1		0.0	0.0	0.0	0.0	0.0	0.0	0.2 0.3 0.4 block			
	o2		0.3	0.0	0.5	0.0	0.0	0.0	0.3 0.15 0.2 block			



3.6.5 Loch

Die Funktion `hole` realisiert kreisförmige oder rechteckförmige Durchbrüche in kreisförmigen oder rechteckförmigen Bereichen. Damit können boolesche Operationen, insbesondere die Subtraktion, in Spezialfällen simuliert werden. Es erfolgt jedoch keine tatsächliche Subtraktion im Sinne einer booleschen Operation. Der eigentliche Sinn der Funktion `hole` besteht in der Generierung der Flächen für die Kombinationen kreisförmige Außenlinie – rechteckförmiges Loch und rechteckförmige Außenlinie – kreisförmiges Loch. Es werden jedoch keine Außenflächen entlang der Außenlinie in der lokalen z–Richtung erzeugt.

Ein Lochobjekt wird zentriert in Bezug auf die Außenfläche im lokalen Ursprung erzeugt. In der Tiefe beginnt ein Lochobjekt im Ursprung des lokalen Koordinatensystems und dehnt sich entlang der negativen z–Achse aus.

Generell gilt: Ein Loch muß stets vollständig im äußeren Umriß enthalten sein. Es darf diesen auch nicht erreichen. Einzige Ausnahme: die Tiefen von Außenumriß und Loch können identisch sein. In diesem Fall ist das Loch durchsichtig; ansonsten besitzt es einen Boden.

Die Erzeugungsparameter sind wie folgt:

1. Außenform

   Bei Angabe des Wertes `"R"` ist die äußere Form des Objekts ein Rechteck, bei Angabe des Wertes `"C"` dagegen ein Kreis.

2. Außenbreite

   Falls die äußere Form des Objekts ein Rechteck ist, gibt dieser Wert die äußere Breite an, ansonsten wird der Radius des Außenumrisses angegeben.

3. Außenhöhe

   Falls die äußere Form des Objekts ein Rechteck ist, gibt dieser Wert die äußere Höhe an, ansonsten wird der Wert ignoriert.

4. Außentiefe

   Der Wert gibt die äußere Tiefe des Objekts an.

5. Rückfläche

   Dieser Wert steuert die Generierung einer Rückfläche entsprechend der stets generierten Vorderfläche. Bei Angabe des Wertes `1` wird die Fläche erzeugt. Ansonsten ist `0` anzugeben.

6. Lochform

   Bei Angabe des Wertes `"R"` ist die Form des Loches ein Rechteck, bei Angabe des Wertes `"C"` dagegen ein Kreis.

7. Lochbreite

   Falls die Form des Loches ein Rechteck ist, gibt dieser Wert die Breite des Loches an, ansonsten wird der Radius des Loches angegeben.

8. Lochhöhe

   Falls die Form des Loches ein Rechteck ist, gibt dieser Wert die Höhe des Loches an; ansonsten wird der Wert ignoriert.

9. Lochtiefe

   Der Wert gibt die Tiefe des Loches an.

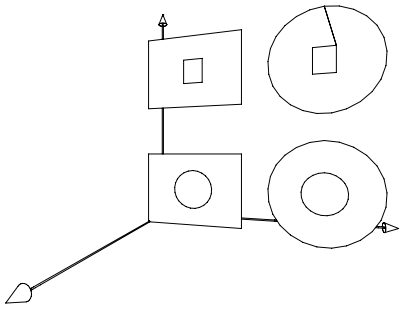
10. Lochoffset in x–Richtung

    Dieser Wert gibt das Offset des Lochmittelpunkts in x–Richtung bezüglich des Ursprungs des lokalen Koordinatensystems, welcher zugleich auch Mittelpunkt des Objekts ist, an.

11. Lochoffset in y–Richtung

    Dieser Wert gibt den Offset des Lochmittelpunkts in y–Richtung bezüglich des Ursprungs des lokalen Koordinatensystems, welcher zugleich auch Mittelpunkt des Objekts ist, an.

odb_ name	obj_ name	ex ist	offs x	offs y	offs z	rot	ctor	...
BAZ	o1		0.2	0.15	0.2	...	"R" 0.4 0.3 0.2 1 "C" 0.08 0.1 0.2 0.0 0.0 hole	
	o2		0.2	0.65	0.2	...	"R" 0.4 0.3 0.2 1 "R" 0.08 0.1 0.2 0.0 0.0 hole	
	o3		0.7	0.15	0.2	...	"C" 0.2 0.2 0.2 1 "C" 0.08 0.1 0.2 0.0 0.0 hole	
	o4		0.7	0.65	0.2	...	"C" 0.2 0.2 0.2 1 "R" 0.08 0.1 0.2 0.0 0.0 hole	



3.6.6 Parametrische Fläche

Die Funktion `surf` erzeugt ein dreidimensionales Objekt auf Basis eines zweidimensionalen Netzes. Die Netzkoordinaten stellen dabei Stützstellen dar, die von der resultierenden Fläche ohne Kanten verbunden werden. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$x_0\ y_0\ z_0\ \ldots x_{udim\times wdim-1}\ y_{udim\times wdim-1}\ z_{udim\times wdim-1}\ udim\ wdim\ umode\ wmode\ \texttt{surf}$$

Die Dimensionen des Netzes lauten *udim* und *wdim*. Dementsprechend müssen $udim \times wdim$ dreidimensionale Koordinaten zur Definition des Netzes angegeben werden. Innerhalb einer jeden Elementarfläche gibt die Rechte–Hand–Regel die Orientierung an[16].

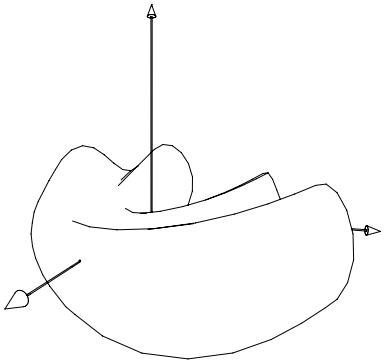
Über die Flags *umode* und *wmode* kann angegeben werden, ob die Fläche entlang der entsprechenden Netzrichtung geschlossen werden soll (`1`) oder nicht (`0`).

Das folgende Beispiel erzeugt eine parametrische Fläche aus 32 Stützpunkten, welche entlang einer Netzrichtung geschlossen ist und anschließend gedreht wurde.

```
-0.150815  0.064026 -0.919388  0.075575 -0.269460 -0.810568 \
 0.329356 -0.102473 -0.677988  0.576438 -0.273374 -0.555442 \
 0.772948  0.063867 -0.448708  0.563432  0.366948 -0.549524 \
 0.327520  0.171928 -0.673550  0.062688  0.355544 -0.804890 \
```

[16] Wenn der Daumen der rechten Hand senkrecht auf der Fläche steht, geben die restlichen Finger der Hand die Orientierung an.

```
-0.490592 -0.095985 -0.379506 -0.371192 -0.281256 -0.307272 \
-0.237910 -0.188485 -0.220444 -0.107800 -0.283430 -0.139519 \
-0.004815 -0.096073 -0.070032 -0.115308  0.072304 -0.136939 \
-0.239162 -0.036040 -0.218088 -0.378620  0.065969 -0.304818 \
-0.565542 -0.069120  0.606108 -0.431332 -0.277990  0.529648 \
-0.275384 -0.175605  0.441702 -0.126945 -0.283430  0.357456 \
-0.004079 -0.074762  0.288498 -0.128341  0.115085  0.359282 \
-0.273712 -0.004849  0.441200 -0.432810  0.110994  0.531496 \
 0.122084 -0.013038  0.885150  0.226666 -0.273824  0.722364 \
 0.348838 -0.145497  0.535098  0.464742 -0.279946  0.355724 \
 0.561242 -0.018840  0.208880  0.464406  0.218190  0.359580 \
 0.350472  0.067949  0.534016  0.226248  0.212398  0.726268 \
8 4 1 0 surf
```



3.6.7 Polygon

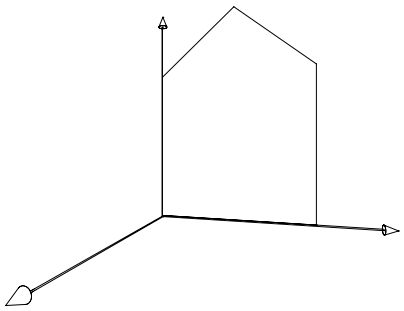
Die Funktion `polyg` erzeugt ein Polygon im Raum auf Basis einer Liste von dreidimensionalen Koordinaten. Diese müssen in Uhrzeigerrichtung angegeben werden. Die letzte Koordinate wird automatisch mit der ersten verbunden. Das somit beschriebene Polygon muß einfach, konvex und planar sein. Das generierte Polygon hat keine Rückfläche.

Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$x_0\ y_0\ z_0\ \ldots x_{n-1}\ y_{n-1}\ z_{n-1}\ n\ \texttt{polyg}$$

Das folgende Beispiel erzeugt ein einfaches Polygon:

```
0.0 0.0 0.0 0.0 0.7 0.0 0.35 1.0 0.0 0.7 0.7 0.0 0.7 0.0 0.0 5 polyg
```

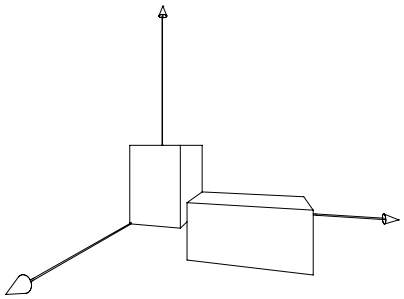


3.6.8 Quader

Die Funktion `block` erzeugt einen homogenen Quader, welcher im Ursprung des lokalen Koordinatensystems beginnt und sich entlang der positiven Achsen des lokalen Koordinatensystems entsprechend ausdehnt.

Die Parametrisierung erfolgt durch Angabe von Breite, Höhe und Tiefe.

odb_ name	obj_ name	ex ist	offs x	offs y	offs z	rot x	rot y	rot z	ctor	mat	attrib	link
BAZ	o1		0.0	0.0	0.0	0.0	0.0	0.0	0.2 0.3 0.4 block			
	o2		0.3	0.0	0.5	0.0	0.0	0.0	0.3 0.15 0.2 block			

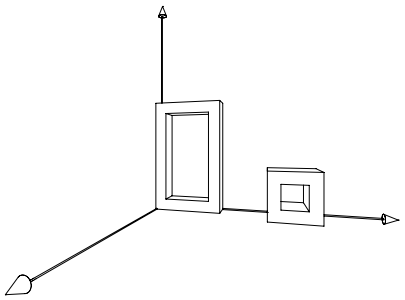


3.6.9 Rahmen

Die Funktion `frame` erzeugt einen Rahmen, der im Ursprung des lokalen Koordinatensystems beginnt und sich entlang der positiven Achsen des lokalen Koordinatensystems entsprechend ausdehnt. Dabei wird aus dem Körper in der lokalen x–y–Ebene ein orthogonales Volumen subtrahiert. Die Dicke des Rahmens in x– und y–Richtung ist gleich. Für die Abmessungen in x– und y–Richtung w und h und für die x/y–Dicke th muß stets gelten: $w, h > 2 \times th$.

Die Parametrisierung erfolgt durch Angabe von Breite, Höhe, Tiefe und Dicke des Rahmens.

odb_ name	obj_ name	ex ist	offs x	offs y	offs z	rot x	rot y	rot z	ctor	mat	attrib	link
BAZ	o1		0.0	0.0	0.0	0.0	0.0	0.0	0.3 0.5 0.1 0.05 frame			
	o2		0.5	0.0	0.0	0.0	0.0	0.0	0.2 0.2 0.2 0.05 frame			



3.6.10 Rotationskörper

Die Funktion `rot` erzeugt ein dreidimensionales Objekt durch Drehung einer dreidimensionalen Definitionskurve. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$axis_x\ axis_y\ axis_z\ x_0\ y_0\ z_0\ \ldots x_{n-1}\ y_{n-1}\ z_{n-1}\ n\ angle\ smooth\ u\ w\ c0\ c1\ \texttt{rot}$$

Die Parameter $axis_x$, $axis_y$ und $axis_z$ spezifizieren die Rotationsachse durch einen normierten Vektor.

Die Parameter x_0 bis z_{n-1} beschreiben die Definitionskurve und der Parameter n enthält die Anzahl der Koordinaten der Definitionskurve. Die Definitionskurve und die Rotationsachse sollten stets in einer Ebene liegen. Koordinaten, die exakt auf der Rotationsachse liegen, sind zu vermeiden. Die Kurve muß nach der Rechte–Hand–Regel wie folgt definiert sein: Wenn der Daumen der rechten Hand in Richtung der Drehung zeigt, geben die restlichen Finger der Hand die Orientierung an.

Der Parameter *angle* gibt den Drehwinkel an. Zur Erzeugung eines homogenen Rotationskörpers muß hierfür der Wert 360.0 angegeben werden.

Der Parameter *smooth* bestimmt, ob die Koordinaten der Definitionskurve linear (`0`) oder mit weichen Übergängen (`1`) miteinander verbunden werden.

Der Parameter u gibt an, ob der letzte und der erste Punkt der Definitionskurve miteinander verbunden werden sollen (`1`) oder nicht (`0`).

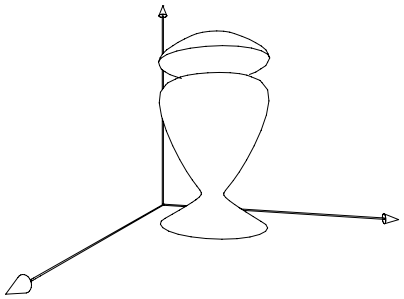
Der Parameter w gibt an, ob der Körper entlang der Drehung geschlossen ist (`1`) oder nicht (`0`). Falls *angle* den Wert 360.0 hat, wird normalerweise der Wert `1` angegeben, ansonsten üblicherweise der Wert `0`.

Der Parameter $c0$ gibt an, ob für die erste und die letzte Koordinate der Definitionsachse eine lineare Deckelfläche senkrecht zur Rotationsachse erzeugt werden soll (`1`) oder nicht (`0`).

Der Parameter $c1$ gibt an, ob für den Fall, daß $angle < 360.0$ und w ungleich 1 ist, die entstehenden inneren Flächen ausgefüllt werden sollen (`1`) oder nicht (`0`).

Das nachfolgende Beispiel erzeugt einen homogenen Rotationskörper um die lokale y–Achse, der anschließend in x– und z–Richtung verschoben wurde.

```
0.0 0.1 0.0 \
0.2 0.0 0.0 0.05 0.1 0.0 0.1 0.2 0.0 0.2 0.5 0.0 0.05 0.55 0.0 0.2 0.6 0.0 0.1 0.7 0.0 \
7 360.0 1 0 1 1 0 rot
```



Die Funktion `rotx` erzeugt einen Rotationskörper um die lokale x–Achse. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$x_0\ y_0\ \ldots x_{n-1}\ y_{n-1}\ n\ angle\ smooth\ u\ w\ c0\ c1\ \texttt{rotx}$$

Die Funktion `roty` erzeugt einen Rotationskörper um die lokale y–Achse. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$x_0\ y_0\ \ldots x_{n-1}\ y_{n-1}\ n\ angle\ smooth\ u\ w\ c0\ c1\ \texttt{roty}$$

Die Funktion `rotz` erzeugt einen Rotationskörper um die lokale z–Achse. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$y_0\ z_0\ \ldots y_{n-1}\ z_{n-1}\ n\ angle\ smooth\ u\ w\ c0\ c1\ \texttt{rotz}$$

3.6.11 Ziehkörper

Die Funktion `sweep` erzeugt ein dreidimensionales Objekt durch lineares Ziehen einer dreidimensionalen Definitionskurve entlang einer vorgegebenen Richtung. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$axis_x\ axis_y\ axis_z\ len\ x_0\ y_0\ z_0\ \ldots x_{n-1}\ y_{n-1}\ z_{n-1}\ n\ smooth\ u\ c0\ c1\ \texttt{sweep}$$

Die Parameter $axis_x$, $axis_y$ und $axis_z$ spezifizieren die Ziehrichtung durch einen normierten Vektor.

Der Parameter *len* gibt die Länge entlang der Ziehrichtung an.

Die Parameter x_0 bis z_{n-1} beschreiben die Definitionskurve, der Parameter n die Anzahl der Koordinaten der Definitionskurve. Die Definitionskurve sollte stets in einer Ebene liegen, auf der außerdem der Ziehvektor senkrecht stehen sollte. Die Kurve muß nach der Rechte–Hand–Regel wie folgt definiert sein: Wenn der Daumen der rechten Hand in Richtung des Ziehvektors zeigt, geben die restlichen Finger der Hand die Orientierung der Definitionskurve an.

Der Parameter *smooth* bestimmt, ob die Koordinaten der Definitionskurve linear (`0`) oder mit weichen Übergängen (`1`) miteinander verbunden werden.

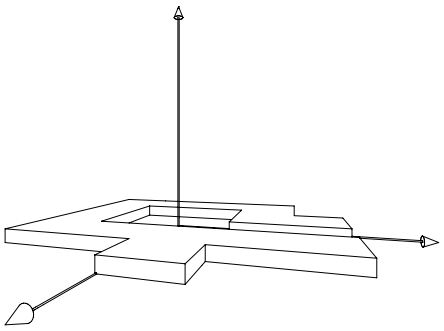
Der Parameter u gibt an, ob der letzte und der erste Punkt der Definitionskurve entsprechend dem Wert von *smooth* miteinander verbunden werden sollen (`1`) oder nicht (`0`).

Der Parameter $c0$ gibt an, ob der Körper Deckelflächen senkrecht zum Ziehvektor erhalten soll (`1`) oder nicht (`0`).

Der Parameter $c1$ gibt an, ob die Verbindung von letzter und erster Koordinate in Form einer entsprechenden Fläche erfolgen soll (`1`) oder nicht (`0`). Fallen beide Koordinaten zusammen, ist hier `0` anzugeben.

Das nachfolgende Beispiel erzeugt einen Ziehkörper entlang der lokalen y–Achse.

```
0.0 1.0 0.0 0.05 \
0.5 0.0 -0.5 -0.5 0.0 -0.5 -0.5 0.0 0.5 0.0 0.0 0.5 0.0 0.0 0.7 0.25 0.0 0.7 \
0.25 0.0 0.5 0.7 0.0 0.5 0.7 0.0 0.25 -0.25 0.0 0.25 -0.25 0.0 -0.25 0.25 0.0 -0.25 \
0.25 0.0 0.1 0.7 0.0 0.1 0.7 0.0 -0.15 0.5 0.0 -0.15 16 0 0 1 1 sweep
```



Die Funktion `sweepx` erzeugt einen Ziehkörper entlang der lokalen x–Achse. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$len\ z_0\ y_0\ \ldots z_{n-1}\ y_{n-1}\ n\ smooth\ u\ c0\ c1\ \texttt{sweepx}$$

Die Funktion `sweepy` erzeugt einen Ziehkörper entlang der lokalen y–Achse. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$len\ x_0\ z_0\ \ldots x_{n-1}\ z_{n-1}\ n\ smooth\ u\ c0\ c1\ \texttt{sweepy}$$

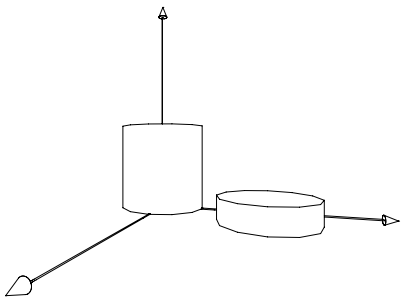
Die Funktion `sweepz` erzeugt einen Ziehkörper entlang der lokalen z–Achse. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$len\ x_0\ y_0\ \ldots x_{n-1}\ y_{n-1}\ n\ smooth\ u\ c0\ c1\ \texttt{sweepz}$$

3.6.12 Zylinder

Die Funktion `cyl` erzeugt einen homogenen Zylinder rotationssymmetrisch zur lokalen y–Achse. Der Zylinder beginnt im Ursprung des lokalen Koordinatensystems und verläuft entlang der positiven y–Achse dieses Koordinatensystems. Die Parametrisierung erfolgt durch die Angabe zweier positiver Zahlen für Länge und Radius als Argumente der Funktion `cyl`.

odb_ name	obj_ name	ex ist	offs x	offs y	offs z	rot x	rot y	rot z	ctor	mat	attrib	link
BAZ	o1		0.0	0.0	0.0	0.0	0.0	0.0	0.4 0.2 cyl			
	o2		0.5	0.0	0.2	0.0	0.0	0.0	0.1 0.2 cyl			



3.6.13 OFML–Referenz

Die Funktion `clsref` erzeugt eine Instanz einer OFML–Klasse. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$p_0\ \ldots p_{n-1}\ n\ \texttt{"}classname\texttt{"}\ \texttt{clsref}$$

Dabei sind p_0 bis p_{n-1} die klassenspezifischen Erzeugungs- bzw. Initialisierungsparameter. Diese werden in folgender Weise auf die Initialisierungsfunktion abgebildet:

$$classname\texttt{::initialize(}pFa\texttt{, }pNa\texttt{, }p_0\texttt{, ..., }p_{n-1}\texttt{)}$$

classname ist der optional voll qualifizierte Name der zu verwendenden OFML-Klasse. Falls er nicht voll qualifiziert ist, wird automatisch der Name der Package, in der sich die ODB befindet, vorangestellt.

3.6.14 ODB–Referenz

Die Funktion `odbref` erzeugt eine Instanz einer ODB–Definition. Die Spezifikation für die `ctor`–Spalte lautet wie folgt:

$$p_0 \ldots p_{n-1}\ n\ \texttt{"}\textit{odbname}\texttt{"}\ \texttt{odbref}$$

Dabei sind p_0 bis p_{n-1} die spezifischen Parameter, auf die in dem referenzierten ODB–Block über die ODB–Parameter `P0` bis `P`*n-1* zugegriffen werden kann.

odbname ist der optional voll qualifizierte Name der zu referenzierenden ODB–Definition. Falls *odbname* nicht voll qualifiziert ist, wird automatisch der Name der Package, in der sich die ODB befindet, vorangestellt.

3.7 Materialzuweisung

Die Spalte `mat` dient zur Zuweisung von Materialien. Falls dieses Feld nicht leer ist, spezifiziert es ein Material oder mehrere Materialien auf folgende Weise:

- Wird durch den Eintrag ein primitiver Typ referenziert, darf der Material–Ausdruck als Ergebnis nur einen Materialnamen liefern. Das entsprechende Material wird an die Objektmethode `setMaterial()` übergeben.
- Wird durch den Eintrag ein Klassentyp referenziert (`clsref`), so darf der Eintrag beliebig viele Materialnamen enthalten, die auch über Vektoren zusammengefaßt sein können. Diese werden zu einem Vektor der Basissprache OFML zusammengefaßt als Argument an die Objektmethode `setMaterials()` übergeben.

  Beispiel. Angenommen, der ODB–Parameter `Mat` sei auf `"foo"` gesetzt und die Materialspalte enthält den Ausdruck `"bar" $Mat "baz" 2]`. Dann wird für das entsprechende Objekt die Methode `setMaterials()` wie folgt aufgerufen:

  ```
  obj.setMaterials(["bar", ["foo", "baz"]])
  ```

- Wird durch den Eintrag ein ODB–Block referenziert (`odbref`), so darf der Eintrag beliebig viele Materialnamen enthalten, die auch über Vektoren zusammengefaßt sein können. Innerhalb des referenzierten ODB–Blocks sind die Materialen über die ODB–Parameter `M0` bis `M`*(n-1)* zugreifbar.

Materialnamen können optional voll qualifiziert angegeben werden. Sind sie nicht oder nur teilweise qualifiziert, wird dem Materialnamen implizit der Name des Paketes, in dem sich die ODB befindet, vorangestellt.

3.8 Attribute

Die Spalte `attrib` kann null oder mehr der folgenden Ausdrücke enthalten.

3.8.1 Selektierbarkeit

Die Selektierbarkeit des Objekts wird durch Ausdruck `0 sel` explizit verboten und mittels `1 sel` explizit erlaubt.

Erfolgt keine Angabe und ist das Objekt durch Referenzierung eines ODB–Blocks (mittels `odbref`) erzeugt worden, so ist es nicht selektierbar. Ist es durch Instantiierung einer OFML–Klasse (mittels `clsref`) erzeugt worden, so hängt die Selektierbarkeit des Objekts von der Implementierung der OFML–Klasse ab. In allen anderen Fällen handelt es sich bei dem Objekt um einen primitiven Typ, der standardmäßig nicht selektierbar ist und für den die Selektierbarkeit auch nicht erlaubt werden sollte.

3.8.2 Kollisionsverhalten

Der Ausdruck `0 cd` schließt das Objekt von der Kollisionsermittlung aus. Erfolgt keine Angabe oder wird der Ausdruck `1 cd` angegeben, wird das Objekt bei der Kollisionsermittlung berücksichtigt.

Dieser Ausdruck darf nur für Einträge verwendet werden, in denen ein ODB–Block (mittels `odbref`) oder eine OFML–Klasse (mittels `clsref`) referenziert werden.

Wird die Kollisionserkennung für ein Objekt ausgeschaltet, so gilt dies auch für alle direkten oder indirekten Nachfahren des Objekts.

3.8.3 Editierverhalten

Das Verhalten eines Objekts in Bezug auf Editoroperationen, z.B. in Einbeziehung des Zwischenspeichers (*clipboard*) wird durch einen Ausdruck der Form *value* `cut` definiert. Dabei sind folgende Werte für `value` zulässig:

`-1` Das Objekt darf generell nicht gelöscht werden.

`0` Das Objekt darf selber nicht gelöscht werden, kann aber im Rahmen einer übergeordneten Instanz gelöscht werden. Im Fall einer versuchten Ausschneideoperation (*cut, delete*) bezüglich des Objekts wird die Operation auf das im Zuge einer Aufwärtstraversierung erste Objekt angewendet, das ausschneidbar ist.

`1` Das Objekt kann gelöscht und in den Zwischenspeicher kopiert werden. Dies ist auch der Fall, wenn das Editierverhalten nicht im Rahmen eines ODB–Eintrages spezifiziert wurde.

`2` Das Objekt kann gelöscht werden, darf aber nicht in den Zwischenspeicher kopiert werden.

3.8.4 Translationsfreiheitsgrade

Mit der Funktion `trx` kann für einzelne Achsen des lokalen Koordinatensystems festgelegt werden, ob das Objekt in die Richtung der jeweiligen Achse verschoben werden kann oder nicht. Die Funktion erwartet ein einzelnes ganzzahliges Argument, dessen Wert sich aus der Addition der erlaubten Achsen ergibt, wobei x–, y– und z–Achse durch 1, 2 und 4 repräsentiert werden. Falls das Argument 0 ist, kann das Objekt nicht verschoben werden.

3.8.5 Rotationsfreiheitsgrade

Mit der Funktion `rtx` kann für einzelne Achsen des lokalen Koordinatensystems festgelegt werden, ob das Objekt um die jeweilige Achse gedreht werden kann oder nicht. Die Funktion erwartet ein einzelnes ganzzahliges Argument, dessen Wert sich aus der Addition der erlaubten Achsen ergibt, wobei x–, y– und z–Achse durch 1, 2 und 4 repräsentiert werden. Falls das Argument 0 ist, kann das Objekt nicht gedreht werden.

3.8.6 Properties

Mit der Funktion `prop` können nach der Erzeugung eines Objekts zusätzliche optionale Parameter gesetzt werden. Dies geschieht durch den Aufruf der Methode `setPropValue()` (siehe Schnittstelle `Property` in der OFML-Spezifikation[17]). Die Funktion erwartet zwei Argumente: Das erste spezifiziert den Key der Property und muss als OFML–Symbol (d.h. mit führendem `@`-Zeichen) angegeben werde. Das zweite Argument gibt den zu setzenden Wert an. Dieser muss dem Typ der Property entsprechen.

Die Funktion kann beliebig oft aufgerufen werden um mehrere Properties zu setzen.

3.8.7 Ebene

Mit der Funktion `layer` kann jedes Objekt einer Ebene (s. Abschnitt 6) zugeordnet werden. Die Funktion erwartet als Argument eine Zeichenkette, die den Namen der Ebene enthält.

3.9 Verweis

Die Spalte `link` wird in dieser Version nicht unterstützt[18].

[17] EasternGraphics GmbH: *OFML – Standardisiertes Datenbeschreibungsformat der Büromöbelindustrie.*
[18] Sie ist zukünftigen Erweiterungen vorbehalten und verweist als Schlüssel in andere Tabellen.

4 Anfügepunkte

4.1 Funktionsweise von Anfügepunkten

Wenn ein neues Objekt in eine Planung eingefügt wird und ein bereits existierendes Objekt selektiert ist, wird versucht, das neue Objekt relativ zu dem selektierten Objekt zu plazieren. Dazu muß das existierende Objekt einen Anfügepunkt besitzen, zu dem das einzufügende Objekt ein passendes Gegenstück hat[19].

Jeder Anfügepunkt hat einen eindeutigen symbolischen Namen, über den er identifiziert werden kann und über den die Zuordnung von gegenüberliegenden passenden Anfügepunkten erfolgt.

Die Position der Anfügepunkte wird relativ zu dem lokalen Koordinatensystem des jeweiligen Objekts angegeben. Ein neues Objekt wird immer so plaziert, daß sein Anfügepunkt an der gleichen Position wie der ihm entsprechende Anfügepunkt des existierenden Objektes ist. Zusätzlich kann das neue Objekt um einen bestimmten Winkel um die durch diesen Punkt verlaufende y–Achse gedreht werden.

Um das Einfügen verschiedener Arten von Objekten an verschiedenen Positionen zu ermöglichen, kann für jedes Objekt eine Liste von Anfügepunkten definiert werden. Beim Einfügen eines Objektes an ein existierendes Objekt wird, sofern der Nutzer keinen Anfügepunkt explizit ausgewählt hat, für das existierende Objekt diese Liste von ihrem Anfang bis zu ihrem Ende durchsucht, bis in ihr ein Anfügepunkt gefunden wird, für den das einzufügende Objekt einen gegenüberliegenden passenden Anfügepunkt definiert und an dem das einzufügende Objekt plaziert werden kann, ohne eine Kollision auszulösen.

4.2 Definition von Anfügepunkten

Die Anfügepunkte eines Objekts werden in der Tabelle *attpt*, die in Tabelle 8 beschrieben ist, definiert.

Feld-nummer	Feld-name	Beschreibung
1	`odb_name`	ODB–Name
2	`name`	symbolischer Name des Anfügepunktes
3	`select`	Auswahl des Anfügepunktes
4	`text_idx`	Index in Text–Tabelle
5	`x_pos`	lokale x–Position des Anfügepunktes
6	`y_pos`	lokale y–Position des Anfügepunktes
7	`z_pos`	lokale z–Position des Anfügepunktes
8	`direction`	Anfügerichtung
9	`rotation`	Rotation des einzufügendes Objektes
10	`mode`	Einfügemode (Kind/Nachbar)

Tabelle 8: Definition von Anfügepunkten

[19] Zusätzlich zu den Anfügepunkten oder statt ihnen kann die Klasse eines ODB–Planungselementes auch ihre eigene Logik zum Einfügen von Objekten implementieren.

Im folgenden werden die einzelnen Spalten dieser Tabelle näher beschrieben:

- `odb_name`
  Die Spalte `odb_name` enthält den Grundnamen des ODB–Namens, für den diese Anfügepunktdefinition gültig ist.

  Bei der Ermittlung der Anfügepunkte eines Objekts wird über den Prefix des ODB–Namens die ODB bestimmt, in der nach den Anfügepunkten gesucht werden soll, und der Grundname des ODB–Namens wird als Schlüssel für die Tabelle *attpt* verwendet.

  In der aktuellen Implementierung werden alle Einträge dieser Tabelle, die dem Schlüssel entsprechen, als potentielle Anfügepunkte in der Reihenfolge ihres Auftretens in der Tabelle geliefert, es sei denn, sie sind in der Spalte `select` explizit deselektiert worden.

- `name`
  Die Spalte `name` enthält den symbolischen Namen des Anfügepunktes. Dieser besteht aus einer beliebigen Folge von Buchstaben, Ziffern und Unterstrichen, wobei das erste Zeichen keine Ziffer sein darf und Groß- und Kleinschreibung berücksichtigt wird.

  Um sicherzustellen, daß die Namen von Anfügepunkten aus unterschiedlichen Paketen nicht kollidieren, sollten die Namen von Anfügepunkten einen möglichst eindeutigen Prefix haben, der sich z.B. aus dem Hersteller- und Serienkürzel des Paketes zusammensetzen kann. Eine Ausnahme für diese Regel stellt die Kombination von Elementen aus verschiedenen Serien **eines** Herstellers dar.

  Der Name eines Anfügepunktes sollte innerhalb aller Einträge der *attpt*–Tabelle mit gleichem ODB–Namen eindeutig sein.

- `select`
  Die Spalte `select` dient zur Auswahl eines Anfügepunktes. Mit ihr kann ein Anfügepunkt explizit freigeschaltet oder gesperrt werden. Der Anfügepunkt ist implizit freigeschaltet, wenn diese Spalte leer ist. Ist sie nicht leer, muß sie einen Ausdruck in Umgekehrter Polnischer Notation enthalten, dessen Ergebnis ein numerischer Wert ist. Ist das Ergebnis 0, ist der Anfügepunkt gesperrt, andernfalls ist er freigeschaltet.

- `text_idx`
  Diese Spalte enthält einen Index in eine Text–Tabelle, die einen den Anfügepunkt beschreibenden Text enthält, der in der Nutzeroberfläche in einem aus der Objekt- und Anfügepunkthierarchie bestehenden Baum verwendet werden kann[20].

- `x_pos`, `y_pos`, `z_pos`
  Diese Spalten enthalten die lokalen Koordinaten der Position des Anfügepunktes in Form eines Ausdrucks in Umgekehrter Polnischer Notation.

- `direction`
  In der Spalte `direction` wird die Richtung festgelegt, in der an diesen Anfügepunkt angefügt werden soll. Die vordefinierten Richtungen sind `R` (rechts), `L` (links), `B` (hinten), `F` (vorne) und `T` (oben). Es können darüber hinaus beliebige andere Richtungen definiert werden.

  Wenn die Spalte leer ist, wird keine konkrete Anfügerichtung vorgegeben. Dies ist für die Ermittlung der gegenüberliegenden Anfügepunkte über die Tabelle *oppattpt*, die in

[20] In der derzeitigen Implementierung ist die Spalte ungenutzt und sollte 0 enthalten.

Abschnitt 4.3 beschrieben ist, von Bedeutung, da in diesem Fall alle in dieser Tabelle angegebenen und vom Namen her identischen Anfügepunkte ohne Berücksichtigung einer eventuell angegebenen Richtung als gegenüberliegende Anfügepunkte betrachtet werden.

- `rotation`
  In der Spalte `rotation` kann eine Rotation des einzufügenden Objektes um die durch den Anfügepunkt verlaufende y–Achse spezifiziert werden. Die Angabe erfolgt als Ausdruck in Umgekehrter Polnischer Notation mathematisch positiv (entgegen dem Uhrzeigersinn) im Gradmaß.

- `mode`
  Mit der Spalte `mode` wird festgelegt, ob das einzufügende Objekt als Kind oder als Nachbar des existierenden Objektes einzufügen ist. Um es als Kind einzufügen, muß diese Spalte ein `C` enthalten, andernfalls ein `S`.

4.3 Definition von gegenüberliegenden Anfügepunkten

Mit der Tabelle *oppattpt*, die in Tabelle 9 beschrieben ist, wird festgelegt, welche Anfügepunkte von verschiedenen Objekten zueinander passen. Diese Festlegung erfolgt aus der Sicht des einzufügenden Objektes, welches für mögliche Anfügepunkte anderer Objekte unter Berücksichtigung von deren Anfügerichtung eine Liste von eigenen passenden Anfügepunkten liefert[21].

Feld-nummer	Feld-name	Beschreibung
1	`odb_name`	ODB–Name
2	`select`	Auswahl des gegenüberliegenden Anfügepunktes
3	`opposite`	Name des gegenüberliegenden Anfügepunktes
4	`direction`	Richtung des gegenüberliegenden Anfügepunktes
5	`att_points`	Liste der eigenen passenden Anfügepunkte

Tabelle 9: Gegenüberliegende Anfügepunkte

Im folgenden werden die in Tabelle 9 aufgelisteten Spalten näher beschschrieben:

- `odb_name`
  Die Spalte `odb_name` enthält den Grundnamen des ODB–Namens des einzufügenden Objektes, dessen Anfügepunkte in der Spalte `att_points` aufgelistet sind.

- `select`
  Die Spalte `select` dient zur Auswahl des in der Spalte `opposite` angegebenen gegenüberliegenden Anfügepunktes. Mit ihr kann dieser explizit freigeschaltet oder gesperrt werden. Er ist implizit freigeschaltet, wenn die Spalte leer ist. Ist sie nicht leer, muß sie einen Ausdruck in Umgekehrter Polnischer Notation enthalten, dessen Ergebnis ein numerischer Wert ist. Ist das Ergebnis 0, ist der gegenüberliegende Anfügepunkt gesperrt, andernfalls ist er freigegeben.

[21] Das einzufügende Objekt wird nach einer Liste seiner eigenen Anfügepunkte gefragt, die als Gegenstück zu dem gerade betrachteten Anfügepunkt des existierenden Objektes in Frage kommen.

- `opposite`
  Die Spalte `opposite` enthält den Namen des gegenüberliegenden Anfügepunktes. Sie dient neben dem ODB–Namen in der Spalte `odb_name` und der Richtung in der Spalte `direction` als Schlüssel beim Zugriff auf die *oppattpt*–Tabelle.

- `direction`
  Die Spalte `direction` enthält die Richtung des gegenüberliegenden Anfügepunktes. Sie dient neben dem ODB–Namen in der Spalte `odb_name` und dem Namen des gegenüberliegenden Anfügepunktes in der Spalte `opposite` als Schlüssel beim Zugriff auf die *oppattpt*–Tabelle.

  Der gegenüberliegende Anfügepunkt wird nur dann in Betracht gezogen, wenn entweder für ihn keine Anfügerichtung spezifiziert worden ist, oder das Feld `direction` in dieser Tabelle leer ist, oder die spezifizierte Anfügerichtung mit der angegebenen Richtung in dieser Tabelle identisch ist.

- `att_points`
  Die Spalte `att_points` enthält eine durch Leerzeichen voneinander getrennte Liste von Anfügepunkten des einzufügenden Objektes mit dem in der Spalte `odb_name` angegebenen ODB–Namen, die zu dem gegenüberliegenden in der Spalte `opposite` angegebenen Anfügepunkt passen.

4.4 Standardanfügepunkte

Neben den nutzerdefinierten Anfügepunkten existieren noch eine Menge von 18 Standardanfügepunkten, die an den acht Ecken, in der Mitte der oberen und unteren Kanten und in der Mitte der Deck- und Bodenfläche des Begrenzungsvolumens eines OFML–Objektes liegen. Die Reihenfolge und Anfügerichtung dieser Anfügepunkte hängt von der aktuellen Planungsrichtung ab. Die Namen dieser Standardanfügepunkte sind in Tabelle 10 beschrieben.

Name		Position
Unten	Oben	
`LBF`	`LTF`	linke vordere Ecke
`CBF`	`CTF`	Mitte der Vorderkante
`RBF`	`RTF`	rechte vordere Ecke
`LBC`	`LTC`	Mitte der linken Kante
`CBC`	`CTC`	Mitte der Boden- bzw. Deckfläche
`RBC`	`RTC`	Mitte der rechten Kante
`LBB`	`LTB`	linke hintere Ecke
`CBB`	`CTB`	Mitte der Hinterkante
`RBB`	`RTB`	rechte hintere Ecke

Tabelle 10: Namen der Standardeinfügepunkte

Der erste Buchstabe des Namens eines Standardanfügepunktes bestimmt die Position des Anfügepunktes in x–Richtung (links: `L`, mitte: `C`, rechts: `R`). Der zweite Buchstabe bestimmt seine Lage in y–Richtung (unten: `B`, oben: `T`). Der dritte Buchstabe schließlich bestimmt seine Lage in z–Richtung (vorne: `F`, mitte: `C`, hinten: `B`).

Mit Hilfe der Tabelle *stdattpt*, die in Tabelle 11 beschrieben ist, ist es möglich, die Verwendung der Standardanfügepunkte für ODB–Objekte an Hand ihres ODB–Names zu steuern. Im einzelnen kann festgelegt werden, ob für ein Objekt mit einem gegebenen ODB–Namen überhaupt Standardanfügepunkte verwendet werden sollen, und wenn ja, ob diese vor oder nach den nutzerdefinierten Anfügepunkten berücksichtigt werden sollen. Des weiteren ist es möglich, nur eine Untermenge der Standardanfügepunkte zu berücksichtigen.

Feld-nummer	Feld-name	Beschreibung
1	`odb_name`	ODB–Name
2	`has_stdattpts`	Generelle Verwendung von Standardanfügepunkten
3	`prep_stdattpts`	Position der Standardanfügepunkte
4	`stdattpts`	Auswahl einer Untermenge der Standardanfügepunkte

Tabelle 11: Standardanfügepunkte

Im folgenden werden die einzelnen Spalten der *stdattpt*–Tabelle näher beschrieben.

- `odb_name`
  Die Spalte `odb_name` enthält den Grundnamen des ODB–Names des jeweiligen Objektes.
- `has_stdattpts`
  Die Spalte `has_stdattpts` steuert, ob Objekte mit dem entsprechenden ODB–Namen über Standardanfügepunkte verfügen oder nicht. Die Spalte muß einen ganzzahligen vorzeichenlosen Wert enthalten. Ist dieser 0, werden keine Standardanfügepunkte verwendet, unabhängig vom Inhalt der anderen Spalten dieser Tabelle. Andernfalls werden die Standardanfügepunkte, wie in den folgenden Spalten angegeben, verwendet.
- `prep_stdattpts`
  Mit der Spalte `prep_stdattpts` wird gesteuert, ob die Standardanfügepunkte vor oder nach eventuellen nutzerdefinierten Anfügepunkten betrachtet werden. Sie muß einen ganzzahligen vorzeichenlosen Wert enthalten. Ist dieser 0, werden sie nach dem nutzerdefinierten betrachtet[22], andernfalls vor diesen.
- `stdattpts`
  Die Spalte `stdattpts` ist entweder leer oder enthält eine Liste von durch Leerzeichen getrennten Namen von Standardanfügepunkten. Im ersten Fall werden alle Standardanfügepunkte berücksichtig, im zweiten Fall nur die angebenen.

Wenn für einen ODB–Namen kein Eintrag in der *stdattpt*–Tabelle ist, so werden für Objekte mit diesem ODB–Namen alle Standardanfügepunkte nach eventuellen nutzerdefinierten Anfügepunkten berücksichtigt.

[22] Dies ist vermutlich der Normalfall.

5 Funktionen

Bei Funktionen wird zwischen eingebauten und nutzerdefinierten Funktionen unterschieden.

Grundsätzlich werden die Funktionsargumente vor dem Aufruf der Funktion notiert. Um zum Beispiel die Quadratwurzel aus 2.0 zu berechnen, ist „`2.0 sqrt`“ zu schreiben.

Im allgemeinen ist zu sagen, daß die Möglichkeiten, die die ODB zur Abarbeitung und Definition von Funktionen bietet, zumindest bei der Erzeugung von 2D–Geometrien kaum ausgenutzt werden dürften. Im Normalfall werden sich durch die 2D–ODB benutzte Ausdrücke und Funktionen auf die Abarbeitung von arithmetischen Standardoperatoren `+`, `-`, `*` und `/` beschränken.

5.1 Eingebaute Funktionen

In den Interpreter für die in der ODB verwendeten Ausdrücke sind neben den in den Abschnitten 2 und 3 beschriebenen Funktionen zur Objekterzeugung und zum Setzen von Attributen weitere, insbesondere mathematische, Funktionen eingebaut.

Der Rückgabewert einiger dieser Funktionen ist eine häufig verwendete Konstante. So gibt zum Beispiel die Funktion `M_PI` den Wert von π zurück. Bei anderen Funktionen ist der Rückgabewert abhängig von einem oder mehrerer Argumente, die die Funktion erwartet. Ein Beispiel dafür ist die Funktion `sin`, die den Sinus ihres Arguments im Bogenmaß berechnet.

Die Tabelle 12 enthält eine Zusammenfassung aller eingebauten Funktionen, die Konstanten zurückgeben. Die eingebauten mathematischen Funktionen sind in Tabelle 13 aufgeführt. In Tabelle 14 werden die eingebauten Funktionen zur Manipulation des Stacks beschrieben. Die Tabelle 15 dokumentiert die Verwendung zweier Funktionen, die besonders für die 2D–ODB interessant sind.

Name	Zurückgegebener Wert	Name	Zurückgegebener Wert
`M_1_PI`	$1/\pi$	`M_2_PI`	$2/\pi$
`M_2_SQRTPI`	$2/\sqrt{\pi}$	`M_2PI`	2π
`M_E`	e	`M_LN10`	$\ln 10 = \log_e 10$
`M_LN2`	$\ln 2 = \log_e 2$	`M_LOG10E`	$\lg e = \log_{10} e$
`M_LOG2E`	$1/\ln 2 = \log_2 e$	`M_PI`	π
`M_PI_2`	$\pi/2$	`M_PI_4`	$\pi/4$
`M_SQRT1_2`	$1/\sqrt{2}$	`M_SQRT2`	$\sqrt{2}$

Tabelle 12: Eingebaute Konstanten

Argumente	Name	Ergebnis	Beschreibung
x	`acos`	y	$y = \arccos x$
x	`asin`	y	$y = \arcsin x$
x	`atan`	y	$y = \arctan x$
$x\ y$	`atan2`	z	$z = \arctan(y/x)$ Die Vorzeichen von x und y werden benutzt, um den Quadranten des Ergebnisses zu bestimmen.
x	`ceil`	y	Berechnet den kleinsten ganzzahligen Wert y, der größer oder gleich x ist.
x	`cos`	y	$y = \cos x$
x	`cosh`	y	$y = \cosh x$
x	`exp`	y	$y = e^x$
x	`fabs`	y	$y = \|x\|$
x	`floor`	y	Berechnet den größten ganzzahligen Wert y, der kleiner oder gleich x ist.
$x\ y$	`fmod`	z	Berechnet den Gleitkommarest von x/y.
x	`log`	y	$y = \ln x$
x	`log10`	y	$y = \lg 10$
x	`modf`	$i\ f$	Teilt das Argument x in den ganzzahligen Anteil i und den gebrochenen Anteil f auf, so daß beide das gleiche Vorzeichen wie x haben.
x	`neg`	y	$y = -x$
$x\ y$	`pow`	z	$z = x^y$
x	`sin`	y	$y = \sin x$
x	`sinh`	y	$y = \sinh x$
x	`sqrt`	y	$y = \sqrt{x}$
x	`tan`	y	$y = \tan x$
x	`tanh`	y	$y = \tanh x$

Tabelle 13: Eingebaute mathematische Funktionen

Argumente	Name	Ergebnis	Beschreibung
n	`argc`		Die Funktion `argc` muß unmittelbar am Anfang einer nutzerdefinierten Funktion aufgerufen werden. Der Parameter n ist die Anzahl der Argumente, die diese nutzerdefinierte Funktion erwartet. Sie entfernt diese Anzahl Werte vom Stack des Aufrufers und macht sie für den Argumentzugriff mittels $\$a$ verfügbar.
x	`dup`	$x\ x$	Die Funktion `dup` dupliziert das oberste Element auf dem Stack.
$x\ y$	`dup2`	$x\ y\ x$	Die Funktion `dup2` dupliziert das zweitoberste Element auf dem Stack.
$s_i \ldots s_2\ s_1\ x$	`dupx`	$s_i \ldots s_2\ s_1\ s_x$	Die Funktion `dupx` dupliziert das x–te Objekt von der Spitze des Stacks.
x	`pop`		Die Funktion `pop` entfernt das oberste Element vom Stack.
$x\ y$	`swap`	$y\ x$	Die Funktion `swap` vertauscht die beiden obersten Elemente auf dem Stack.
$s_x \ldots s_2\ s_1\ x$	`swapx`	$s_1 \ldots s_2\ s_x$	Die Funktion `swapx` vertauscht das oberste Element des Stacks mit dem x–ten Element von der Spitze des Stacks.

Tabelle 14: Funktionen zur Manipulation des Stacks

Argumente	Name	Ergebnis	Beschreibung
x	`utos`	s	Die Funktion `utos` wandelt entsprechend den in der Benutzungsoberfläche vorgenommenen Einstellungen zur Formatierung von Einheiten einen Gleitkommawert x in eine Zeichenkette s um.
x	`atos`	s	Die Funktion `atos` wandelt entsprechend den in der Benutzungsoberfläche vorgenommen Einstellungen zur Formatierung von Winkeln einen Gleitkommawert a in eine Zeichenkette s um.

Tabelle 15: Funktionen für 2D–ODB

5.2 Nutzerdefinierte Funktionen

Nutzerdefinierte Funktionen werden in der Funktionstabelle angelegt. Diese Funktionstabelle hat den in Tabelle 16 dargestellten Aufbau.

Feld-nummer	Feld-name	Beschreibung
1	`name`	Name der Funktion
2	`body`	Körper der Funktion

Tabelle 16: Funktionstabelle

In der ersten Spalte wird der Name der Funktion angegeben. Der Funktionsname kann aus einer beliebig langen Folge von Buchstaben[23], Ziffern und Unterstrichen bestehen, wobei das erste Zeichen ein Buchstabe oder Unterstrich[24] sein muß.

Die zweite Spalte enthält den Körper der Funktion in Form eines Ausdrucks in Umgekehrter Polnischer Notation.

5.2.1 Funktionsargumente

Eine nutzerdefinierte Funktion kann beliebig viele Argumente haben, einschließlich keine.

Für Funktionen ohne Argumente sind keine besonderen Vorkehrungen zu treffen.

Für Funktionen mit Argumenten muß am Anfang des Funktionskörpers die Argumentanzahl, gefolgt von dem Aufruf der eingebauten Funktion `argc`, stehen. Dadurch wird die angegebene Anzahl an Argumenten von dem lokalen Stack des Ausdrucks, der die Funktion aufgerufen hat, entfernt und für die Dauer der Abarbeitung der Funktion zwischengespeichert. Die Funktion kann dann an beliebiger Stelle auf die Argumente mittels $\$n$ zugreifen, wobei n die Nummer des Arguments ist. Die Nummerierung der Argumente beginnt bei 0.

5.2.2 Rückgabewerte

Eine nutzerdefinierte Funktion kann eine beliebige Anzahl an Rückgabewerten haben, einschließlich keine.

Um einen oder mehrere Werte zurückzugeben, werden diese am Ende der Abarbeitung des Funktionskörpers einfach auf dem Stack gelassen. Nach der Rückkehr der Funktion stehen diese Werte in der gleichen Reihenfolge an der Spitze des lokalen Stacks des Ausdrucks, der die Funktion aufgerufen hat. Es liegt dann in der Verantwortung des aufrufenden Ausdrucks, die Rückgabewerte vom Stack zu entfernen.

[23] Es sind nur die Buchstaben `A` bis `Z` und `a` bis `z` erlaubt. Umlaute dürfen folglich nicht verwendet werden.
[24] Von der Verwendung eines Unterstrichs am Anfang eines Funktionsnamens wird abgeraten, da derartige Namen für interne Zwecke reserviert sind.

5.2.3 Beispiel

Das folgende Beispiel zeigt die Funktion `DIST`, die die Entfernung zwischen zwei durch ihre x– und y–Koordinaten definierten Punkte bestimmt. Die Argumente werden durch den Aufrufer in der Reihenfolge x_0, y_0, x_1, y_1 auf den Stack gelegt. Wenn die Funktion zurückkehrt, sind die Argumente vom Stack entfernt und durch das Ergebnis der Funktion ersetzt.

`name`	`body`
`DIST`	`4 argc $2 $0 - dup * $3 $1 - dup * + sqrt`

In der folgenden Tabelle wird anhand des lokalen Stacks der Funktion die Abarbeitung des Funktionskörpers verdeutlicht.

Stack	Token	Operation
	`4`	`4` ⇒ Stack
`4`	`argc`	Übernehmen der 4 Argumentwerte vom Stack des aufrufenden Ausdrucks
	`$2`	x_1 ⇒ Stack
x_1	`$0`	x_0 ⇒ Stack
x_1 x_0	`-`	$dx = x_1 - x_0$; x_1 und x_0 vom Stack entfernen und durch dx ersetzen
dx	`dup`	dx ⇒ Stack
dx dx	`*`	$dx^2 = dx \times dx$; beide dx vom Stack entfernen und durch dx^2 ersetzen
dx^2	`$3`	y_1 ⇒ Stack
dx^2 y_1	`$1`	y_0 ⇒ Stack
dx^2 y_1 y_0	`-`	$dy = y_1 - y_0$; y_1 und y_0 vom Stack entfernen und durch dy ersetzen
dx^2 dy	`dup`	dy ⇒ Stack
dx^2 dy dy	`*`	$dy^2 = dy \times dy$; beide dy vom Stack entfernen und durch dy^2 ersetzen
dx^2 dy^2	`+`	$sq = dx^2 + dy^2$; sowohl dx^2 wie auch dy^2 vom Stack entfernen und durch sq ersetzen
sq	`sqrt`	$dist = \sqrt{sq}$; sq vom Stack entfernen und durch $dist$ ersetzen
$dist$		Rückgabewert

6 Ebenen

6.1 Funktionsweise von Ebenen

Über die jeweilige `layer`–Funktion (s. Abschnitt 2.8.7 bzw. 3.8.7) können Objekte einer Ebene zugeordnet werden. Damit können mehreren Objekten, unabhängig von ihrer Position in der Objekt-Hierarchie, gleichzeitig Eigenschaften wie Sichtbarkeit, Farbe usw. zugewiesen werden.

6.2 Definition von Ebenen

Die Definition der Eigenschaften von 2D-Ebenen erfolgt ausschließlich über die Applikation.

Die 3D-Ebenen werden in der Tabelle *layer* definiert. Die Definition von Ebenen ist optional. Für nicht definierte Ebenen werden voreingestellte Werte verwendet. Die Werte in dieser Tabelle sind ihrerseits wieder Vorgabewerte, die durch die Applikation überschrieben werden können.

Feld-nummer	Feld-name	Beschreibung
1	`layer_name`	Name der Ebene
2	`attributes`	Eigenschaften

Tabelle 17: Definition von 3D-Ebenen

Im folgenden werden die einzelnen Spalten dieser Tabelle näher beschrieben:

- `layer_name`
  Die Spalte `layer_name` enthält den Namen der Ebene. Dabei sind folgende Zeichen zulässig: Ziffern, Buchstaben, `_` (Unterstrich), `-` (Bindestrich) und `$` (Dollar-Zeichen).

  Layer-Namen sollten der OLAYERS-Spezifikation entsprechen[25].

- `attributes`
  Über vordefinierte Funktionen wird in dieser Spalte das Setzen von Ebenen-Eigenschaften ermöglicht. Der Aufruf der Funktionen erfolgt in Umgekehrter Polnischer Notation, d.h., die Argumente stehen vor dem Namen der Funktion.

  Zur Zeit ist nur die Funktion `visible` definiert. Hat das Argument dieser Funktion den ganzzahligen Wert 0, sind die Objekte auf dem Layer unsichtbar. Dies betrifft Rendering (Echtzeit, Fotorealismus), Druckausgabe und Grafikexport.

[25] Verband Büro-, Sitz- und Objektmöbel e.V.: *OLAYERS – OFML-kompatible Layer.*

A Konfigurationsdatei für `ebmkdb`

```
name    ex_odb

comment
Dies ist die Konfigurationsdatei, die benoetigt wird, um aus den
*.csv Dateien mittels ebmkdb eine EBase-Datenbank zu erstellen.
Bis auf den Namen in der ersten Spalte, der von dem jeweiligen
Paketnamen abgeleitet werden sollte, und diesem Kommentar sollte
sie unveraendert uebernommen werden.
end comment

table   odb2d                   odb2d.csv       variable_width
fields  10
field   1       odb_name        vstring            delim ;  trim  hidx
field   2       level           uint16             delim ;
field   3       visible         string          10 delim ;  trim
field   4       x_offs          string          10 delim ;  trim
field   5       y_offs          string          10 delim ;  trim
field   6       rot             string          10 delim ;  trim
field   7       x_scale         string          10 delim ;  trim
field   8       y_scale         string          10 delim ;  trim
field   9       ctor            vstring            delim ;  trim
field   10      attrib          vstring            delim ;  trim

table   odb3d                   odb3d.csv       variable_width
fields  13
field   1       odb_name        vstring            delim ;  trim  hidx
field   2       obj_name        vstring            delim ;  trim
field   3       visible         string          10 delim ;  trim
field   4       x_offs          string          10 delim ;  trim
field   5       y_offs          string          10 delim ;  trim
field   6       z_offs          string          10 delim ;  trim
field   7       x_rot           string          10 delim ;  trim
field   8       y_rot           string          10 delim ;  trim
field   9       z_rot           string          10 delim ;  trim
field   10      ctor            vstring            delim ;  trim
field   11      mat             vstring            delim ;  trim
field   12      attrib          vstring            delim ;  trim
field   13      link            string          10 delim ;  trim

table   funcs                   funcs.csv       variable_width
fields  2
field   1       name            string          10 delim ;  trim  hidx
field   2       body            vstring            delim ;  trim

table   attpt                   attpt.csv       variable_width
fields  10
field   1       odb_name        vstring            delim ;  trim  hidx link
```

```
field   2       name            string          32 delim ;  trim
field   3       select          string          10 delim ;  trim
field   4       text_idx        uint32             delim ;  trim
field   5       x_pos           string          10 delim ;  trim
field   6       y_pos           string          10 delim ;  trim
field   7       z_pos           string          10 delim ;  trim
field   8       direction       string           2 delim ;  trim
field   9       rotation        string          10 delim ;  trim
field   10      mode            string           1 delim ;  trim

table   stdattpt                stdattpt.csv    variable_width
fields  4
field   1       odb_name        vstring            delim ;  trim  hidx
field   2       has_stdattpts   uint8              delim ;  trim
field   3       prep_stdattpts  uint8              delim ;  trim
field   4       stdattpts       vstring            delim ;  trim

table   oppattpt                oppattpt.csv    variable_width
fields  5
field   1       odb_name        vstring            delim ;  trim  hidx link
field   2       select          vstring            delim ;  trim
field   3       opposite        string          32 delim ;  trim
field   4       direction       string           2 delim ;  trim
field   5       att_points      vstring            delim ;  trim

table   layer                   layer.csv       variable_width
fields  2
field   1       layer_name      vstring            delim ;  trim  hidx
field   2       attributes      vstring            delim ;  trim
```