ProgressXML

Version 1.3 / Revision 2024-01-28

An innovative data transmission concept for precasters and reinforcers

Edited by:

PROGRESS GROUP

In cooperation with:

VeriCon Ingenieurs BV & VeriCon Solutions BV, Veldhoven (NL)
Precast Software Engineering GmbH, Salzburg (A)
Gesys GmbH & Co. KG, Kißlegg (D)
IDAT GmbH, Darmstadt (D)
Softbauware GmbH, Langen (D)

Index:

1 GENERAL 6
1.1 Scope of application *6*
1.2 Default values *6*
1.3 Character encoding *7*
1.4 Versioning *7*
1.5 Compatibility with UNICAM *8*
1.6 PXML Delegate Files and PXML Include Files *9*
2 STRUCTURE OVERVIEW 14
3 DETAIL SPECIFICATIONS 21
3.1 Global ID *21*
3.1.1 Unambiguity of the GlobalID 21
3.1.2 Special case: GlobalID of DocInfo Table 21
3.1.3 Automatic generation of GlobalIDs 21
3.1.4 Late generation of GlobalIDs 22
3.2 DocInfo *22*
3.2.1 GlobalID 22
3.2.2 Document Version 22
3.2.3 Comment 22
3.2.4 ConvertConventions 22
3.2.5 Mode 23
3.3 Order *24*
3.3.1 Order Information 24
3.3.2 Import Source Information 25
3.3.3 ApplicationName, ApplicationGUID, ApplicationVersion 25
3.4 OrderInfo, OrderInfoVal *26*
3.5 Product (Element) *26*
3.5.1 ElementNo 27
3.5.2 ProductType 27
3.5.3 PieceCount 28
3.5.4 Data transfer for double walls: TurnWidth, TotalThickness, DoubleWallsGap 28
3.5.5 Comment 29
3.5.6 RotationPosition 29
3.5.7 Stacking Information 29
3.5.8 Project Coordinates 30
3.5.9 Supplementary Product Information 31
3.6 ElementInfo *31*
3.6.1 Fields of ElementInfo entries 31
3.6.2 Predefined ElementInfo types 32
3.6.3 ElemInfoVal 35
3.7 Slab (Element Part) *35*
3.7.1 PartType 35
3.7.2 Geometric Slab Placement (X/Y/Z, RotX/Y/Z) 35
3.7.3 Slab Production Directives (ProdX/Y/Z, ProdRotX/Y/Z) 36
3.7.4 Geometric Placement and Production Directives for Double Walls 36
3.7.5 Various Slab Information 37
3.7.6 Multi-Layer Elements 37
3.7.7 Legacy Slab Fields 37
3.7.8 Simplified geometry representation 38
3.8 Outline *39*
3.8.1 Geometric Outline Placement (X, Y, Z, RotX, RotY, RotZ) 39
3.8.2 Height 39
3.8.3 Name 39
3.8.4 GenericInfo 39
3.8.5 MountingInstruction (only for Mountpart) 39
3.8.6 MountPartType, MountPartArticle (only for Mountpart) 40
3.8.7 MountPartProperties (only for Mountpart) 40
3.8.8 Concrete Properties (only for lots) 40
3.8.9 Layer 40
3.8.10 ObjectID 41
3.8.11 Shape, SVertex 41
3.9 Steel *46*
3.9.1 Geometric Steel Placement (X, Y, Z, RotX, RotY, RotZ) 47
3.9.2 ToTurn (only for steel mesh) 47
3.9.3 StopOnTurningSide (only for steel mesh) 47

3.9.4 Name 47
3.9.5 MeshType 47
3.9.6 WeldingDensity (only for steel mesh) 48
3.9.7 BorderStrength 48
3.9.8 Generic Steel Info 48
3.9.9 Steel Production Directices (ProdX/Y/Z, ProdRotX/Y/Z) 48
3.9.10 Layer 49
3.9.11 ObjectID 50
3.10 Bar 50
3.10.1 ShapeMode 50
3.10.1.1 ShapeMode "realistic" 50
3.10.1.2 ShapeMode "schematic" 50
3.10.1.3 ShapeMode "polygonal" 51
3.10.1.4 Automatic determination of ShapeMode and mixed representation 52
3.10.2 ReinforcementType (reinforcement layers) 52
3.10.2.1 Definition of reinforcement type 52
3.10.2.2 How to set the reinforcement types 53
3.10.2.3 Upper reinforcement layers 53
3.10.3 SteelQuality 54
3.10.4 PieceCount, Diameter, X, Y, Z 54
3.10.5 RotZ 54
3.10.6 ArticleNo 55
3.10.7 NoAutoProd 55
3.10.8 ExtIronWeight 55
3.10.9 Bin 55
3.10.10 Pos 55
3.10.11 Note 55
3.10.12 Machine 55
3.10.13 BendingDevice 55
3.10.14 Spacer 56
3.10.14.1 Type 56
3.10.14.2 Position 56
3.10.15 WeldingPoint 56
3.10.15.1 WeldingOutput 56
3.10.15.2 Position 56
3.10.15.3 WeldingPointType, WeldingPrgNo 56
3.10.15.4 GroupID 57
3.10.16 Segment 58
3.10.16.1 Segment-Orientation (RotX, BendY) 58
3.10.16.2 Segment-Length (L) 58
3.10.16.3 Bending-Radius (R) 58
3.10.16.4 External dimensions 60
3.10.16.5 External length of segment 61
3.10.16.6 Height and width of segment 61
3.10.16.7 Rules for computing external dimensions 61
3.10.16.8 Conventional external dimensions 62
3.10.16.9 General computations for the bending radius 62
3.10.16.10 Arcs and spirals in traditional "spiral form" 62
3.10.16.11 Arcs in ordinary PXML form 63
3.10.16.12 General computations for coordinate rotation 66
3.10.16.13 Conversion from or to UNICAM 68
3.10.16.14 Conversion from or to BVBS-BF2D 70
3.10.16.15 Conversion from or to BVBS-BF3D 70
3.10.16.16 Conversion from or to 3D-BF2D 71
3.10.17 Canonical Bar representation 72
3.11 Girder 74
3.11.1 PieceCount, X, Y, Z, GirderName, Length, AngleToX 74
3.11.2 NoAutoProd 74
3.11.3 Height, TopExcess, BottomExcess 74
3.11.4 Weight, TopFlangeDiameter, BottomFlangeDiameter 74
3.11.5 GirderType 74
3.11.6 MountingType 75
3.11.7 ArticleNo 75
3.11.8 Machine 75
3.11.9 Period, PeriodOffset 75
3.11.10 Width 75
3.11.11 AnchorBar 75
3.11.12 GirderExt 75
3.11.12.1 Type = splicePos 76

3.11.12.2 Type = FixingPos 76
3.11.12.3 Type = GirderGripPos 76
3.11.12.4 Type = MeshGripPos 76
3.11.12.5 Type = SupportPos 76
3.11.12.6 Export to UNICAM 76
3.11.13 Section 76
3.11.13.1 Fields in the Section table 76
3.11.13.2 Assignment of the Section data 77
Calculations for girder shapes 77
3.12 Alloc 79
3.12.1 GuidingBar 79
3.12.2 Determination of direction excluding GuidingBar 79
3.12.3 Region 80
3.13 SteelExt 81
3.14 Feedback 82
3.14.1 Production Test Service (PTS) 82
3.14.2 Machine Return 83
3.14.3 Fields of the Feedback Block 83
3.14.3.1 Feedback attributes 83
3.14.3.2 Feedback fields 83
3.14.4 FbVal 84
3.14.4.1 Wire information 88
3.14.4.2 Welding point information 88
3.14.4.3 Spacer information 88
3.14.5 Examples of PTS messages 88
3.14.6 Examples of Machine Return messages 90
3.14.7 Examples of FbVal entries 91
3.14.7.1 Example Bar 91
3.14.7.2 Example Steel 91
3.14.7.3 Example Slab 91
3.14.7.4 Example Girder 92
3.14.8 Types of communication of PTS 92
3.14.8.1 Communication on file basis 92
3.14.8.2 Communication via web services 93
3.14.9 Types of communication of Machine Return 94
3.14.9.1 Communication on file basis 94
3.14.9.2 Communication via web services 94
3.14.10 Parallel PTS servers and series of PTS servers 94
3.14.11 Filter, classify and sort PTS feedback messages 95
4 PROPOSALS FOR FUTURE EXTENSIONS 96
5 VERSION HISTORY 97

1 General

1.1 Scope of application

The data format of ***progressXML*** (hereinafter also referred to as **PXML**) is a data format based on XML for the generation of data and production control and scheduling at precasting plants.

In particular, there are two different areas of application:

- interface between systems of different manufacturers →*Standard Tags,* and
- internal (proprietary) storage of data of CAD/CAM systems →*Internal Tags.*

This document has been devised to describe the **Standard Tags**. These are those fields of the PXML format that are bindingly defined and that may be used for data interchange between different systems. It is *not* necessary to use all the Standard Tags, but one should try to exclusively use these Standard Tags for data interchange only.

In addition to these Standard Tags, any software producer may define or use any kind of proprietary **Internal Tags** for its internal data representation. These should simply be ignored whenever data are interchanged between different systems.

It is recommended to always put a prefix `I_` in front of any internal tags; this will help avoid name clashes with standard tags that may be introduced at a later time (standard tags never open with `I_`)[1].

From the above remarks, we derive the following requirements on cross-system PXML import modules:

a) Exclusively Standard Tags will be read.
b) The absence of Standard Tags must be tolerated (default values are accepted, see Section 1.2).
c) The PXML file may have any number of additional tags (viz. internal tags) that will be simply ignored for import.

1.2 Default values

Default values will be attained where a tag is missing. These default values are consistently specified for each data type:

- **string**: ""
- **double**: 0
- **bool**: false

For other than the above data types, default values should never be attained. This is specifically true for integer values: as these may often include IDs that may also be 0, there is no default value rule in place for integers.

[1] It is a variant of this convention to use the prefixes of **I_P_** or **I_V_**: here, the former are used for *persistent* internal tags, viz. those tags that are also written on file, and the latter are use for *volatile* internal tags that are *not* written on file and that would generally contain redundant information only (cache fields).

1.3 Character encoding

It is recommended to use UTF-8 encoding in PXML files. This will help avoid many problems or complications respectively.

However, as this is an XML file, selection of the UTF-8 code is not mandatory, and exclusively the encoding rules for XML documents will apply only:

a) If there is no encoding declaration[2] and no BOM (Byte Order Mark), this will be UTF-8.
b) For UTF-16 or UTF-32 encodings, a respective BOM entry is mandatory. An encoding declaration is not stringently required, but is still recommended.
c) If an obsolete (viz. not Unicode-capable) encoding is used, an explicit encoding declaration must be provided. This will typically be "iso-8859-1" or "windows-1252" for what is called the standard "ANSI" Code.

1.4 Versioning

There are two version identifiers: **Major Version** and **Minor Version**. The combination of *Major Version /Minor Version* coincides with the version identifier of this document.

Please note: These version identifiers merely relate to the standard tags only; if there is versioning for internal tags, this must occur via an additional proprietary version identifier.

Major Version (Schema Version)

The *Major Version* is determined in the XML-Namespace. The namespace has the following format:

`http://progress-m.com/ProgressXML/Version1`

The *Major Version* will only be changed if the schema syntax changes in an incompatible manner.

Changes of the *Major Version* should be avoided whenever possible. A change of the *Major Version* will normally only be necessary if the previous version is expected to be further extended in future (at least temporarily), representation that way an independent branch of the specification.

Minor Version

The *Minor Version* is entered in the DocInfo block of data.

The various Minor Versions must be mutually compatible syntactically. That is to say, one version may have tags that do not exist in the other version, or one and the same tag may have different semantic meanings in different versions respectively; but the schema syntax must be compatible.

Example #1: an angle detail that originally was in degrees will be interpreted as a tenth-degree value in a new version.

Example #2: an integer angle detail in an old version will not be used anymore; and a new tag with a floating point angle detail will be introduced instead.

Please note: Supplements that are fully downward compatible may also be added within a Minor Version. Typically, these are fields that have been additionally included in the PXML standard.

[2] Here, by **Encoding Declaration** we understand a header in the format of **<?xml version="1.0" encoding="iso-8859-1" ?>**.

1.5 Compatibility with UNICAM

Purpose

When defining the ProgressXML format, we have tried to achieve extensive compatibility with the UNICAM CAD-CAM interface. More specifically, it should be possible for conversions of the type

UNICAM →ProgressXML→ UNICAM

to get back the original file (apart from a few exceptions).

This requirement has made some elements, terms or structures being introduced into this specification that actually seem rather irrational.

Where reference is made to the UNICAM documentation without quoting the version, as a general rule, this will be meant to be a reference to versions 5.2 or 6.0 respectively of the UNICAM interface.

UNICAM elements not supported

Some elements of the UNICAM CAD-CAM interface have deliberately not been integrated in PXML:

1) **Actual quantity:** will always be set to 0 for export to UNICAM.
2) **End hook bending shapes:** only free bending shapes will be supported; upon import from UNICAM, end hook bending will be converted to free bending shapes.

1.6 PXML Delegate Files and PXML Include Files

Production data is frequently provided by two distinct systems: a CAD system and an EPR system. The CAD system provides geometrical data for the elements to be produces, while the ERP system typically cares about header data like order numbers and delivery dates. The **PXML Delegate File** mechanism is a technique that allows to merge information coming from those distinct systems into one single PXML document.

The *PXML Delegate File* itself is typically generated by the ERP system. It therefore typically contains order heading data, and optionally (with batch production) provides some target piece count values, too. But this file doesn't specify any geometrical details about the elements to be produced. Instead it "delegates" those geometrical details to an **Include File** that has to be provided by a CAD system.

Syntactically, the *Delegate File* is just an ordinary PXML file. But instead of enclosing geometrical data directly, it contains *Include* directives that specifies *Include File* paths[3]. The *Include Files* (CAD files) are again ordinary PXML files[4]. The example listed below demonstrates how one *Delegate File* is merged with two *Include Files* resulting in a single complete PXML file[5]:

Delegate File \\SrvXY\Examples\delegate.pxml:

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
	<DocInfo>
		<MajorVersion>1</MajorVersion>
		<MinorVersion>3</MinorVersion>
	</DocInfo>
	<Order>
		<OrderNo>100000000041</OrderNo>
		<DeliveryDate>2013-11-05T10:50:07+01:00</DeliveryDate>
		<Product>
			<ElementNo></ElementNo>	<!--Ignore3-->
			<PieceCount>3</PieceCount>
			<Include>\\SrvXY\Examples\CADFiles\abcd1.pxml</Include>
		</Product>
		<Product>
			<PieceCount>5</PieceCount>
			<Include>CADFiles\abcd2.pxml</Include>
			<Slab>
				<X>600</X>
			</Slab>
		</Product>
	</Order>
</PXML_Document>
```

[3] The path to an *Include File* can be either absolute, or relative to the directory in which the *Delegate File* resides.
[4] *Include Files* are sometimes provided in other formats, too (e.g. UNICAM or BVBS); such files have to be converted into PXML when being imported.
[5] All items marked with "Ignore" in the listings have been added to the example for definition purposes only. This items should not be present at all in a real world example.

CAD File \\SrvXY\Examples\CADFiles\abcd1.pxml:

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
	<DocInfo>
		<MajorVersion>1</MajorVersion>
		<MinorVersion>3</MinorVersion>
	</DocInfo>
	<Order>
		<Component>C1</Component>
		<Product>
			<ElementNo>E1</ElementNo>
			<ProductType>DW</ProductType>
			<Slab>
				<PartType>01</PartType>
			</Slab>
		</Product>
	</Order>
</PXML_Document>
```

CAD File \\SrvXY\Examples\CADFiles\abcd2.pxml:

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
	<DocInfo>
		<MajorVersion>1</MajorVersion>
		<MinorVersion>3</MinorVersion>
	</DocInfo>
	<Order>	<!--Ignore5-->
	</Order>
	<Order>
		<OrderNo>aa2</OrderNo>	<!--Ignore1-->
		<Component>C2</Component>	<!--Ignore6-->
		<Product>
			<ElementNo>E2</ElementNo>
			<ProductType>DW</ProductType>
			<PieceCount>1</PieceCount>	<!--Ignore4-->
			<Slab>
				<PartType>01</PartType>
			</Slab>
		</Product>
		<Product>	<!--Ignore2-->
			<ElementNo>E3</ElementNo>
		</Product>
	</Order>
	<Order>
		<Product>	<!--Ignore2-->
			<ElementNo>E5</ElementNo>
		</Product>
	</Order>
</PXML_Document>
```

Resulting merged File:

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
	<DocInfo>
		<MajorVersion>1</MajorVersion>
		<MinorVersion>3</MinorVersion>
	</DocInfo>
	<Order>
		<OrderNo>100000000041</OrderNo>
		<Component>C1</Component>
		<DeliveryDate>2013-11-05T10:50:07+01:00</DeliveryDate>
		<Product>
			<ElementNo>E1</ElementNo>
			<ProductType>DW</ProductType>
			<PieceCount>3</PieceCount>
			<Slab>
				<PartType>01</PartType>
			</Slab>
		</Product>
		<Product>
			<ElementNo>E2</ElementNo>
			<ProductType>DW</ProductType>
			<PieceCount>5</PieceCount>
			<Slab>
				<X>600</X>
			</Slab>
			<Slab>
				<PartType>01</PartType>
			</Slab>
		</Product>
	</Order>
</PXML_Document>
```

The *Include* integration is done according the following rules:

a) An *Include* directive can, in theory, be present on all levels of the PXML hierarchy. But the most common usage is to put *Include* operations exclusively on *Product* level (as in the example listed above). The hierarchical level of the *Include* directive will herein be called **Include Level**.

b) All *Include File* items that are hierarchically above or on same level as *Include Level* have to be ignored if they are already set in the original *Delegate* file (see "Ignore1" in the example) or already set by a previously included *Include* file (see "Ignore6" in the example). See "Ignore3" and "Ignore4" for some more examples. Empty strings are considered as null values in this circumstances

c) If the *Include File* has more than one object on *Include Level*, only the first matching Object is considered (see "Ignore2" in the example).

Besides the simple *Include* directive as mentioned so far, a **complex Include directive** may be specified as follows:

```
<?xml version="1.0"?>
<Include>
	<FileName>\\SrvXY\CADFiles\abcd1.pxml</FileName>
	<Filter>
		<IncludeFilter>
			<DataTableName>Slab</DataTableName>
			<DataColumnName>PartType</DataColumnName>
			<Condition>\s*01\s*</Condition>	<!--RegEx condition-->
		</IncludeFilter>
		<IncludeFilter>
			<DataTableName>Steel</DataTableName>
			<DataColumnName>Name</DataColumnName>
			<Condition>\s*cageXy\s*</Condition>	<!--RegEx condition-->
		</IncludeFilter>
	</Filter>
</Include>
```

This complex *Include* specification has to be embedded into the string field as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
	<DocInfo>
		<MajorVersion>1</MajorVersion>
		<MinorVersion>3</MinorVersion>
	</DocInfo>
	<Order>
		<OrderNo>100000000041</OrderNo>
		<Product>
			<PieceCount>3</PieceCount>
			<Slab>
				<Steel>
					<Include>
&lt;?xml version="1.0"?>
&lt;Include&gt;
	&lt;FileName&gt;\\SrvXY\CADFiles\abcd1.pxml</FileName&gt;
	&lt;Filter&gt;
		&lt;IncludeFilter&gt;
			&lt;DataTableName&gt;Slab&lt;/DataTableName&gt;
			&lt;DataColumnName&gt;PartType</DataColumnName&gt;
			&lt;Condition&gt;\s*01\s*&lt;/Condition&gt;
		&lt;/IncludeFilter&gt;
		&lt;IncludeFilter&gt;
			&lt;DataTableName&gt;Steel&lt;/DataTableName&gt;
			&lt;DataColumnName&gt;Name&lt;/DataColumnName&gt;
			&lt;Condition&gt;\s*cageXy\s*&lt;/Condition&gt;
		&lt;/IncludeFilter&gt;
	&lt;/Filter&gt;
&lt;/Include&gt;
					</Include>
				</Steel>
			</Slab>
		</Product>
	</Order>
</PXML_Document>
```

This example includes the first *Steel* block that satisfies the condition of having the name “cageXY” and being part of a first half slab.

UNICAM Delegate File

A UNICAM file can also be interpreted as a *delegate* file (starting from UNICAM version 6.0).

The SLABDATE info fields 1 to 4 are used here:

Info1: contains the delegate file marker, namely the fixed text "#include

Info2+Info3+Info4: contains the path to the PXML file (the path is formed by binding these 3 fields together, whereby the spaces at the end of each of these fields are cut away).

The path specification can contain the following macros:

<A>: Order name (from line 3 of the HEADER)

<E>: Element name

<P>: Project (line 4 of the HEADER)

<K>: Customer (line 12 of the HEADER)

An example of the info lines would be as follows:

Info1: **#include**

Info2: **\\srvXY\CAD**

Info3: **<K>\<P>**

Info4: **<A>_<E>.PXML**

The UNICAM *delegate* file can also contain a "rotation angle of the element" (SLABDATE, line 4, position 55-57 – for version 6.0). This rotation can now have already been carried out in CAD, or it can be specified subsequently by the control system. Therefore, this angle, minus the angle from *Product.RotationPosition*, must be used as the rotation request on merging. I.e. when creating the merged file, the element must be rotated by the angle of the mentioned angle difference. The rotation value from the UNICAM *delegate* file must then be used as the new value for *Product.RotationPosition*.

2 Structure overview

The following representation describes the structure of the PXML file. Details regarding the various fields are provided further down.

The figures in red on the right specify how often an entry may be made or has to be made respectively:

- **1** = precisely once
- **0,1** = 0 or once
- **>=1** = at least once
- **n** = any number of times (including 0)

The length of the character string is basically arbitrary. Upon export to UNICAM, however, the character strings will be reduced to a maximum length, or will be filled with blank characters to reach a certain length respectively.

The text-items shown here (e.g. aaaaaaaa) are mere examples. Unless stated otherwise, all texts are arbitrary alphanumerical character strings; however, please note that some of these free texts will have to be converted to numerical values for UNICAM export. Those values that are already numerical in the PXML definition will be annotated in the Table with **Int**, **Bool**, or **Double**.

<PXML Document>		**1**
<DocInfo GlobalID="aaa">		**1**
<MajorVersion>1</MajorVersion>	Int	1
<MinorVersion>3</MinorVersion>	Int	1
<Comment>aaaaaaaaaaaaa</Comment>		0,1
<ConvertConventions>aaa#bbb#ccc</ConvertConventions>		0,1
<Mode>		**n**
<ID>aaaaa</ID>		1
<Val>true</Val>		0,1
</Mode>		
</DocInfo>		
<Order GlobalID="aaa">		**n**
<OrderNo>aaaaaaaa</OrderNo>		0,1
<Structure>cccc</Structure>		0,1
<Building>cccc</Building>		0,1
<Storey>cccc</Storey>		0,1
<SubStorey>cccc</SubStorey>		0,1
<Component>bbbbb</Component>		0,1
<DrawingNo>dddddd</DrawingNo>		0,1
<DrawingDate>dd.mm.yyyy</DrawingDate>		0,1
<DrawingRevision>ee</DrawingRevision>		0,1
<DrawingAuthor>aaaaaaaaaaaaaaaaaaa</DrawingAuthor>		0,1
<ErpProjectUnit>dddddd</ErpProjectUnit>		0,1
<DeliveryDate>2010-04-16T11:46:48.933+02:00</DeliveryDate>		0,1
<GenericOrderInfo01>aaaaaaaaaaaaaaaaaaaa</GenericOrderInfo01>		0,1
: : : :		0,1
<GenericOrderInfo20>aaaaaaaaaaaaaaaaaaaa</GenericOrderInfo20>		0,1
<Comment>aaaaaaaaaaaaaaaaaaaa</Comment>		0,1
<OrderArea>1234.5</OrderArea>	Double	0,1
<ImportSource>aaaaaaaaaaaaaaaaaaaa</ImportSource>		0,1
<ImportSourceType>aaaaaaaaaaaaaaaaaaaa</ImportSourceType>		0,1

<ApplicationName>aaaaaaaaaaaaaaaaaaa</ApplicationName>		0,1
<ApplicationGUID>aaaaaaaaaaaaaaaaaaa</ApplicationGUID>		0,1
<ApplicationVersion>aaaaaaaaaaaaaaaaaaa</ApplicationVersion>		0,1
<OrderInfo Type="AccountingPosition" GlobalID="aaa">		**n**
<Code>aaaaa</Code>		0,1
<OrderInfoVal Type=”aa” V=”bb” U="mm" Culture="en"/>		**n**
</OrderInfo>		
<Product GlobalID="aaa">		**n**
<ElementNo>111</ElementNo>		0,1
<ProductType>11</ProductType>		0,1
<TotalThickness>6666</TotalThickness>	Double	0,1
<DoubleWallsGap>777</DoubleWallsGap>	Double	0,1
<PieceCount>1111</PieceCount>	Int	0,1
<TurnWidth>3500</TurnWidth>	Double	0,1
<Comment>aaaaaaaaaa</Comment>		0,1
<RotationPosition>111</RotationPosition>	Double	0,1
<StackNo>111</StackNo>		0,1
<StackID>111</StackID>		0,1
<StackingSequence>111</StackingSequence>		0,1
<StackingLevel>111</StackingLevel>		0,1
<StackingX>11111</StackingX>	Double	0,1
<StackingY>11111</StackingY>	Double	0,1
<StackingZ>11111</StackingZ>	Double	0,1
<StackingAngle>111</StackingAngle>	Double	0,1
<StackingRotY>111</StackingRotY>	Double	0,1
<StackingRotX>111</StackingRotX>	Double	0,1
<P1X>111111</P1X>	Double	0,1
<P1Y>111111</P1Y>	Double	0,1
<P1Z>111111</P1Z>	Double	0,1
<P2X>111111</P2X>	Double	0,1
<P2Y>111111</P2Y>	Double	0,1
<P2Z>111111</P2Z>	Double	0,1
<P3X>111111</P3X>	Double	0,1
<P3Y>111111</P3Y>	Double	0,1
<P3Z>111111</P3Z>	Double	0,1
<AdditionInfo>aaaa</AdditionInfo>		0,1
<UnloadingInfo>aaaa</UnloadingInfo>		0,1
<TransportInfo>aaaa</TransportInfo>		0,1
<ItemPosition>aaaa@bbbb@cccc@dddd</ItemPosition>		0,1
<ElementInfo Type="AccArea" Inventory=true GlobalID="aaa">		**n**
<Code>aaaaaaa</Code>		0,1
<Description>aaaaaaa</Description>		0,1
<ObjectID>aaaaaaa</ObjectID>		0,1
<PieceCount>22</PieceCount>	Int	0,1
<Val1>123.4</Val1>	Double	0,1
<Val2>-987.2</Val2>	Double	0,1
<Unit>m³</Unit>		0,1
<Details>aaaaaaa</Details>		0,1
<ElemInfoVal Type=”Width” V=”225.3”/>		**n**

	</ElementInfo>		
	<Slab GlobalID="aaa">		**n**
(Legacy)	<SlabNo>111</SlabNo>		0,1
	<PartType>11</PartType>		0,1
(Legacy)	<ProductAddition>44</ProductAddition>		0,1
	<ProductionWay>aa</ProductionWay>		0,1
(Legacy)	<NumberOfMeansOfTransport>666</NumberOfMeansOfTransport>		0,1
(Legacy)	<TransportSequence>777</TransportSequence>		0,1
(Legacy)	<PileLevel>888</PileLevel>		0,1
(Legacy)	<TypeOfUnloading>99</TypeOfUnloading>		0,1
(Legacy)	<MeansOfTransport>00</MeansOfTransport>		0,1
	<ExpositionClass>aaaaaaa</ExpositionClass>		0,1
	<SlabArea>11.111</SlabArea>	Double	0,1
	<SlabWeight>55555.5</SlabWeight>	Double	0,1
	<ProductionThickness>2222</ProductionThickness>	Double	0,1
	<MaxLength>11111</MaxLength>	Double	0,1
	<MaxWidth>22222</MaxWidth>	Double	0,1
	<IronProjectionLeft>±3333</IronProjectionLeft>	Double	0,1
	<IronProjectionRight>±4444</IronProjectionRight>	Double	0,1
	<IronProjectionBottom>±5555</IronProjectionBottom>	Double	0,1
	<IronProjectionTop>±6666</IronProjectionTop>	Double	0,1
	<X>111111</X>	Double	0,1
	<Y>222222</Y>	Double	0,1
	<Z>222222</Z>	Double	0,1
	<RotX>111111</RotX>	Double	0,1
	<RotY>222222</RotY>	Double	0,1
	<RotZ>222222</RotZ>	Double	0,1
	<ProdX>111111</ProdX>	Double	0,1
	<ProdY>222222</ProdY>	Double	0,1
	<ProdZ>222222</ProdZ>	Double	0,1
	<ProdRotX>111111</ProdRotX>	Double	0,1
	<ProdRotY>222222</ProdRotY>	Double	0,1
	<ProdRotZ>222222</ProdRotZ>	Double	0,1
(Legacy)	<OrderPosition>aaaaaaaa</OrderPosition>		0,1
(Legacy)	<ProductGroup>bbbb</ProductGroup>		0,1
(Legacy)	<SlabType>33</SlabType>		0,1
(Legacy)	<ItemDesignation>c...c</ItemDesignation>		0,1
(Legacy)	<ProjectCoordinates>111111 222222…</ProjectCoordinates>		0,1
(Legacy)	<PositionInPileX>111111</PositionInPileX>	Double	0,1
(Legacy)	<PositionInPileY>111111</PositionInPileY>	Double	0,1
(Legacy)	<PositionInPileZ>111111</PositionInPileZ>	Double	0,1
(Legacy)	<AngleInPile>111111</AngleInPile>	Double	0,1
	<GenericInfo01>aaaaaaa…aaaaaa</GenericInfo01>		0,1
	<GenericInfo02>aaaaaaa…aaaaaa</GenericInfo02>		0,1
	<GenericInfo03>aaaaaaa…aaaaaa</GenericInfo03>		0,1
	<GenericInfo04>aaaaaaa…aaaaaa</GenericInfo04>		0,1
	<ReforcemInfo></ReforcemInfo>		0,1
	<Outline Type="lot" GlobalID="aaa">		**n**
	<X>22222</X>	Double	0,1

<Y>22222</Y>	Double	0,1
<Z>22222</Z>	Double	0,1
<RotX>22222</RotX>	Double	0,1
<RotY>22222</RotY>	Double	0,1
<RotZ>22222</RotZ>	Double	0,1
<Height>22222</Height>	Double	0,1
<Name>bbbb...bbbb</Name>		0,1
<GenericInfo01>bbbb...bbbb</GenericInfo01>		0,1
<GenericInfo02>bbbb...bbbb</GenericInfo02>		0,1
<MountingInstruction>2</MountingInstruction>		0,1
<MountPartType>33</MountPartType>		0,1
<MountPartArticle>aaaaaaaaaa</MountPartArticle>		0,1
<MountPartIronProjection>333</MountPartIronProjection>	Double	0,1
<MountPartDirection>±555</MountPartDirection>	Double	0,1
<MountPartLength>66666</MountPartLength>	Double	0,1
<MountPartWidth>77777</MountPartWidth>	Double	0,1
<ConcretingMode>aa</ConcretingMode>		0,1
<ConcreteQuality>aaaaaaaa</ConcreteQuality>		0,1
<UnitWeight>4.444</UnitWeight>	Double	0,1
<Volume>33.333</Volume>	Double	0,1
<Layer>aaa</Layer>		0,1
<ObjectID>aaaaaa</ObjectID>		0,1
<Shape GlobalID="aaa">		**n**
<Cutout>false</Cutout>	Bool	0,1
<RefHeight>33.333</RefHeight>	Double	0,1
<SVertexGlobalID="aaa">		**n**
<X>11111</X>	Double	0,1
<Y>22222</Y>	Double	0,1
<Bulge>33333</Bulge>	Double	0,1
<LineAttribute>aaaa</LineAttribute>		0,1
<Profile>-10 10|0 0</Profile>		0,1
<DX>22222</DX>	Double	0,1
<DY>22222</DY>	Double	0,1
</SVertex>		
</Shape>		
</Outline>		
<Steel Type="mesh" GlobalID="aaa">		**n**
<X>22222</X>	Double	0,1
<Y>22222</Y>	Double	0,1
<Z>22222</Z>	Double	0,1
<RotX>22222</RotX>	Double	0,1
<RotY>22222</RotY>	Double	0,1
<RotZ>22222</RotZ>	Double	0,1
<ToTurn>true</ToTurn>	Bool	0,1
<StopOnTurningSide>true</StopOnTurningSide>	Bool	0,1
<Name>2</Name>		0,1
<GenericInfo01>aaa…aaa</GenericInfo01>		0,1
: : : :		0,1
<GenericInfo06>aaa…aaa</GenericInfo06>		0,1

<MeshType>2</MeshType>		0,1
<WeldingDensity>100</WeldingDensity>	Int	0,1
<BorderStrength>2</BorderStrength>	Int	0,1
<ProdX>±123</ProdX>	Double	0,1
<ProdY>±123</ProdY>	Double	0,1
<ProdZ>±123</ProdZ>	Double	0,1
<ProdRotX>±123</ProdRotX>	Double	0,1
<ProdRotY>±123</ProdRotY>	Double	0,1
<ProdRotZ>±123</ProdRotZ>	Double	0,1
<Layer>aaa</Layer>		0,1
<ObjectID>aaa</ObjectID>		0,1
<Bar GlobalID="aaa">		**n**
<ShapeMode>realistic</ShapeMode>		0,1
<ReinforcementType>3</ReinforcementType>		0,1
<SteelQuality>aaa</SteelQuality>		0,1
<PieceCount>55555</PieceCount>	Int	0,1
<Diameter>666</Diameter>	Double	0,1
<X>±88888</X>	Double	0,1
<Y>±99999</Y>	Double	0,1
<Z>±77777</Z>	Double	0,1
<RotZ>±123</RotZ>	Double	0,1
<ArticleNo>bbbbbbbbbb</ArticleNo>		0,1
<NoAutoProd>false</NoAutoProd>	Bool	0,1
<ExtIronWeight>444.444</ExtIronWeight>	Double	0,1
<Bin>123</Bin>		0,1
<Pos>aaa</Pos>		0,1
<Note>aaa</Note>		0,1
<Machine>aaa</Machine>		0,1
<BendingDevice>1</BendingDevice>		0,1
<Spacer GlobalID="aaa">		**n**
<Type>222</Type>	Int	0,1
<Position>33333</Position>	Double	0,1
</Spacer>		
<WeldingPointGlobalID="aaa">		**n**
<WeldingOutput>77</WeldingOutput>	Double	0,1
<Position>33333</Position>	Double	0,1
<WeldingPointType>111</WeldingPointType>	Int	0,1
<WeldingPrgNo>222</WeldingPrgNo>	Int	0,1
<GroupID>222</GroupID>		0,1
</WeldingPoint>		
****		**n**
<RotX>±123</RotX>	Double	0,1
<BendY>±123</BendY>	Double	0,1
<L>33333</L>	Double	0,1
<R>22</R>	Double	0,1
****		
</Bar>		
<Girder GlobalID="aaa">		**n**
<PieceCount>55555</PieceCount>	Int	0,1

<X>±88888</X>	Double	0,1
<Y>±99999</Y>	Double	0,1
<Z>±77777</Z>	Double	0,1
<GirderName>aaaaaaaaaa</GirderName>		0,1
<Length>55555</Length>	Double	0,1
<AngleToX>±999</AngleToX>	Double	0,1
<NoAutoProd>true</NoAutoProd>	Bool	0,1
<Height>222</Height>	Double	0,1
<TopExcess>222</TopExcess>	Double	0,1
<BottomExcess>222</BottomExcess>	Double	0,1
<Weight>33.333</Weight>	Double	0,1
<TopFlangeDiameter>44</TopFlangeDiameter>	Double	0,1
<BottomFlangeDiameter>44</BottomFlangeDiameter>	Double	0,1
<GirderType>2</GirderType>	Int	0,1
<MountingType>1</MountingType>	Int	0,1
<ArticleNo>aaa</ArticleNo>		0,1
<Machine>aaa</Machine>		0,1
<Period>000</Period>	Double	0,1
<PeriodOffset>111</PeriodOffset>	Double	0,1
<Width>80</Width>	Double	0,1
<AnchorBarGlobalID="aaa">		**n**
<Type>222</Type>	Int	0,1
<Length>111</Length>	Double	0,1
<Position>33333</Position>	Double	0,1
</AnchorBar>		
<GirderExt Type="SplicePos" GlobalID="aaa">		**n**
<Position>33333</Position>	Double	0,1
<Flags>0</Flags>	Int	0,1
<Val0>33333</Val0>	Double	0,1
<Val1>33333</Val1>	Double	0,1
<Val2>33333</Val2>	Double	0,1
<Val3>33333</Val3>	Double	0,1
</GirderExt>		
<Section GlobalID="aaa">		**n**
<L>195</L>	Double	0,1
<S>-100</S>	Double	0,1
<F>50</F>	Double	0,1
</Section>		
</Girder>		
<Alloc Type="Bar" GlobalID="aaa">		**n**
<GuidingBar>2</GuidingBar>	Int	0,1
<Region GlobalID="aaa">		**n**
<IntervalCount>5</IntervalCount>	Int	0,1
<Pitch>111</Pitch>	Double	0,1
<IncludeBegin>true</IncludeBegin>	Bool	0,1
<IncludeEnd>true</IncludeEnd>	Bool	0,1
<RefIndex>4</RefIndex>	Int	0,1
</Region>		
</Alloc>		

<SteelExt Type="Xyz" GlobalID="aaa">		**n**
<Info>aaaaaaaaaa</Info>		0,1
</SteelExt>		
</Steel>		
</Slab>		
</Product>		
</Order>		
<Feedback ItemType="Bar" GlobalID="aaa">		**n**
<MessageType>error</MessageType>		0,1
<Code>123ABC</Code>		0,1
<InfoValue>123XYZ</InfoValue>		0,1
<PieceCount>3</PieceCount>	Int	0,1
<MaterialType>16A</MaterialType>		0,1
<MaterialBatch>12345@AR177228C</MaterialBatch>		0,1
<MaterialWeight>12345</MaterialWeight>	Double	0,1
<ProdDate>2010-07-30T09:06:05+02:00</ProdDate>		0,1
<Machine>aaa</Machine>		0,1
<Description Culture="en" Text="aaaaaaaaaa"/>		n
<FbVal T=”OrdNo” V=”AA00048386”/>		**n**
</Feedback>		
</PXML_Document>		

3 Detail specifications

3.1 Global ID

Each PXML Table may have an attribute titled **GlobalID** to facilitate global identification of the respective item. Here, "global" is to be understood as cross-system, viz. as opposed to intra-system IDs that would be merely known within a subsystem only.

Typically, the *GlobalID* is assigned by the system that generates the data (viz. in the CAD or the master computer respectively), and is then applied by the subsystems in order to be used for feedbacks to the higher-ranking system.

Some systems use a numerical *GlobalID*, others use a string (such as a GUID string, for example). However, similar to practically all PXML fields, it is also true for the *GlobalID* that its use is optional, and it is thus possible to do without any *GlobalID* altogether, or to use the *GlobalID* in merely some PXML Tables only.

3.1.1 Unambiguity of the GlobalID

The *GlobalID* should be unambiguous within any one item type, so for instance, two different *Bar* items should have different *GlobalID*s (this is not only true for 2 *Bar* items belonging to the same *Steel* block, but also *Bar* items from different *Steel* blocks shall have different *GlobalIDs*). The *GlobalID*s of different item types (such as *Bar* or *Girder*, for example), on the other hand, may have overlaps.

However, unambiguity of the *GlobalID* is not a definite rule of PXML, but merely a requirement of the application, which may be more or less pronounced as a function of the specific application. Thus, for a *PTS* query it may be sufficient if the *GlobalID*s within any one query document are unambiguous. For production feedback from machinery, however, you will typically need a more universal unambiguity of the *GlobalID*s. (*PTS* queries and production feedback will be explained in more detail further below).

3.1.2 Special case: GlobalID of DocInfo Table

For the *DocInfo*Table, the *GlobalID* has a slightly modified meaning: here, it does not merely identify the table entry (as there is no more than one *DocInfo*table entry), but rather the whole document.

3.1.3 Automatic generation of GlobalIDs

If the system generating the data does not deliver any *GlobalID*, the subsystem may generate this identifier on its own to then provide purposeful feedback. Here, the following pattern shall be used:

ParentItemGlobalID.ItemIndex

Here, "ParentItemGlobalID" is the *GlobalID* of the parent item immediately overlying, and "ItemIndex" is the zero-based element index of the respective item within its immediate parent environment. As *Order* items do not have a parent, merely an ItemIndex is used here (no prefix).

Example 1: A *Bar* item has the *GlobalID* "ABC"; here, the *GlobalID*s can be generated for the segments of this reinforcing bar as follows:
"ABC.0"
"ABC.1"
"ABC.2"
and so forth.

Example 2: All PXML items are without a *GlobalID*. Now, the following *GlobalID* can be automatically generated for the fifth rebar in the third *Order* block:

"2.0.0.0.4"
(*Order i*ndex = 2, *Product* index = 0; *Slab* index = 0, *Steel* index = 0; *Bar* index = 4).

3.1.4 Late generation of GlobalIDs

Where a cross-system identification of items is necessary, *GlobalIDs* are helpful or even necessary. In other cases, however, such IDs can be a hindrance because they introduce artificial differentiation where this would not be necessary: If two PXML objects are identical in content, it can be software-technically efficient to handle them using a common object; however, this optimization would be thwarted by different *GlobalIDs*. If it is necessary to differentiate between the two objects from an application perspective, it also makes sense to have different *GlobalIDs*. However, if no differentiation is required from the application point of view, the *GlobalIDs* would introduce differentiation in an artificial and disruptive way.

It therefore makes sense to generate *GlobalIDs* in the overall process only when they are actually needed.

3.2 DocInfo

The *DocInfo* block must be there precisely once per document.

3.2.1 GlobalID

The *GlobalID* entry is optional and is used as identification of the whole document, especially in regard to PTS feedback.

3.2.2 Document Version

The *MajorVersion* and *MinorVersion* denote the main version of the underlying PXML specification. See section 1.4.

3.2.3 Comment

In the *Comment* field any comment be entered on the document. The *Comment* is optional, but should be placed above all in *PTS* feedback documents to describe the state of the sending system (identification of the system, software version, parameter version).

3.2.4 ConvertConventions

When, for some reason, a PXML file is not PXML-compliant, one can annotate the nonconformity in the *ConvertConventions* filed. When importing such a file, the data can then be converted according that information, in order to get a fully PXML-compliant data set.

If several conventions should be specified, they have to be separated by a # sign.

PXML-based systems work internally without considering the *ConvertConventions*. The *ConvertConventions* are to be processed immediately when data is imported; after having carried out the required conversions, the *ConvertConventions* field should be cleared.

Basically there are no *ConvertConventions* that must necessarily be accepted. The following conventions are accepted often, though:

- **SegmentsHaveOuterLen**: Indicates, that the Segment.L values indicate the outer length of the segments, and not (as required in PXML) the center dimensions.

3.2.5 Mode

Additional information that would typically include processing instructions intended for the data receiver may be specified via *Mode* entries.

Each *Mode* entry will be composed of an *ID* field that will define the meaning of the entry and of a *Val* field that will hold the value.

The following *Mode* IDs are defined in the PXML standard[6]:

- **ProdLayout**: Possible values are "true" or "false".
  This *Mode* instruction specifies whether or not the elements have already been arranged and laid out ready for production.
  For prefabricated component circulation systems, this means in detail: if *ProdLayout=true* these are data that were arranged and laid out on production pallets, and if *ProdLayout=false* these are CAD data the element coordinates of which have not as yet been aligned to production pallets.
  Please note: if *ProdLayout=true* the PXML document (viz. the file) will correspond to precisely one production unit (such as one circulation pallet). That is to say, the data will be "portioned" by production units such that the receiving system will know what elements will go into one and the same production unit.
  Please note regarding the PTS check: If *ProdLayout=false* the PTS server will check the elements separately, thus ignoring their absolute position (here, the PTS server will assume that the elements will be positioned in their ideal position later on while they will be assigned to the pallets). However, the PTS server will accept the rotational orientation of the elements as being fixed, viz. it will *not* try to assume the ideal rotational orientation autonomously[7].
- **RequestedCulture:** You can use one or several of these entries to specify what national languages are of interest. Thus, it can be communicated to a PTS server which are the languages in which the message texts are to be supplied, for example.
  These entries must be specified through the ISO 639 Language Codes, optionally extended by the ISO 3166 Country Codes (such as "en" or "en-US", for example). If several languages are requested, the respective number of *Mode* entries must be specified accordingly.
- **EstimateProdTime:** Possible values are "true" or "false".
  This *Mode* directive instructs a PTS server whether or not to do a production time computation.
  This option is intended to facilitate that the production time simulation (which may require a huge computational effort) will only be used if it is really required.
- **EnableProduction:** Possible values are "true" or "false".
  This *Mode* directive declares that data is approved for production. This directive is used to distinguish between final CAD data (and thus ready for production) and preliminary versions of the same data.

---

[6] In addition, each application is free to define its own additional *Mode ID*s for internal purposes only. These are then to begin with "I_" such as to avoid conflicts with potential later extensions of the standard.

[7] It is with intent that a different treatment is defined for the position offset and for the rotational orientation respectively: the former will not be checked if *ProdLayout=false*, whereas the latter will always be checked. It would be lacking in practical relevance to request that a CAD system ideally position the slabs from a production-technical point of view when it creates a PTS request as this would rather be an essential task of the pallet assignment process. On the other hand, it would not work to have the PTS server check out all possible rotational orientations autonomously. This is because, on the one hand, the PTS server knows little about the practicability of the rotation of the elements, and on the other hand there may be false PTS check results when systems are interlinked whereby different sub-systems may assume different rotational orientations. (On principle, this argument could also be used for the positioning offset, but here it is less relevant in practice).

- **EnableReinforcement:** Possible values are "true" or "false".
  This *Mode* directive is similar to the *EnableProduction* directive, but restricts the approval to reinforcement production. This directive is particularly useful in systems requiring a longer lead time in reinforcement preparation. In such cases reinforcement production may need to start before the final drawing approval is achieved.
- **EnableProcurement:** Possible values are "true" or "false".
  This *Mode* directive is similar to the *EnableProduction* directive, but restricts the approval to material procurement.

Example of *Mode* sections:

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
  <DocInfo GlobalID="7C7E1FC5-0A46-48c5-A3B2-249D75B70BCF">
    <MajorVersion>1</MajorVersion>
    <MinorVersion>3</MinorVersion>
    <Mode>
      <ID>ProdLayout</ID>
      <Val>true</Val>
    </Mode>
    <Mode>
      <ID>RequestedCulture</ID>
      <Val>en</Val>
    </Mode>
    <Mode>
      <ID>RequestedCulture</ID>
      <Val>fr-BE</Val>
    </Mode>
    <Mode>
      <ID>EstimateProdTime</ID>
      <Val>false</Val>
    </Mode>
  </DocInfo>
</PXML_Document>
```

3.3 Order

The *Order* section may appear any number of times (or not at all). It holds production units with mutually matching order information.

Please note: an *Order* section does *not* have to hold *all* the production units of a commission. However, is must not hold production units for different commissions.

In the order section, basically, the information is stored which is independent of the “elementation”, i.e. independent of how building parts are divided in precast-able elements. These data are therefore typically independent of the details of the shop drawings describing the precast elements.

3.3.1 Order Information

OrderNo: order identifier.

Structure: Group of buildings, e.g. First row of a row house settlement.

Building: Single building.

Storey: description of storey.

Substorey: Section of a storey.

Component: component.

DrawingNo: drawing identifier.

DrawingDate: the date on which the drawing was generated or revised respectively.

DrawingRevision: the revision number of the drawing.

DrawingAuthor: the person who made up the drawing.

ErpProjectUnit: Subproject ID of ERP. Is often to equal to *DrawingNo*. But while *DrawingNo* follows a classification defined in the CAD, *ErpProjectUnit* refers to a project structuring in ERP.

DeliveryDate: scheduled delivery date.

GenericOrderInfo01[8]: Project – Name

GenericOrderInfo02: Project – Addition

GenericOrderInfo03: Project – Address

GenericOrderInfo04: Project – Info

GenericOrderInfo05: Location – Name

GenericOrderInfo06: Location – Street

GenericOrderInfo07: Location – Zip Code

GenericOrderInfo08: Location – City

GenericOrderInfo09: Customer – Name

GenericOrderInfo10: Customer – Street

GenericOrderInfo11: Customer – Zip Code

GenericOrderInfo12: Customer – City

GenericOrderInfo13..20: Application specifically defined additional information.

Comment: Any comment regarding the Order.

OrderArea: Accounting area in m^2 of the whole order (i.e. related to the *OrderNo* in question).

This value is typically used for communication from CAD to ERP.

3.3.2 Import Source Information

ImportSource: If the Order was imported from another file, the file name can be entered in *ImportSource*.

ImportSourceType: In *ImportSourceType,* the type of the source file can be specified here (such as BVBS or Unitechnik 5.2, for example).

3.3.3 ApplicationName, ApplicationGUID, ApplicationVersion

These information are devised to identify the application by means of which the data has been generated or revised respectively. In case of doubt, this will determine how data-content is to be interpreted.

Typically, these fields will be evaluated upon loading from the file or the clipboard respectively, and will then be set to zero (after conversion, if appropriate). These values should not be used within the application.

The **ApplicationName** is typically the title of the application, but does not include a version and no claim to unambiguity.

The **ApplicationGUID** should uniquely identify the application, but again without quoting a more precise version (here, a separate GUID may be entered for each major version, if appropriate). The GUID should have the format of "59303B9F-B7E1-42bf-857A-9F6574A37433".

The **ApplicationVersion** should hold a version detail as precise as possible; typically, this is a character string in the format of "4.21.2471.40371".

[8] For UNICAM import or export respectively, the first 12 nos. of GenericOrderInfo entries (if any) will be mapped onto the 12 nos. lines of Construction project / Construction site / Owner.

3.4 OrderInfo, OrderInfoVal

[Designed in cooperation with Precast Software Engineering GmbH, Salzburg (A).]

The **OrderInfo** table contains additional order block related entries. The type of entries is application-dependent and is designated by the **Type** attribute. For instance, wide-ranging project guidelines can be described here.

The *OrderInfo* Table just has **Code** field: it is an alphanumerical Code in the sense of an article code or another type of matchcode. The meaning of the **Codes** is defined only within a given *OrderInfo*. I.e. different *OrderInfo* types have individual systematics for the **Code** field.

The **OrderInfoVal** table is a sub-table of *OrderInfo*. Each *OrderInfoVal* item has a **Type** attribute that determines the meaning of the entry. The **V** attribute contains an alphanumerical string and is the value itself. Additional (optional) attributes are the **Culture** attribute (language) and the **U** attribute (Unit) [9].

The standard defines the following *Type* attributes[10]:

- OrderInfo Type = "**DrawingTemplate**":A „DrawingTemplate“ item indicated the project template to be used when creating the detailing drawings in CAD. This is used when the ERP system is controling the CAD drawing creation process and the ERP is thus ordering drawings from the CAD. The *OrderInfo* entry itself specifies only the *Code* which identifies the drawing template. Related *OrdeInfoVal* items may specify more details:
  - OrderInfoVal Type = "**LocalizedName**": A localized name.
  - OrderInfoVal Type = "**Description**": A localized description text.
- OrderInfo Type = "**AccountingPosition**":
  An „AccountingPosition“ entry describes an accounting position of the order. By transferring the available accounting positions from ERP to CAD, the CAD designer can directly associate the correct accounting positions to the elements he is drawing.
  The *OrderInfo* item itself contains only the *Code* of the accounting position. (It is the ID if the ERP accounting position and will then be referenced in the field *Product.ItemPosition* when elements are transmitted from CAD to ERP). Further details of the accounting position may be specified in *OrderInfoVal* subentries:
  - OrderInfoVal Type = "**LocalizedName**": A localized name.
  - OrderInfoVal Type = "**Description**": A localized description..
  - OrderInfoVal Type = "**PositionInBOQ**": Designation of the releted position in the BOQ.
  - OrderInfoVal Type = "**PositionInOrder**": Designation of the releted position in the order.
  - OrderInfoVal Type = "**PSE.AccPos.XYZ**": An additional accounting position information as it is provided by a *Precast Software Engineering* systemen with field name XYZ.

Example:

```
<OrderInfo Type="AccountingPosition">
  <Code>U1.F1.Stairs</Code>
  <OrderInfoVal Type="LocalizedName" V="Geschoss 1, Treppen" Culture="de"/>
  <OrderInfoVal Type="LocalizedName" V="Floor 1, Stairs" Culture="en"/>
  <OrderInfoVal Type="PSE.AccPos.IncludedReinforcement" V="0.23" U="kg/m²"/>
</OrderInfo>
```

3.5 Product (Element)

For double walls, the **Product** is the complete wall; it includes both wall shells.

[9] For unit specification, the same applies to the unit of the ElemenInfo entry.
[10] In addition, application specific types can be defined. In order to avoid naming conflicts with future extensions oth the standard, the application specific type names should start with "I_ ".

The information of UNICAM-SLABDATE are partly accommodated in the Product segment, and partly in the Slab segment respectively.

3.5.1 ElementNo

[Replaces *Slab.SlabNo*, see section 3.7.7]

Number or name of the precast element.

3.5.2 ProductType

The ProductType is roughly equivalent to the Product specifications of the UNICAM-HEADER.

Any number of type details may be used here. The following types are fixedly defined:

- **00:** element and roof slab
- **DW:** double wall
- **03:**prestressed slab
- **04:** isolating slab
- **05:** facade element
- **06:** solid floor
- **07:** silo slab
- **08:** constructional part
- **09:** solid wall
- **10:** hollo core slab
- **11:** sandwich panel
- **16:** brick floor
- **19:** brick wall
- **NW:** zero wall
- **TW:** thermo wall
- **36:** light-weight concrete full-thickness floor
- **39:** light-weight concrete solid wall
- **GML:** Generic Multi Layer element
- **BM:** beam
- **CL:** column
- **ST:** stairs
- **MD:** module
- **DT:** double-tee
- **InSitu:** In-situ concrete element. Elements of this type are not precast items; instead, they are cast in situ concrete parts of the building. Typical example: upper layer of floors.

Please note: If different ProductType values occur, the Order block will be split up into several HEADER blocks for UNICAM export (in UNICAM, the ProductType appears in the HEADER block).

Please note: Here, the types 01 or 02 are *not* used for the double wall, but rather the type **DW**; in the PartType of the Slab, a differentiation will then be made between wall half #1 and wall half #2.

Virtual Element

If no *ProductType* is specified (or an empty string is given), the element is a **Virtual Element**: such an element is used to transfer additional project-related information that cannot be assigned to any

real element. Examples are project-related additional services that are listed for billing reasons or additional materials that cannot be assigned to any individual element.

3.5.3 PieceCount

Target quantity.

3.5.4 Data transfer for double walls: TurnWidth, TotalThickness, DoubleWallsGap

The following definition will apply to double walls (or other double-wall-like elements with separately produced slabs):

The single slabs will be represented in their rotational position as they are on the single pallets prior to turning; that is to say, the first wall half (= the wall half to be turned) must be shown turned in relation to the finished product.

That is to say, the representation of the single slabs shows the production-engineering situation, but not the finished product. The double wall is thus described in its *opened* form.

In order to know how the finished element will look like it is not good enough to consider the single slabs, but one must also know how the two wall halves are joined together. This additional information is provided by setting **TotalThickness** and **TurnWidth**[11].

The **DoubleWallsGap** describe the resulting gap between the two parts after turning the first part. Actually, this value is redundant as it results from the other given geometric details. It isn't thus a fundamental value and it is just provided for compatibility reasons.

A more formal and mathematical description of the special geometrical treatment of double walls is given in section 3.7.4.

Please note for the import of UNICAM data from CAD systems:

If UNICAM data are applied from a CAD system, an assumption must be made regarding the double wall coordinates used. Depending on the CAD system or the setting of the CAD system, there may be fairly different definitions which, however, are not obvious from the UNICAM file. The conventions used most frequently are as follows:

a) The double wall halves are transferred in an **open representation**, that is to say like in PXML. As there is no *TurnWidth* field in UNICAM, at this point, we must know what pallet width is being assumed in the CAD system; failing this, it will not be possible to correctly interpret the data. The best choice would be, to use *TurnWidth*=0 for the open representation in UNICAM; unfortunately, this setting is hardly ever used.

b) The second wall half is transferred similarly as in case a), and thus similarly as in PXML. The first wall half, on the other hand, will be transferred in a **pseudo-turned form**: the data is obtained from the open representation by turning by 180° around the axis *Y*=*TurnWidt*/2 (="turning in") and subsequently flipping on the XY-plane.

c) The **inverse pseudo-turned form** is similar to case b), but the double wall is viewed from below here, i.e. looking at the exterior of the second wall half. When importing these data, we have to rotate the whole wall through 180° around the Z-axis, plus we then have to pseudo-turn the second wall half.

The *pseudo-turned form* and the *inverse pseudo turned form* are linked by a transformation ***Myz***:

$$M_{yz} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This transformation describes a mirroring on the *YZ* plane. This transformation can be used to transform the *pseudo turned form* into the *inverse pseudo turned form* and vice versa (in fact, *Myz* is an involutory matrix, i.e. it is identical with its inverse). Moreover, the *inverse pseudo turned form* can be converted to the correct *open form*, by first applying the "wrong" transformation (i.e. the one that would be appropriate for the *pseudo turned form*) and subsequently applying *Myz*. The same is true when converting from *pseudo turned form*. Therefore, *Myz* is a quite simple and universal way of switching between the two forms.

[11] The choice of *TurnWidth* is actually free as long as it is considered correctly when setting *Slab.Y* and *Slab.ProdY*. It is best to set *TurnWidth* to 0.

3.5.5 Comment

Any comment regarding the product.

3.5.6 RotationPosition

An angle in degrees, which indicates how the element is rotated relative to the original CAD drawing (The rotation is typically performed during the pallet assignment process and is motivated by production optimization considerations).

The rotation of the element takes place around the Z-axis in positive rotation direction (i.e. counter-clockwise). Since the first part of a double wall is laid out in turned mode, its rotation angle has to be counted in negative rotation direction.

Best practice is to not rotate the element and thus not use the *RotationPosition* field. Instead of rotating the element, the *Slab.ProdRotX/Y/Z* fields should be used[12].

3.5.7 Stacking Information

StackNo

[Replaces *Slab.NumberOfMeansOfTransport*, see section 3.7.7]

Identifier of the transport stack.

The *StackNo* is a text field and may even contain structured values. For instance, two stacks, which are to be transported together, can be held together via a designation structure of the form "17.1" and "17.2".

StackID

Unique database ID of the stack. This ID identifies the stack across all orders.

StackingSequence

[Replaces *Slab.TransportSequence*, see section 3.7.7]

Sequence within the transport stack.

The element with the lowest value has to be placed first on the stack[13].

StackungLevel

[Replaces *Slab.PileLevel*, see section 3.7.7]

Pile level within the transport stack. This field is of interest solely if there might be more elements on one pile level. Otherwise the *StackSequence* indication should suffice.

Moreover this field is mainly just an informal information for the operators. An automatic stacking machine would definitely use the exact stacking coordinates (see below).

[12] One difficulty in using *RotationPosition* is that you have to "guess" the position of the original (non-rotated) element in order to correctly interpret the project coordinates and the stack coordinates, because project and stack coordinates refer to the original (non-rotated) element. However, the rotational/displacement performed in CAD is not clearly known, because only the rotation is specified in *RotationPosition*, but not the displacement. The exact rule for the back calculation therefore depends on the CAD. In many cases the back transformation has to be done by first rotating back the element and then moving it such that its enveloping rectangle (i.e. the smallest x/y-rectangle containing all lot-*Outlines*) is positioned at (0, 0).

[13] Most often the PXML *StackingSequence* field corresponds exactly to the UNICAM field "Assembly sequence in Transport pile number". However, the UNICAM specification stated the latter should be the *inverse* stacking order and thus *inverse* to the PXML *StackingSequence*. But most CAD systems doesn't follow this specification and use the UNICAM field as a stacking sequence field, i.e. exactly the same way as the PXML *StackingSequence*.

StackingX/Y/Z/Angle/RotY/RotX

[Replaces *Slab.PositionInPileX/Y/Z* and *Slab.AngleInPile*, see section 3.7.7]

Position coordinates of the element on the transport stack.

The position of the element on the stack results by applying the following transformations in the sequence given below:

1) Rotation *StackingAngle* around the absolute Z axis.
2) Rotation Stacking*RotY* around the absolute Y axis.
3) Rotation *StackingRotX* around the absolute X axis.
4) Translation by *StackingX/Y/Z*.

(All angles are given in degrees).

3.5.8 Project Coordinates

[Replaces *Slab.ProjectCoordinates*, see section 3.7.7]

The project coordinates describe the position of the element within the real-world building.

A set of 3 points P_1, P_2 and P_3 has to be given:

- P_1 is the point within the building where the precast element has to be positioned.
- The vector $\overrightarrow{P_1P_3}$ describes the axis within the building where the X-Axis of the element is mapped to.
- The vector $\overrightarrow{P_1P_2}$ describes the axis within the building where the Y-Axis of the element is mapped to[14].

Note:

The project coordinates describe a rotation and displacement in space. The somewhat peculiar representation using the 3 mentioned points has been chosen for compatibility with UNICAM. However, it is simpler and more common to describe such transformation by a rotary matrix R and displacement vector $\vec{t}$. A point of the element representation is then transferred to the building coordinates by first rotating it with R and then moving with $\vec{t}$.

For the given R and $\vec{t}$, the points *P1*, *P2*, *P3* can be calculated as follows (note, however, that *P1*, *P2*, *P3* are not uniquely determined):

$$P_1 = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}, \qquad P_2 = \begin{pmatrix} R_{12} + t_x \\ R_{22} + t_y \\ R_{32} + t_z \end{pmatrix}, \qquad P_3 = \begin{pmatrix} R_{11} + t_x \\ R_{21} + t_y \\ R_{31} + t_z \end{pmatrix}$$

If, on the other hand, P_1, P_2, P_3 are given, R und $\vec{t}$ can be calculated as follows:

$$\vec{t} = \begin{pmatrix} P_{1x} \\ P_{1y} \\ P_{1z} \end{pmatrix}$$

$$\begin{pmatrix} R_{11} \\ R_{21} \\ R_{31} \end{pmatrix} = \left(\overrightarrow{P_1P_3}\right)^0, \qquad \begin{pmatrix} R_{12} \\ R_{22} \\ R_{32} \end{pmatrix} = (\overrightarrow{P_1P_3} \times \overrightarrow{P_1P_2}) \times \overrightarrow{P_1P_3})^0, \qquad \begin{pmatrix} R_{13} \\ R_{23} \\ R_{33} \end{pmatrix} = \begin{pmatrix} R_{11} \\ R_{21} \\ R_{31} \end{pmatrix} \times \begin{pmatrix} R_{12} \\ R_{22} \\ R_{32} \end{pmatrix}$$

Here the following notation for unit vectors is used:

$$(\vec{a})^0 := \frac{\vec{a}}{|\vec{a}|}$$

[14] This definition requires $\overrightarrow{P_1P_2}$ to be orthogonal to $\overrightarrow{P_1P_3}$. While this condition might be usually given, it cannot be guaranteed. A more general (and thus mathematically more correct) approach is to define the Y-Axis as being given with the double cross product $(\overrightarrow{P_1P_3} \times \overrightarrow{P_1P_2}) \times \overrightarrow{P_1P_3}$.

3.5.9 Supplementary Product Information

AdditionInfo

[Replaces *Slab.ProductAddition*, see section 3.7.7]
Additional information related to the product type.

UnloadingInfo

[Replaces *Slab.TypeOfUnloading*, see section 3.7.7]
Information on how to handle the element at unloading (tilting).

TransportInfo

[Replaces *Slab.MeansOfTransport*, see section 3.7.7]
Information related to the means of transport or a transport related grouping of elements.
The transport container type may be specified here.

ItemPosition

[Replaces *Slab.OrderPosition, Slab.ProductGroup, Slab.SlabType* and *Slab.ItemDesignation*, see section 3.7.7.
More specifically:
Product.ItemPosition = Slab.OrderPosition@Slab.ProductGroup@Slab.SlabType@Slab.ItemDesignation]
Item information typically related to ERP system references.
The value of the field can have a multi-part structure (but does not have to). The individual parts must then be separated by @ signs (example: "123 @ AFX5").
A possible use of this field is the specification of the ERP accounting position to which the element is assigned. Typically, reference is made here to ID values that have previously been transferred to the CAD via *Order.OrderInfo* entries (of type *AccountingPosition*).

3.6 ElementInfo

[Designed in cooperation with Precast Software Engineering GmbH, Salzburg (A).]
The *ElementInfo* table contains additional entries for the *Product* block. The type of entries depends on the application and is distinguished by the **Type** attribute.
The *ElementInfo* entries are mainly used for transferring data from CAD to ERP. Typically, summarizing quantities and accounting articles are needed in this context, without going into too detailed geometrical information. Accordingly, the CAD to ERP data transmission is often only up to the depth of the *Product* level; details of the underlying structures are considered in a summarizing manner in *ElementInfo* items.
ElemenInfo entries can also be used for transferring extra data to the production system. So, for instance, a production master computer might need additional accounting information in order to be able correctly display production reports and target production cycle times.
The **Inventory** flag indicates whether the element info entry is provided for inventory management, too. The supplied quantities are then physically accurate and can be used for warehouse management tasks. If the **Inventory** field is not set, or is set to *false*, the related quantities might describe consolidated amounts, which are rather intended for accounting purposes.

3.6.1 Fields of ElementInfo entries

The exact meaning of each field is dependent on the *Type* attribute. But basically their meaning can be summarized as follows:

- **Code:** an alphanumeric code like an article code or another type of "match code". The PXML standard does not specify values for the *Code*. That is, the actual codes are system-specific and agreed between the CAD and ERP systems[15].
- **Description:** A free textual description or designation.
- **ObjectID:** Unique identification of the object, to which the *ElementInfo* enty is referred to. It is also possible to have several *ElementInfo* entries referring the same object, thus having the same *ObjectID*. A such way multiply referred underlying object may also be referred from within different *Product* blocks.
- **PieceCount:** An integer value multiplier for the entire *ElementInfo* item. If the value is not specified, usually a quantity of 1 is assumed. An *ElementInfo* entry with a *PieceCount* value of *n* is to be considered as equivalent to *n* identical *ElementInfo* items with *PieceCount* 1.
- **Val1, Val2:** Quantity or size values. The exact meaning and the underlying unit are *Type*-dependent. For the calculation of total quantities these values have typically to be multiplied by *PieceCount*.
- **Unit:** unit of measurement for the values of *Val1* and *Val2*. The standard defines the folowing units: ***n*** (= piece count), ***mm***, ***cm***, ***m***, ***km***, ***m²***, ***m³***, ***L*** (=Liter), ***kg***, ***t***, ***EUR*** (as well as all other currency codes according ISO 4217). If no measurement unit is specified, it can be assumed that SI basic units are used (m, m², m³, kg). Composite units can be specified by use of "*" and "/", e.g. **kg/m²**, **N*m**.
- **Details:** Depending on the *Type* attribute value, far-reaching details can be entered here. In complex cases a whole structured data object can be described by providing a complete XML string.

In addition, application-specific fields are allowed (I_P_ fields).

3.6.2 Predefined ElementInfo types

The following *ElementInfo* Type values are defined in the PXML standard[16]:

- ElementInfo Type = "**AccArea**":
  Accounting area of the element.
  The value provided here follows agreed accounting conventions and may differ from the real geometrical surface are.
  Val1: Value of the area in m^2.
- ElementInfo Type = "**AccAreaExt**":
  Extended accounting area of the element.
  This type is similar to "AccArea", but additional areas are included, typically the area covered by iron projections.
  Val1: Value of the area in m^2.
- ElementInfo Type = "**AccAreaAdd**", "**AccAreaExtAdd**":
  Additional accounting area of the element.
  Here it is possible to specify another additional accounting area that is not yet included in *AccArea* or *AccAreaExt.*
  Val1: Value of the area in m^2.
- ElementInfo Type = "**EffortArea**":
  Effort area of the element.
  A value given in m^2 that estimates the total production costs of the element. The value is

[15] This "freedom" in the used of the *Code* field is the main difference to the *Type* field, as the latter has a much more predefined meaning.
[16] In addition, additional application-specific types may be defined. In order to prevent naming conflicts with future PXML extensions, the application specific type values should start with prefix "I_P_".

typically calculated by using real or accounting area figures as a basis and adding some virtual area values to it that represent additional production efforts. The resulting total area is then somehow proportional to the production costs. The compressively produced *EffortArea* is a good indicator for the plant's production output.
Val1: Value in m^2.

- ElementInfo Type = "**ArchitecturalPart**":
  Architectural unit (e.g. wall or floor) that is superordinate to the element.
  Elements that belong to the same parent architectural unit should have a common *ObjectID*
  Code: This can be used as a distinction between different types of architectural units
  Val1: Total area of the architectural unit in m^2 (accounting value, typically including cutouts).
  Val2: Partial area (in m^2) of the element, belonging to the precast element in question (accounting value, typically including cutouts). This value is potentially identical with *AccArea*; then it can also be omitted.
- ElementInfo Type = "**Outlet**":
  Cutout in the project's building.
  Such a cutout may span over several precast elements and may therefore be referred from within different *Product* blocks. It is therefore crucial to identify the underlying project's cutout by specifying the *ObjectID*[17].
  Val1: Total area of the project cutout in m^2.
  (Alternatively, a number of recesses can also be specified in *Val1*. But then *Unit* must be set to "*n*").
  Val2: Partial area of the cutout (in m^2), belonging to the precast element in question.
  ObjectID: Unique ID of the cutout in the CAD project. If all cutouts of a project have to be summated, all those with identical ObjectID have to be counted only once.
  If the cutout should also be assigned to an architectural unit (*ArchitecturalPart*), the *ObjectID* should have the following structure:
  abc.xyz
  Here *abc* is the *ObjectID* of the *ArchitecturalPart* and *xyz* is an identification of the cutout within the *ArchitecturalPart*.
  Code: This can be used as a distinction between different cutout types[18].
- ElementInfo Type = "**MountPart**":
  Mounting part, fitting.
  As well as described for "*Outlet"*, even an object referred in *MountPart* can belong to more than one precast element. Again, the ObjerctID is used for defining this connection.
  ObjectID: Unique ID of the mount part in the CAD project. If mount parts of a project have to be summated, all those with identical ObjectID have to be counted only once
  If the mountpart should also be assigned to an architectural unit (*ArchitecturalPart*), the *ObjectID* should have the following structure:
  abc.xyz
  Here *abc* is the *ObjectID* of the *ArchitecturalPart* and *xyz* is an identification of the mountpart within the *ArchitecturalPart*.
  Code: Article code of the mount part.
  Val1, Val2: Value parameters of the mount part. The exact meaning of those values depends on the mount part type, hence on the *Code* field.

[17] Most often big cutouts has to be subtracted for accounting purposes. In some implementations the distinction between "big" and "small" cutouts is already done in CAD, in other implementations it done in ERP. In the former case the CAD may transfer only "big" cutouts to ERP, in the latter case all cutouts need to be transferred.
[18] If, for instance, the CAD system makes a distinction between big and small cutouts but all cutouts (big and small ones) are transferred as "*Outlines*", the *Code* field may be used to specify this distinction.

- ElementInfo Type = "**SteelBar**":
  Round steel reinforcement.
  Code: An article or match code that typically identifies a wire type (with its wire diameter, steel quality, steel supplier, etc.).
  Val1: Weight in *kg*. Depending on the value of the *Inventory* field, these are a real warehouse management relevant quantities, or just accounting values. If both aspects are of interested, two distinct *ElementInfo* entries have to be provided.
  Details: Here, details for production processing may be provided (e.g., specify that a reinforcement layer may be produced via flexible mesh welding machine or by means of a single bar handling robot)[19].
- ElementInfo Type = "**LatticeGirder**":
  Lattice girder reinforcement.
  Code: An article or match code that typically identifies a lattice girder type
  Val1: Weight in *kg*. (Real or only for accounting purposes, depending on the **Inventory** field value).
- ElementInfo Type = "**StdMesh**":
  Standard mesh reinforcement.
  Code: An article or match code that typically identifies a standard mesh type
  Val1: Area in m^2. (Real or only for accounting purposes, depending on the **Inventory** field value).
- ElementInfo Type = "**BentMesh**":
  Bended standard mesh.
  Code: An article or match code that identifies a bent mesh type to the extent that pricing can be calculated via the provided *Val*-fields.
  Val1: Weight in *kg* (Real or only for accounting purposes, depending on the **Inventory** field value).
- ElementInfo Type = "**Cage**":
  Reinforcement cage.
  Code: An article or match code identifying the cage type.
  Val1: Weight in *kg* (Real or only for accounting purposes, depending on the **Inventory** field value).
- ElementInfo Type = "**Concrete**":
  Concrete lot.
  Code: An article or match code which identifies the concrete mixture.
  Val1: Volume in m^3 (real or an accounting quantity, see *Inventory* field).
- ElementInfo Type = "**ErpProjectUnit**":
  ERP project unit assigned tot he element. The values typically refer to the IDs that have been transmitted via *Order.ErpProjectUnit* from ERP to CAD.
  Code: The Code of *ErpProjectUnit*.
- ElementInfo Type = "**PriceQty**":
  A size or quantity indication which is relevant for the price calculation. (It may be a surface, a volume, a number of pieces, or whatever else can be used as a multiplier for a price calculation. It is even possible to specify the price itself).
  Val1: value of the price defining amount.
- ElementInfo Type = "**PlanningQty**":
  A size or quantity indication which is relevant for production planning. This quantity is often used for determining production capacity, as well. Sometimes the unit used for the *PlanningQty* differs from the unit used in *PriceQty*. There are cases, for instance, where

[19] Alternatively, a processing mode could be defined via the *Code* field, as well.

PriceQty is given in *m³* and *PlanningQty* is given in number of pieces.
Val1: value of the planning relevant amount.

- ElementInfo Type = "**TransportInfo**":
  Miscellaneous information about the means of transport. Typically, a type identifier is specified in the *Code* field (container type) and names and dimensions can be specified in *ElemInfoVal* entries.

3.6.3 ElemInfoVal

When the fields defined in the *ElementInfo* entry are not sufficient, additional values can be added by using *ElemInfoVal* entries.
The following table describes the *ElemInfoVal* types defined by the standard.

Type	Description	Example
Len	Length	1234.5
Width	Width	456.8
Height	Height	123.4
Qlty	Quality (ex. material quality)	Q123
Name	Designation	abcd

The String could also contain other internal fields which should have the Prefix "I_".

Example MountPart:

```
<ElementInfo Type="MountPart">
  <Code>FK77</Code>
  <ElemInfoVal Type="Height" V="100"/>
  <ElemInfoVal Type="Width" V="225.3"/>
  <ElemInfoVal Type="Len" V="310.1"/>
  <ElemInfoVal Type="Qlty" V="Q123"/>
</ElementInfo>
```

3.7 Slab (Element Part)

3.7.1 PartType

We can use any type details. The meaning of PartType will be a function of the higher-ranking ProductType.
The following PartType values are fixedly defined for the DW Product type:

- 01: double wall 1st stage, and
- 02: double wall 2nd stage

In addition to the main parts of an element, there are also supplementary parts, such as closing plates. The code 05 is intended for these:

- 05: Supplementary part

3.7.2 Geometric Slab Placement (X/Y/Z, RotX/Y/Z)

These fields describe translation and rotation that are used for geometrically placing the slab (= element part) within the element.
The sequence of the placement operations is as follows[20]:

[20] The described sequence has to be observed strictly. The sequence may look somewhat unusual but is motivated by compatibility considerations.

1) Translation (*X*, *Y*, *Z*)
2) Rotation *RotZ* around the absolute Z axis.
3) Rotation *RotY* around the absolute Y axis.
4) Rotation *RotX* around the absolute X axis.

The translation offsets are in mm and the rotation angles are in DEG.

3.7.3 Slab Production Directives (ProdX/Y/Z, ProdRotX/Y/Z)

When transferring data from CAD, the *Slab.X/Y/Z/RotX/RotY/RotZ* values are only used to determine the position of the element parts (wall halves) relative to each another. If, in addition to this, the production placement of the element parts should be specified, some more fields are required: **ProdX**, **ProdY**, **ProdZ**, **ProdRotX**, **ProdRotY**, **ProdRotZ**. The position on the production pallet is given by applying the following transformations in the order listed here:

1) Rotation *ProdRotX* around the absolute X axis.
2) Rotation *ProdRotY* around the absolute Y axis.
3) Rotation *ProdRotZ* around the absolute Z axis.
4) Translation (*ProdX*, *ProdY*, *ProdZ*)

In total, the position of an element part on the production pallet results from the following subsequent transformations being executed one behind the other:

1) *Slab.X/Y/Z*
2) *Slab.RotZ*
3) *Slab.RotY*
4) *Slab.RotX*
5) *Slab.ProdRotX*
6) *Slab.ProdRotY*
7) *Slab.ProdRotZ*
8) *Slab.ProdX/ProdY/ProdZ*

3.7.4 Geometric Placement and Production Directives for Double Walls

The **double-wall elements** (i.e. the product types **DW**, **NW**, **TW**) represent a special case. For these elements, historical reasons imply that the slab placement of the first wall half (= the wall half to be turned) is essentially determined by *Product.TurnWidth*/*TotalThickness*.

More precisely, the effective geometric placement and production directive values of wall half 1 are calculated as follows:

- $Slab1.X_{Resulting} = Slab1.X$
- $Slab1.Y_{Resulting} = Slab1.Y - Product.TurnWidth$
- $Slab1.Z_{Resulting} = Slab1.Z - Product.TotalThickness$
- $Slab1.RotZ_{Resulting} = Slab1.RotZ$
- $Slab1.RotY_{Resulting} = Slab1.RotY$
- $Slab1.RotX_{Resulting} = Slab1.RotX + 180°$
- $Slab1.ProdRotZ_{Resulting} = Slab1.ProdRotZ$
- $Slab1.ProdRotY_{Resulting} = Slab1.ProdRotY$
- $Slab1.ProdRotX_{Resulting} = Slab1.ProdRotX + 180°$
- $Slab1.ProdX_{Resulting} = Slab1.ProdX$
- $Slab1.ProdY_{Resulting} = Slab1.ProdY + Product.TurnWidth$

- $Slab1.ProdZ_{Resulting} = Slab1.ProdZ + Product.TotalThickness$

This calculation rule is just a formal algorithm for what has already been described for double walled elements in section 3.5.4.

Note: It must be emphasized that these special rules apply only to the half slab 1, i.e. to the slabs with *PartType* "1" or "01" (and only for product types DW, NW, TW). If you want to avoid this special condition and uses explicit placement specification instead, just avoid setting the *PartType* to "1" or "01". If explicit placement specification is used, the *PartType* may be left empty, since the role of the single slabs can be deduced by their *RotX* value. Alternatively, you could use other *PartType* values, such as "P01".

3.7.5 Various Slab Information

There is a compilation of various pieces of slab information, which, in detail, corresponds to the respective fields in the info block of the UNICAM-SLABDATE.

Some of these information items are redundant as they can be computed from other variables. These fields are mainly present for UNICAM-compatibility reasons. This is especially true for the fields **MaxLength**, **MaxWidth**, **IronProjectionLeft**, **IronProjectionRight**, **IronProjectionBottom**, **IronProjectionTop**: in addition of being fully redundant, these fields suffer from the defect of not being invariant when the slab is rotated (i.e. they would have to be adjusted upon rotation of the slab). It is therefore recommended to try to not use these fields. Instead, the underlying geometric detail information should be used to calculate the respective values.

ReforcemInfo, merely for compatibility with UNICAM, includes the details of the UNICAM-REFORCEM Info block.

The **Generic Slab Info** fields are available for freely definable additional information

3.7.6 Multi-Layer Elements

Several concrete layers (lots) may be specified in any one Slab. Or, an element part could be composed of several layers each of which has its own reinforcement or its own mount parts (according to UNICAM-LAYERS).

In PXML, such multi-layer elements are implemented by setting the *Layer* values within the *Outline*- and *Steel*-sections.

3.7.7 Legacy Slab Fields

The fields listed below were originally defined on *Slab* level (for historical reasons motivated mainly by compatibility with UNICAM). However, the information they contain is on *Product* level, i.e. an information on element level (as against to slab level).

The implementation of *PXML-Delegate Files* made it necessary to bring these fields to where they really belong, i.e. to the *Product* level. These *Slab* fields were therefore replaced by the corresponding *Product* fields.

- *Slab.SlabNo* → *Product.ElementNo*
- *Slab.NumberOfMeansOfTransport* → *Product.StackNo*
- *Slab.TransportSequence* → *Product.StackingSequence*
- *Slab.PileLevel* → *Product.StackingLevel*
- *Slab.PositionInPileX* → *Product.StackingX*
- *Slab.PositionInPileY* → *Product.StackingY*
- *Slab.PositionInPileZ* → *Product.StackingZ*

- *Slab.AngleInPile* → *Product.StackingAngle*
- *Slab.ProjectCoordinates* → *Product.P1X/P1Y/P1Z/P2X/P2Y/P2Z/P3X/P3Y/P3Z*
- *Slab.ProductAddition* → *Product.AdditionInfo*
- *Slab.TypeOfUnloading* → *Product.UnloadingInfo*
- *Slab.MeansOfTransport* → *Product.TransportInfo*
- *Slab.OrderPosition,*
  Slab.ProductGroup,
  Slab.SlabType,
  Slab.ItemDesignation, → *Product.ItemPosition*

The resulting obsolete fields are referred to herein as **Legacy Slab Fields**. Conceptually, they would be dispensable, but they need, in fact, to be continued for backward compatibility reasons. More precisely, the following backward compatibility strategy is recommended:

1) Applications using PXML as their internal data structure should no longer use (internally) the *Legacy Slab Fields*; instead, they should use the related *Product* field.
2) A process reading a PXML file should read the *Product* fields, if available, and should fall back to the *Legacy Slab Fields*, if the related *Product* fields haven't been set. (More specifically, if a field isn't set on *Product* level, the value of the first *Slab* is taken, where the related field is set to a non-empty value).
3) A process writing a PXML file can restrict itself to write the *Product* fields if and only if it can be sure that the reading process will be able to read those fields (i.e. the reading software already implements the "new" schema with those fields at *Product* level). In case of doubt, the writing process should adopt a maximum compatibility policy and also write all *Legacy Slab Fields*[21].

3.7.8 Simplified geometry representation

Many legacy systems do not process the following values:

- *Slab.RotX/RotY/RotZ*
- *Slab.ProdX/ProdY/ProdZ*
- *Slab.ProdRotX/ProdRotY/ProdRotZ*
- *Outline.RotX/RotY/RotZ*
- *Steel.RotX/RotY/RotZ*
- *Steel.ProdX/ProdY/ProdZ*

In the simplified geometry representation, we strive to reduce these values to 0 to stay compatible with such legacy systems. In order to minimize information losses, the values mentioned fields are incorporated in other fields as much as possible[22].

---

[21] In theory a writing process could even restrict itself to write only the *Legacy Slab Fields*, without also writing the related *Product* fields. There are, however, cases where writing the *Product* level fields is indispensable, e.g. when using PXML Delegate Files or when sending data to an ERP system that doesn't need to know about *Slab* details.

[22] The major information loss occurs when Outline or Girder objects are to be rotated in the X or Y axes. For these operations it can be suggested to just transform the x and y coordinated. More specifically, if a rotation by angle φ in the X axis has to be performed, the following would be done:

$$Y := cos\varphi \cdot Y$$
$$\alpha := arctan2(cos\varphi \cdot sin\alpha, cos\alpha)$$

Here *Y* stands for all Y coordinated that apply to *SVertex* and α stands for *Girder.AngleToX*.
A similar calculation applies for a rotation in the Y axis:

$$X := cos\varphi \cdot X$$

3.8 Outline

By **Outline**, we understand a general geometric boundary. The **Type** attribute determines the type of the enclosed item:

- **lot**
- **mountpart**

Some tags of the Outline segment are merely intended for certain *Outline*-types only, and will be ignored for the other *Outline*-types.

The **Lot** *Outlines* describe a **Concrete Lot** with contours, cutouts and concrete data.

The **Mountpart** *Outlines* describe mount parts.

3.8.1 Geometric Outline Placement (X, Y, Z, RotX, RotY, RotZ)

These fields describe translation and rotation that are used for placing the *Outline* object within the *Slab*. The sequence of the placement operations is as follows[23]:

1) Translation (X, Y, Z)
2) Rotation RotZ around the absolute Z axis.
3) Rotation RotY around the absolute Y axis.
4) Rotation RotX around the absolute X axis.

The translation offsets are in mm and the rotation angles are in DEG.

Position of Installation:

In UNICAM (version 6.0 or higher), there is an indication of a **Position of installation** related to the Slab zero point. This value is redundant up to a certain degree as the position of installation is derived from the position and geometry of the mount part. (Though this is only true for known mount parts, but only those of this type can be automatically installed or placed). For this reason, there is no such position detail included in PXML.

However, the X/Y Offset of the mount part can be set such that it coincides with the position of installation (the vertex coordinates must be compensated accordingly). It is recommended to proceed in this way for data interchange with UNICAM.

3.8.2 Height

Height in mm (thickness of the concrete layer, height or depth of the mount part).

3.8.3 Name

Identifier of the item. When reading UNICAM concrete layers, the concrete layer identification is put into this field. When reading UNICAM mountparts, the mountpart designation is used.

3.8.4 GenericInfo

Freely usable informational lines.

3.8.5 MountingInstruction (only for Mountpart)

Instructions for installation. The following applies following the UNICAM Installation Identifier:

- 0 = the part is being installed,

$$\alpha := arctan2(sin\alpha, cos\varphi \cdot cos\alpha)$$

With this approach a 180° rotation in the X or Y axis becomes a reflection on the XZ or YZ plane.

[23] The described sequence has to be observed strictly. The sequence may look somewhat unusual but is motivated by compatibility considerations.

- 1 = the part is merely being drawn only,
- 2 = the part is merely being installed,
- 3 = the part is neither being drawn nor installed,
- 4 = the part is being installed into reinforcement, and
- 5 = the part is being automatically installed.

3.8.6 MountPartType, MountPartArticle (only for Mountpart)

This corresponds to the respective UNICAM definitions.

However, the *MountPartType* "**21**" is defined in a more general way as being "**void form**". Inserts of this type can be used for modelling cutouts and recesses in concrete. In this light, there are two alternative ways for defining cutouts: either by using a cutout-Shape or by using a voiding mountpart. The latter, while being a more complex approach, is much more flexible[24].

3.8.7 MountPartProperties (only for Mountpart)

This corresponds to the respective UNICAM definitions:

- **MountPartIronProjection** = rebar projection in mm. These are rebars that protrude from a mount part.
- **MountPartDirection** = The direction of the rebar projection or the orientation of the mounting part respectively.
  The angle detail is within the range of]-180°, 180°]. For export to UNICAM, this is converted to [0°, 360°].
- **MountPartLength/MountPartWidth** = length / width in mm.
  These values are optional, and are usually set to facilitate automatic trimming to size of the part. In this sense, these are the production dimensions of the mount part.
  Basically, these values are only required for mount parts the size of which is variable and not fully defined through the item ID.
  The precise meaning of these two dimensions will depend on the type of the mount part. There may be types that will merely require one of these two dimensions, or else meanings other than those of the length or width may be assigned to these two values.

3.8.8 Concrete Properties (only for lots)

- **ConcretingMode** = concreting flag.
- **ConcreteQuality** = concrete quality or grade (such as B25, for example). In addition to the actual concrete quality, further (plant-specific) information can also be specified here, e.g. "B25.Red" for a colored concrete.
- **UnitWeight** = bulkdensity in kg/dm³.
- **Volume** = target concrete volume in m³.

3.8.9 Layer

Used only for multi-layer elements; defines the layer to which the item belongs.

[24] Please note: a voiding mountpart as described here requires always to be marked as to be installed in its MountingInstruction, since a non-installation instruction would indicate that the part doesn't have a voiding effect.

3.8.10 ObjectID

The *ObjectID* enables the object to be identified by means of an ID, which is typically set by the CAD system. In contrast to the *GlobalID*, the *ObjectID* is not necessarily unique, as it is possible to link several *Outline* and/or *Steel* objects via a common *ObjectID* by declaring that way their affiliation to a common underlying physical object.

The ObjectID can also have a multi-level structure by separating partial IDs with dots. This makes it possible to create complex multi-level object hierarchies.

Example:

We consider the following objects:

A) ObjectID = “5”
B) ObjectID = “23”
C) ObjectID = “23.1”
D) ObjectID = “23.1.1”
E) ObjectID = “23.1.2”
F) ObjectID = “23.1.2”

Object A stands on its own, while objects B, C, D, E and F belong together as they form group "23".

Objects C, D, E and F also form group "23.1", which is a subgroup of group "23".

Objects E and F form group "23.1.2", which is a subgroup of group "23.1".

Using the *ObjectID*, it is also possible to connect *Steel* objects with *Outline* objects or to create complex structures in which both *Steel* and *Outline* objects occur in any nesting. A connection to *ElementInfo* objects can also be established via the *ObjectID*.

In principle, it is also possible to use the *ObjectID* to connect objects that are located in different elements. This is useful, for example, if you have large mountparts that extend over several elements.

The *ObjectID* also defines a partial order between the objects, in the sense that, for instance, "23" comes before "23.1" ([25]). This order relation can now be used to define priorities for common properties: for example, if you want to use a common article code for the group "23", you refer to the article code of B because B is the first object in this group (in the sense of the partial order mentioned).

3.8.11 Shape, SVertex

A **Shape** holds a sequence of points (**SVertex**) forming a polygon. The polygon is closed by connecting the last point to the first point (without, however, listing the first point twice).

Cutout:

If this field is set to *true*, the *Shape* object will describe a cutout, i.e. a hole in the surrounding shape.

RefHeight:

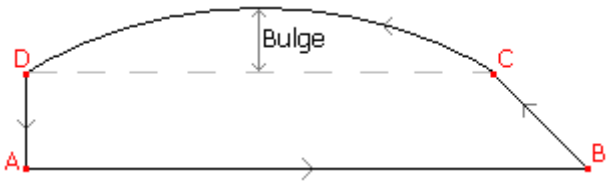
A reference z-position used in conjunction with sloped edges. See below.

SVertex:

AnSVertex describes a corner mark of the polygon (a dot in the plane).

A **Bulge** can be assigned to each *SVertex*: if this is unequal 0, the connection of the vertex to its follower vertex is an arc of the specified height (in mm).

[25] Only a partial order is clearly defined, not a total order. While there is a clear order relation between B and C, the relation between E and F can no longer be defined without additional conditions. And even the relationship between A and B can only be determined if you specify whether you are using purely alphabetical sorting or so-called natural sorting.



This figure shows a *Shape* with 4 vertices. A, B and D are to have a *Bulge=0*, whereas the value of *Bulge* for C is positive; with a negative value of *Bulge*, the arc would be curved inwards, i.e. the sign of *Bulge* indicates the arc orientation class.

Special case of merely one SVertex:

If the *Shape* is composed of merely one single *SVertex*, this will be interpreted as a *Circle*. The *SVertex* will specify the circle midpoint, and the amount of *Bulge* will be equivalent to the circle diameter (which may also be 0); the sign of *Bulge* indicates the arc orientation class.

Line Attributes:

A **LineAttribute** can be specified for each *SVertex* to which the most varied application-specific meanings can be assigned. Thus, information regarding the type of formwork may be specified via the *LineAttribute* of an element contour.

Most often, the *LineAttribut* will be a four-digit hexadecimal figure that will be interpreted as a bit field. Here, the various bits will have the following meaning:

Bit 00 [0001]: No chamfer at the bottom.
Bit 01 [0002]: Special formwork.
Bit 02 [0004]: Grouting joint.
Bit 03 [0008]: No chamfer at the top.
Bit 04 [0010]: Curvature at the bottom.
Bit 05 [0020]: Spring (formwork including a groove).
Bit 06 [0040]: Groove (formwork including a spring).
Bit 07 [0080]: Curvature on top.
Bit 08 [0100]: Clean edge.
Bit 09 [0200]:
Bit 10 [0400]:
Bit 11 [0800]: Supporting formwork (formwork designed to support a mount part).
Bit 12 [1000]: Window formwork.
Bit 13 [2000]: Contour formwork.
Bit 14 [4000]: Do not fix formwork in place.
Bit 15 [8000]:

Open polygon curve:

According to the above definition, the vertices of a *Shape* always form a closed polygon curve. An open polygon curve can only be implemented in that you go back all the way to the first point from the last point. More specifically, an *openarc* will be implemented through a Shape with 2 vertices *A* and *B* that will satisfy $A_{Bulge}=-B_{Bulge}$. If, in addition to that, the two *Bulge* values equal *0*, this will give the special case of a simple *line segment*.

Please note: open polygon curves (isolated line segments or arcs) will always have a surface area of 0. They are thus only useful for *Mountpart* Outlines; for *Lot* Outlines, open polygon curves would be interpreted as contours or cutouts with a surface area of 0, which obviously is meaningless. Hence,

a contour or cutout must never be represented as a sequence of isolated line segments, but must always be represented via polygons.

Edges with profiles:

The **Profile** field allows the exact definition of the geometric profile of the edges. The field contains a string of the form

$$z_0|p_0\ z_1|p_1\ z_2|p_2\ \dots\ z_n|p_n$$

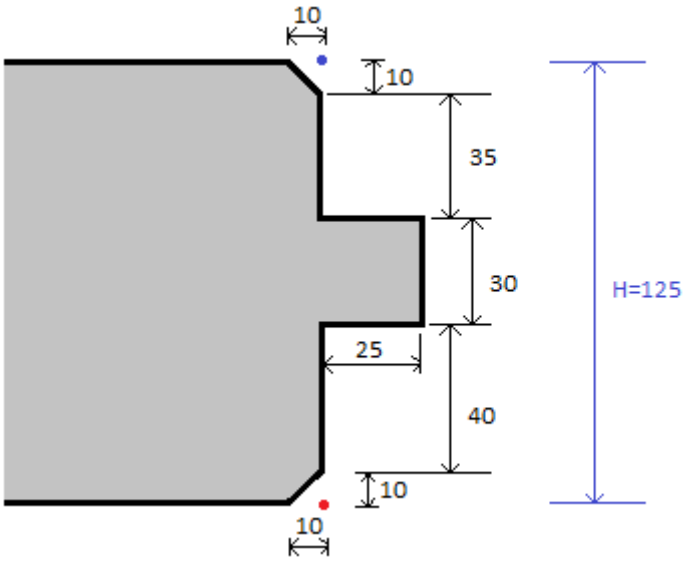
Thus, a sequence of number pairs separated by spaces is specified (the numbers themselves are arbitrary XML floating point values)[26].

The specification of the first and the last Z-value is optional: if the first Z-value is omitted, it is assumed to be 0; if the last Z-value is omitted, the thickness of the concrete layer is assumed for it. If both Z values are omitted, the profile string has the following format:

$$p_0\ z_1|p_1\ z_2|p_2\ \dots\ z_{n-1}|p_{n-1}\ p_n$$

Here the z_i are vertical values (z-values) and the p_i are horizontal values, with positive horizontal values protruding from the concrete[27]. It should be expressly pointed out that the *Profile* field refers to the concrete shape; in the case of recesses and inserts (formwork), the values are to be interpreted in a mirrored manner, i.e. a negative p_i value protrudes out from the insert.

Example:



This concrete profile is described by the following *Profile* entry:

```
<Profile>0|-10 10|0 50|0 50|25 80|25 80|0 115|0 125|-10</Profile>
```

The red and the blue dot in the drawing describe the lower and the upper reference point of the nominal edge.

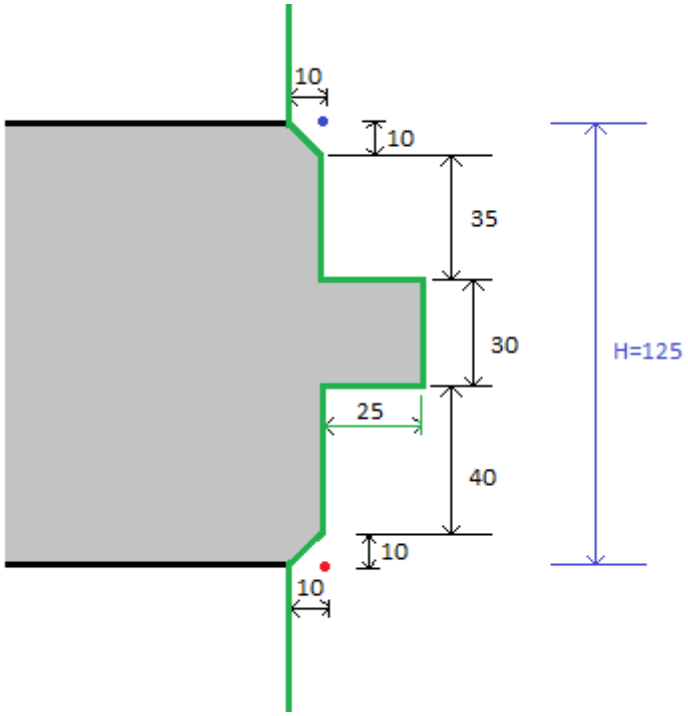
[26] The specification of the first and the last Z-value is optional: if the first Z-value is omitted, it is assumed to be 0; if the last Z-value is omitted, the thickness of the concrete layer plus *Outline* Z-offset is assumed for it. If both Z values are omitted, the profile string has the following format:

```
p0 z1|p1 z2|p2 … zn-1|pn-1 pn
```

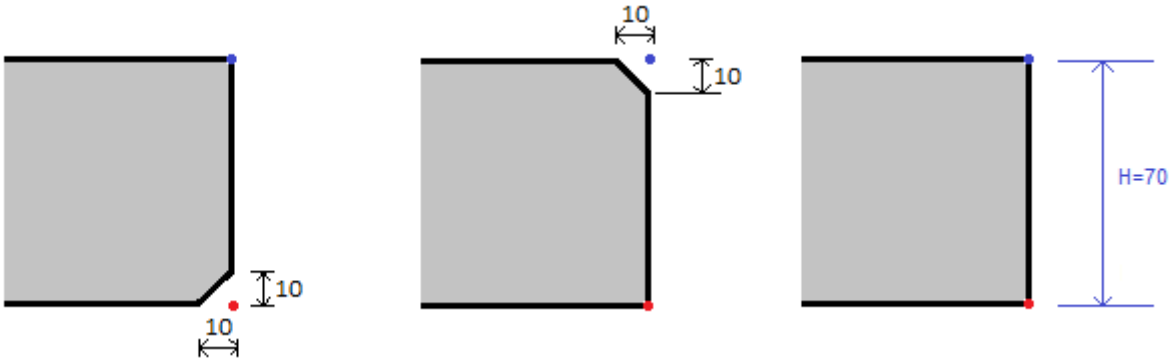
However, this short format is only supported for backward compatibility. The use of it is not recommended.

[27] The definition that positive horizontal values protrude from the concrete assumes that a clear distinction can always be made between "inside" and "outside". This is always possible for polygons with a non-null area, but polygons with less than 3 points don't have a defined orientation. In such cases, the orientation has to be assumed explicitly by assuming it positive for all concrete edges and negative for the holes in concrete and for mountparts.

The profile described in this way is extended downward and upward to infinity, resulting in a profile defined for all z values (see green line):



The following examples describes the important cases of "bottom chamfer", "top chamfer" and "no chamfer":



`<Profile>60|0 70|-10</Profile>`

`<Profile>0|-10 10|0</Profile>` `<Profile>0|0</Profile>`

The `Zi` refer to an “absolute” zero that is supposed to be at *–Outline.Z*. However, this detail is relevant for multilayer *Slabs* only (see below).

By definition, the profile string is never empty. So, if there is an empty string in the *Profile* field, it means that no *Profile* has been specified.

The *Profile* field belongs to a specific *SVertex*, but, like the *LineAttribute*, it refers to the entire edge following *SVertex*.

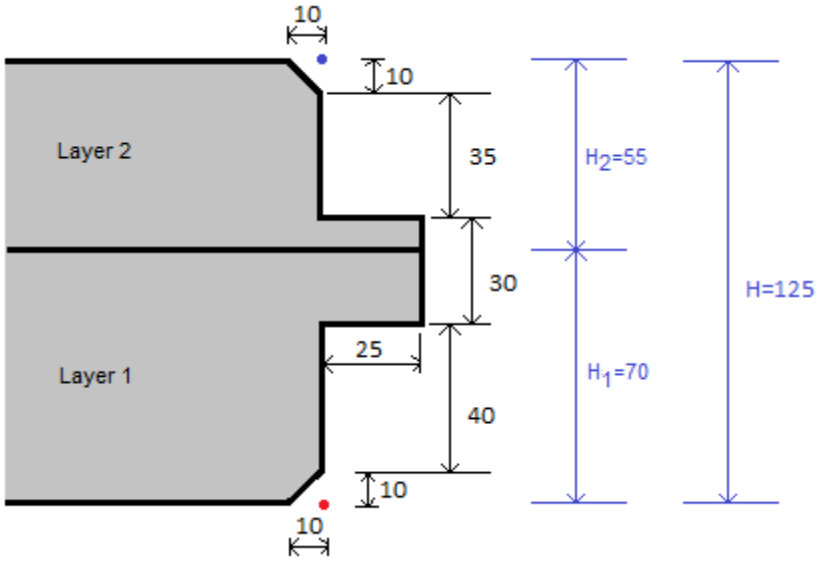
The *Profile* field provides an exact geometric description of the concrete contour. This creates a certain redundancy to line attributes because the line attribute also contains contour information such as chamfer, tongue and groove. However, the line attribute should be understood more as a production directive: it indicates how production technology is to achieve a desired contour profile. But in the end, it is up to the machine’s logic to decide whether following the indications given by the line attributes or directly deducing all from geometry.

Profile for multilayer Slabs:

In the case of multi-layer *Slabs* (sandwich elements), a *Profile* is considered that covers the entire Z-region of the *Slab* and thus extends beyond the individual concrete layer. For each layer, only that section of the profile is relevant which lies in the Z-area of the respective layer.

To be able to use the same *Profile* code for superimposed concrete layers, we define that the Z-values of the *Profile* string refer to the absolute Z-location. Consequently, to obtain local Z values (which are related to the *Outline* origin), one must subtract *Outline.Z* from all z_i

Example:



Both concrete layers should have the same *Profile* here, i.e.

```
<Profile>0|-10 10|0 50|0 50|25 80|25 80|0 115|0 125|-10</Profile>
```

If a multilayer *Slab* also contains insulating layers, these insulating layers must also match the overall *Profile*. A distinction must be made here between 2 cases:

a) Insulations that are inserted as fittings: These insulations are produced in a separate production step. Their shape is not formed by the concrete formwork. These insulations are to be represented as *mountpart Outlines* and do not have the same profile code as the concrete layers below or above them (the *Profile* code would also be interpreted inverted for *mountparts*).

b) Insulations that are poured in liquid or foam form and consequently formed by the concrete formwork: These insulations are to be represented as concrete layers, i.e. as *lot Outlines*. They then carry the same *Profile* code as the concrete layers below or above them.

Sloped edges (DX, DY, RefHeight)[28]*:*

The fields X and Y describe the vertex coordinates on the Z-position given in **Shape.RefHeight** (which is referred to the object's Z-position). Typically, *RefHeight* is 0 and thus X and Y are referred to the Z-base-position of the object.

On all other z-levels the X/Y-coordinated are given by

*X + DX * (Z – RefHeight)*

*Y + DY * (Z – RefHeight)*

[28] This section describes the representation of oblique edges by using the fields *DX* and *DY*. However, when describing the lateral concrete shape, it is recommended to use *Profile*, instead of *DX/DY*, since the *Profile* concept is simpler and more powerful for lateral shapes.

Therefore, if *DX* or *DY* are different from 0, the outline edge is not parallel to the Z-axis, but is obliquely, instead. For instance, with a *DX*-value of 1 (and *DY* = 0) the edge has an inclination of 45 degree to the z-axis.

That way, oblique prisms and pyramids can easily be represented. In order to realize more general polyhedra, a combination of several outline objects might have to be used.

It should be noted that *DX* and *DY* refer to an *SVertex*; an edge is therefore influenced by both adjacent vertices. In this aspect, the *DY*/*DY* concept differs significantly from the *Profile* concept, since in the *Profile* concept an edge is always determined only by the *Profile* value of the previous vertex[29].

Note: When two adjacent vertex-edges do not lie in a common plane (i.e. they are skewed to each other), they cannot be connected via a flat surface. In such a case, it isn't obvious how to define the joining surface between those two edges.

For a 3D graphical representation, it might be accurate enough to simply represent the joint surface by two triangles – this is the easiest way, especially when considering that 3D graphics are typically build on a composition of triangles.

But in order to have a well-defined and uniform volume calculation rule, it is necessary to give a precise definition of the joining surfaces. Here the PXML recommendation is to consider the body as a series of straight prisms with infinitesimal heights. The volume calculation is then given as follows:

$$V = \frac{Height}{12}\sum_{i=0}^{n-1}\Big((6X_{L,i} + 3X_{H,i})Y_{L,i+1} - (6Y_{L,i} + 3Y_{H,i})X_{L,i+1} + (3X_{L,i} + 2X_{H,i})Y_{H,i+1} - (3Y_{L,i} + 2Y_{H,i})X_{H,i+1}\Big)$$

Here X_L, Y_L are the coordinates on the lower side of the object and X_H, Y_H are the coordinates on the upper side of the object. The index *i* iterates through the polygon vertices; *Height* is the total height of the object.

Simplified volume calculation: For a single insulated **geometric** item the above exact volume calculation can be carried without any problems. But for sloped parts with holes or inclined parts or concrete lots partially penetrated from inclined mount-parts it might become quite complex to carry out an exact volume calculation. For most applications it will therefore be reasonable to use a simplified volume calculation, which can be achieved by assuming all DX and DY values to be 0.

3.9 Steel

The *Steel* section is a grouping of round bar irons or lattice girders.

Type attribute

The following Types are available:

- **none:** loose reinforcement, typically loose rebar and lattice girders with no assembly instructions and often no indication of their spatial arrangement.
- **mesh:** welded or otherwise fastened reinforcement with mandatory indication of the exact spatial arrangement of the bars and lattice girders.
- **cage:** Synonymous with *mesh*. The distinction between *mesh* and *cage* is only historically conditioned and has at most plant-specific significance.
- **extiron:** reinforcement provided separately that is not specified in detail. This may be in the format of loose rebars or more complex units such as mesh or cages. The lattice girder section is not used for Extiron steel blocks.

Summarizing, we can say that the difference between *"none"* and *"mesh/cage"* is that with *"none"* we just have a list of bars, while with *"mesh/cage"* we have in addition a defined assembly process on *Steel* level. In this case the *MeshType* (see below) can give further instructions for the *Steel* level processing.

[29] *Profile* and *DX*/*DY* are alternative description methods that are difficult to use together. Theoretically, however, it is possible to combine both specifications: the displacements in *X* and *Y* of *Profile* and those of *DX*/*DY* must then be added.

3.9.1 Geometric Steel Placement (X, Y, Z, RotX, RotY, RotZ)

These fields describe translation and rotation that are used for placing the *Steel* object within the *Slab*. The sequence of the placement operations is as follows[30]:

1) Translation (X, Y, Z)
2) Rotation RotZ around the absolute Z axis.
3) Rotation RotY around the absolute Y axis.
4) Rotation RotX around the absolute X axis.

The translation offsets are in mm and the rotation angles are in DEG.

3.9.2 ToTurn (only for steel mesh)

Possible values are: **true**, **false**.

This specifies whether or not the mesh is to be supplied and delivered in its turned condition (turning occurs along the longitudinal axis).

3.9.3 StopOnTurningSide (only for steel mesh)

Possible values are: **true**, **false**.

This specifies whether or not the stop side is also to be the turning side.

3.9.4 Name

Identifier for the steel block (such as the mesh identifier, for example).

3.9.5 MeshType

For *Steel* blocks of the *mesh*/*cage* type, machine production of the entire unit is possible in principle. A production process can then be specified via the **MeshTyp**:

- 0: Standard process.
- 1: Mesh bending 2D: this mesh is to be fed to a beam bending machine.
- 2: Manual or semi-automated production of a reinforcement module (e.g., by manual welding of automatically positioned stirrup series).
- 3: Mesh bending 3D: this mesh is to be fed to a single-head bending machine.
- 4: cover mesh (such as the cover of a cage, or of a solid wall reinforcement).
- 5: cover mesh 2D; same as type '1', but will be supplied and delivered together with a type '4'.
- 6: cover mesh 3D; same as type '3', but will be supplied and delivered together with a type '4'.
- 7:
- 8: loose mesh: will not be produced by the steel machine, but will be added manually (usually, such a mesh is retrieved from the mesh warehouse).
- 9: application-specific type.

In addition to the type identifier as mentioned above, *MeshType* may hold other type-specific details; the following use is recommended:

[30] The described sequence has to be observed strictly. The sequence may look somewhat unusual but is motivated by compatibility considerations.

- 8#StandardSheet:R188A
  Loose mesh, to be made by means of stocked mesh of the R188A type.
- 9#ProgressInfo8080:123#ProgressInfo8090:abc
  Here, an application-specific type is specified that will be described in more detail via two application-specific parameters (in this example, these parameters are identified as "ProgressInfo8080" and "ProgressInfo8090" respectively; the values of "123" or "abc" respectively are assigned to these parameters).

That is to say, several parameters may be specified, separated by # characters; the parameter name and parameter value are separated by a colon.

Note: The *MeshType* itself describes the basic manufacturing process, but without specifying in detail how the individual rebars are to be processed. The role of the individual rebars within the defined manufacturing process is then only determined by the *ReinforcementType* of them (see Section 3.10.2).

3.9.6 WeldingDensity (only for steel mesh)

Welding density in % (integer value).

Values between 0 and 100 may be specified. In addition, high-order digits may be used to encode additional information:

a = *WeldingDensity* **mod** *1000*

b = *WeldingDensity* **div** *1000.*

The value of *a* will determine the welding density; the value of *b* is available for additional plant-specific information.

If *WeldingDensity* has a value of *0* or *DBNull* respectively, this will stand for "Default"; depending on the plant or facility, this may be 0%, 100% or any other value.

Please note: the specified welding density is to be understood as *inner* welding density; *additional* welding points will be inserted at the edge of the mesh (see *BorderStrenght*).

3.9.7 BorderStrength

Reinforcement of the edge of the mesh. A value of *0* or *DBNull* respectively stands for the plant-specific default. A value of 1 means that the outermost row will be welded at a rate of 100%; a value of 2 means that the two outermost rows will be welded at a rate of 100%.

3.9.8 Generic Steel Info

Freely usable informational lines.

3.9.9 Steel Production Directices (ProdX/Y/Z, ProdRotX/Y/Z)

The *Steel Production Directives* fields describe a placement of the *Steel* block (typically the cage) during reinforcement, which, however, does not relate to the finished product, but is merely to be understood as a recommendation to the reinforcement production machine. That is to say, the reinforcement cage is *not* to be installed into the concrete element in its rotated or moved form but is merely to be positioned preliminarily only for ease of production. Of course, the reinforcement production machine is at liberty to follow this recommendation, or to decide independently which way around the cage is to be turned for ease of production respectively[31].

The ProdRot angles must be specified in DEG units, and the sequence of the operations is as follows:

1) Rotation *ProdRotZ* around the absolute Z axis.

[31] Please note: usually, rotation merely bears on the round-bar steel only, but not the lattice girders.

2) Rotation *ProdRotY* around the absolute Y axis.
3) Rotation *ProdRotX* around the absolute X axis.
4) Translation *ProdX/Y/Z*

Finally, the production position of the reinforcement results from the concatenation of the following operations:

1) *Steel.X/Y/Z*
2) *Steel.RotZ*
3) *Steel.RotY*
4) *Slab.RotX*
5) *Slab.X/Y/Z*
6) *Slab.RotZ*
7) *Slab.RotY*
8) *Slab.RotX*
9) *Slab.ProdRotX*
10) *Slab.ProdRotY*
11) *Slab.ProdRotZ*
12) *Slab.ProdX/ProdY/ProdZ*
13) *Steel.ProdRotZ*
14) *Steel.ProdRotY*
15) *Steel.ProdRotX*
16) *Steel.ProdX/Y/Z*

If you combine rotational and shifting operations of a step into one rotational shift *H* each, you have the following sequence:

1) H_{Steel}
2) H_{Slab}
3) $H_{SlabProd}$ → Assignment
4) $H_{SteelProd}$ → Arrangement

Or written as operator multiplication:

$$H_{total} = H_{SteelProd} \cdot H_{SlabProd} \cdot H_{Slab} \cdot H_{Steel}$$

Often $H_{SlabProd}$ is referred to as **"assignment"** (pallet assignment, bed assignment) and $H_{SteelProd}$ as **"arrangement"**.

Process sequence of assignment and arrangement: Im Arbeitsvorbereitungsprozess wird die *Belegung* meistens vor dem *Arrangement* festgelegt. Das passt gut zur oben definierten Operator-Reihenfolgen, da das *Arrangement* sich dann auch eine vorgegebene *Belegung* bezieht. In Szenarien komplexer Bewehrungsproduktion kann es aber auch vorkommen, dass das *Arrangement* bereits optimiert wird, bevor die *Belegung* festgelegt wird. Da sich das *Arrangement* aber auf eine gegebene *Belegung* bezieht, muss beim nachträglichen Verändern der *Belegung* auch das *Arrangement* angepasst werden, um das effektive *Arrangement* faktisch unverändert zu lassen.

Konkret: wenn wir mit $\hat{H}_{SteelProd}$ das *Arrangement* bezeichnen, das vor dem Setzten der *Belegung* optimiert wurde, so muss man, wenn man nachträglich $H_{SlabProd}$ setzt, $H_{SteelProd}$ anpassen:

$$H_{SteelProd} = H_{SlabProd} \cdot \hat{H}_{SteelProd} \cdot H_{SlabProd}^{-1}$$

3.9.10 Layer

Used only for multi-layer elements; defines the layer to which the item belongs.

3.9.11 ObjectID

See the corresponding section on *Outline* objects, section 3.8.10.

3.10 Bar

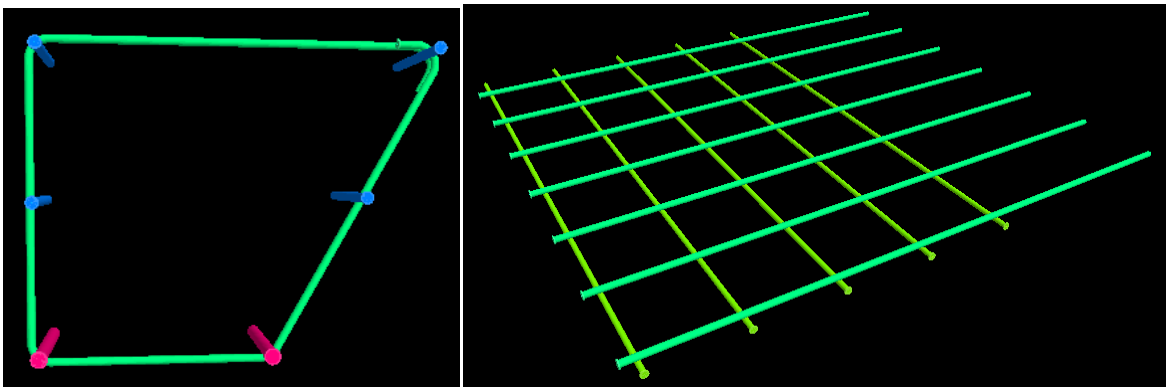
A *Bar* entry corresponds to round-bar steel element, viz. a straight or bent rebar.

3.10.1 ShapeMode

Via *ShapeMode,* one can select the type of representation for the rebar geometry. For straight rebars, this option is rather immaterial; for complex bending shapes, however, data representation can be largely adjusted to the internal representation in CAD through appropriate selection of *ShapeMode*. The effort or time used to implement data export from CAD can thus be decisively reduced.

3.10.1.1 ShapeMode "realistic"

All rebars are specified with the correct bending radii and correct spatial coordinates (the bending radii may also be given via the *BendingDevice*). The relative spacing of the rebars must also be taken correctly into consideration. This representation can be used without any limitation as it directly reflects the product to be produced, thus leaving no room for interpretation.

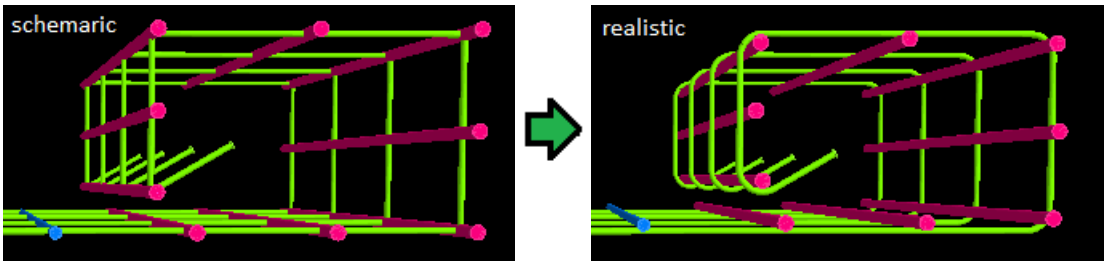


In this representation, the type of reinforcement has no geometric meaning as the geometry is clearly defined even without a need to know the type of reinforcement.

The *realistic ShapeMode* is the recommended representation type, because it is most universal and unambiguous.

3.10.1.2 ShapeMode "schematic"

*Schematic*Mode is a simplified representation whereby the rebars will be drawn with a bending radius of 0:



For bending angles of up to 90°, the bending shape simply is rather more "angular" than in reality. For bending angles above 90°, however, the outer shape will be shown distorted.

Here, the spatial absolute position of the rebars should be set as follows:

- Independent rebars (*Master* rebars): here, the rebar coordinates are set such that the L0 segment (= main or master segment) takes the real position.

- Dependent rebars (*Slave* rebars): *Slave* rebars are typically strung along the *Master* rebar such that the rebars intersect at their core.

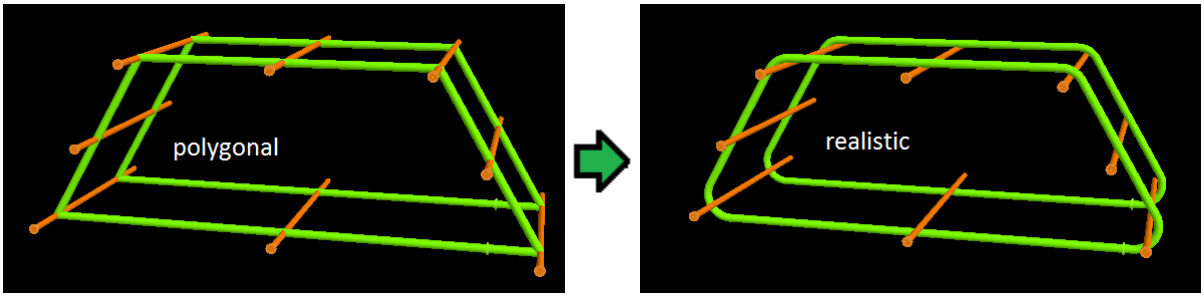
Master/*Slave* relationships may be defined either explicitly through a master rebar staggering pattern[32], or else implicitly via the type of reinforcement: two rebars of a type of reinforcement of 1and 2 will be mutually interrelated in a *Master*/*Slave* relationship if they intersect at the rebar core (here, the type 1 rebar will be the *Master* and the type 2 rebar will be the *Slave*)[33].

If we specify the rebars such that they intersect at the core, we will need a formal additional rule that will specify in what way the rebar layers will be mutually staggered in reality. Hence, the following shall apply to *Master*/*Slave* rebars that intersect each other at their core:

For straight *Master* rebars, the *Slave* rebars will be on top (viz. type 1reinforcement at the bottom, type 2reinforcement at the top). For bent *Master* rebars, the *Slave* rebars will be *inside*[34].

3.10.1.3 ShapeMode "polygonal"

Polygonal representation is similar to *schematic* representation, and the majority of the rules described above can be applied without any modification. Contrary to *schematic* representation, however, not the common rebar lengths are specified here, but rather the edge length of the polygon that covers the real bending shape.



For bending up to 90°, the representations of *schematic* and *polygonal* tally with each other. However, both representations differ when it comes to acute bends: the *polygonal* representation remains true to the real shape of the rebar, whereas the *schematic* representation modifies the outer proportions[35].

For "external radius bends" on the rebar ends, the *polygonal* representation will often appear unnatural, and for 180° bends it will even be impractical (as the polygon legs would be infinitely long). In such an instance, you may want to divide an acute bending angle α into two angles$\alpha_1$andα_2, and introduce an additional edge in between these bends that will have the following length:

$$L_k = R \cdot \left(\tan\frac{\alpha_1}{2} + \tan\frac{\alpha_2}{2}\right), \quad \alpha_1 + \alpha_2 = \alpha$$

Whereby R is the bending radius at the rebar center or core.

Typically, $\alpha_1$is given by the outer shape of the polygon, and $\alpha_2$merely completes the bend such as to obtain the desired overall angleα. The polygon edge $L_k$to be introduced artificially can then be determined as described above. Following conversion to the *realistic* mode of representation, the additional edge will be reduced to a segment of a length of 0, which will then be omitted for logical reasons – the two bending angles$\alpha_1$and $\alpha_2$will then merge into one overall angleα.

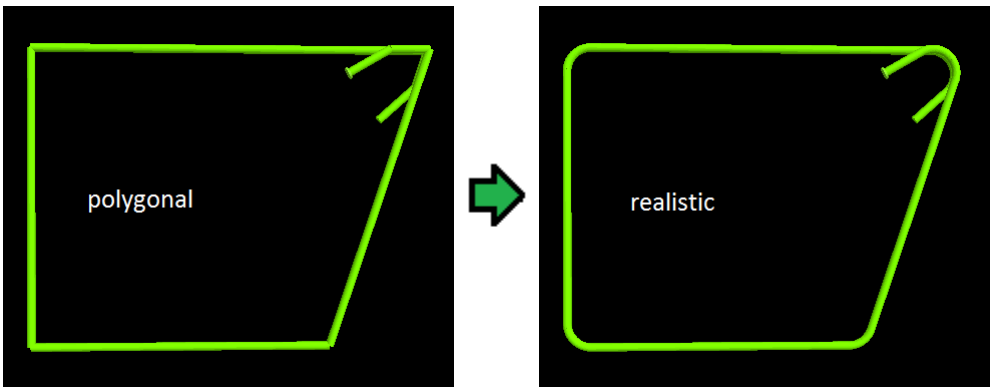
---

[32] Master rebar staggering pattern: see Section 3.12.1.

[33] Similarly, a type 4 rebar may be a *Slave* in its relation to a type 2 rebar. However, this multi-state dependence is only used for very complex cages. (For a definition of the types of reinforcement, refer to Section 3.10.2).

[34] In practical cases, it will be fairly obvious which is *inside* and which is *outside*. A general mathematically precise definition of *outside* or *inside*can be stipulated as follows:
If $\vec{v_\iota}$are the vectors of the rebar segments and$\vec{a} := \sum_{i=0}^{n-2} \vec{v_\iota} \times \overrightarrow{v_{\iota+1}}$, then $\vec{a} \times \vec{v_\iota}$will face *inwardly*.

[35] In the above example, the cage has an identical height in both, *polygonal* representation and *realistic*representation; in *schematic*representation, it would have a smaller height.



Having regard to this splitting of angles, the *polygonal* mode of representation will turn into a very universal one that will have an edge over the *schematic* mode of representation[36].

The dependence of L_k on R will introduce a certain redundancy into the data as L_k conclusively results from R. As a matter of fact, we may select L_k to be smaller or even set it to 0 as the system receiving these data will know that each polygon leg will have to have at least the length mentioned above. Then there will be a deviation of the polygon from the real rebar shape, however, this typically only relates to end roundings, such that the overall shape of the rebar will be displayed undistortedly.

3.10.1.4 Automatic determination of ShapeMode and mixed representation

Often, no *ShapeMode* will be specified, specifically when the data are imported from UNICAM or BVBS files. As different CAD systems will use different types of representation, data without specified *ShapeMode* must be considered to be incomplete[37].

In UNICAM files, the *ShapeMode* is sometimes (as an extension of the formal definition) mapped in the spare field RODSTOCK, line 5, columns 9-11:

- 000 = *undefined*
- 001 = *schematic*
- 002 = *polygonal*

Some CAD systems use a **mixed representation**: the bended bars are drawn in *schematic* mode, but the straight bars are set at their actual position in space. The bars do not intersect in the core and therefore there is no automatic position adjustment of those bars.

3.10.2 ReinforcementType (reinforcement layers)

3.10.2.1 Definition of reinforcement type

The types of reinforcement largely follow the UNICAM definition; however, additional definitions were introduced for cage production:

0 = no definition.

1 = first rebar layer; **for cage production: stirrups**.

2 = second rebar layer; **for cage production: longitudinal rebar**.

3 = spears.

[36] The mentioned division of angle may also be done in *schematic* representation; this will help avoid having to specify bending angles of more than 90°, and would result in perfect congruence between the *schematic* and *polygonal* modes of representation. However, the advantage of the *polygonal* mode of representation lies in the option of specifying acute bending angles through acute polygon angles geometrically true – this is not feasible in the *schematic* mode of representation.

[37] There are useful approaches to systematically conjecture the intended *ShapeMode*: Data with an explicit specification of bending radii or bending roll diameters will typically appear in *realistic* mode; data that do not come with these details will mostly appear in *schematic* mode. However, this rule does not apply unconditionally.

4 = other reinforcement (third rebar layer).

5 = upper reinforcement of first rebar layer.

6 = upper reinforcement of second rebar layer.

7 = upper reinforcement other rebars (upper reinforcement of third rebar layer).

8 = loose rebars (not welded, not connected to any other rebars).

For Extirons, the ReinforcementType is equivalent to UNICAM Extiron-Type.

3.10.2.2 How to set the reinforcement types

The exact meaning of the individual reinforcement types depends on the actual manufacturing process. If there are several manufacturing processes in a plant, the meaning of the *ReinforcementType* field may therefore vary with the *MeshType* of the *Steel* block (see also section 3.9.5).

In general, however, the following recommendations can be given for the use of reinforcement types:

ReinforcementType 1, 2 and 4:

For flat wire mesh, one would select the reinforcement layer immediately above the desired layer of the rebars: those rebars that are at the bottom in the production pallet (= on the outside in the double wall) will form layer #1, the rebars on top of that will form layer #2.

The following stipulation is recommended for bent wire mesh or cages:

a) If bending occurs in one direction only, this should be the first layer that will be bent.

b) If bending occurs in either direction, it should be feasible to bend the first layer last when the wire mesh is unrolled.

Here, item b) is to be understood such that, when unrolling the wire mesh, the first layer must be unrolled first, then the second one. When unrolling the first layer, adjacent layer #2 rebars will be carried along; when the second layer is unrolled thereafter, *no* layer #1 rebars will be carried along anymore[38].

Layer #1 may also be considered to be the *main layer* or *stable layer*: coordinate corrections for the wire diameter or bending radii will be dominated by layer #1; layer #2 will adapt to layer #1, but not the other way around.

ReinforcementType 8:

ReinforcementType 8 identifies a rebar as a "special case" within the manufacturing process in question. If, for example, a steel block is produced using a mesh welding system (which can possibly be specified by the *MeshType*), then *ReinforcementType 8* would specify that the bar in question is not to be welded. If, on the other hand, the entire *Steel* block is to be subjected to a manual manufacturing process, this should be specified via the *MeshType*.

3.10.2.3 Upper reinforcement layers

Reinforcement types 1, 2 and 4 together form a coherent set of "layers" as described above. Reinforcement types 5, 6 and 7 form a second independent set for which the same rules apply as for the first set of layers.

The second ("upper") set of layers will only be absolutely necessary if two independent sets of layers are required within one *Steel* block (layers in different*Steel* blocks independent of each other anyway). In practice, this should occur very rarely only as complex reinforcement cages are usually split into

[38] Should there be any rebars that must be carried along when the second layer is unrolled, the same must be assigned to a third layer (for this purpose, select ReinforcementType = 4, "other reinforcement"). To be able to produce such rebars as mesh rebars, a beam bending machine is required in the longitudinal direction and in the transverse direction; failing this, such rebars must be produced as loose rebars, and welded manually. Rebars not connected to others, viz. that are not to be unrolled or bent with the other rebars, will be assigned to ReinforcementType #8.

several *Steel* blocks which will usually be produced at different times even (for example, cf. the "cover mesh" for solid walls).

Thus, the upper layers are mainly required for compatibility to UNICAM. PXML implementations should treat the upper and lower layers equally as there may well be a *Steel* block that will only include upper layers[39].

3.10.3 SteelQuality

Steel quality.

3.10.4 PieceCount, Diameter, X, Y, Z

Quantity, diameter, and offsets.

3.10.5 RotZ

To determine the position of a rebar, the coordinate system is (after having been moved by X/Y/Z) rotated through the angle ***RotZ*** through the z-axis. All other indications of directions or orientations will then relate to the coordinate system thus rotated (see also the segment orientation, Section 3.10.16.1).

The allowable range for *RotZ* is:

$$RotZ \in]-180°,\ 180°].$$

RotZ defines the direction or orientation of installation of the rebar, and is roughly equivalent to the angle to the x-axis in UNICAM (however, the latter being in the range of [-360, 360°[). Thus, assuming the first section of the rebar (or the whole rebar respectively) to be in the xy plane, *RotZ* will be the angle enclosed by the rebar and the x-axis.

For the general case, it is somewhat more complex to define *RotZ*:

Let's assume, the bending plane of the first bending intersects the xy plane; *RotZ* would then be the angle enclosed by the line of intersection with the x-axis.

Or more specifically: if $\vec{v}_0$ and $\vec{v}_1$ are the first two sections of the rebar, then

$$RotZ = arctan\left(\frac{v_{0y}v_{1z} - v_{0z}v_{1y}}{v_{0x}v_{1z} - v_{0z}v_{1x}}\right) + k\pi.$$

(It may indeed happen that the denominator is 0; the result will then of course be $\pm\frac{\pi}{2}$. The result will only be really undefined if the denominator and the numerator are 0; this case will occur if $\vec{v}_0$ and $\vec{v}_1$ are parallel or antiparallel, or if both of them lie in the XY plane, or if we have only one single segment).

In addition, the above equation contains the undetermined addend of $k\pi$, where k equals 0 or is ±1. As a matter of fact, this consideration does not define this detail as one direction or orientation is initially as good as the one opposite by 180 degrees. To define *k*, one could request that the projection of $\vec{v}_0$ define the direction or orientation, that is to say that[40]:

$$\vec{v}_r \cdot \vec{v}_0 \geq 0 \text{ for } \vec{v}_r := (cos(RotZ), sin(RotZ), 0)$$

Derivation of the equation for *RotZ*:

Let's assume λ to be such that

[39] The upper steel mesh ("cover mesh") of solid walls, for example, might exclusively include upper layers only. However, this should be irrelevant for a PXML implementation.

[40] This limitation does not only result in an intuitively meaningful *RotZ*, but also simplifies other computations, as we will see in Section 3.10.16.12.

$$\vec{v}_0 - \lambda\ \vec{v}_1 = \begin{pmatrix} x_p \\ y_p \\ 0 \end{pmatrix}.$$

The point (x_p, y_p) is the projection of $\vec{v}_0$ in the XY plane for projection along the vector $\vec{v}_1$. From the Z component of the above equation, we obtain $\lambda = v_{0z}/v_{1z}$.

Because of $RotZ = arctan(\ y_p/x_p)$, we obtain the above relationship for *RotZ*.

As mentioned, there will be a problem if $x_p = y_p = 0$. In this instance, it will help to merely consider the projection of the first section only:

$$RotZ = arctan2(v_{0y}, v_{0x}),\ \ (\text{or } RotZ = 0 \text{ for } v_{0x} = v_{0y} = 0).$$

Remark: Basically, it is unnecessary to specify *RotZ* as any change of direction or orientation can be implemented for any *RotZ* using appropriate segment angles *RotX* and *BendY* (see Section 3.10.16.1). *RotZ* was merely introduced to be able to treat the significant case of constant bending plane in a straightforward and illustrative manner. For this reason, the above definition of *RotZ* is to be understood as a mere recommendation only. On principle, it is within everybody's discretion to set *RotZ* as he or she may see fit, or not to use it at all (i.e. to leave it at *RotZ=0*).

3.10.6 ArticleNo

Article identifier of the round steel rebar.

3.10.7 NoAutoProd

This will be set if the rebar is *not* to be automatically produced by the steel machine.

It is typically used for in-stock products that were pre-produced in standard lengths and that are thus excluded from just-in-time production.

3.10.8 ExtIronWeight

The weight of an *ExtIron*rebar in kg. This is exclusively used for *ExtIron*rebars only (as these generally do not have any dimensional details).

3.10.9 Bin

To assign a bin (such as a carriage bin of the steel machine, for example).

3.10.10 Pos

Text field for entering the single rebar drawing reference.

3.10.11 Note

Text field for entering a comment regarding the rebar.

3.10.12 Machine

Text field for specifying the production machine.

A machine-internal production list may optionally be treated as a separate machine in itself. In this case, it is recommended to select the following format:

MSR:2

In this example, "MSR" indicates the machine as such, and "2" is the number of the production list.

3.10.13 BendingDevice

Bending device. This is a text field in which you would normally specify the diameter (in mm) of the bending die to be used. As this is a text field, however, this indication may also be of a more

general nature (such as a description of the die). This field is mapped onto the header field 's' in BVBS[41].

Often, the specification of the *BendingDevice* will imply a minimal bending radius. For a bending die of a diameter *D,* a wire with a diameter *d* will have a minimum bending radius of

$$r_{min} = \frac{D}{2} + \frac{d}{2} + k_r.$$

(k_r being the radius correction value due to spring-back, i.e. the difference between the inner radius of the bent wire and the radius of the bending matrice. Simplifying, this value is most often considered as being 0)[42].

This will, of course, result in a potential conflict with the bending radius indication of the segment (see Section 3.10.16.3). In such a case, the larger bending radius will always be relevant.

3.10.14 Spacer

A Spacer entry describes a single spacer. As opposed to some other formats, spacers will always be listed separately in PXML (so there will be no option to enter a fixed pitch or division).

3.10.14.1 Type

Spacer Type; corresponds typically to the concrete cover in mm, divided by 5.

3.10.14.2 Position

Position of the spacer along the rebar. The position details will relate to the *theoretical* lengths.

3.10.15 WeldingPoint

3.10.15.1 WeldingOutput

Welding output in %.

3.10.15.2 Position

Position of the welding point along the theoretical path of the rebar.

3.10.15.3 WeldingPointType, WeldingPrgNo

WeldingPointType contains a type identifier for the welding point.

WeldingPrgNo directly determines the welding program to be used. The exact meaning of this value depends on *WeldingPointType* and the used machine.

The following values of *WeldingPointType* are predefined:

- **0:** Undefined type
- **1:** Generic Welding Point
- **-1:** Point that must not be welded
- **-2:** Label fixing point

[41] In UNICAM, this field (as an extension of the formal definition) will be mapped onto the spare field RODSTOCK line #5, columns #5-7, but merely numerical values are entered here, and no more than three (3) characters only.

[42] The consideration of k_r is only necessary when high accuracy is needed. In practice, this can occur when stirrups or cages are processed automatically. Then the overall geometry of the bent bar need to be well known, and its geometry depends on k_r if some bending angles exceed 90 degrees. But above all, there is the need to know the exact real segment lengths on unrolled bars. It is therefore recommended to determine k_r by measuring the real lengths on the bars. Doing so, k_r will not just consider the spring-back factor, but will also include the fact that center line of the wire may be lengthened due to bending (this will result in a slightly increased k_r).
It must be noted that k_r is not a constant value. The spring back value is typically more important for small wire diameters.

- **-3:** Color point (if multiple colors are available, the *WeldingPrgNo* may be used for specifying the color type)
- **-17**: Mark for main segment ("L0" segment). A segment that is marked in this way should be used as a central and stable segment during production.
- **-23**: **BendingStep**. Marks a bending step in which the specified segment is moved in a bending action.

  The *WeldingPrgNo* determines whether the bending occurs at the begin or the end of the segment: a negative *WeldingPrgNo* specifies to bend at the end.

  The *GroupID* may be used for grouping bendings and identifying these groups for external references. The sequence of BendingSteps occurrences within a bar defines a supposed bending sequence when producing the bar. While the effective bending sequence may differ from that, the supposed bending sequence is relevant when determining welding points between layers.
- **-29**: **GuidedPoint**. Marks a point on a "guided bar" that is connected to a "guiding bar" in such a way that the guided bar moves when the guiding bar is moved or bent up.

  The *GroupID* is used to assign a *GuidedPoint* to the respective *GuidingPoint*.
- **-30**: **GuidingPoint**. Marks a point on a "guiding bar" that is connected to a "guided bar" in such a way that the guided bar moves when the guiding bar is moved or bent up.

  The *GroupID* is used to assign a *GuidedPoint* to the respective *GuidingPoint*.
- **-100**: **GrippingPoint**. Marks a point where a bar is gripped to transport the bar or mesh.

The negative type values specify points that are not really welding points but are somewhat similar in their handling.

3.10.15.4 GroupID

The **GroupID** field can be used to link different welding points to each other. Typically, welding takes place on the cross point of two bars. Both such bars have a *PXML WeldingPoint* placed on that cross point and these two *PXML WeldingPoint* items can be linked together by assigning the same *GroupID* to them.

3.10.16 Segment

A *Segment* is a section of a round steel rebar. A straight rebar consists of precisely one segment; bentrebars have one segment for each section thereof (i.e. for *n* bends, there will be *n+1* segments).

The first section will commence at the X/Y/Z coordinates specified for the respective rebar; each further section will then commence at the end of a respective previous section.

It is optional to specify the **Type** attribute. The *Type* attribute may have the following values:

- Type = "normal": the segment describes a normal line segment (this is the default-type).
- Type = "spiral": the segment describes a spiral. See Section 3.10.16.10

3.10.16.1 Segment-Orientation (RotX, BendY)

For each section, two angles **RotX** and **BendY** will be specified that will describe a rotation of the coordinate system: the coordinate system will at first be rotated around the x-axis, through the *RotX* angle; and will then be rotated around the new negative y-axis through *BendY*[43].

$$RotX \in [-90°, 90°] \text{ (conditional only[44])}.$$

$$BendY \in]-180°, 180°] \text{ (conditional only[45])}.$$

Now, the x-axis of the rotated coordinate system specifies the direction of the respective segment. The rotation of the various sections will add up, that is to say for the second section , the coordinate system of the first section (rotated previously already) will be rotated yet again.

RotX rotates the bending plane, and *BendY* reflects the bending angle (except for the first segment).

3.10.16.2 Segment-Length (L)

The **Segment length L** (in mm) corresponds to what is often called the **theoretical length** of the section (as measured at the center of the rebar). If the bending radius *R* is larger than *0*, the length *L* will deviate from the real length of the rebar.

For the representation of *spirals* (see Section 3.10.16.10), L will indicate the *arc radius*.

3.10.16.3 Bending-Radius (R)

The **Bending radius R** describes the curvature of the bends (in mm).

If this value is *0*, the various systems can autonomously take a bending radius, and make a respective correction of the length.

The bending radius indication of the *first segment* must be ignored and is always assumed to be 0. The same will apply to the first segment after a spiral.

For spirals, the value of *R* is also ignored (see Section 3.10.16.10).

The side length *L* of a segment is to be understood as a *theoretical length*; for*R>0,* the *real length* is different from *L*.

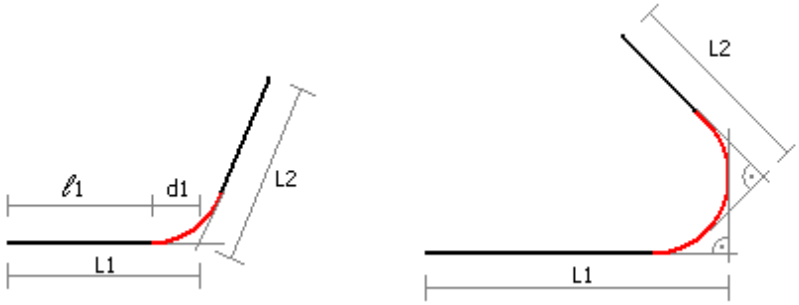
---

[43] The *negative* Y-axis is used for *BendY* to keep the bending angle compatible to BVBS and UNICAM.

[44] By definition, *RotX* may take any value. However, it is recommended to use values between ±90° as this range is adequate to represent any shape: *RotX* values off this range can be transformed into this range through adding (or subtracting) 180° if all following *BendY* are inverted at the same time. Such restriction to the range between ±90° makes representation more unambiguous, and will avoid the case of *RotX*=±180° that would unnaturally complicate things, as this case could also be described via *RotX*=0.

[45] The restriction of *BendY* to values between ±180° will only be rational for a bending radius of *R=0*; for*R>0, BendY* may be within the range of ±360°. Finally, to facilitate the representation of spirals of any length in an easy manner, it is allowed in PXML to specify any value for*BendY*, viz. even values beyond the range of ±360°.

For a definition of theoretical lengths, refer to the following Figures:



For angles not larger than 90°, *R* will have *no* effect on the bending shape, but merely results in the corners being rounded. The basic shape will then be given by the theoretical lengths. For bending angles beyond 90°, *R* will have a direct effect on the overall dimensions.

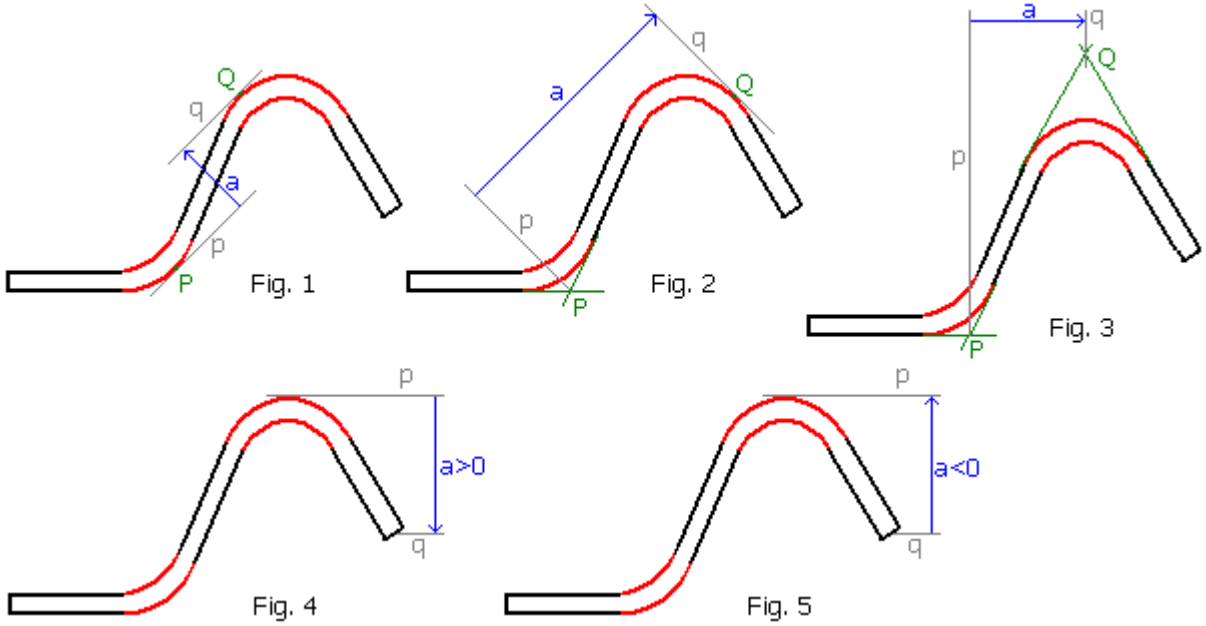
Radius definition via specification of ***BendingDevice***:

As an alternative to directly entering the radius in the segment, there may be an implicit determination of the bending radius via the *BendingDevice* (see Section 3.10.13). If both are given (viz. *BendingDevice* and *R*), the larger radius will be relevant.

3.10.16.4 External dimensions

Normally, *center dimensions* are always taken into consideration, i.e. all dimensions relate to the center of the rebar, often called the *core* of the rebar. In some instances (specifically when designing complex bending shapes), it will be helpful to consider external dimensions instead.

A segment's **external dimension** will only be specified unambiguously once a **direction of dimensioning** will have been specified.



The above Figures show how the segment's external dimensions are defined for different directions of dimensioning. The *direction of dimensioning* (viz. direction of the blue arrow) defines two *extension lines p* and *q* (the gray lines) that are orthogonal to the direction of dimensioning. The segment's *external dimension* is equivalent to the distance between the extension lines *p* and *q*. The segment's *external dimension* will be *positive* if the perpendicular from *p* to *q* is parallel to the direction of dimensioning; this measure will be *negative* if the perpendicular from *p* to *q* is antiparallel to the direction of dimensioning (*p* is the extension line at the *beginning* of the segment, *q* is the extension line at the *end* of the segment respectively).

Fig. 1: If possible, *p* and *q* will be tangents (of the arc exterior).

Fig. 2: Here, *p* cannot be a tangent. Instead, *p* runs through the point of intersection *P* of the two end-tangents of the arc.

Fig. 3: Here, neither *p* nor *q* is a tangent; the lines are defined via the points of intersection *P* and *Q*.

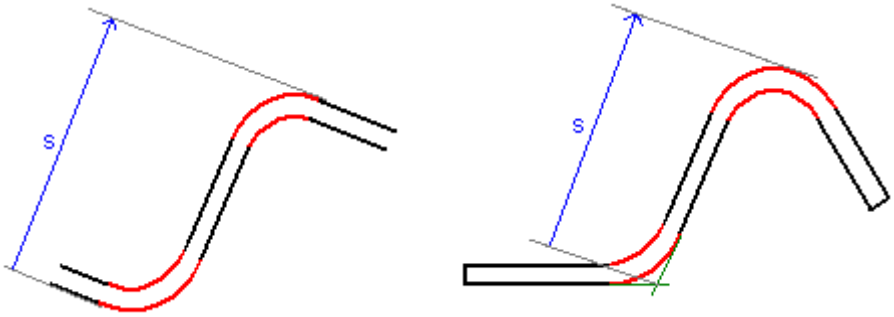
Fig. 4: If a segment has no bend at one end (first or last segment), dimensioning will relate to the rebar center (however, this is not true for *conventional external dimensions*; see Section 3.10.16.8).

Fig. 5: The positive distance is always counted from *p*to *q* (*p* at the beginning of the segment). If this is opposed to the direction of dimensioning, the distance will be *negative*.

For bending angles equal to or larger than 180°, the above definition is ambiguous. In such an instance, the extension or projection line is drawn such that it is the tangent of the first section of the bend (as if the bending angle was less than 180°).

Formal definition: The **measuring points** *P* and *Q* are either tangent points or points of intersection of the end-tangents (whereby merely arcs smaller than 180° are taken into consideration only). The segment's **external dimension** is the inner product of the vector PQ with the unit vector in the direction of dimensioning.

3.10.16.5 External length of segment



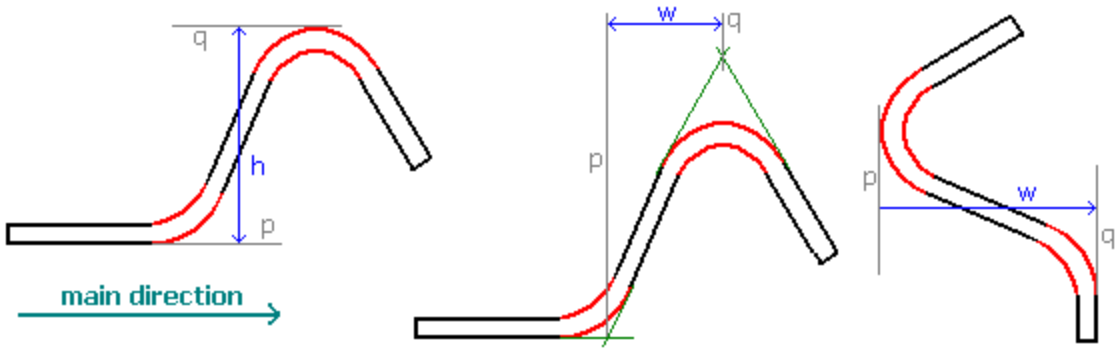
The **external length** of a segment is the segment's*external dimension* that runs in the direction of the segment. (The segment's*external dimension* is defined according to Section 3.10.16.4.).

3.10.16.6 Height and width of segment

To be able to use the **height** or **width** of a segment, we first need to define a **main direction**. It then holds:

Width = segment external dimension in the main direction, and

Height = segment external dimension perpendicular to the main direction (at an angle of +90° in relation to the main direction).



3.10.16.7 Rules for computing external dimensions

The definition of the external dimension may extended to cover a *series* of segments. Here, it can be easily demonstrated that the external dimensions are **additive**, i.e. the external dimension of the series of segments is equal to the sum of the external dimensions of the various segments[46].

Such additivity may also be used to partition an individual segment into elementary subsegments; here, a normal segment may be treated as a series of 3 segments:

1) a segment with a bend at the beginning, a straight section of a length of 0, no bend at the end;
2) a straight section; and
3) a segment without a bend at the beginning, a straight section of a length of 0, a bend at the end.

[46] Naturally, said additivity of the external dimensions will only apply if the same direction of dimensioning is used for all external dimensions and if the correct sign is assigned to all external dimensions. For example, a segment being orthogonal to the direction of dimensioning and having 90° at the beginning and -90° at the end respectively will have an external dimension that will be equal to the negative wire diameter.

3.10.16.8 Conventional external dimensions

If a segment end does not have a bend, the external dimension will relate to the rebar center (rebar core). This has been defined that way so as to create theoretical relations that are as simple and as possible.

For the user, however, it may seem rather unnatural to have a reference to the rebar center at the beginning and the end of the rebar each. For this reason, the term of **conventional external dimension** is introduced here: here, no reference is made to the rebar center at the beginning and end of the rebar, but rather to the exterior of the rebar, viz. to that side which is also used for the dimensioning of the other end of the segment[47].

3.10.16.9 General computations for the bending radius

The following holds for the length *b* of the arc of a bend around α:

$$b = R \cdot |\alpha|.$$

The following holds for the straight section ℓ:

$$\ell = L - d, \quad d = R\,tan(\frac{\alpha_M}{2}), \quad \alpha_M := min(\,|\alpha|,\ 90°)$$

Thus, the following holds for the real length L_R of the rebar:

$$L_R = \ell + \frac{b}{2} = L + \frac{b}{2} - d = L + R(\frac{|\alpha|}{2} - tan(\frac{\alpha_M}{2})).$$

Naturally, all of the above will only hold if $L >= d$, or, in other words, if:

$$L >= R\,tan(\frac{\alpha_M}{2}).$$

There are various options to force these conditions; here, the easiest one obviously is to increase *L* to the above minimum value, if required.

3.10.16.10 Arcs and spirals in traditional "spiral form"

Traditionally, arches and spirals are dealt in a particular form and are somehow explicitly described as special arc-like shapes. To maintain compatibility with older systems, PXML offers such a possibility, too, i.e. it defines a particular *spiral* segment type[48].

A *spiral* segment is identified through the *Type* identifier "spiral".

For spirals, the segment parameters have the following meaning:

- ***BendY***: turning angle of the spiral
  (may take any value, viz. is *not* limited to ± 360°).
- ***L***: radius of the spiral.
- ***R***: does not have any relevance and is thus ignored[49].
- ***RotX***: defined the pitch of the helix[50].
  The increase in height per torsion is $G = L \cdot \sin\,(RotX)$.

Connecting the arc to the straight segment:

---

[47] This is not defined for a straight rebar. However, in this instance it doesn't make a difference whether we relate dimensioning to the core of the rebar or to an exterior of the rebar.

[48] Although *spiral* segments are provided in PXML, newer systems should consider to describe the arches in normal PXML representation, as will be described in Section 3.10.16.11.

[49] Please note: the Specification declares that *R* be ignored, and thus may take any value. It is *not* at the discretion of the implementations to make any other use of *R* and thus to restrict the freedom when setting *R*. This declaration is necessary because the radius *R* is often set for all segments via higher-ranking parameters (e.g. bending die parameters) – in such an instance, it should not be necessary to exclude spirals.

[50] The option of specifying a *true RotX* for spirals is refrained from deliberately as this will hardly ever be necessary in practice. However, should this actually be necessary on a few rare occasions, an auxiliary segment of a length of 0 must be prefixed to the spiral.

If arcs (spirals) are connected to straight segments, the transition between the arc and the line segment will always be *without a bend*. At the beginning of the arc, this must be so due to the mere fact of the data format: as *BendY* describes the arc angle, it is not possible to additionally specify a bending angle. At the end of the arc, viz. at the beginning of the next straight segment, it would be basically possible to specify a bend. However, this degree of freedom is not to be used, i.e. the following segment is to have *BendY=0*. This restriction is meaningful with a view to having symmetrical relationships at the beginning and the end of the arc, and to altogether simplifying the computational relationships. Moreover, this restriction blends in well with the BVBS encoding and the functional principle of the bending machines: in either case, bends must be encoded via separate segments at the beginning and the end of the arc.

Spirals and type of representation:

The treatment of spirals is basically defined for the *realistic* type of representation only (see *ShapeMode*, Section 3.10.1). The *schematic* or *polygonal* concepts of representation cannot be combined with the concept of representation for spirals in any meaningful way, and both concepts also belong to very different domains in terms of applications engineering.

3.10.16.11 Arcs in ordinary PXML form

In PXML, an arc can also be represented merely as a bending. In such an instance, the arc will not be separately identified as such and will merely differ from a standard bending in that it has a large bending radius and that it thus will typically not be produced through bending over a die plate, but rather through other machine devices. As a matter of fact, however, it may be better not to anticipate the type of machine processing in the data, but rather to leave this to the machine software instead; here, PXML data will be limited to a geometrical description of the product without actually specifying the manufacturing process or technology.

If arcs are described using standard PXML segments the following will have to be taken into consideration:

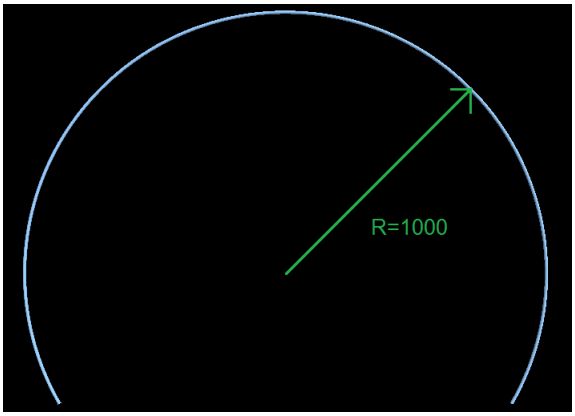
a) There may be no bending at the beginning of the first segment. That is to say, at least two segments will be required to render a curved iron. (And this is the only real drawback of the standard PXML segments as compared to the *spiral* segments.)

b) If there is a bending upstream of the arc, the same must be accommodated in a separate segment. (This is the same for *spiral* segments.)

c) An arc is already fully defined by its radius and angle. The L-value of the segment is redundant and should have precisely the value of $L = R\ tan(\alpha_M/2)$ (see Section 3.10.16.9). To avoid this redundancy in the data, it is recommended to set *L=0*. (Except where there is a straight segment downstream of the arc in which case L is accordingly larger than$R\ tan(\alpha_M/2)$).

d) Representation of a flute height of spirals is (currently) unprovided for.

Example 1: simple arc



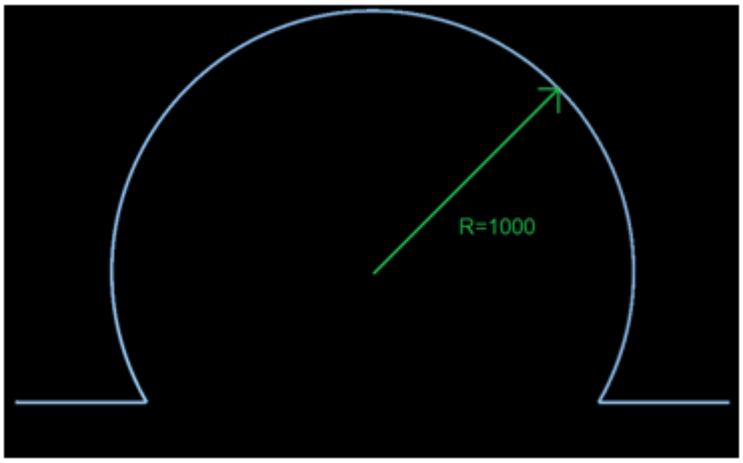
An arc with an arc radius of 1,000 mm (core dimension) and an angle of 240° will be represented via two segments:

Segment 1: L=0, BendY=0, R=0

Segment 2: L=0, BendY=240, R=1000

The length of this arc is 4189 mm; this length is not explicitly specified as it can be calculated from the radius and angle. Again, the segment length *L* (as mentioned above) is not specified, or is simply set to be 0.

Example 2: arc with end hooks



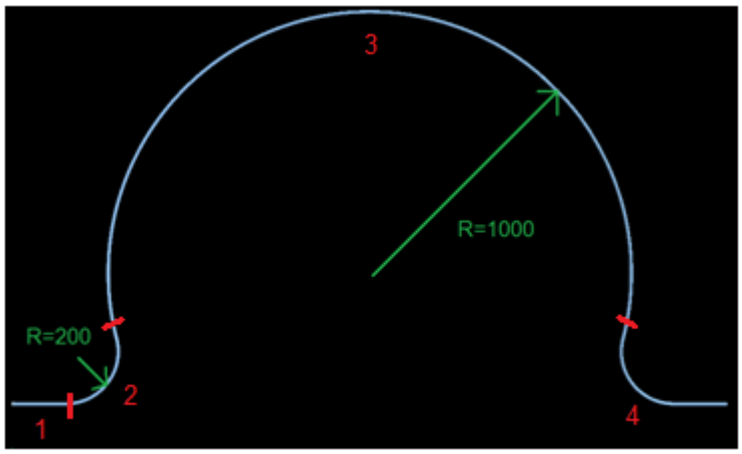
Segment 1: L=500, BendY=0, R=0

Segment 2: L=0, BendY=120, R=0

Segment 3: L=0, BendY=-240, R=1000

Segment 4: L=500, BendY=120, R=0

Example 3: arc with end hooks with a bending radius



Segment 1: L=404, BendY=0, R=0

Segment 2: L=0, BendY=104, R=200

Segment 3: L=0, BendY=-208, R=1000

Segment 4: L=404, BendY=104, R=200

The red lines and red numbers in the figure delimit the segments.

The shortening of the angles by approx. 16° (=120°-104°) can be determined via the following relationship:

$$\Delta\alpha = \alpha - \operatorname{acos}\frac{R_2 + R_3 \cdot \cos\alpha}{R_2 + R_3} = 120° - \operatorname{acos}\frac{200 + 1000 \cdot \cos 120°}{200 + 1000} = 15{,}52°$$

3.10.16.12 General computations for coordinate rotation

There are three types of coordinate rotation:*RotZ*, *RotX* and *BendY*. Rotation around the z-axis (*RotZ*) occurs only once for each rebar, and is virtually freely selectable as has been explained above. Rotations around the x-axis or the y-axis (*RotX* or *BendY*) can be specified for each section.

Let's assume $\underline{v}$being the coordinate representation of a vector in the original coordinate system and $\underline{v}'$ the respective representation in the rotated system; then, in such an instance, transformation will be described through a **rotation matrix** *D* as follows:

$$\underline{v}' = D \cdot \underline{v}.$$

The following holds for the rotations mentioned above[51]:

$$D_{RotZ} = \begin{pmatrix} \cos RotZ & \sin RotZ & 0 \\ -\sin RotZ & \cos RotZ & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$D_{RotX} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos RotX & \sin RotX \\ 0 & -\sin RotX & \cos RotX \end{pmatrix}$$

$$D_{BendY} = \begin{pmatrix} \cos BendY & 0 & \sin BendY \\ 0 & 1 & 0 \\ -\sin BendY & 0 & \cos BendY \end{pmatrix}$$

$$D_{RotX,BendY} = D_{BendY} \cdot D_{Rotx} = \begin{pmatrix} \cos BendY & -\sin RotX \cdot \sin BendY & \cos RotX \cdot \sin BendY \\ 0 & \cos RotX & \sin RotX \\ -\sin BendY & -\cos BendY \cdot \sin RotX & \cos BendY \cdot \cos RotX \end{pmatrix}$$

(These details already take into consideration that *BendY* describes a rotation around the *negative* y-axis).

Now, let's assume $\underline{v} = (v_x, v_y, v_z)$ being the representation of a unit vector that will transition into the representation $\underline{x}^0 = (1,0,0)$ when the coordinates are rotated as $D_{RotX,BendY}$. It hence follows that $D_{RotX,BendY} \cdot \underline{v} = \underline{x}^0$, and further that $\underline{v} = D^T_{RotX,BendY} \cdot \underline{x}^0$. From this it follows that:

$$v_x = cos(BendY),\ v_y = -sin(RotX)\ sin(BendY),\ v_z = cos(RotX)\ sin(BendY).$$

Through combining these equations, we arrive at:

$$tan(RotX) = -\frac{v_y}{v_z}.$$

If, now, we restrict *RotX* to the range of [-90°,90°] (which is always possible as it was mentioned above), we arrive at:

$$\boxed{RotX = -arctan\frac{v_y}{v_z}.}$$

On given *RotX* we are now able to determine sin(*BendY*) in two possible ways:

$$sin(BendY) = \frac{v_y}{-sin(RotX)},\quad sin(BendY) = \frac{v_z}{cos(RotX)}.$$

Since cos(*RotX*) or sin(*RotX*) may be zero (but never both of them), we have to choose the numerically more stable option, i.e. the one with the bigger denominator.

With $v_x = cos(BendY)$ the value of cos(*BendY*) is given, too, and thus *BendY* can be identified uniquely by

$$BendY = arctan2(sin(BendY),\ cos(BendY))$$

Putting this together we finally get

[51]Rotation matrices are always *orthogonal*, i.e. we have$D^{-1}=D^T$.

$$BendY = \begin{cases} arctan\,2\left(\frac{v_y}{-sin(RotX)},\ v_x\right), & |sin(RotX)| \geq |cos(RotX)| \\ arctan\,2\left(\frac{v_z}{cos(RotX)},\ v_x\right), & |sin(RotX)| < |cos(RotX)| \end{cases}$$

We will have a special case for $v_y = v_z = 0$. Obviously, this will be the case whenever $BendY = 0$ or $BendY = 180°$ respectively. The above "arctan" term for the computation of *RotX* will then be undefined, and as a matter fact *RotX* will not be clearly definable in such an instance as two successive *RotX* will rotate around the same axis when *BendY*=0 or $BendY = 180°$ respectively; in such an instance only the sum of these *RotX* will be clearly defined, but not each value separately. If we do not care about this ambiguity, we will still get a valid result: *RotX* will then have a random value that will be dominated by rounding errors[52]; if we continue computing consistently using this "random" *RotX*, the result will still be correct altogether (the randomness of *RotX* will be compensated in the next *RotX*). However, it is better to treat this special case in an explicit way as follows:

$$\text{for } BendY_i = 0\text{:}\quad RotX_i + RotX_{i+1} \rightarrow RotX_i,\ \ 0 \rightarrow RotX_{i+1}$$

That is to say, the sum $RotX_i$ and $RotX_{i+1}$ will be fully absorbed in $RotX_i$; $RotX_{i+1}$ will be set to 0[53].

On the other hand, we will have a certain problem if *BendY* is (by approximation) ±180°, thus in the case of $\underline{v} = (v_x, v_y, v_z) = (-1,0,0)$. Here, too, *RotX* is arithmetically undefined, as well as the sign of *BendY*. In practice, however, the bending radius is different from 0 so much that *BendY* and *RotX* are clearly defined; however, the bending radius information is not included in the simplified representation of straight subsegments. Here, *RotX* and *BendY* must be defined via additional information, or maybe via a modified *v* that would supply non-vanishing y or z components by taking into consideration the actual bending direction[54].

[52] The numerical special case of the NaN result must also be assigned a real value (e.g. 0).

[53] Such an approach will guarantee that *RotX* will only be different from 0 within the bending shape if there is a real rotation of the bending shape. For planar bending shapes, *RotX* will only be required to specify the orientation angle of the whole rebar, and in such an instance you always want this angle to be provided in $RotX_0$ (but not in $RotX_1$).
Finally, it should be noted that the case of *BendY*=0 will hardly ever occur inside the bending shape (that would be a succession of two segments without a bend in between – which would really be a degenerate case, so to say). However, the first *BendY* angle is very often 0, viz. for bars where the first segment lies in the XY plane.
For the sake of completeness, it should be noted that the case of $v_y \neq 0$, $v_z = 0$ is not a problem numerically speaking. The "arctan" feature for the determination of *RotX* will then yield ±90°, where the sign comes about randomly from rounding errors again (that is to say, v_z may be slightly positive or slightly negative, depending on the rounding error). This randomness will then be compensated again in the next computations and will not be a problem as no "artificial" value different from 0 has been introduced.

[54] This consideration does not hold for the first segment as the real bending radius will be 0 in this case (for the first segment, *BendY* merely specifies an orientation, but not a bend). However, this is not a problem as it will be possible to avoid the case of *BendY*=±180° for the first segment anyway if the restriction from Section 3.10.5 is duly taken into consideration, which requires *RotZ* to be selected such that $\vec{v}_r \cdot \vec{v}_0 \geq 0$; here, $\vec{v}_0$ is the first segment and $\vec{v}_r := (cos(RotZ), sin(RotZ), 0)$. This condition is equivalent to $\cos(BendY_0) \geq 0$ and thus to $BendY_0 \in [-90°, 90°]$.

3.10.16.13 Conversion from or to UNICAM

In UNICAM, there is a restriction in that the first segment must lie in the XY plane. Hence, in UNICAM possibly an additional horizontal **dummy segment** must be introduced the length of which is 0. This *dummy segment* will not be required in PXML[55]. Another difference between UNICAM and PXML is found in the bending plane: whereas, in UNICAM, all bends must lie within the plane, the bending plane may be rotated in PXML prior to each bend (*RotX*); of course, this degree of freedom will be lost upon conversion PXML → UNICAM.

Hereinafter, the following terms will be used for the UNICAM values:

α_{toX} = angle to the x-axis

α_L = orientation angle of the bending plane (0 is for the vertical bending plane)[56]

d_i length of the segment i (i=0, 1, 2, ...)

α_i = angle at the end of the segment i (i=0, 1, 2, ...)

UNICAM → PXML excluding Dummy Segment ($d_0 \neq 0$):

$$RotZ = \alpha_{toX}$$
$$RotX_0 = -\alpha_L, \quad BendY_0 = 0, \quad L_0 = d_0$$
$$RotX_i = 0, \quad BendY_i = \alpha_{i-1}, \quad L_i = d_i, \quad i \geq 1$$

UNICAM → PXML including Dummy Segment (d0=0):

$$RotZ = \alpha_{toX}$$
$$RotX_0 = -\alpha_L, \quad BendY_0 = \alpha_0, \quad L_0 = d_1$$
$$RotX_i = 0, \quad BendY_i = \alpha_i, \quad L_i = d_{i+1}, \quad i \geq 1$$

PXML → UNICAM, general considerations:

Using the*dummy segment*, conversion from PXML to UNICAM will be fairly straightforward. The special cases excluding a *dummy segment*, on the other hand, will have to be treated separately as, here, the information on $BendY_0$ cannot be directly transmitted (the *dummy segment* that could carry this information does not exist in these instances). The information on $BendY_0$ must hence be somehow included in the other variables (and the case excluding the *dummy segment* is specifically characterized in that this is possible).

Besides, one will always have to make a differentiation between an ordinary rebar and a bent rebar respectively: ordinary rebars have no bending plane, bent rebars do have a bending plane that will be defined by the first real bend ($BendY_1$). Naturally, for conversion to UNICAM, we have to assume that all real bends will lie in the same plane.

PXML → UNICAM for only one segment:

Special case of $RotX_0 = \pm 90°$ or $BendY_0 = 0°$ (no*dummy segment* required):

$$\alpha_L = 0$$
$$\alpha_{toX} = RotZ - BendY_0 \, sgn(\sin RotX_0)$$
$$d_0 = L_0, \quad \alpha_0 = 0$$

Special case of $BendY_0 = \pm 180°$ (no*dummy segment* required, but inversion of orientation):

$$\alpha_L = 0$$
$$\alpha_{toX} = RotZ + 180°$$

[55] In PXML, it is basically allowed to have segments with a length of 0, but it is not necessary (and hence not reasonable) to use a horizontal DummySegment.

[56] Here, α_L will be assumed to be in the range of]-180°,180°]. Depending on the format, in the UNICAM file, α_L will be either directly saved, or a respective angle in the range of [0, 360°[.

$$d_0 = L_0, \quad \alpha_0 = 0$$

All other cases (including a *dummy segment*):

$$\alpha_L = -RotX_0$$
$$\alpha_{toX} = RotZ$$
$$d_0 = 0, \quad d_1 = L_0, \quad \alpha_0 = BendY_0, \quad \alpha_1 = 0$$

PXML → UNICAM for several segments:

As the bending plane is defined by the first bend, it is reasonable to assume *BendY1*≠0. If the first bend is degenerate (a bending angle of 0), any adjacent segments will be condensed in merely one segment (in terms of data or theoretically).

$$\alpha_L = -arcsin(\sin R\,otX_0 \cos R\,otX_1 + \cos B\,endY_0 \cos R\,otX_0 \sin R\,otX_1)$$

$\alpha_{toX} = \arg(a,\ b)$, *where*

$$a = \cos RotX_0 \cos RotX_1 \cos RotZ - \sin RotX_1 \cdot (\cos BendY_0 \cos RotZ \sin RotX_0 + \sin BendY_0 \sin RotZ)$$

$$b = \cos RotZ \sin BendY_0 \sin RotX_1 + \sin RotZ \cdot (\cos RotX_0 \cos RotX_1 - \cos BendY_0 \sin RotX_0 \sin RotX_1)$$

For *RotX₀*=±*90°* or *BendY₀*=*0* or *BendY₀*= ±*180°* (no Dummy Segment):

$d_i = L_i, \quad \alpha_i = BendY_{i+1}, \quad i \geq 0$ (the last angle being 0).

For *RotX₀* ≠ ±*90°* and *BendY₀* ≠ *0* and *BendY₀*≠±*180°* (including Dummy Segment):

$$d_0 = 0$$
$$\alpha_0 = arg(u, v), \quad where$$
$$u = cos\ B\ endY_0\ cos\ R\ otX_0\ cos\ R\ otX_1 - sin\ R\ otX_0\ sin\ R\ otX_1$$
$$v = cos\ R\ otX_0\ sin\ B\ endY_0$$

$d_i = L_{i-1}, \quad \alpha_i = BendY_i, \quad i \geq 1$ (the last angle being 0).

However, the case excluding*dummy segment* will still require some preprocessing. Prior to export, the bending shape must be transformed such that the following conditions will be satisfied:

$$RotX_0 \in [-90°, 90°]$$
$$BendY_0 = 0$$
$$RotX_1 = 0$$

This **dummy rectification**can be performed as follows:

dummyrectification for *RotX₀* = ±*90°*:

$$RotZ_{new} = RotZ_{old} - BendY_{0,old}\ sgn(\sin R\,otX_{0,old})$$
$$BendY_{0,new} = 0$$
$$RotX_{0,new} = RotX_{0,old} + RotX_{1,old}$$
$$RotX_{1,new} = 0$$
$$RotX_{0,new} \rightarrow [-90°, 90°]$$

Dummy rectification for *BendY₀*=*0*:

$$RotX_{0,new} = RotX_{0,old} + RotX_{1,old}$$
$$RotX_{1,new} = 0$$
$$RotX_{0,new} \rightarrow [-90°, 90°]$$

Dummy rectification for *BendY₀*=±*180°*:

$$RotZ_{new} = RotZ_{old} + 180°$$
$$BendY_{0,new} = 0$$
$$RotX_{0,new} = 180° - RotX_{0,old} + RotX_{1,old}$$
$$RotX_{1,new} = 0$$

$$RotX_{0,new} \rightarrow [-90°, 90°]$$

Where

$$RotX_{0,new} \rightarrow [-90°, 90°]$$

is a transformation that will arrange for $RotX_0$ to be within this restricted angular range. So, $RotX_0$ will have to be changed by $\pm 180°$, if necessary; this will be compensated by inverting all *BendY*values ($BendY_0$ is not affected as this value has previously been set to 0 already).

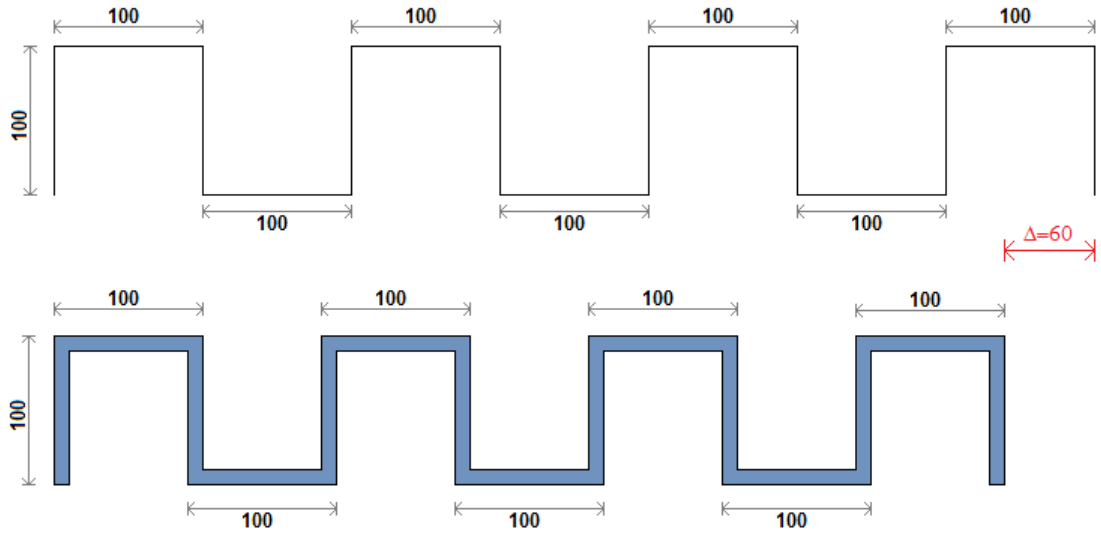
3.10.16.14 Conversion from or to BVBS-BF2D

In the BVBS format, there is an option of specifying a bending radius for each bend. Deviating from the BVBS specification, some machines will interpret all bends with a specified bending radius as a spiral (and a specified length, if any, will then be read as arc length). For export to BVBS, it is thus recommended not to write the bending radii into the geometry block, but rather to specify the same by indicating the bending roll in the header block instead: The bending roll diameter will be set to twice as much as the largest bending radius indicated. The radius details for spirals must of course be specified as a radius in the geometry block; the value of the spiral diameter is taken from the L parameter of PXML segment (see Section 3.10.16.10).

3.10.16.15 Conversion from or to BVBS-BF3D

The BVBS specifications for BF3D are (as opposed to BF2D) anything but complete, consistent or conclusive[57]. It is thus not surprising that many mutually incompatible implementations are being circulated.

Something that is inconsistent, for example, is the concept of whether to consider outer dimensions or center line dimensions in BF3D. In principle, outer dimensions are common practice in BVBS. However, there is a prevailing opinion by the majority that center line dimensions should be used for BF3D. As a matter of fact, the use of outer dimensions in BF3D is rather out of the ordinary as the BF3D vectors are not within or on the iron path anymore as we can see from the following example:



(The BF3D format is shown on top, the real iron geometry is shown below, when assuming that outer dimensions are specified in BF3D.)

Something that is also inconsistent is the representation of bending angles beyond 90°. Here, there are at least three variants that are being used:

a) To conceive the BF3D vectors as pure length vectors or direction vectors (however, this is not consistent anymore for bending angles of 180°).

[57] Here, we refer to the generally accepted BVBS specification of 2000 – we are not aware of any more recent versions.

b) To divide the angle in partial angles whereby each partial angle is less than or equal to 90°.

c) Polygonal representation (by analogy with the polygonal representation in PXML). From a certain size of an angle (135°, for example), the bending angle is divided in parts (as bending angles close to 180° are not feasible anymore in this representation).

Another issue yet to be clarified is how to represent the absolute position of the rebar. Are the "bar block" entries to be used for that? ("X", "Y" or "E"?). Some implementations introduce separate private fields x/y/z in the "Private" block.

Due to this divergence of opinions, it is hence not possible to provide a consistent definition for BF3D. However, when we try to combine the most common BF3D concepts in one shared perspective, we arrive at the following definition which is at least compatible with many major existing implementations:

1) BF3D considers center line dimensions (viz. not outer dimensions).
2) The geometry of the BF3D vectors coincides with that of the polygonal representation in PXML. Bending angles beyond 90° can be divided as previously described for the polygonal representation in PXML. Bending angles close to 180° must be divided. Vice versa (as for the polygonal PXML representation), short segments in between two bends must be seen as nothing but pure auxiliary segments; at the machine the two bends must be combined into one single bend once again.
3) If absolute coordinates of the rebar are to be specified, this will be done in the "Private" block using the fields "x", "y" and "z".

3.10.16.16 Conversion from or to 3D-BF2D

Due to the significant weaknesses of BF3D, it would be better doing completely without BF3D and use a slightly extended BF2D instead. In fact, it is sufficient to provide the possibility of specifying a *RotX*, of course only in those cases in which this is required. The extension proposed here provides the option of specifying a *RotX* value separated by"~" from the bending angle parameter in BF2D. Example:

w30~90@

In this example we have *RotX*=90° and *BendY*=30°.

For compatibility with the standard BF2D, however, it is absolutely necessary to omit the *RotX* addition if *RotX* (almost) equals 0.

3.10.17 Canonical Bar representation

As already mentioned, a given *Bar* geometry doesn't have a unique *RotZ/RotX$_i$/BendY$_i$* representation: different combinations of *RotZ/RotX$_i$/BendY$_i$* may result in the same geometry.

We have mentioned some additional recommendations which lead to a quite unique representation, and we are going to formalize these rules here. However, it must be pointed out that it is not strictly required to follow these rules. They are only meant to define a geometry's representation which is unique and is easy to be processed by legacy systems that deal with 2D information only.

The rules are[58]:

1) $RotX_1 = 0$. [59]
2) $RotX_i \in [-90°, 90°] \quad \text{for} \quad i > 0$.
3) $BendY_0 \in [-90°, 90°]$.
4) Observe one (and only one) of the two following restrictions:
   a. $RotX_0 \in [-90°, 90°]$ (→ **Legacy Canonical** form)
   b. $BendY_1 > 0$ (→ **Separation Canonical** form) [60]

If a *Bar* representation fulfills these conditions with variant **4a**, we call it a **Legacy Canonical** form. If the conditions with variant **4b** are fulfilled, we call it a **Separation Canonical** form.

The *Legacy Canonical* form is the friendliest form for legacy systems since it minimizes the use of all *RotX$_i$*, including *RotX$_0$*. With this convention, most bars can be represented without using *RotX$_i$* values at all.

However, the condition *4a* is somewhat problematic from a fundamental point of view since it is not invariant against modifications of the bar placement: modifying the rotation placement of a bar may break condition *4a* and for getting it fulfilled again it would be necessary to change the internal representation of the bending shape (by inverting all *BendY$_i$*). This means, that condition *4a* breaks the separation of *placement* and *internal shape*.

This problem can be solved by applying condition *4b* instead of *4a*: this leads to a uniquely defined "inner" shape representation of the bar which is completely independent from the bar's placement[61].

Restrict *BendY$_0$* or *RotX$_0$*:

In the above specification, *BendY$_0$* is constrained to [-90°,90°]. Without this condition one could restrict *RotX$_0$* to [-90°,90°] without losing the mentioned rotation invariance. But then, for example, quite normal cranked bars aligned in x-direction would have a representation with *RotZ=BendY$_0$*=180°, which would generally be considered disturbing. The *Separation Canonical form* has therefore been defined to accept *RotX$_0$*=180° rather than *BendY$_0$*=180°.

However, if, in special legacy applications, the restriction of *RotX$_0$* becomes more important than that of *BendY$_0$*, one can take advantage of the following fact:

$$D_{BendY}(BendY_0) \cdot D_{RotX}(RotX_0) \cdot D_{RotZ}(RotZ) = D_{BendY}(BendY_0 + 180°) \cdot D_{RotX}(180° - RotX_0) \cdot D_{RotZ}(RotZ + 180°)$$

I.e., you can change *RotX$_0$* to *RotX$_0$*–180° if you increase *BendY$_0$* and *RotZ* by 180° at the same time. This way you can change a *Separation Canonical* form into a form with restricted *RotX$_0$*.

---

[58] We assume here that all $BendY_i \neq 0$ for all $i > 0$. Without this condition, you could insert additional segments at any point, to which no real geometry corresponds. If there are such segments, you can always take them away by joining them with the previous segment.

[59] I.e., the first bending plane dominates the representation. Since most bars have only one bending plane, bending plane rotation is avoided completely for these cases.

[60] If there is only 1 segment, condition *4b* is always to be considered as fulfilled.

[61] Having *BendY$_1$>0* is not important by itself. The reason for introducing this condition is only for getting unique *RotZ, RotX$_0$, BendY$_0$*. Without restricting *BendY$_1$* to positive values, it is always possible to add 180°a given *RotX$_0$* and compensate this by inverting all *BendY$_i$*.

3.11 Girder

3.11.1 PieceCount, X, Y, Z, GirderName, Length, AngleToX

AngleToX specifies the direction in the xy-plane (starting from the x-axis).
The values are within the range of]-180°, 180°].

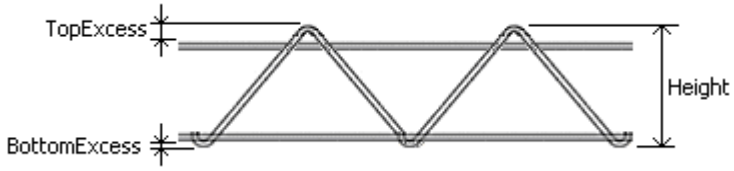
X and *Y* refer to the X/Y position of the top flange (core dimension of the top chord).

Z refers to the Z position of the bottom flanges (core dimension of the bottom chord).

3.11.2 NoAutoProd

This must be set if the lattice girder is *not* to be produced automatically.

3.11.3 Height, TopExcess, BottomExcess



Height is the Z difference between the lowest point and the highest point.
TopExcess is the Z distance of the highest point to the upper edge of the upper chord.
BottomExcess is the Z distance of the lowest point to the bottom edge of the bottom chord.

3.11.4 Weight, TopFlangeDiameter, BottomFlangeDiameter

TopFlangeDiameter and *BottomFlangeDiameter* are given in mm; *Weight* is given in Kg.

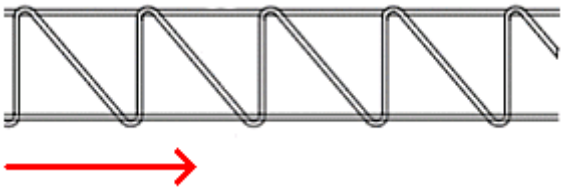
3.11.5 GirderType

GirderType is a numerical Type identifier. It is recommended to use the following values:

0 = standard girder (without any additional definition).
1 = general basic reinforcement girder.
2 = shear force girder.
3 = supplementary girder.
6 = Versagirder
7 = left-sided Versagirder
8 = right-sided Versagirder

This corresponds to the unit digit of the UNICAM type of girders.

Orientation of the shear force girders: the orientation of the shear force girders is defined such that the inclined flanks are sloping:



3.11.6 MountingType

Mounting instruction:

0 = no indication.

1 = to be fixed manually (rework required).

2 = to be placed manually.

This corresponds to the tens digit of the UNICAM type of girders.

3.11.7 ArticleNo

Article designation of the lattice girder.

3.11.8 Machine

Text field to specify the production machine.

A machine-internal production list may optionally be treated as a separate machine of its own. In such an instance, it is recommended to select the following format:

GTA:2

In this example, "GTA" specifies the machine as such, and "2" is the number of the production list.

3.11.9 Period, PeriodOffset

In the *Period* field, the lattice girder period can be specified (in mm). Here, a value of 200 would be typical. If the value is not specified, or if it is 0, an application-dependent default must be assumed.

The *PeriodOffset* field specifies at what distance from the beginning of the lattice girder there will be the first period low point. For an uncut lattice girder, this value will normally be 0; for a grid-compliant cut lattice girder, this value will be either 0 or one half of the period length; for a randomly cut lattice girder, all values are possible, either positive or negative ones.

Export to UNICAM: For export to UNICAM, the fields *Period* and *PeriodOffset* are written into the two spare fields at the beginning of line #5 of the BRGIRDER block.

3.11.10 Width

In the *Width* field, the width of the lattice girder can be entered (in mm); this is measured on the extreme outside on the lattice girder. Here, a typical value would be 80. If this value is not specified, or if it is 0, an application-dependent default must be assumed.

3.11.11 AnchorBar

Anchor rods.

Type: type identifier.

Length: length in mm.

Position: laying position in mm.

3.11.12 GirderExt

Generic sub-table of the lattice girder.

The specific meaning of the fields will be dependent on the type.

For all types, there is the **Position** field: this describes a position in mm, measured from the beginning of the lattice girder.

The **Flags** field holds 32 bits that may be used type-specific and/or application-specific. The meaning of the fields **Val0, …, Val3** is also type-specific and/or application-specific.

3.11.12.1 Type = splicePos

Splice point at which the lattice girder was split and subdivided at the lattice girder position.

- Val0: phase jump mm at the welding point. This is equivalent to the length of lattice girder that was 'omitted' at the splice position. This value may also be negative.
- Val1: length of overlap at the splice point. There will be a double lattice girder along this line segment. This value may also be negative.

3.11.12.2 Type = FixingPos

The point at which the reinforcement is fixed in place (welded or tacked).

The values Val0, …, Val03 will depend on the application. For applications with fixed welding tables, the welding program will be provided Val0. In other applications, the welding current, the welding time or similar may be provided.

3.11.12.3 Type = GirderGripPos

The position at which the lattice girder will be gripped for installation (gripping position for handling a single lattice girder).

3.11.12.4 Type = MeshGripPos

The position at which the lattice girder will be gripped when the element will be handled.

3.11.12.5 Type = SupportPos

The position at which a supporting spool piece must be installed.

3.11.12.6 Export to UNICAM

For export to UNICAM, the GirderExtRows will be entered as lattice girder welding points. The welding output will contain the type information:

- Welding output = **0xx**: *Unknown*.
- Welding output = **1xx**: *SplicePos*.
- Welding output = **2xx**: *FixingPos*.
- Welding output = **3xx**: *GirderGripPos*.
- Welding output = **4xx**: *MeshGripPos*.
- Welding output = **5xx**: *SupportPos*.

The value of **'xx'** will be filled with ***k = Round(Val0 / 5.0)***. To make sure that *k* will be within the range of 0 to 100, the following transformation is done additionally: *k* is restricted to [-50, 49], and 100 is added for negative values (that is to say, xx = 93 will be set for *k* = -7).

3.11.13 Section

The period pattern of the lattice girder can be specified by using the *Section* table. If no *Section* entries are given, the higher-ranking specifications in the *Girder* entry or default values respectively will applied.

A *Section* entry describes a period section that always begins with a period low point, and ends with a period low point.

3.11.13.1 Fields in the Section table

- **L**: Length of a period section in mm. Typical values are around 200 mm.
  The value should always be positive: $L > 0$.

- **S**: Shift of the top diagonal wire point in relation to the center of the period section. For a normal lattice girder, this value will be 0. For shear force girders, this value will typically be around -100. However, the value should be confined to be within the related section, i.e. $|S| \leq \frac{L}{2}$.
  The value of *S*, however, describes the said displacement in an "idealized" form: if ***V*** is the actual displacement (in mm), *S* is defined as follows:
  $$S := V \cdot \frac{L}{L - 2 \cdot (r_B + r_T)}, \quad \text{hence } V = S \cdot \frac{L - 2 \cdot (r_B + r_T)}{L}$$
  Here r_T and r_B are the upper and lower diagonal wire bending radii. I.e. if the bending radii are taken into account, the actual displacement is less than the value indicated in *S*.[62]
- **F**: Length of the flat part (generally only for shear force girders with a low height and double welded bottom chord). The flat part is always located on the one side of the section, where the diagonal wire is less steep. In other words, if S is negative, the flat part is on the positive end of the section (and vice versa). Moreover, the restriction should always be $0 \leq F \leq 2 \cdot |V|$ observed[63].

3.11.13.2 Assignment of the Section data

Section entries are assigned to the effective lattice girder sections according to the following rules:

a) The *Section* entries describe a recurring pattern. If, for example, the three *L*-data are specified to be 195, 195, 200, this means that the length of the first two periods is 195 mm each and that the length of the third period is 200 mm. The fourth and fifths periods will then have a length of 195 mm again, and so forth.

b) The *Section* data merely describe a pattern, they do not provide any information regarding the actual length of the lattice girder. This is rather defined in the *Length* field of the *Girder* entry.

c) The period pattern normally starts at the beginning of the lattice girder. If, however, a *PeriodOffset* value other than 0 is specified, the beginning of the pattern will be shifted accordingly[64].

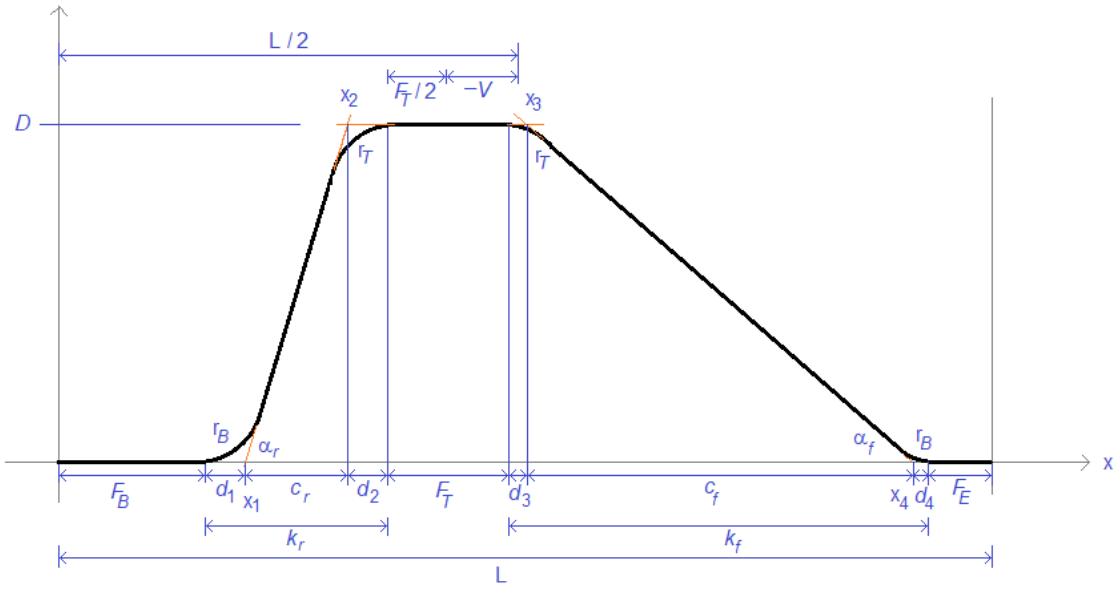
Calculations for girder shapes

The following calculation analyzes the shape of the diagonal wire in its upright position:

[62] This "idealized" definition for *S* has the following motivation: the actual displacement *V* depends on the bending radii. However, these may not be known, so it is advantageous to define the value *S* so that it is typically independent of the bending radii. In concrete terms, a value of *S=L/2* always leads to a completely vertical diagonal wire section, independent of the bending radii.

[63] The flat part is meant to avoid that shear force girders with a small height value have too weakly rising diagonal wires. That's why the flat part is located on the side where the weakly rising diagonal wire is.
The restriction $F \leq 2 \cdot |V|$ helps to avoid two unnatural conditions: first it avoids that *F* would invert the shear direction and second it excludes the undefined case where *F* is positive and *S* is *0* (in this case the position of the flat part would be undefined). The limit case of $F = 2 \cdot |V|$ refers to a diagonal wire with symmetrical shape.

[64] Please note: If the value of *PeriodOffset* is a positive value, the beginning of the period pattern will be shifted into the lattice girder. The first section of the lattice girder will thus be positioned upstream of the beginning of the period pattern. However, this section will still have a defined structure as the pattern recurs periodically, and is thus also defined upstream of its starting point. (Upstream of the beginning of the pattern, the end of the same pattern will string together just because the pattern recurs in an infinite sequence).



The PXML data provides the following values[65]:

F_B, F_T, F_E, L, V, r_B, r_T, D

These values define the whole geometry as follows:

$$k_r = \frac{L}{2} + V - \frac{F_T}{2} - F_B, \qquad k_f = \frac{L}{2} - V - \frac{F_T}{2} - F_E$$

$$d_1 = r_B \cdot \tan\frac{\alpha_r}{2}, \qquad d_2 = r_T \cdot \tan\frac{\alpha_r}{2}, \qquad d_3 = r_T \cdot \tan\frac{\alpha_f}{2}, \qquad d_4 = r_B \cdot \tan\frac{\alpha_f}{2}$$

$$c_r = \frac{D}{\tan\alpha_r} = \frac{D \cdot (1 - \tan^2\frac{\alpha_r}{2})}{2\tan\frac{\alpha_r}{2}}, \qquad c_f = \frac{D}{\tan\alpha_f} = \frac{D \cdot (1 - \tan^2\frac{\alpha_f}{2})}{2\tan\frac{\alpha_f}{2}}$$

With $t_r := \tan\frac{\alpha_r}{2}$ and $t_f := \tan\frac{\alpha_f}{2}$ we get:

$$d_1 = r_B \cdot t_r, \qquad d_2 = r_T \cdot \mathrm{t}_r, \qquad d_3 = r_T \cdot \mathrm{t}_f, \qquad d_4 = r_B \cdot \mathrm{t}_f$$

$$c_r = \frac{D \cdot (1 - \mathrm{t}_r^2)}{2 \cdot \mathrm{t}_r}, \qquad c_f = \frac{D \cdot (1 - \mathrm{t}_f^2)}{2 \cdot \mathrm{t}_f}$$

Because of

$$k_r = d_1 + d_2 + c_r, \qquad k_f = d_3 + d_4 + c_f$$

We get

$$k = g \cdot t + \frac{D \cdot (1 - \mathrm{t}^2)}{2 \cdot \mathrm{t}}, \quad g := r_B + r_T$$

($k := k_r$ and $t := t_r$ or $k := k_f$ and $t := t_f$)

Now this can be solved for t:

$$t = \frac{\sqrt{D^2 - 2Dg + k^2} - k}{D - 2g}$$

[65] With the current PXML definition the value of F_T is always *0* and only one of the values F_B or F_E is can be different from 0. However, the following analysis considers the more general case of arbitrary nonnegative values of F_T, F_B, F_E. r_B and r_T are not directly given in PXML. Typically, they are assumed to be $\frac{5W}{2}$, where W is the diameter of the diagonal wire.

3.12 Alloc

An *Alloc*block will describe a staggering pattern for rebars or lattice girders respectively.

This is specifically used to describe *cages*: here, the *Alloc*block will describe the laying pattern of the stirrups.

Type-Attribute

The following types are available:

- **Bar:** The *Alloc* block relates to rebars (stirrups, staggered rebars).
- **Girder:** The *Alloc* block relates to lattice girders.

3.12.1 GuidingBar

If there is a valid rebar index in the *GuidingBar* field, staggering will occur along this rebar. The *Region* values will then define the guiding positions on this guiding rebar, i.e. the "guided" rebars will be arranged at these guiding positions along the guiding rebar.

Definitions:

- **Guiding bar:** the rebar that specifies the path for staggering.
- **Guiding positions:** those points or positions on the guiding rebar at which the guided rebars will be arranged.
- **Guided bar:** the rebar that is to be "guided" by the guiding rebar.
- **Allocated bars:** those rebars that are positioned at the real points or positions that are derived from the staggering scheme.

The following rules will apply:

a) The path defined by the guiding bar will run along the rebar core and will have the bending radius of 0 if there are bends.

b) To determine the coordinates of the allocated bars, add the *relative staggering position coordinates* to the rebar coordinates. Here, the *relative staggering positions* are the staggering positions on the guiding bar, related to the beginning of the guiding bar. The original (unstaggered) guided bar must hence be positioned at the starting point of the guiding bar.

c) If the guiding bar has bends, the staggered rebars will be additionally rotated. Such rotation will occur after shifting and around the guiding position. For each guiding bar bend, and up to the respective guiding position, rotation will then occur around the axis of the bend[66].

3.12.2 Determination of direction excluding GuidingBar

If there is no *GuidingBar*, the direction will be determined by the staggered object itself:

The rebar will determine the starting point via the respective X/Y/Z coordinates of the *Bar* items. The direction will be determined as follows:

1) Rotate the coordinate system around *RotZ* (viz. around the positive z-axis).

[66] Here, the first *BendY* of the guiding bar should *not* be read as a bend; that is to say, this only refers to the bend within the rebar.
At this juncture, it may come as a surprise that *RotZ* and *BendY0* of the guiding bar are not included in the allocatedrebars. On the one hand, there are practical reasons for that as it would be rather non-descriptive if *RotZ* of the guided bar would not be absolute, but relative in relation to the guiding bar. On the other hand there are actually rather fundamental reasons that prevent us from including *RotZ* and *BendY0* of the guiding bar in the allocatedrebars: *RotZ* and *BendY0* are not unambiguous for a given guiding bar as there are different combinations of these values that are equivalent for the guiding bar, but that would bring about different geometrical relations for the allocatedrebarsif they would be included.

2) Rotate the coordinate system around $RotX_0$ (viz. around the positive x-axis).
3) Rotate the coordinate system around $BendY_0$ (viz. around the negative y-axis).
4) Rotate the coordinate system around $RotX_1$ (viz. around the positive x-axis); do this only if there is more than one segment.
5) The negative y-axis will then face positive pitch values. This is the axis via which the first bend will be defined.

Please note: in UNICAM, the pitch value is defined differently for longitudinal rebars and transverse rebars respectively[67].

- For rebars in a +X direction (0°): UnicamPitch = –PxmlPitch,
- For rebars in a –X direction (180°): UnicamPitch = PxmlPitch,
- For rebars in a +Y direction (90°): UnicamPitch = PxmlPitch,
- For rebars in a –Y direction (-90°): UnicamPitch = –PxmlPitch, and
- For all other angles: single rebars must be specified in UNICAM.

The UNICAM definition appears to be somewhat more familiar, but cannot be consistently defined for generic angles.

The same applies similarly to **lattice girders**: the starting point is specified via the lattice girder coordinates: the direction of the pitch is defined as follows:

1) Rotate the coordinate system around AngleToX (viz. around the positive z-axis).
2) The negative y-axis will then face the direction of positive pitch values. This is the axis via which the first bend will be defined.

Again, we see the same difference to the UNICAM definition as mentioned above.

3.12.3 Region

The staggering pattern is composed of a sequence of *Regions*: each *Region* includes the following information:

- **IntervalCount:** number of intervals.
  This value is integer and nonnegative.
- **Pitch:** Interval width in mm. This value may also be non-integer, or may also be negative respectively.
- **IncludeBegin:** If this is set, the beginof the interval is included in the region.
- **IncludeEnd:** If this is set, the end of the interval is included in the region.
- **RefIndex:** Integer value that determines the referenced item (rebar or lattice girder respectively). A value of 0 means that the first rebar (or the first lattice girder respectively) of the respective *Steel* block is referenced.
  If *RefIndex* is NULL, negative, or too large, it is deemed not allocated.
  The *Region* will then define an interval offset without, however, allocating a rebar.

Comments:

i) The **length** of the *Region* is derived from **L = IntervalCount * Pitch**.
ii) A *Region* will define positions.
The position *k* of the *m*-th Region will be computed as follows:
$P_{m,k} = \sum_{i=0}^{m-1} L_i + k \cdot Pitch_m\,, \quad L_i = IntervalCount_i \cdot Pitch_i.$
At first, the position thus computed is merely a scalar (linear) value. Via the procedures

[67] For example, refer to Unitech Interface Definition, 4.9.5, Item 10: "Positive pitch means positive coordinate direction, …, use of the pitch is ONLY allowed for angles of 0°, 90°, 180° or 270°."

from Sections 3.12.1 and 3.12.2, a specific point in space can then be determined on this basis.

iii) If the *Region* has no valid *RefIndex*, no stirrups have been allocated to it[68].

iv) If the *Region* has a valid *RefIndex*, at least *IntervalCount-1* stirrups will be allocated. If *IncludeBegin* is set, another stirrup will be added. If *IncludeEnd* is set, another stirrup will be added, too. Thus, a maximum of *IntervalCount+1*stirrups can be allocated to the *Region*.

v) Usually, *IncludeBegin* of a region and *IncludeEnd* of the preceding region will be synchronized. That is to say, real freedom only exists at the beginning of the first *Region* and at the end of the last *Region* respectively. However, such synchronization is not absolutely required.

vi) Due to the above pattern, an *Alloc*block will yield a quantity for a stirrup referenced in the same. Now, this quantity, which should be called **AllocCount,** is computed as follows:

$$AllocCount = \sum_{i=0}^{RegionCount-1} D_i \cdot (IntervalCount_i - 1 + E_i + B_i), \quad AllocCount >= 0$$

$B_i = 1$ if $IncludeBegin$ is set

$E_i = 1$ if $IncludeEnd$ is set

$D_i = 1$ if the respective stirrup occurs in Region_i, otherwise 0.

Please note: The*AllocCount*value is nonnegative by definition although it could be negative for *IntervalCount=0* in absolute terms of figures.

Please note: An *Alloc*block will define its own *AllocCount* value for each stirrup that is referenced in the *Alloc*block.

vii) The total of the *AllocCounts* for one stirrup specifies how many stirrups have an allocated guiding position. Usually, this number will be the same as the number of stirrups (*Bar.PieceCount*). However, this compliance is not absolutely required, that is to say, *Bar.PieceCount* may have excess or missing stirrups.
If there are any **excess stirrups**, these will be allocated to their original X/Y/Z position; that is to say, these excess stirrups will be considered separately from the *Alloc*blocks.
If there are any **missing stirrups**, the last stirrups of the last *Alloc*blocks will be deemed to be missing.

viii) A *Region* with ***Pitch=0*** can be considered in terms of figures just like any other *Region*. This special case is usually used to represent stirrups that are not welded, but that are included in the supply unmounted (typically at the front or rear ends of a cage).

ix) The definition of the *Alloc* or *Region* data structures will allow for one rebar to be referenced in several *Regions* that may even be in different *Alloc*blocks. Usually, however, a rebar will only be referenced in *Region* blocks that belong to one single *Alloc*block only. That is to say, the rebar usually has no more than one staggering pattern (=*Alloc*block) only. To the contrary, one staggering pattern often has several different rebars.

3.13 SteelExt

Additional entries to the Steel block. The type of these entries is application-dependent and will be differentiated via the *Type* attribute. The meaning of the *Info* field will depend on the *Type* attribute. In addition, there will usually be application-specific fields (I_P_-Tags).

[68] For the sake of simplicity, only *stirrups* are mentioned here. However, this is similarly true for other staggered rebars or for lattice girders respectively.

3.14 Feedback

The **Feedback** Table is essentially different from the other PXML Tables: the *Feedback* Table is not intended to describe any production data, but rather includes the machine feedback. As will be explained in more detail below, there are two types of feedback:

- *PTS* message: will provide information regarding the producibility of certain Items in advance.
- *Machine-Return*: will provide information regarding any production done.

For data transfer from the CAD to the machine, there will usually be no *Feedback* Table. For the feedback from the machine (or the test system respectively) to the CAD, there will often only be the *Feedback* Table, but no other data; in some cases, however, the machine may include production data in a PXML feedback file as well.

3.14.1 Production Test Service (PTS)

The **Production Test Service** (abbreviated **PTS**) allows a CAD system or control system to issue a query to the production system to scan forproducibility of a product. Data may be generated under the direct involvement of the machine capabilities; thus, we can achieve a high level of process reliability and optimality of the production sequence.

Demand for PTS systems has specifically arisen in connection with modern mesh bending facilities. These facilities have an ever increasing scope of performance which, however, cannot be described by way of a few threshold or limit values due to the increasing complexity of such facilities. The fully automatic producibility of a complex reinforcement cage will depend on such a variety of details that the same can only be reliably reviewed under the direct involvement of the machine software; and this is exactly what is to be achieved by means of the *PTS*.

PTS is composed of two parts:

- **PTS Server:** this is a service application that must be provided by the machine manufacturer; this service will run on the CAD workstations, or will be centralized somewhere within the network respectively. The *PTS s*erver will receive and respond to queries; to this end, it must take into consideration the computational logic and the parameters of the machine software.
- **PTS Client:** this must be provided by the CAD manufacturer in the CAD application (and / or in the control system). By means of the *PTS* Client, queries are issued to the *PTS* Server from within the CAD application. The server response must then be visualized in the CAD system.

Allocation of a response to the request previously sent will occur via the *GlobalID* of the *DocInfo*Table. This ID should thus be set individually for each request.

The **Request** of the *PTS* Client to the PTS Server will be sent in the format of a PXML document that will contain production data, viz. *Order* blocks and the associated child structures.

The **Feedback** of the *PTS* Server will be sent in the format of a PXML document that will contain *Feedback* blocks. If the *PTS* Server modifies any production data of its own account, it can feed back *Order* blocks in addition to the *Feedback* blocks in an attempt to describe the data thus modified[69]. However, such feedback of the *Order* blocks (= production data) should be seen as an optional additional function; the central element of the feedback are the *Feedback* blocks as such.

[69] Naturally, the production data fed back must make reference to the original data via *GlobalID*. If an Item of the original data in the feedback is referenced several times (because an original rebar had to be split into several items, for example), the same *GlobalID*must be specified in all relevant feedback items.

3.14.2 Machine Return

Machine Return is a functionality that is closely related to the *PTS* feedback. However, as opposed to *PTS,* this is not a test feedback, but rather a production feedback of the production machines to the higher-ranking control system. Here, for the most part, produced quantities will be fed back.

If the production machine modifies any production data of its own account, it may be helpful to also specify the modified production data in the feedback (viz. *Order* blocks plus child structure). This (optional) functionality may even provide for data that are entered in the machine manually be maintained back to the higher-ranking control system.

A further application area of production feedbacks is used by reporting events from MES systems to the higher-ranking ERP system.

3.14.3 Fields of the Feedback Block

3.14.3.1 Feedback attributes

1) **ItemType**: ... specifies the object type to which the feedback relates. Typically a PXML Table name is specified here.
2) **GlobalID**: ... specifies to which production data Item the feedback relates. To be able to unambiguously allocate a *Feedback* entry, we need both, the *ItemType* and the *GlobalID*. Merely both of these together will identify the production Item concerned.

Both attributes mentioned are to be specified precisely once for each *Feedback* block.

3.14.3.2 Feedback fields

1) **MessageType**: ... defines the type of the message. The following values are available:
   a. **info**: Pure information only, no warning character. "info" is also the default *MessageType* that will be assumed if there is no specification.
   b. **hint**: Hint at a low warning level. For example, this will be used for items that can be produced without any problem, but for which attention is to be drawn to the fact of lack of optimization in relation to the production speed.
   c. **warning**: Warning of medium level. This item can be produced, but with some difficulty only; it may be of reduced quality, or it may impose major strain on the machine while being produced.
   d. **error**: Error. This item cannot be produced the way it is proposed (viz. this item will not be produced at all, or with a strongly modified shape only).
   e. **program**: Feedback which is intended for automatic evaluation by the PTS client. It isn't primarily meant to be displayed to the user. Typically, such messages do not have a description text.
2) **Code**: Alphanumerical code that identifies the meaning of the *Feedback* message. The list of codes including their assigned meaning will be provided by the machine manufacturer. For *PTS* messages, it is recommended to specify unambiguous error codes via the *Code* field. Within the limits of *Machine Return* messages, the *Code* may be used to identify feedback events. Here, typical codes would include *imported* (imported in terms of data), *started* (production has started), *produced* (produced as a single item; for concrete elements: concreted), *embedded* (installed, welded; for concrete elements: warehoused), *delivered* (delivered, transferred to the downstream system; for concrete elements: taken out of the warehouse / lifted).
   For MES→ERP feedback events and codes a fully standardized specification has been given in the **PXML Web API** (see www.pxml.eu).

3) **InfoValue**: Additional information for *Feedback* message. A numerical or alphanumerical value can be provided. The interpretation of this value depends on the *Code* field.
4) **PieceCount**: Quantity (integer value); ... specifies how many pieces of the specified item are fed back. Typically, this will read "1".
   This field is used for messages of the type of *Machine Return*, and will typically not be present in *PTS* messages.
5) **MaterialType**: Type of the material used.
   For *Bar* items, typically the wire diameter is entered here; if there are different wire types to any one diameter, a more generic alphanumerical wire ID will be entered here.
   For lattice girders, typically the girder type is entered here.
   This field is used for messages of the type of *Machine Return*, and will usually not be present in *PTS* messages.
6) **MaterialBatch**: Batch identification of the material used.
   The following format is recommended for round-bar steel from coil:
   "12345@AR177228C". Here, "12345" is an ID to unambiguously identify the coil; "AR177228C" is the steel batch. As the coil batch can be derived from the ID, it is often useful to merely specify the ID only (viz. only "12345"); of the ID is not known, it will be adequate to merely specify the batch only, starting with a field separator '@' (viz. "@AR177228C").
   This field will be used for messages of the type *Machine Return*, and is usually not present in *PTS* messages.
7) **MaterialWeight**: The weight of the material used in kg (total weight for all units produced).
   This field is used for messages of the type of *Machine Return*, and is usually not present in *PTS* messages.
8) **ProdDate**: Production timestamp (production end point). This is typically specified in *Machine Return* messages to record the precise time of production.
9) **Machine**: Identification of the machine or of the PTS server that generated the message. This field is particularly important if several PTS server are connected in series. (If within a physical machine there is the requirement to distinguish between different production lists, a format like MMM:X is recommended. This is the same recommendation as has been made for other PXML machine fields. In this sample MMM identifies the physical machine and X identifies the production list within that machine).
10) **Description**: Optional text entries to describe the message in text format. Several languages may be specified. Here, language codes must be specified by means of ISO 639 for the *Culture* attribute, optionally extended by the ISO 3166 country codes; viz., we will have indications in the format "en" or "en-US" respectively.
    The description text may also be long or multi-line. In addition to the error message as such, there may be recommendations for correction, or information on corrections made automatically.
    For example: "Structural module Y=50mm not satisfied. Rebar shifted by -25 mm in Y."
    A *Feedback* entry should have merely one *Description* entry for each language only.

In *Machine Return* messages in which merely the produced number of pieces is reported, the *MessageType* will not be specified, or will be set to "info". However, a production message or a warning may be included as a matter of principle such that a *Machine Return* message may also be combined with other *MessageType*s.

3.14.4 FbVal

For *Feedback* messages it is possible to include additional values with *FbVal* entries. Each *FbVal* entry has two attributes: a type *T* and a value *V*.

The following tables describe the *FbVal* types and values defined by the standard.

T	Description	Example
OrdNo	Order number	AA00048386
Customer	Customer identification	XyConstruction
ElemNo	Element number	5522A
PartType	Element part (typically used with DW/TW elements)	01
Pos	Denomination of the item position	17B
ShiftID	ID of the production shift to which the feedback is referring to.	MorningShift
ShiftStart	Start time of the production shift to which the production feedback is related. The shift's start is especially relevant for evaluations dealing with shifts crossing midnight; for such shifts the associated shift day may be different from the day of the timestamp reported in the *ProdDate* of the feedback entry.	2017-02-20T16:32:33+01:00
PalDataID	Identification of the logical production unit	123456
PalNo	Identification of the physical production unit	43
Size	Nominal dimension as requested by CAD data. Typically the nominal length in mm. For meshes: typically X and Y dimensions are in a form 5740x1320. The nominal dimension together with the nominal type should contain enough information for the adding inventory posting.	600
ProdSize	Nominal dimension as actually produced. This value may differ from the *Size* value in cases where the machine modifies dimensions in order to bring them in the range accepted by the machine.	800
NomTy	Requested production type (from CAD). In "Bar" entries a Bending form denomination could be used In "Girder" entries the field usually contains the girder type.	M10
ProdTy	Type which has actually been produced. Could deviate from NomTy. For "Girder" entries the field typically indicates the parametrized girder type name in the machine (this type name is usually linked to the NomTy over Alias) For "Bar" entries the field could indicate the bending form denomination or the number of bendings in a form "B5" for a bar with 5 bendings.	B5
ProdArticle	ERP Item code of the produced product.	AXL554

Total_Kg	Real total weight in Kg of the actually produced item.	9.123
Wr	Wire properties (see Wire information description below). For “Bar” entries the field indicates wire information of the bar itself. For “Steel” entries the field indicates either wire information about one single bar or several bars of the same type that are composed together.	D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=7.756 Art=GD12345
Wr_Tp	Wire properties for the top flange of the girder (see Wire information description below).	D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=7.756
Wr_B1	Wire properties for the left bottom flange of the girder (see Wire information description below).	D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=7.756
Wr_B2	Wire properties for the right bottom flange of the girder (see Wire information description below).	D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=7.756
Wr_D1	Wire properties for the left diagonal wire of the girder (see Wire information description below).	D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=7.756
Wr_D2	Wire properties for the right diagonal wire of the girder (see Wire information description below).	D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=7.756
Wld	Information about one generic welding point (see Welding point information)	X=2663.4 I=9774
Wld_Tp	Information about one top wire welding point (see Welding point information)	P=7322 T=226 X=2563.4 I=9664
Wld_B1	Information about one left bottom wire welding point (see Welding point information)	X=2663.4 I=9774
Wld_B2	Information about one right bottom wire welding point (see Welding point information)	I=9333 T=230 X=2560.4
Wld_Target	Target values of generic welding points (see Welding point info) If it does not contain the X/Y position then the values refer to all welding points of that item.	T=300 I=9800
Wld_Tp_Target	Target values of the top wire welding points (see Welding point info) If it does not contain the distance to the beginning of the girder then the values refer to all welding points of the girder.	T=350 I=10200
Wld_B1_Target	Target values of the left bottom wire welding points (see Welding point info) If it does not contain the distance to the beginning of the girder then the values refer to all welding points of the girder.	T=300 I=9800
Wld_B2_Target	Target values of the right bottom wire welding points (see Welding point info)	T=300 I=9800

	If it does not contain the distance to the beginning of the girder then the values refer to all welding points of the girder.	
Spacer	Information about one spacer (see Spacer information).	H=20 D=95 X=3055.7 Y=282.1
Area	In a “Slab” entry the field indicates the area of the element in m².	15.22
CutBegin CutEnd WeldBegin WeldEnd BendBegin BendEnd PlaceBegin PlaceEnd	Start and/or end time of various production events can be specified. The following events are defined: *Cut*: Cutting on specified dimensions *Weld*: Welding. *Bend*: Bending *Place*: Placing of items on their target location. Typically, there is only one entry per *Feedback* block for each of these events. This means that for a *Steel* block, the information refers to the entire *Steel* block.	2017-02-20T16:32:33+01:00
m2	Produced area in square meters. The value is determined by the machine according to its own criteria. Typically, the wrapping xy-box of the produced part is used.	17.32
RefItem_XX	Referenced item. Here **XX** is only a placeholder and must be effectively replaced by a table name. Specifically, there are types like RefItem_**Bar**, RefItem_**Segment**, etc. The value to be indicated is the GlobalID of the referenced item. Typically, such entries are used in PTS messages that take refer further "causing" items along with the actually affected item. An example would be a bending that cannot be executed because another bar is too close. This other bar would then be a problem-causing item that can be indicated with RefItem_Bar.	12345ABC
HltBarTIntvl	Interval on Bar (given with theoretical length positions) that indicate a section which should be highlighted. This is typically used in PTS feedbacks, for example to highlight a bending where a problem occurs. Since these are positional values on a Bar, it is intended for Bar items, or items that are below the Bar level. The sequence in which the two interval limits are specified may be ascending or descending; the display client should choose a representation in which it is recognizable which is the first and which is the second interval limit.	1320;1420

3.14.4.1 Wire information

Contains a string that is composed of:

Field	Description	Example
D	Diameter [mm]	12
Qlty	Steel quality	BS300
Mtl	Coil-Info@Charge	83@C25408
Art	Article Code of the Raw Material	GD12345
Len	Real length [mm]	5730
Kg	Real weight [kg]	2.756

Example: D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=7.756 Art=GD12345
Note: this wire information is related to the consumed material (raw material)

3.14.4.2 Welding point information

Contains a string that is composed of:

Field	Description	Example
X	Distance X to the beginning of the welded item (e.g. girder or mesh) [mm]	322
Y	Distance Y to the beginning of welded item (e.g. girder or mesh) [mm]	257
I	Welding current [mA]	8185
P	Welding pressure [mBar]	9074
T	Welding time [ms]	231
WH	ID of welding head used	1

Example: X=0 I=9855.5 P=8861 T=221

3.14.4.3 Spacer information

Contains a string that is composed of:

Field	Description	Example
H	Height [mm]	25
D	Diameter [mm]	95
X	X position For „Slab“ entries the position is relative to the element.	4234.9
Y	Y position For „Slab“ entries the position is relative to the element.	122.7

Example: H=25 D=95 X=4234.9 Y=122.7

The string could also contain other internal fields which should have the Prefix “I_”.
The sequence of the fields is not relevant and not every field has to be inserted.
Also the sequences of the different welding points or wires are not bound and every entry is facultative.

3.14.5 Examples of PTS messages

The detailed list of possible *PTS* messages will be defined by the machine manufacturer. Hence, the following list should be merely read as an example only:

Examples of ItemType="Segment":

- Maximum segment length exceeded. [error]
- Segment too short. [error]
- Segment as L0 too short. [error]
- Bending too close as L0 too short. [error]
- Invalid bending angle. [error]

Examples of ItemType="Bar":

- Invalid bar diameter. [error]
- Bar with too few Welding points. [error]
- Bar too close to next bar. [error]
- Bar too close to next bar – corrected. [warning]
- Multiple bars aligned on same y; production will be slow. [hint]

Examples of ItemType="Steel":

- Invalid mesh size (too large). [error]
- Invalid mesh size (too small). [error]
- Mesh not transportable. [error]
- Mesh not transportable – corrected. [warning]

As these messages may not always be clearly visually relatable on the CAD monitor, the *ItemType* should be identifiable from the *Description* text.

The following example shows the PXML document of a *PTS* feedback:

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
  <DocInfo GlobalID="7C7E1FC5-0A46-48c5-A3B2-249D75B70BCF">
    <MajorVersion>1</MajorVersion>
    <MinorVersion>3</MinorVersion>
    <Comment>MSystem PTS 3.77.12.123, List 1, Params 2010-07-12</Comment>
  </DocInfo>
  <Feedback ItemType="Bar" GlobalID="12345">
    <MessageType>error</MessageType>
    <Code>MaxBarLen</Code>
    <Description Culture="en" Text="Maximum bar length exceeded."/>
    <Description Culture="de" Text="Max. Eisenlänge überschritten."/>
  </Feedback>
  <Feedback ItemType="Bar" GlobalID="2057">
    <MessageType>warning</MessageType>
    <Code>MinBarLen</Code>
    <Machine>BGM</Machine>
    <Description Culture="en" Text="Bar too short."/>
    <Description Culture="de" Text="Eisen zu kurz."/>
  </Feedback>
  <Feedback ItemType="Segment" GlobalID="5523">
    <MessageType>error</MessageType>
    <Code>DistCBar</Code>
    <Machine>BGM</Machine>
    <Description Culture="en" Text="Bending too near to crossing bar."/>
    <Description Culture="de" Text="Biegung zu nahe an querendem Eisen."/>
    <FbVal T="HltBarTIntvl" V="1320;1420"/>
  </Feedback>
</PXML_Document>
```

In the above example, we see 2 *PTS Feedback* messages: an error message for rebar "12345", and a warning for rebar "2057". In this example, "MinBarLen" would be a warning code relating a rebar that has been automatically extended by the machine; this rebar can thus be produced in the automatically corrected form.

Particular attention should be paid to the *Comment* entry in the *DocInfo*section: here, we should see an identification of the *PTS* server from which it follows when its parameters were last balanced and matched with the machines. Ideally, the *PTS* Client should visualize this information for the user.

3.14.6 Examples of Machine Return messages

The following example shows the PXML document of a *Machine Return* message that reports the production of 2rebars.

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
  <DocInfo GlobalID="7C7E1FC5-0A46-48c5-A3B2-249D75B70BCF">
    <MajorVersion>1</MajorVersion>
    <MinorVersion>3</MinorVersion>
  </DocInfo>
  <Feedback ItemType="Bar" GlobalID="2057">
    <PieceCount>3</PieceCount>
    <MaterialType>16A</MaterialType>
    <MaterialBatch>12345@AR177228C</MaterialBatch>
    <ProdDate>2010-07-30T09:06:03+02:00</ProdDate>
  </Feedback>
  <Feedback ItemType="Bar" GlobalID="2058">
    <PieceCount>1</PieceCount>
    <MaterialType>8</MaterialType>
    <MaterialBatch>12345@AR177228C</MaterialBatch>
    <ProdDate>2010-07-30T09:06:05+02:00</ProdDate>
  </Feedback>
</PXML_Document>
```

The following example shows the PXML document of a *Machine Return* message that reports the production of 3 numbers of an element; here, it is also specified what material has been used and how many kilograms were required.

```
<?xml version="1.0" encoding="utf-8"?>
<PXML_Document xmlns="http://progress-m.com/ProgressXML/Version1">
  <DocInfo GlobalID="7C7E1FC5-0A46-48c5-A3B2-249D75B70BCF">
    <MajorVersion>1</MajorVersion>
    <MinorVersion>3</MinorVersion>
  </DocInfo>
  <Feedback ItemType="Slab" GlobalID="12007">
    <PieceCount>3</PieceCount>
    <ProdDate>2010-07-30T09:06:03+02:00</ProdDate>
  </Feedback>
  <Feedback ItemType="Slab" GlobalID="12007">
    <MaterialType>10</MaterialType>
    <MaterialBatch>12345@AR177228C</MaterialBatch>
    <MaterialWeight>123.7</MaterialWeight>
    <ProdDate>2010-07-30T09:06:05+02:00</ProdDate>
  </Feedback>
  <Feedback ItemType="Slab" GlobalID="12007">
    <MaterialType>12</MaterialType>
    <MaterialBatch>12345@AR177228C</MaterialBatch>
    <MaterialWeight>162.4</MaterialWeight>
    <ProdDate>2010-07-30T09:06:05+02:00</ProdDate>
  </Feedback>
  <Feedback ItemType="Slab" GlobalID="12007">
    <MaterialType>KTS810</MaterialType>
    <MaterialWeight>98.7</MaterialWeight>
    <ProdDate>2010-07-30T09:06:05+02:00</ProdDate>
  </Feedback>
</PXML_Document>
```

Here, the feedback blocks all relate to the same *Slab* entry. The first feedback block specifies that 3 numbers were produced, the others the material consumption used for this *Slab* entry.

Please note: It is within the discretion of the applications to relate the feedbacks to coarse or fine hierarchy levels. Thus, for example, the feedback can be made solely for the whole *Order* entry, viz. to merely feed back the fact that this *Order* entry has been processed, and maybe to report how much and what kind of material has been used in the process. Other applications, on the other hand, might provide feedback on a *Bar* level. However, when feeding back different hierarchy levels, please note that **summable variables** (such as the weight) must always be **accumulative**: if, for example, *Bar* feedbacks are reported with a weight, and *Slab* feedbacks are reported with a weight, the reported *Slab* weight should only describe the additional weight that is not included in the *Bar* feedbacks.

3.14.7 Examples of FbVal entries

3.14.7.1 Example Bar

```
<Feedback ItemType="Bar" GlobalID="12345">
  <ProdDate>2010-07-30T09:06:05+02:00</ProdDate>
  <Machine>MSR1:2</Machine>
  <FbVal T="OrdNo" V="AA00048386"/>
  <FbVal T="ElemNo" V="5522A"/>
  <FbVal T="Size" V="5740.4"/>
  <FbVal T="ProdTy" V="4"/>
  <FbVal T="NomTy" V="M10"/>
  <FbVal T="Wr" V="D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=2.756"/>
</Feedback>
```

3.14.7.2 Example Steel

```
<Feedback ItemType="Steel" GlobalID="12345">
  <ProdDate>2010-07-30T09:06:05+02:00</ProdDate>
  <Machine>MSystem:1</Machine>
  <FbVal T="OrdNo" V="AA00048386"/>
  <FbVal T="ElemNo" V="5522A"/>
  <FbVal T="Size" V="5740x1320"/>
  <FbVal T="ProdTy" V="4"/>
  <FbVal T="NomTy" V="M10"/>
  <FbVal T="Wr" V="D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=7.756"/>
  <FbVal T="Wr" V="D=16 Qlty=BS300 Mtl=83@C25408 Len=810 Kg=9.003"/>
</Feedback>
```

3.14.7.3 Example Slab

```
<Feedback ItemType="Slab" GlobalID="12345">
  <ProdDate>2010-07-30T09:06:05+02:00</ProdDate>
  <Machine>MeshSpacer:1</Machine>
  <FbVal T="OrdNo" V="AA00048386"/>
  <FbVal T="ElemNo" V="5522A"/>
  <FbVal T="Area" V="5.5"/>
  <FbVal T="Spacer" V="H=25 D=95 X=122.3 Y=122.7"/>
  <FbVal T="Spacer" V="H=25 D=95 X=122.3 Y=650.6"/>
  <FbVal T="Spacer" V="H=25 D=95 X=122.3 Y=1333.2"/>
  <FbVal T="Spacer" V="H=25 D=95 X=1508.7 Y=122.7"/>
  <FbVal T="Spacer" V="H=25 D=95 X=1508.7 Y=650.6"/>
  <FbVal T="Spacer" V="H=25 D=95 X=1508.7 Y=1333.2"/>
  <FbVal T="Spacer" V="H=25 D=95 X=3296.8 Y=122.7"/>
  <FbVal T="Spacer" V="H=25 D=95 X=3296.8 Y=650.6"/>
  <FbVal T="Spacer" V="H=25 D=95 X=3445.7 Y=1234.8"/>
</Feedback>
```

3.14.7.4 *Example Girder*

```
<Feedback ItemType="Girder" GlobalID="12345">
  <ProdDate>2010-07-30T09:06:05+02:00</ProdDate>
  <Machine>Versa1:2</Machine>
  <FbVal T="OrdNo" V="AA00048386"/>
  <FbVal T="ElemNo" V="5522A"/>
  <FbVal T="Size" V="800"/>
  <FbVal T="ProdTy" V="KT82012"/>
  <FbVal T="NomTy" V="KT82011"/>
  <FbVal T="Wr_Tp" V="D=12 Qlty=BS300 Mtl=99@C25408 Len=810 Kg=2.756"/>
  <FbVal T="Wr_B1" V="D=6 Qlty=BS300 Mtl=81@C25422 Len=850 Kg=0.689"/>
  <FbVal T="Wr_B2" V="D=6 Qlty=BS300 Mtl=80@C250 Len=850 Kg=0.689"/>
  <FbVal T="Wr_D1" V="D=5 Qlty=BS300 Mtl=83@C2555 Len=1212.7 Kg=0.95"/>
  <FbVal T="Wr_D2" V="D=5 Qlty=BS300 Mtl=94@C25476 Len=1212.7 Kg=0.95"/>
  <FbVal T="Wld_B1" V="X=0 I=9876.5 P=8862 T=230"/>
  <FbVal T="Wld_B2" V="X=0 I=9855.5 P=8861 T=221"/>
  <FbVal T="Wld_Tp" V="X=100 I=9855.5 P=8861 T=340"/>
  <FbVal T="Wld_B1" V="X=200 I=9876.5 P=8862 T=433"/>
  <FbVal T="Wld_B2" V="X=200 I=9855.5 P=8861 T=255"/>
  <FbVal T="Wld_Tp" V="X=295 I=9855.5 P=8861 T=340"/>
  <FbVal T="Wld_B1" V="X=410 I=9816.5 P=6862 T=221"/>
  <FbVal T="Wld_B2" V="X=410 I=9855.1 P=8861.6 T=533"/>
  <FbVal T="Wld_Tp" V="X=500 I=8153.5 P=9871 T=340"/>
  <FbVal T="Wld_B1" V="X=610 I=9816.5 P=8862 T=113.4"/>
  <FbVal T="Wld_B2" V="X=610 I=9855.1 P=7861.6 T=60"/>
  <FbVal T="Wld_Tp" V="X=700 I=9458.3 P=9848.4 T=340"/>
  <FbVal T="Wld_B1" V="X=790 I=7200.5 P=8862 T=222"/>
  <FbVal T="Wld_B2" V="X=791 I=9855.1 P=8761.6 T=221"/>
  <FbVal T="Wld_Tp_Target" V="T=340 I=10200 P=8750"/>
  <FbVal T="Wld_B1_Target" V="T=300 I=9800 P=6000"/>
  <FbVal T="Wld_B2_Target" V="T=300 I=9800 P=7000"/>
</Feedback>
```

3.14.8 Types of communication of PTS

There are many options for communication between the *PTS* Client and the *PTS* Server. However, each *PTS* Server system should offer at least the options mentioned hereinafter.

3.14.8.1 *Communication on file basis*

The client and the server communicate via two directories, viz. a **Request** directory and a **Feedback** directory. The server will watch the *Request* directory; whenever there is a file there, the file will be read and deleted, and a response file will be provided in the *Feedback* directory that will have the same name as the request file.

The server offers the option of defining any number of *Request/Feedback* directory pairs such that there will be an own directory pair for each client.

Each *Request/Feedback* directory pair will correspond to a parameter block (production list, production mode[70] or similar) on the *PTS s*erver; it may thus happen that a *PTS* client will address different *Request/Feedback* directory pairs.

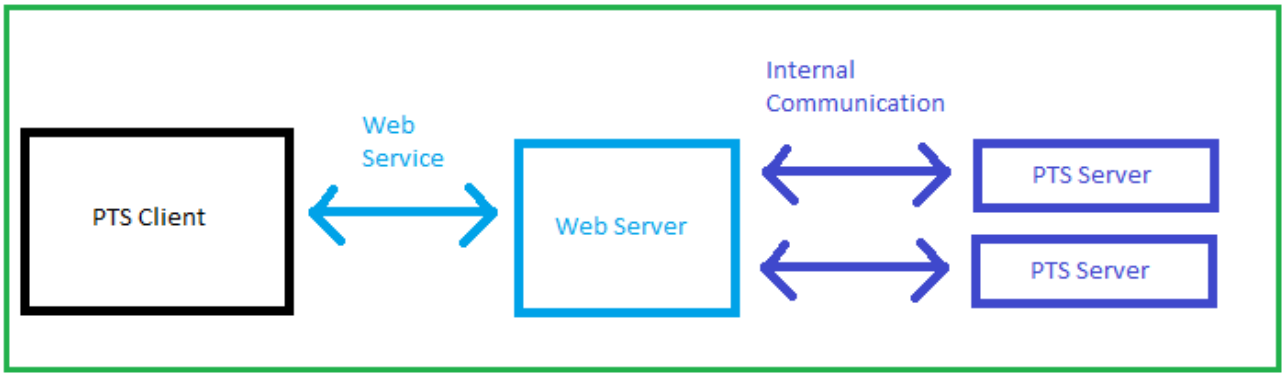
[70] An example of a *production mode* variant is as follows: for a PTS request, some client systems want the machines to be considered as fully available at 100% (this requirement is typical of CAD PST clients); other PTS clients (such as automatic transfer station view of the master computer, for example), on the other hand, want the machines to be taken into consideration according to their current operating condition such that a deactivated bending head will be allowed for, for example.

3.14.8.2 Communication via web services

(The principle of PTS communication via web services is explained here. The complete formal definition of the PXML Web API can be found at www.pxml.eu)

The client and the server communicate by means of web services.

Actually, there are two servers involved: the **web server** and the **PTS server**. The web server is a kind of **gateway** that provides the web services to the final client. The web server itself routes the PTS request to the underlying PTS servers.



Login

Communication has to be initiated by calling a **login service** in order to get an **authentication token**:

POST https://*mydomain.org*/Authentication/login

Authorization: Basic *dXNlclhZOnBhc3N3b3JkQUJD*

In this example "*mydomain.org*" is the address of the web service and „*dXNlclhZOnBhc3N3b3JkQUJD*" is a Base64 coded sequence of *user:password* (in the example used here, the user is „userXY" and the password is „passwordABC").

The resulting authentication token is returned in the header of the http response. This token must be used for all subsequent API calls by setting it in the header of the call (Token: {token}).

Put a PTS resource on the server (= send a PTS inquiry):

The client starts a PTS inquiry by putting a new resource on the web server:

POST https://*mydomain.org*/PTS

The **PXML document** of the inquiry is transmitted as byte array in the body of the POST statement. In addition, a **PtsServerPath** is specified in order to identify the underlying PTS server that has to process the inquiry. This is necessary since a single web server can have several underlying PTS servers (each PTS server corresponding to a single machine configuration).

The body can be formatted in JSON or XML.

Example of a body in JSON:

```
{
    "PtsServerPath": "MSytem/List1",
    "PxmlDocument": "QEA="
}
```

(„QEA=" is a sample of a Base64 coded byte array that should contain the PXML document).

The web service returns an alphanumerical **TestID** to the client.

Query the state of the PTS resource:

A dedicated GET web service returns the state of the PTS processing cycle:

GET https://*mydomain.org*/ PTS/{*TestID*}/Status

This statement returns a string with one of the following values:

- **WaitingForFeedbackFromPtsServer**: The PXML Document has been sent from the web server to the PTS server and the web server is waiting for feedback.
- **RequestNotSentToPtsServer**: Sending the PXML Document from web server to PTS server has failed.
- **FeedbackAvailable**: The PTS result is available.
- **RequestToPtsServerTimedOut**: The query from web server to PTS server timed out.

Reading the Feedbacks:

As soon as the feedback is available on the web server (state *FeedbackAvailable*), the client can retrieve the result:

GET https://*mydomain.org*/ PTS/{*TestID*}/Feedback

This query returns a PXML feedback file as a byte array.

Delete a PTS resource on the web server:

If a PTS resource is no longer needed, it can be deleted by the following command:

DELETE https://*mydomain.org*/ PTS/{*TestID*}

Some servers may delete automatically old PTS resources after a certain time.

3.14.9 Types of communication of Machine Return

3.14.9.1 Communication on file basis

The machine continuously writes *PXML* files that are read and optionally deleted by the overriding control system.

To facilitate evaluation for the reading system, the file names should reflect the sequence of the written files (either by means of a time stamp in the file name, or of a consecutive ID in the file name respectively).

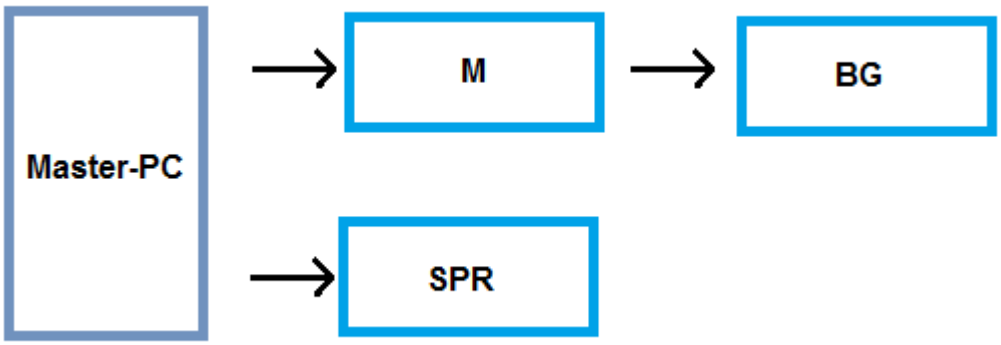
It is at the discretion of the machine to generate an own file for each *Feedback* message, or to pack several feedback messages into merely one file. However, once a file has been written, it must not be modified anymore. That is to say, the machine must not open an existing file again, such as to add more *Feedback* blocks, for example. At that point, this concept is clearly different from the rules of the Unitechnik-Report-Files.

3.14.9.2 Communication via web services

See www.pxml.eu.

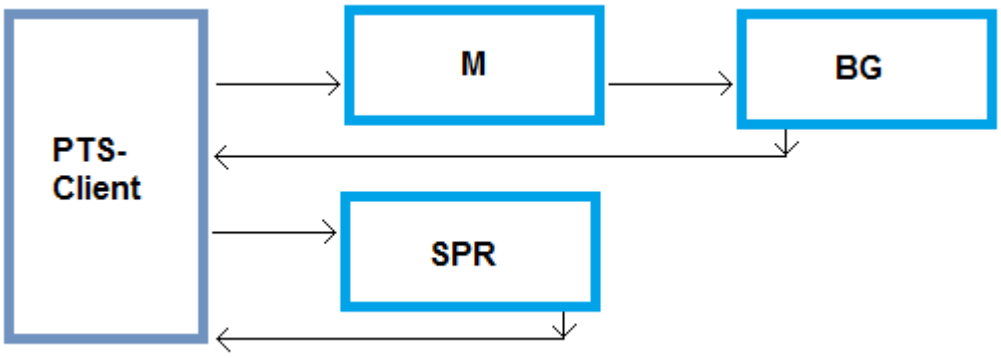
3.14.10 Parallel PTS servers and series of PTS servers

A complex production system may have several PTS servers. A mesh welding plant M, for instance, may be followed by a mesh bending plant BG, and in parallel there may be a shuttering robot SPR:



The machine M and the machine BG are cascaded, in the sense that BG is supplied by M with data. The control system ("Master PC") sends data in parallel to M and SPR.

The 3 machines are represented by 2 parallel PTS servers:



A PTS client should be able to query multiple parallel PTS servers.

The combination M/BG is viewed from the outside as one single PTS system: the transfer of PTS feedback file from M to BG isn't visible from the outside. This linkage is necessary, since M may slightly modify the production data, and BG needs to work with the modified data. In contrast, SPR operates quite independently of M.

3.14.11 Filter, classify and sort PTS feedback messages

The PTS server provides only a rough classification of messages, specifying them as errors, warnings, hints or information messages (See section 3.14.3.2). Since the server can serve multiple clients that may have different tasks and concepts, it is not a server's task to filter, sort or classify PTS messages. The PTS server reports everything from the perspective of the machine: it just reports which parts can be produced, which cannot, and which can be produced with restrictions – the server doesn't categorize messages as important or unimportant.

It is the responsibility of the client application to provide opportunities to display PTS messages in a suitable way. This requires the client to have mechanisms to filter and sort PTS messages. Ideally, there is also the opportunity to highlight some particular messages.

4 Proposals for future extensions

See the German version of the document.

5 Version History

Version 1.1:

New fields in the Bar section

- **Bin:** bin

Version 1.2:

New fields in the Bar section

- **Pos:** position
- **Note:** comment
- **Machine:** machine allocation
- **BendingDevice:** bending device
- **NoAutoProd:** replaces the *AutomaticProduction*field (but is inverted)

New fields in the *Girder* section

- **NoAutoProd:** replaces the *AutomaticProduction*field (but is inverted)

Fields in the *Slab* section have been removed

- LateralFormworkType
- LongitudinalFormworkType
- PlasterThicknessBottom
- PlasterQualityBottom
- UnitWeightBottom
- PlasterVolumeBottom
- PlasterThicknessTop
- PlasterQualityTop
- UnitWeightTop
- PlasterVolumeTop

New *Outline* fields

- **Z**-coordinate. replaces *MountPartInstallHeight*
- **Height**. replaces*MountPartThickness*.

Removed *Outline* fields

- MountPartThickness
- MountPartInstallPosX
- MountPartInstallPosY
- MountPartInstallHeight
- Area

New *Outline* type: *lot*

This is used to represent concrete lots. Also replaces the types of *contour* and *cutout*.

New *Shape* field: *Cutout*

If this is set, the respective Shape polygon is a cutout.

Removing the Concrete Layer elements

There are no Concrete Layer elements anymore as the concrete layer property is included in the *Lo* outline. For export to UNICAM, a global concrete layer must be specified; here, the information of the first *lot* will be copied once again.

The *LayerType* (identifier of the layer) is omitted and is not supported anymore.

Removing the contour or cutout *Outline* Types.
These Outline types are not required anymore as they are replaced by the *lot* Outline.
If there is an overall contour with several concrete layers in UNICAM, it will be split up into individual surface-congruent *lot* elements.

Nee definition of the multi-layer elements.
Introduction of the **MasterLayer** flag; new definition of the multi-layer concept PXML.

Default values are defined uniformly for each data type.
All individual default assignments are thus invalid.

Version 1.2 – Later Additions

New fields in the *Steel* section (added on 2007-08-14)

- **Steel.WeldingDensity:** welding density.
- **Steel.BorderStrength:** strength of the steel mesh border.

New field in the *Bar* section (added on 2007-09-25)

- **Bar.Ident:** numerical ID for system-wide identification.

Nee fields in the *Steel* section (added on 2007-10-03)

- **Steel.GenericInfo03…06:** free information blocks (previously, there were only two (2) information blocks).

New *Steel* type (added on 2007-10-13)

- **Steel.Type = "cage":** cage for cage production facilities.

Alloc Table introduced (added on 2007-11-17)

- **Tables "Alloc" and "Region":** laying patterns for stirrups, guiding bars or lattice girders respectively.

SteelExt Table introduced (added on 2007-12-05)

- **Table "SteelExt":** Collection of application-specific supplementary data to the Steel block.

New field in the *Girder* section (added on 2008-07-21)

- **Girder.Machine:** Text field for specification of the machine or list.

New fields in the *Girder* section (added on 2008-11-28)

- **Girder.Period:** lattice girder period.
- **Girder.PeriodOffset:** distance of the first period low point from the beginning of the lattice girder.

GirderExt Table introduced (added on 2008-12-12)

- **Table "GirderExt":** Generic subtable to GirderRow.

New fields in the *Girder* section (added on 2009-02-18)

- **Girder.GirderType:** The meaning of this field has been defined more precisely. A thrust girder type has been defined.
- **Girder.MountingTyp:** Mounting type for lattice girder.

New field in the *Girder* section (added on 2009-03-10)

- **Girder.BottomFlangeDiameter:** Bottom chord diameter in mm.

New field in the *Girder* section (added on 2009-03-19)

- **Girder.TopExcess, Girder.BottomExcess:** Excess length of the period rebar beyond the upper or bottom chord respectively.

Export to UNICAM for *GirderExtRows* (added on 2009-04-01)

- Export occurs via the lattice girder welding lines.

New field in the *Product* section (added on 2010-02-11)

- **Product.TurnWidth:** Assumed pallet width for double walls.

New field in the *Order* section (added on 2010-06-15)

- **Order.DeliveryDate:** Delivery date.

Global ID introduced (added on 2010-07-29)

- **<Table>.GlobalID:** Cross-system ID appearing in all PXML Tables.
- **Bar.Ident** is now obsolete, and has been removed from the standard.

ProdRot introduced (added on 2010-07-29)

- **Steel.ProdRotX/Y/Z:** Recommended rotation for production of reinforcement added.
- **Slab.AngelOfRotation:** is now obsolete, and has been removed from the standard.

PTS Specification (added on 2010-07-29)

- **Production Test Service:** Feedback from the inspection system.
- **Machine Return:** Feedback from the machine.

Clarification regarding the Character Encoding (added on 2010-07-29)

- **Character-Encoding:** The conventional established XML Encoding Rules will apply.

Proposals for extension (added on 2010-07-29)

- A separate chapter has been added that contains possible future extensions.

New MeshType values (added on 2010-07-29)

- 5 = cover mesh 2D; same as Type '1', but will be supplied and delivered together with Type '4'.
- 6 = cover mesh 3D; same as Type '3', but will be supplied and delivered together with Type '4'.

Nee fields in the Girder section (added on 2010-08-18)

- **Girder.Width:** Girder width in mm, between the two outermost points.

New fields in the Feedback section (added on 2010-11-25)

- **MaterialWeight:** Weight of the reported material.

New field in the Slab section (added on 2011-01-31)

- **ExpositionClass:** Exposition class.

New field in the Bar section (added on 2011-03-16)

- **ShapeMode:** Type or mode of representation of the bending shape.

With the introduction of this field, the issue of "schematic" representation has also been revised. This option, which has been defined rather vaguely only so far, has been defined more precisely, and has been adjusted to the reality of existing implementations in the process.

Extension of the definition of Steel.MeshType (added on 2011-04-22)

Besides specifying the Type as such, it should also be possible to specify type-specific details.

PXML specification now available in English, too (2011-06-03)

New field in the Feedback section (added on 2011-10-04)

- **InfoValue:** Additional information value.

New field in the DocInfo section (added on 2012-02-02)

- **ConvertConventions:** for declared non conformity with the specification.

Multi-layer handling has been simplified (2012-02-08)

- **Outline.Layer:** Has been added.
- **Steel.Layer:** Has been added.
- **Slab.MasterLevel:** Has been removed.

Parallel PTS servers and series of PTS servers (2012-09-14)

New field in the Feedback section (added on 2012-11-29)

- **Machine:** Identification of the Machine or PTS-Server originating the message.

Additional comments to conversion to and from BVBS (2013-06-10)

Additions for lattice girders with variable grid length (Added on 2013-11-10)

- **PeriodOffset:** No longer restricted to values between 0 and 200.
- **Section**: New sub-table for individual periods.

Mode table added (2013-22-25)

New field in the Product section (Added on 2013-11-27)

- **RotationPosition**.

Version 1.3 (2015-03-01)

In version 1.3 many key enhancements and even structural changes have been introduced. However, if all compatibility recommendations are observed, this new version is fully backward compatible with PXML 1.2. Older systems which have been made for version 1.2, can therefore work directly with the version 1.3 without modification.

Replacement of *Legacy Slab Fields* with corresponding new Product fields

- **See section 3.7.7**.

Introduction of *PXML Delegate Files*

- **See section 1.6**.

Definition of an element's *Reference Position*

The recommendation of not using cutouts in mountparts has been removed

Introduction of 3D geometry for Outlines

- **New Shape/SVertex fields DX, DY, RefHeight, see section 3.8.11**.
- **New Outline field ObjectID, see section 3.8.10**.

New field in the Order section

- **OrderArea, see section 3.3.1**.

New field in the Product section

- **ElementInfo, see section 3.6**.

Clarifying additions for rules of double wall composition and project coordinates.

- **For double wall composition see section 3.5.4.**
- **For project coordinates see section 3.5.8.**

Version 1.3 – Later Additions

2016-03-24

a) Definition of **Simplified volume calculation**. See section 3.8.11.
b) Clarification of the *Outline.Name*-Compatibility with UNICAM. See section 3.8.3.
c) Correction of the Include-Merge-Rules: Substructures of the delegate-files may coexist with substructures from the include file. See section 1.6.

2016-04-06

a) New table **FbVal** as sub table of the *Feedback* table. See section 3.14.4.
b) Add examples for the usage of **FbVal**. See section 3.14.7.
c) New table **ElemInfoVal** as sub table of the *ElemInfo* table. See section 3.6.3.
d) New field **F** in *Section*-entry of the *Girder* table. See section 3.11.13.1.
e) New girder-types for **Versagirder** in the *Girder* table. See section 3.11.5.

2017-02-20

a) New mode directives: **EnableProduction**, **EnableReinforcement**, **EnableProcurement**. See section 3.2.5.
b) Additional fields for the transfer of the building structure: **Structure**, **Building**, **SubStorey**. See section 3.3.1.
c) New field **ErpProjectUnit**. See section 3.3.1.
d) New *ElementInfo*-types **Concrete, ErpProjectUnit, PriceQty, PlanningQty**. See section 3.6.2.
e) Unit specifier **Unit** in *ElementInfo*. See section 3.6.2.
f) Unit specifier **U** in *ElemInfoVal*. See section 3.6.3.
g) New tables OrderInfo and OrderInfoVal. See section 3.4
h) New fields in *Girder* table: DiagonalWireDiameter (section 3.11.4), ArticleNo (section 3.11.7)
i) Addition: Recommendation for structuring the *StackNo*, see section 3.5.7.
j) Additional FbVal-Types: **Customer**, **Pos, ShiftID, ShiftStart**. See section 3.14.4.

2017-07-07

a) New FbVal-Value „**ProdArticle**“ and new Wr-Field „**Art**“. See section 3.14.4.

2018-06-30

a) PTS Communication: TCP/IP socket communication has been replaced by web services communication. See section 3.14.8.2.
b) Description of concept of using mountparts as “**void**” solids. See section 3.8.6.
c) Introduction of **placement rotation for Slab**. See section 3.7.2.
d) Introduction of **Production Derectives for Slab**. See section 3.7.3
e) Introduction of **placement rotation for Outline**. See section 3.8.1.
f) Introduction of full **stacking geometry**. See section 3.5.7.
g) The definition of *Reference Position* has been removed.
h) *Product.RotationPosition* has been declared as obsolete. See section 3.5.6.
i) Introduction of complex Include directives. See section 1.6.
j) Intrduced new *FbVal* types: **CutBegin**, **CutEnd**, **WeldBegin**, **WeldEnd**, **BendBegin**, **BendEnd**, **PlaceBegin**, **PlaceEnd**. See section 3.14.4.
k) Intrduced new *FbVal* types: **Wld**, **Wld_Targer, m2**. See section 3.14.4.
l) Introduction of new fields for the welding point feedback: **Y**, **WH**. See section 3.14.4.2.

2018-07-03

a) Introduction of **Simplified geometry representation.** See section 3.7.8.

2019-02-28

a) Introduction of **3D-BF2D.** See section 3.10.16.16.

b) Introduction of *SVertex*.**Profile**. See section 3.8.11.
c) Introduction of *Product*.**StackID**. See section 3.5.7.
d) Introduction of **placement rotation for Steel**. See section 3.9.1.
e) Additional clarification of the *Girder.Section* definition. See section 3.11.13.1.

2019-12-19

a) Additional clarification on the *Girder.Section* definition. See section 3.11.13.
b) Added recommendation for generation of *GlobalIDs*. See section 3.1.4.
c) Defined concept of *Virtual Element*. See section 3.5.2.

2021-01-08

a) Extended the *WeldingPoint* definition by specifying predefined *WeldingPointType* values and by adding the field *GroupID*. See section 3.10.15.
b) Added more *FbVal Type* definitions. See section 3.14.4.
c) Detailed definition of *SVertex.Profile* for multilayer *Slabs*. See section 3.8.11.

2021-05-03

a) Clarification on *SVertex.Profile* definition. See section 3.8.11.
b) New standard tags for *ElementInfo*: **AccAreaAdd, AccAreaExtAdd**, **ArchitecturalPart**, **TransportInfo**. See section 3.6.1.
c) New standard tags for *ElemInfoVal*: **Name**. See section 3.6.2.

2021-06-24

a) Correction of *Slab.ProdRotX/Y/Z* sequence. See section 3.7.3.

2022-11-17

a) Introduction of *Slab.ProdX/Y/Z*. See section 3.7.3.
b) Added *WeldingPointType* definitions -23, -29, -30. See section 3.10.15.
c) Clarifications on usage of *Steel.Type*, *Steel.MeshType*, *Bar.ReinforcementType*.
d) Added ProductTypes BM, CL, ST, MD, DT. See section 3.5.2

2023-03-23

a) Added new *ProductType* value **InSitu**. See section 3.5.2.
b) Added new *PartType* value **05**. See section 3.7.1.
c) Correction of documentation error regarding standard DW *Production Directives*. See section 3.7.4

2024-01-28

a) Added new *FbVal* type **HltBarTIntvl**. See section 3.14.4.
b) new *WeldingPoint* type **GrippingPoint**. See section 3.10.15.3.
c) Bugfix in example 3 of section 3.10.16.11.
d) Introduced *ObjectID* for *Steel* items and refined description for *ObjectIDs*. See sections 3.8.10 and 3.9.11.