ProgressXML

Index:

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 plant.

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.

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>0</MajorVersion>	Int	1
<MinorVersion>0</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
<Component>bbbbb</Component>		0,1
<Storey>cccc</Storey>		0,1
<DrawingNo>dddddd</DrawingNo>		0,1
<DrawingDate>dd.mm.yyyy</DrawingDate>		0,1
<DrawingRevision>ee</DrawingRevision>		0,1
<DrawingAuthor>aaaaaaaaaaaaaaaaaaa</DrawingAuthor>		0,1
<Comment>aaaaaaaaaaaaaaaaaaa</Comment>		0,1
<ImportSource>aaaaaaaaaaaaaaaaaaa</ImportSource>		0,1
<ImportSourceType>aaaaaaaaaaaaaaaaaaa</ImportSourceType>		0,1
<DeliveryDate>2010-04-16T11:46:48.933+02:00</DeliveryDate>		0,1
<GenericOrderInfo01>aaaaaaaaaaaaaaaaaaa</GenericOrderInfo01>		0,1
: : : :		0,1
<GenericOrderInfo20>aaaaaaaaaaaaaaaaaaa</GenericOrderInfo20>		0,1
<ApplicationName>aaaaaaaaaaaaaaaaaaa</ApplicationName>		0,1
<ApplicationGUID>aaaaaaaaaaaaaaaaaaa</ApplicationGUID>		0,1
<ApplicationVersion>aaaaaaaaaaaaaaaaaaa</ApplicationVersion>		0,1
<Product GlobalID="aaa">		**n**

<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
<Slab GlobalID="aaa">		**n**
<SlabNo>111</SlabNo>		0,1
<PartType>11</PartType>		0,1
<ProductAddition>44</ProductAddition>		0,1
<ProductionWay>aa</ProductionWay>		0,1
<NumberOfMeansOfTransport>666</NumberOfMeansOfTransport>		0,1
<TransportSequence>777</TransportSequence>		0,1
<PileLevel>888</PileLevel>		0,1
<TypeOfUnloading>99</TypeOfUnloading>		0,1
<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
<OrderPosition>aaaaaaaa</OrderPosition>		0,1
<ProductGroup>bbbb</ProductGroup>		0,1
<SlabType>33</SlabType>		0,1
<ItemDesignation>c...c</ItemDesignation>		0,1
<ProjectCoordinates>111111 222222…</ProjectCoordinates>		0,1
<PositionInPileX>111111</PositionInPileX>	Double	0,1
<PositionInPileY>111111</PositionInPileY>	Double	0,1
<PositionInPileZ>111111</PositionInPileZ>	Double	0,1
<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
<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
<Shape GlobalID="aaa">		**n**
<Cutout>false</Cutout>	Bool	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
</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
<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
<ProdRotX>±123</ProdRotX>	Double	0,1
<ProdRotY>±123</ProdRotY>	Double	0,1
<ProdRotZ>±123</ProdRotZ>	Double	0,1
<Layer>aaa</Layer>		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
	</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
	<Machine>aaa</Machine>		0,1
	<Period>000</Period>	Double	0,1
	<PeriodOffset>111</PeriodOffset>	Double	0,1
	<Width>80</Width>	Double	0,1

<AnchorBar GlobalID="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
</Section>		
</Girder>		
<Alloc Type="Bar" GlobalID="aaa">		**n**
<GuidingBar>2</GuidingBar>	Int	0,1
<Region>		**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
</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).

Automatic allocation of *GlobalID*s, if any, must be made immediately after the import, viz. before any items can be filtered out or summarized respectively via filter functionalities.

Generally, it must be advised against the use of automatically generated *GlobalID*s as these must be seen as rather unreliable. This will be specifically true where filter or grouping or blocking functionalities are provided in export modules that might cause a change in the number of some items.

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 **Fehler! Verweisquelle konnte nicht gefunden werden.**.

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[3]:

[3] 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.

- **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[4].
- **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.

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>2</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>
```

[4] 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).


```
    </Mode>
    <Mode>
      <ID>EstimateProdTime</ID>
      <Val>false</Val>
    </Mode>
  </DocInfo>
</PXML_Document>
```

1.1 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.

1.1.1 Specific Order Info

OrderNo: order identifier (any text).

Component: component (any text).

Storey: description of storey (any text).

DrawingNo: drawing identifier (any text).

DrawingDate: the date on which the drawing was generated or revised respectively (any text).

DrawingRevision: the revision number of the drawing (any text).

DrawingAuthor: the person who made up the drawing (any text).

1.1.2 Delivery Date

Delivery date.

1.1.3 GenericOrderInfo

The entries *GenericOrderInfo01* through *GenericOrderInfo20* are informational entries with a freely selectable meaning.

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.

1.1.4 Comment

Any comment regarding the Order.

1.1.5 ImportSource, ImportSourceType

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

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

1.1.6 ApplicationName, ApplicationGUID, ApplicationVersion

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

1.2 Product (Element)

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

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

1.2.1 ProductType

The 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:** hollow floor
- **11:** sandwich element
- **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

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.

1.2.2 TotalThickness, DoubleWallsGap

These are identically equal to the respective UNICAM-SLABDATE fields. Both values are in mm.

The DoubleWallsGap is only required for double walls, of course.

1.2.3 PieceCount

Target quantity.

1.2.4 TurnWidth, data transfer for double walls

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

The single slabs will be represented the way they rest 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.

However, this definition also has a drawback: to understand 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.

Within a production facility, this information is usually available implicitly: there is a fixed pallet width and thus a fixed tilting axis located halfway along the pallet width. That is to say, it is this tilting axis (= the axis of the tilting table) that will determine how the two wall halves will be joined together. However, if CAD data are available that have not been prepared for a fixed pallet width, one will encounter a problem; the element to be produced is not fully defined in terms of its geometry.

This indeterminateness may be remedied through explicit specification of the **TurnWidth** (= tilting table width). This variable will described the assumed pallet width. The two wall halves are mutually displaced such that the desired double-wall product will be formed through turning along the axis *Y=TurnWidth*/2. If the facility has a pallet width different from the one assumed, the coordinates must be adjusted accordingly when pallet assignment is made.

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 **opened 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.

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 format**: the data partly correspond to the turned slab; however, the Z coordinates are not transformed, the shape and orientation of the rebars or lattice girders will also not be transformed. So, technically speaking, we have no turning for the rebars or lattice girders, but rather a Y-displacement by *pallet width-2*Ym*, whereby *Ym* is the Y-centerline of the rebar or lattice girder respectively (Who would come up with a transformation like that?!). That is to say, the first wall half is represented *incorrectly* geometry-wise as the data neither match the unturned slab nor the turned slab; however, as opposed to variant a) we do have an advantage here in that the position of the wall halves in relation to each other is uniquely defined (However, this advantage could also be obtained by representing the first wall half correctly turned; here, it is hardly comprehensible why this much-confusing *pseudo-turning* is used so often.).

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.

1.2.5 Comment

Any comment regarding the product.

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.

1.3 Slab (Element Part)

1.3.1 SlabNo

This is intended to identify the slab. It corresponds to *Element Part Number* in UNICAM 5.2, or to *Element Part Name* in higher versions respectively.

1.3.2 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

1.3.3 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 are redundant as they can be computed from other variables; however, the values are still listed explicitly such as to provide the opportunity for CAD applicationsto freely set these values.

As opposed to UNICAM, there is also a Z coordinate for the slab in PXML.

The **handling and stacking properties** are assigned to the Slab (= Element part) although these are normally properties of the product (= Element). This selection improves the compatibility with UNICAM and provides more flexibility as it would also be conceivable to transport element parts from one production site to another.

1.3.4 Slab-Type

For conversion from or to UNICAM, SlabType is mapped to the UNICAM element type. Technically speaking, however, SlabType in PXML is a property of the element **part**.

Whenever possible, this value should not be used at all.

1.3.5 ProjectCoordinates

This covers the 9 project coordinates (separated by blank spaces) as mentioned in the UNICAM specification.

1.3.6 Coordinates of Element in Pile

Coordinates of the element on the pile. This corresponds to the UNICAM specification (version 6.0 or higher).

Here, as opposed to UNICAM, the angle is not limited to [0, 359°], and it is even recommended to use the range of]-180°, 180°].

1.3.7 ReforcemInfo

Merely for compatibility with UNICAM; includes the details of the UNICAM-REFORCEM Info block.

1.3.8 Generic Slab Info

Freely definable additional information.

1.3.9 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 fitments or mounting parts (according to UNICAM-LAYERS).

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

1.4 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 fitments or mounting parts.

1.4.1 X/Y/Z

Offset of all data in relation to the Slab zero point.

For fitments or mounting parts, the Z coordinate is the overall height.

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 fitment or mounting part. (Though this is only true for known fitments or mounting 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 fitment or mounting 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.

1.4.2 Height

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

1.4.3 Name

Identifier. In UNICAM 5.2, identifiers are available for fitments or mounting parts only; for UNICAM 6.0 or higher, identifiers are also available for cutouts. In PXML, identifiers are also available for contours, but it is recommended not to use this for contours.

1.4.4 GenericInfo

Freely usable informational lines.

1.4.5 MountingInstruction (only for Mountpart)

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

- 0 = the part is being installed,
- 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.

1.4.6 MountPartType, MountPartArticle (only for Mountpart)

This corresponds to the respective UNICAM definitions.

1.4.7 MountPartProperties (only for Mountpart)

This corresponds to the respective UNICAM definitions:

- **MountPartIronProjection** = rebar projection in mm. These are rebars that protrude from a fitment or mounting 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 fitment or mounting part.
  Basically, these values are only required for fitments or mounting 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 fitment or mounting 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.

1.4.8 Concrete Properties (only for lots)

- **ConcretingMode** = concreting flag.
- **ConcreteQuality** = concrete quality or grade (such as B25, for example).
- **UnitWeight** = bulkdensity in kg/dm³.
- **Volume** = target concrete volume in m³.

1.4.9 Layer

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

1.4.10 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:

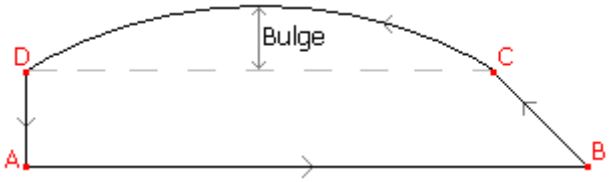
If this field is set to *true*, the *Shape* object will describe a cutout.

The *Cutout* field is merely used with*lot* outlines only.

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).



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 fitment or mounting 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.

1.5 Steel

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

Type attribute

The following Types are available:

- **none:** loose rebars or lattice girders.
- **mesh:** steel mesh (possibly also including lattice girders). Also cages made of steel mesh.
- **cage:** reinforcement cage.
  Typically, stirrups and longitudinal rebars are specified (preferably, longitudinal rebars with RotZ=0 and stirrups with RotZ=90).
- **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.

1.5.1 X/Y/Z

Offset of all data in relation to the Slab offset.

1.5.2 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).

1.5.3 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.

1.5.4 Name

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

1.5.5 MeshType

Mesh type:

- 0: standard mesh.
- 1: bent mesh 2D; this mesh is to be fed to a beam bending machine.
- 2: reinforcement module.
- 3: bent mesh 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: (vacant for extension; not as yet specified; should not be used at this juncture).
- 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:

- 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.

1.5.6 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*).

1.5.7 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%.

1.5.8 Generic Steel Info

Freely usable informational lines.

1.5.9 ProdRotX/Y/Z

The **ProdRot**fields describe a rotation of the reinforcement block (typically the cage), 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 form, but is merely to be rotated 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.

The 3 ProdRot angles must be specified in DEG units, and will describe the rotation as follows:

- at first, the whole cage will be rotated around the X-axis,
- then, it will be rotated around the Y-axis of the rotated system, and
- then, it will be rotated around the Z-axis of the rotated system.

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

1.5.10 Layer

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

1.6 Bar

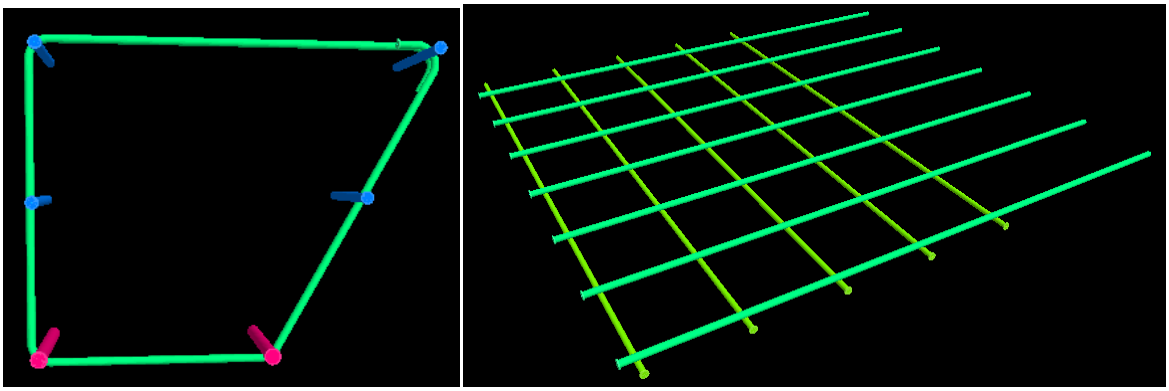
A *Bar* entry corresponds to round-bar steel element, viz. a straight or bent rebar.

1.6.1 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.

1.6.1.1 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.

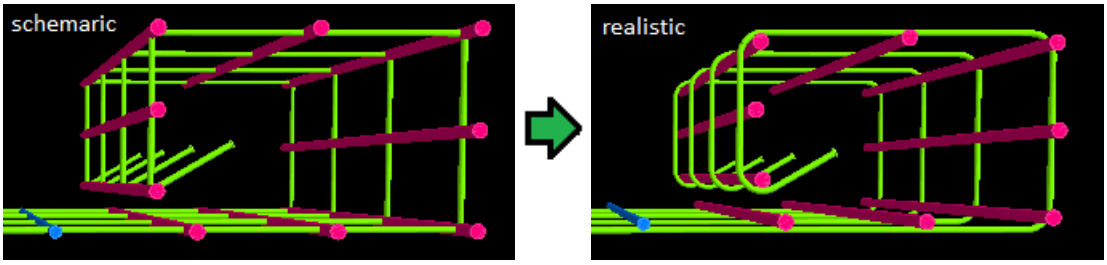


In this representation, the type of reinforcement has no geometric meaning as the geometry is clearly defined even without a need to know the type of reinforcement.

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

1.6.1.2 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.

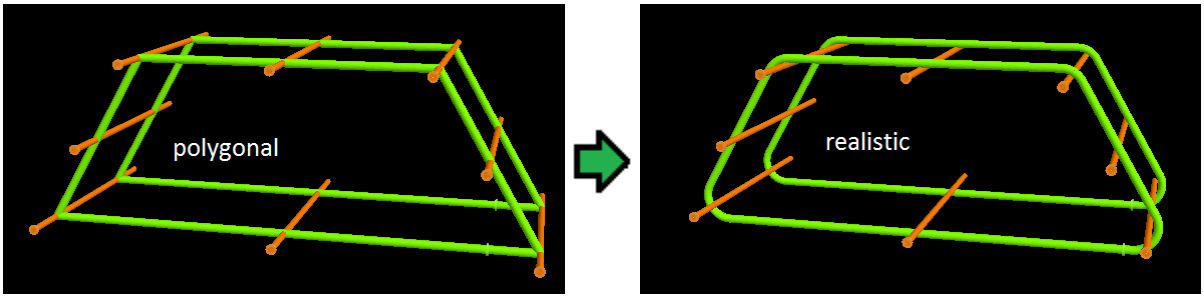
Master/*Slave* relationships may be defined either explicitly through a master rebar staggering pattern[5], or else implicitly via the type of reinforcement: two rebars of a type of reinforcement of 1and 2 will be mutually interrelated in a *Master*/*Slave* relationship if they intersect at the rebar core (here, the type 1 rebar will be the *Master* and the type 2 rebar will be the *Slave*)[6].

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*[7].

1.6.1.3 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.



[5] Master rebar staggering pattern: see Section 1.8.1.

[6] 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 1.6.2).

[7] 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 $\vec{v_i}$are the vectors of the rebar segments and$\vec{a} := \sum_{i=0}^{n-2} \vec{v_i} \times \overrightarrow{v_{i+1}}$, then $\vec{a} \times \vec{v_i}$will face *inwardly*.

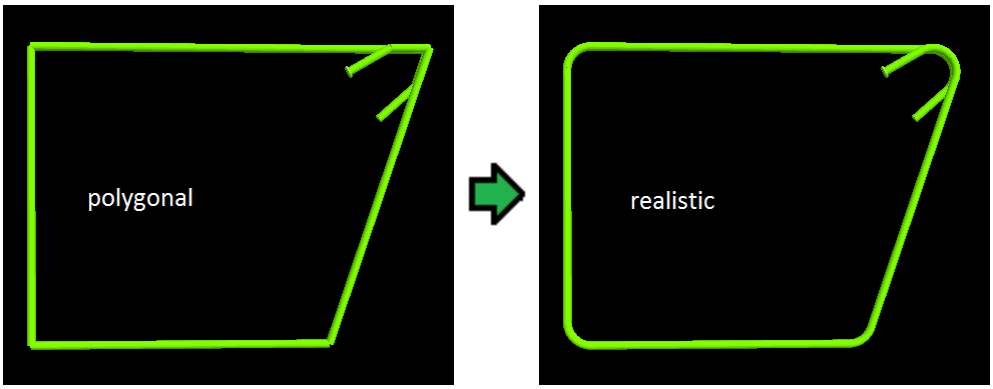
For bending up to 90°, the representations of *schematic* and *polygonal* tally with each other. However, both representations differ when it comes to acute bends: the *polygonal* representation remains true to the real shape of the rebar, whereas the *schematic* representation modifies the outer proportions[8].

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 α_1 and α_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, α_1 is given by the outer shape of the polygon, and α_2 merely completes the bend such as to obtain the desired overall angle α. 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 α_1 and α_2 will then merge into one overall angle α.



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[9].

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.

1.6.1.4 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[10].

[8] 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.

[9] 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.

[10] 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.

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.

1.6.2 ReinforcementType (reinforcement layers)

1.6.2.1 Definition of reinforcement layers

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.

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.

1.6.2.2 Definition of reinforcement layers

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[11].

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.

[11] 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.

1.6.2.3 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 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[12].

1.6.3 SteelQuality

Steel quality.

1.6.4 PieceCount, Diameter, X, Y, Z

Quantity, diameter, and offsets.

1.6.5 RotZ

To determine the position of a rebar, the coordinate system is at first 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 1.6.16.1).

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 bend 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 $\pm\pi$. 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

[12] 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.

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[13]:

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

Derivation of the equation for *RotZ*:

Let's assume λ to be such that

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

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:

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

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 1.6.16.1). *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*).

1.6.6 ArticleNo

Article identifier of the round steel rebar.

1.6.7 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.

1.6.8 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).

1.6.9 Bin

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

1.6.10 Pos

Text field for entering the single rebar drawing reference.

[13] This limitation does not only result in an intuitively meaningful *RotZ*, but also simplifies other computations, as we will see in Section 1.6.16.12.

1.6.11 Note

Text field for entering a comment regarding the rebar.

1.6.12 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.

1.6.13 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[14].

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

$$R_{\min} = D/2 + d/2 + k_r .$$

(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)[15].

This will, of course, result in a potential conflict with the bending radius indication of the segment (see Section 1.6.16.3). In such a case, the larger bending radius will always be relevant.

1.6.14 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).

1.6.14.1 Type

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

1.6.14.2 Position

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

1.6.15 WeldingPoint

1.6.15.1 WeldingOutput

Welding output in %.

---

[14] 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.

[15] 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.

1.6.15.2 Position

Position of the welding point along the rebar.

1.6.15.3 WeldingPointType, WeldingPrgNo

WeldingPointType contains a type identifier for the welding point. *WeldingPrgNo* directly determines the welding program to be used. Normally, no more than one of these two details will be used only.

1.6.16 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 1.6.16.10

1.6.16.1 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*[16].

$$RotX \in]-90°, 90°] \text{ (conditional only}^{17}\text{)}.$$

$$BendY \in]-180°, 180°] \text{ (conditional only}^{18}\text{)}.$$

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).

1.6.16.2 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 1.6.16.10), L will indicate the *arc radius*.

1.6.16.3 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 1.6.16.10).

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*.

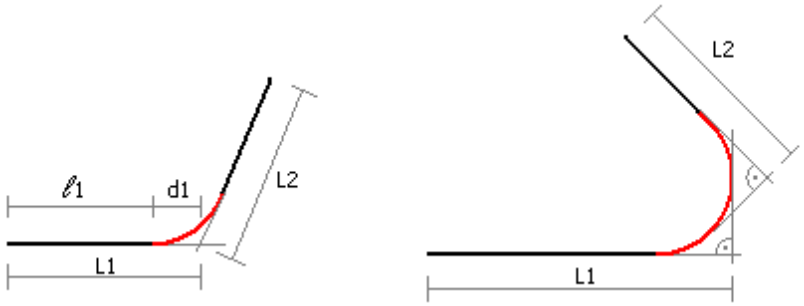
---

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

[17] 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)πif 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.

[18] 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.

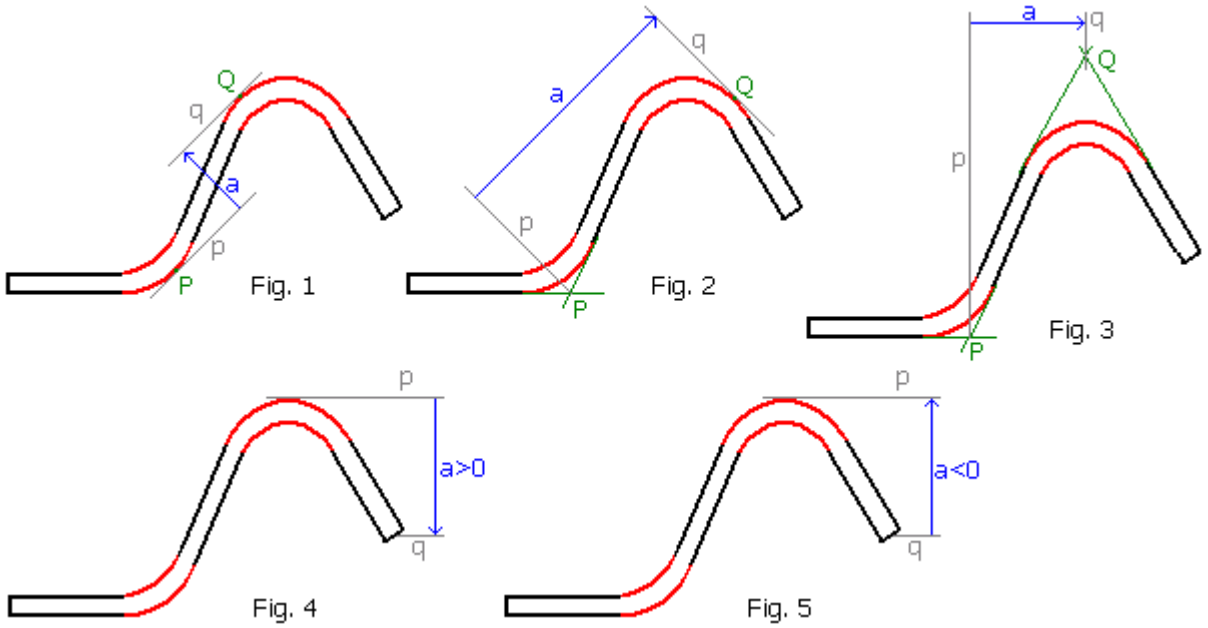
Radius definition via specification of ***BendingDevice***:

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 1.6.13). If both are given (viz. *BendingDevice* and *R*), the larger radius will be relevant.

1.6.16.4 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*.

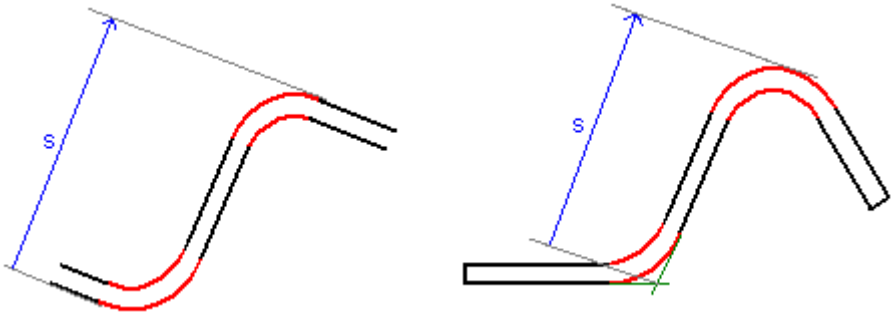
Fig. 4: If a segment has no bend at one end (first or last segment), dimensioning will relate to the rebar center (however, this is not true for *conventional external dimensions*; see Section 1.6.16.8).

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.

1.6.16.5 External length of segment



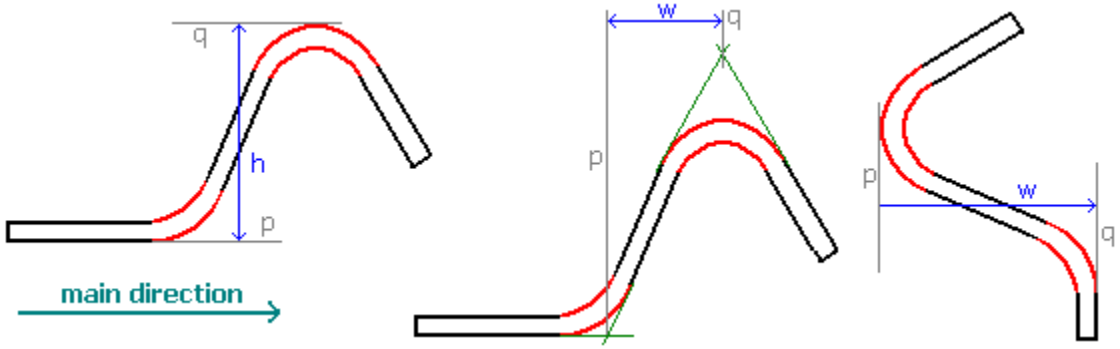
The **external length** of a segment is the segment's*external dimension* that runs in the direction of the segment. (The segment's*external dimension* is defined according to Section 1.6.16.4.).

1.6.16.6 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).



1.6.16.7 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[19].

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.

[19] 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.

1.6.16.8 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[20].

1.6.16.9 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.

[20] 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.

1.6.16.10 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[21].

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[22].
- ***RotX***: defined the pitch of the helix[23].
  The increase in height per torsion is $G = L \cdot \sin RotX$.

Connecting the arc to the straight segment:

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 1.6.1). 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.

1.6.16.11 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:

---

[21] 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 1.6.16.11.

[22] 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.

[23] 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.

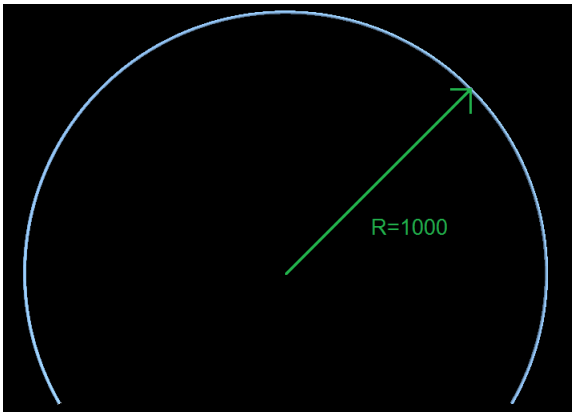
a) There may be no bending at the beginning of the first segment. That is to say, at least two segments will be required to render a curved iron. (And this is the only real drawback of the standard PXML segments as compared to the *spiral* segments.)

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 **Fehler! Verweisquelle konnte nicht gefunden werden.**1.6.16.9). 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



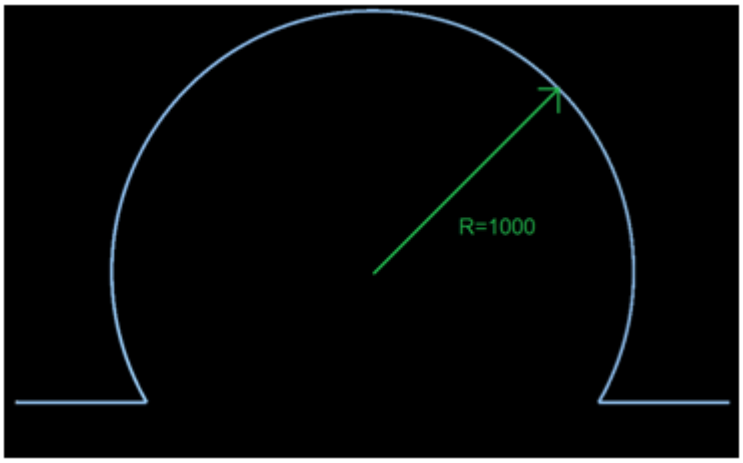
An arc with an arc radius of 1,000 mm (core dimension) and an angle of 120° will be represented via two segments:

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



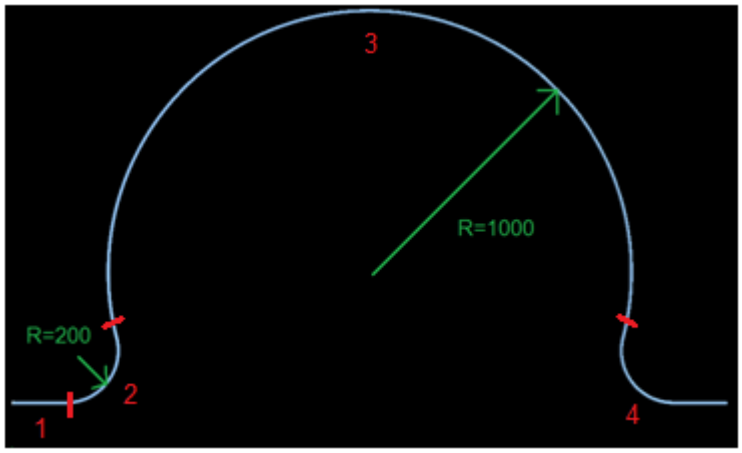
Segment 1: L=500, BendY=0, R=0

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=0
Segment 3: L=0, BendY=-208, R=1000
Segment 4: L=404, BendY=104, R=0
The red lines and red numbers in the figure delimit the segments.

1.6.16.12 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[24]:

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

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

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

(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:

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

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:

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

This restriction to the range of]-90°,90°] will now also bring about that $\cos(RotX) \geq 0$. From $v_z = \cos(RotX)\sin(BendY)$, it then follows that $\operatorname{sgn}(v_z) = \operatorname{sgn}(\sin(BendY))$. Finally, with $v_x = \cos(BendY)$, we arrive at:

$$\boxed{BendY = \begin{cases} \arccos(v_x), & v_z \geq 0 \\ -\arccos(v_x), & v_z < 0 \end{cases}.}$$

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

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

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[25]; 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:

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

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[26]. 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[27].

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

[26] 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 barswhere 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.

[27] 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 1.6.5 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 $\vec{v}_r := (\cos(RotZ), \sin(RotZ), 0)$.
You will see this as follows:
$\underline{v}_r = D_{RotZ} \cdot (\cos(RotZ) \quad \sin(RotZ) \quad 0)^T$ is the coordinate representation of $\vec{v}_r$ after the *RotZ*coordinates were rotated. If we plug the rotation matrix coordinates in for D_{RotZ}, we will obtain $\underline{v}_r = (1 \quad 0 \quad 0)^T$. Now, if we assume that $\underline{v} = (-1 \quad 0 \quad 0)^T$ (which is the same as if *RotZ*=±180°), we will obtain:
$\underline{v}_r \cdot \underline{v}_0 = (1 \quad 0 \quad 0) \cdot (-1 \quad 0 \quad 0)^T = 1$, which obviously is a contradiction to $\vec{v}_r \cdot \vec{v}_0 \geq 0$.

1.6.16.13 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[28]. 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)[29]

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):

$$\alpha_L = 0$$

$$\alpha_{toX} = RotZ - BendY_0 \operatorname{sgn}(\sin RotX_0)$$

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

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

[28] 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.

[29] 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°[.

$$\alpha_L = 0$$

$$\alpha_{toX} = RotZ + 180°$$

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

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).

$$\alpha_L = -\arcsin(\sin RotX_0 \cos RotX_1 + \cos BendY_0 \cos RotX_0 \sin RotX_1)$$

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

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

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

For $RotX_0 = \pm 90°$ or $BendY_0 = 0$ or $BendY_0 = \pm 180°$ (no Dummy Segment):

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

For $RotX_0 \neq \pm 90°$ and $BendY_0 \neq 0$ and $BendY_0 \neq \pm 180°$ (including Dummy Segment):

$$d_0 = 0$$

$$\alpha_0 = \arg(u, v), \quad where$$

$$u = \cos BendY_0 \cos RotX_0 \cos RotX_1 - \sin RotX_0 \sin RotX_1$$

$$v = \cos RotX_0 \sin BendY_0$$

$$d_i = L_{i-1}, \quad \alpha_i = BendY_i, \quad i \geq 1$$ (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 $RotX_0 = \pm 90°$:

$$RotZ_{new} = RotZ_{old} - BendY_{0,old} \operatorname{sgn}(\sin RotX_{0,old})$$

$$BendY_{0,new} = 0$$

$$RotX_{0,new} = RotX_{0,old} + RotX_{1,old}$$

$$RotX_{1,new} = 0$$

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

Dummy rectification for $BendY_0 = 0$:

$$RotX_{0,new} = RotX_{0,old} + RotX_{1,old}$$

$$RotX_{1,new} = 0$$

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

Dummy rectification for $BendY_0 = \pm 180°$:

$$RotZ_{new} = RotZ_{old} + 180°$$

$$BendY_{0,new} = 0$$

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

$$RotX_{1,new} = 0$$

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

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).

1.6.16.14 Conversion from or to BVBS-BF2D

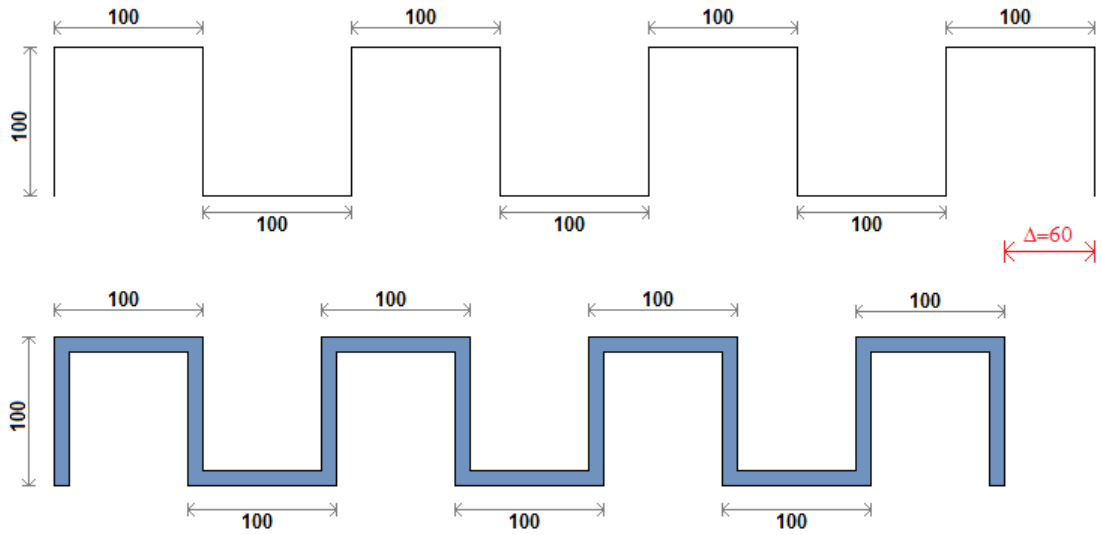
In the BVBS format, there is an option of specifying a bending radius for each bend. Deviating from the BVBS specification, some machines will interpret all bends with a specified bending radius as a spiral (and a specified length, if any, will then be read as arc length). For export to BVBS, it is thus 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 1.6.16.10).

1.6.16.15 Conversion from or to BVBS-BF3D

The BVBS specifications for BF3D are (as opposed to BF2D) anything but complete, consistent or conclusive[30]. 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:

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



(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°).

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".

1.7 Girder

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

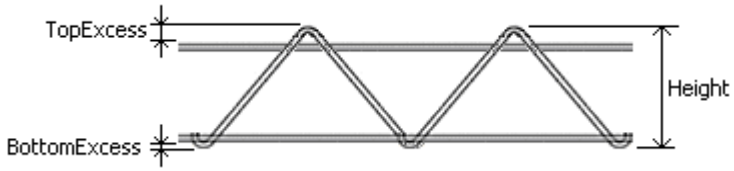
AngleToX specifies the direction in the xy-plane (starting from the x-axis).
The values are within the range of]-180°, 180°].

X, *Y*, and *Z* describe the position of the center line that is located in the middle of the two bottom chord rebars (core dimension of the bottom chord).

1.7.2 NoAutoProd

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

1.7.3 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.

1.7.4 Weight, TopFlangeDiameter, BottomFlangeDiameter

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

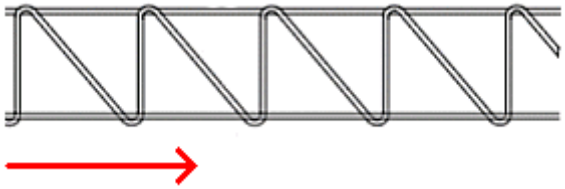
1.7.5 GirderType

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.

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:



1.7.6 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.

1.7.7 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.

1.7.8 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.

1.7.9 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.

1.7.10 AnchorBar

Anchor rods.

Type: type identifier.

Length: length in mm.

Position: laying position in mm.

1.7.11 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.

1.7.11.1 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.

1.7.11.2 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.

1.7.11.3 Type = GirderGripPos

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

1.7.11.4 Type = MeshGripPos

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

1.7.11.5 Type = SupportPos

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

1.7.11.6 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).

1.7.12 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.

1.7.12.1 Fields in the Section table

- **L**: Length of a period section in mm. Typical values are around 200 mm.
- **S**: Shift of the top flange welding spot in relation to the center of the period section (in mm). For a normal lattice girder, this value will be 0. For shear force girders, this value will typically be around -100.

1.7.12.2 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[31].

d) Only non-zero values will be used to generate the *Section* pattern. Plus, the *Section* fields are independent of each other in terms of their sequence. The two following *Section* tables are thus precisely equivalent[32]:

L	**S**
195	-50
195	0
200	-50
195	0
195	-50
200	0

L	**S**
195	
195	
200	
	-50
	0

[31] 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).

[32] Hence, each implementation may even generate a separate table for each *Section* value (or, as one more alternative, use a table with one value and a field type identifier).

1.8 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.

1.8.1 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[33].

1.8.2 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).

[33] 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[34].

- 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.

1.8.3 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:

$$P_{m,k} = \sum_{i=0}^{m-1} L_i + k \cdot Pitch_m, \quad L_i = IntervalCount_i \cdot Pitch_i .$$

[34] 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°."

At first, the position thus computed is merely a scalar (linear) value. Via the procedures from Sections 1.8.1 and 1.8.2, 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[35].

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.

1.9 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).

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

1.10 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.

1.10.1 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[36]. 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.

[36] 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.

1.10.2 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.

1.10.3 Fields of the Feedback Block

1.10.3.1 Feedback attributes

1) **ItemType**: ... specifies the PXML Table name of the Item to which the feedback relates.
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.

1.10.3.2 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).
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).
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.

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.

1.10.4 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>2</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>
</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.

1.10.5 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>2</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>2</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.

1.10.6 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.

1.10.6.1 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[37] or similar) on the *PTS* server; it may thus happen that a *PTS* client will address different *Request/Feedback* directory pairs.

1.10.6.2 Communication via TCP/IP sockets

Communication occurs via a TCP port. By default, this will be port 56693; however, other port values are to be used. Connection buildup will be initiated by the client; connection clearing as well. It is at the discretion of the client to permanently use an active link, or to newly build up a connection for each request respectively.

Different servers will have different IP or port numbers. If different parameter blocks are to be addressed on a server, this will also be done via different port numbers.

The request stream and the response stream will each enclose a PXML_Document. The feedback stream can be unambiguously associated to a request via the *GlobalID* of the *DocInfo*Table.

1.10.7 Types of communication of Machine Return

1.10.7.1 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.

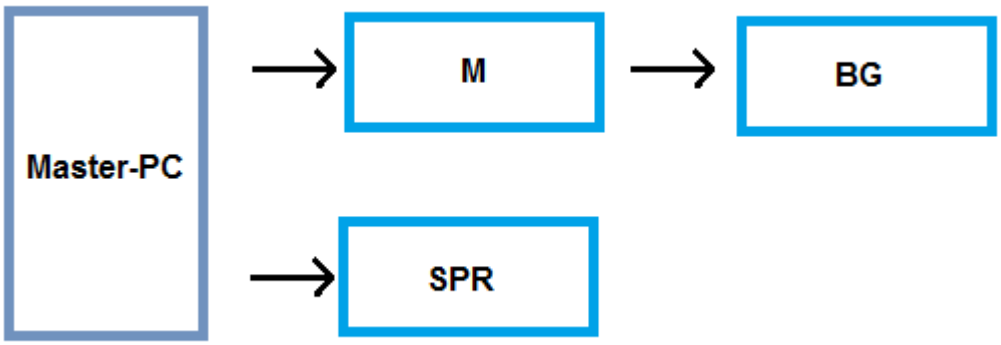
1.10.7.2 Communication via TCP/IP sockets

Machine Return communication via sockets is not bindingly defined at this juncture.

1.10.8 Parallel PTS servers and series of PTS servers

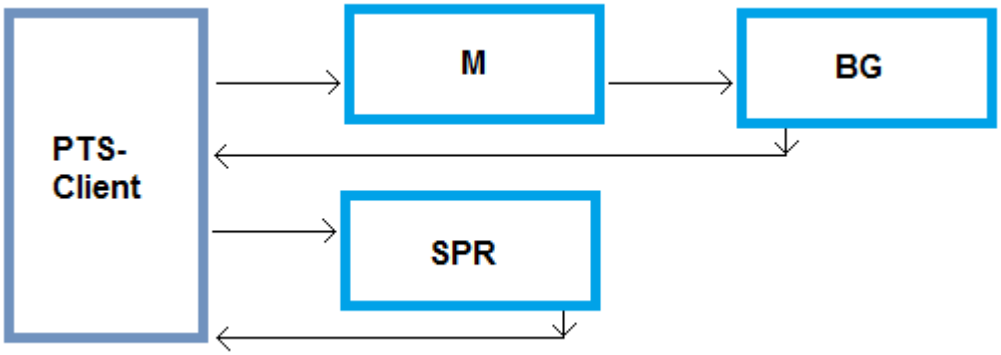
A complex production system may have several PTS servers. A mesh welding plant M, for instance, may be followed by a mesh bending plant BG, and in parallel there may be a shuttering robot SPR:

[37] 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.



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.

1.10.9 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 1.10.3.2). 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.

2 Proposals for future extensions

See the German version of the document.

3 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.

ProdRotintroduced (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 (2016-06-10)

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**.

