Technical Paper
PLCopen Technical Committee 6

XML Formats for IEC 61131-3

Version 1.01 –Official Release

DISCLAIMER OF WARANTIES

THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS AND MAY BE SUBJECT TO FUTURE ADDITIONS, MODIFICATIONS, OR CORRECTIONS. PLCOPEN HEREBY DISCLAIMS ALL WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, FOR THIS DOCUMENT. IN NO EVENT WILL PLCOPEN BE RESPONSIBLE FOR ANY LOSS OR DAMAGE ARISING OUT OR RESULTING FROM ANY DEFECT, ERROR OR OMISSION IN THIS DOCUMENT OR FROM ANYONE'S USE OF OR RELIANCE ON THIS DOCUMENT.

Copyright © 2003 - 2005 by PLCopen. All rights reserved.

Date: 10/06/05

The following paper

XML Formats for IEC 61131-3

is a document representing the results of the work done in the PLCopen Technical Committee 6 - XML. This release 1.0 is based on the work done on the version 0.99, 'Release for Comments', as published in April 2004, as well as the feedback received.
This specification has been written thanks to the following members of the TC6 XML:

Dieter Hess	3S
Dirk Schubel	3S
Alexander Fay	ABB
Johan Gren	ABB
Josef Papenfort	Beckhoff
Uwe Thomas	Beckhoff
Matthias Riedl	ifak
Thomas Brandl	Indramat
Michael Sperber	infoteam Software
Christina Böttger	infoteam Software
Wolfgang Horn	IST
Dietmar Berlesreiter	Keba
Thomas Baier	Kirchner Soft
Andreas Weichelt	KW Software
Dimitrij Kirzhner	KW Software
Hansjörg Hotz	Matsushita
Monique Atali-Ringot	Rockwell Automation
Paul Brooks	Rockwell Automation
Heinz Dieter Ferling	Schneider Automation
Hans Peter Otto	Siemens
Achim Koch	SMS Demag
Gerd Schneider	Softing
Les Powers	Triconex
Eelco van der Wal	PLCopen

Change Status List:

Version number	Date	Change comment
V 0.0	06/05/2002	Preliminary draft with additions from PLCopen
V0.1	14/01/2003	Results of the meeting at infoteam software
V 0.2	25/02/2003	Results of the meeting at Beckhoff Elektronik
V 0.3	06/05/2003	Results of the meeting at Matsushita Electric Works
V 0.4	17/06/2003	Results of the meeting at Kirchner SOFT
V 0.5	15/07/2003	Results of the meeting in Amsterdam
V 0.6	2/10/2003	Results of the meeting at KW Software, Lemgo, Germany
V 0.7	28/11/2003	Results of the meeting in Neurenberg, with further editing by EvdW
V 0.71	31/03/2003	Hotz: All elements that describe IEC 61131-3 object are entered
V 0.8	09/04/2004	Final version before release as 0.99 – EvdW
V 0.99	16/04/2004	Released as Version 0.99 – in combination with scheme
V 0.99 A	14/02/2005	Based on the feedback on the Version 0.99. Meeting Feb. 8+9, 05
V 0.99 B	18/04/2005	Changed pictures included. US proof reading.

V 1.0	27/04/2005	Last minor changes done. Examples added cf. new xsd. Official release
V 1.01	10/06/2005	Minor update: “refLocalId” is required. Text and examples updated.

Table of Contents

1 Introduction

Since the release of the IEC 61131-3 programming standard, users want to be able to exchange their programs, libraries and projects between development environments. Although this was not the intent of the standard itself, it was a task that the independent organization PLCopen committed itself to.

IEC 61131-3 is focused on the software development environment. As such it is just a part of a total solution. The other parts are a structure of tools like:

- networking tools
- debugging tools
- simulators
- documentation tools

Therefore PLCopen had decided to develop interfaces towards these support tools. This has resulted in a workgroup named TC6 for XML (eXtended Markup Language). This committee has defined an open interface, which supports different kinds of software tools, and provides the ability to transfer the information that is on the screen to other platforms. This screen information does not only contain textual information, but also graphical information. This can include the position and size of the function blocks, and how they are connected.
The design of the 'transferred' program itself has to remain the same after the transfer, so not to be altered in look and feel. The wide variety of possibilities, especially in the graphical tools, has to be brought under one umbrella. Originally, PLCopen looked to the STEP standard to do this. STEP can be looked at as an earlier version of XML, but the graphical part was limited. The STEP protocol was used for the PLCopen Portability Level, but showed a lack of graphical definitions. This meant that, without extensive work, the graphical languages could not be transferred, and the original goals could not be fulfilled.

PLCopen wants to be able to transfer a control project without much additional effort, from one development environment to another without loosing information even when it is incomplete, e.g. not compilable without errors. This of course is also valid for the POUs, and especially for the User Derived Function Block libraries. XML provides the right technology for this.
As such it will be more than an export / import tool from one development environment to another. From the moment that this format is available, it is just a small step to feed a documentation tool with the information, for instance. Actually, it is not important where this XML-code is coming from, as long as it is recognizable and useable. It could be generated by other tools like simulation and modeling tools, and consumed by verification, documentation, and version control tools.
To support this principle, all relevant information will be exported. The importing tool has to be intelligent in filtering which parts of this information are useful and needs to be imported With this approach, PLCopen creates a complete new market, in which the focus is on reusability of software development from libraries up to complete control projects.

1.1. Purpose

This document presents the representation of the complete project within the IEC 61131-3 environment based on current XML technologies, including the common elements with *Sequential Function Chart* (SFC), the two textual languages Structured Text (ST) and *Instruction List* (IL), and the two graphical languages *Function Block Diagram* (FBD), and *Ladder Diagram* (LD). The formats are specified through corresponding XML schema. This is an independent file, with the

.xsd extension, and as such part of this specification. A description of these schemas is contained in this document. It is assumed that the reader of this document is familiar with the basic technologies.

The described formats are destined for the import and export of IEC 61131-3 *Projects* and *Program Organization Unit*s (POUs). These items can be under development, and so incomplete. As such there is no verification on their applicability or their correctness.

In principle all information is made available in the exported XML file. The intelligence is in the importing function (One exception is the generation of the coordinate system information in free-style graphical editors – this is generated in the export functionality)
Vendor specific information and attributes can be included in the export file and possible deleted during import, if applicable. The supplier specific information should not deal with any of the logic part of the program.

This means that filtering is done on the import – suppliers have to take care that the extensions of the XML schemes for internal purposes is done in such a way that deletion of the info does not effect the functionality of the project. This could be done via an additional, supplier specific XML scheme, besides the PLCopen defined version, linked via an URL or file to the source.

Concerning the exchange of graphical language constructs between different Programming Systems, the focus is on logical information with optional explicit graphics.

Concerning consistency – the XML description is the valid description. Other descriptions are added for clarification, etc., and no consistency to the XML description is guaranteed.

1.2. Short introduction into XML

XML stands for extended Markup Language, providing the basis for the well-known HTML (Hyper Text Markup Language) that is used extensively on the internet.
XML has several advantages:

1. It is extendable
2. The data included can be checked for consistency with the scheme provided
3. Different schemes provide a possibility to check the incompatibilities

The W3C consortium calls XML "a common syntax for expressing structure in data." Structured data refers to data that is tagged for its content, meaning, or use. For example, whereas the <H1> tag in HTML specifies text to be presented in a certain typeface and weight, an XML tag would explicitly identify the kind of information: <BYLINE> tags might identify the author of a document, <PRICE> tags could contain an item's cost in an inventory list - all the way down to <DOGFOODBRAND> if that's the level of detail required.

A schema is defined as a formal specification of element names that indicates which elements are allowed in an XML document, and in what combinations. It also defines the structure of the document: which elements are child elements of others, the sequence in which the child elements can appear, and the number of child elements. It defines whether an element is empty or can include text. The schema can also define default values for attributes. A schema also provides for extended functionality such as data typing, inheritance, and presentation rules.

By separating structure and content from presentation, the same XML source document can be written once, then displayed in a variety of ways: on a computer monitor, within a cellular-phone

display, translated into voice on a device for the blind, and so forth. It will work on any communications devices that might be developed; an XML document can thus outlive the particular authoring and display technologies available when it was written. Check www.xml.org for more information.

2 Scope

The scope of this specification is defined by identified "Use Cases", focussed to the application areas in which these schemas could be used. The following uses cases have been identified:

2.1. Use Case – Exchange format for programming tools (all IEC languages)

- Exchange at POU or project level.
- (One time) migration to another system – a certain amount of manual work could be needed
- Parallel use of multiple systems - a certain amount of manual work could be needed

The items can be under development, and so incomplete. As such there is no verification on their applicability, their consistency, or their correctness. For less manual work during import, it is recommended that the data is consistent and valid when exported.

2.2. Use Case: Interface to producers of graphical and logical information

In this case the XML scheme provides an interface to a producing tool. An example of such a producer is a high level engineering tool that creates graphics and logical information on FB, program, and /or project level.
The producer of graphical and logical information will generate an XML file. This XML file can be based on some lists or tables defining the components to be used (e.g. parts of a plant) and on some template information for generating connections between these parts.
One major requirement for continuous development is the possibility to store custom data (e.g. a foreign key) with the elements in the generated XML and these elements should still be there when importing the XML into the programming tool and re-exporting to XML. This goal can be reached by specifying, in the XML, which attributes have to be preserved even if they are of no specific meaning for some XML processor. To support this, at three levels custom data can be added to objects: Project, data- and POU type, and variable (including FB instances).

2.3. Use Case: Interface to consumer of graphical and logical information.

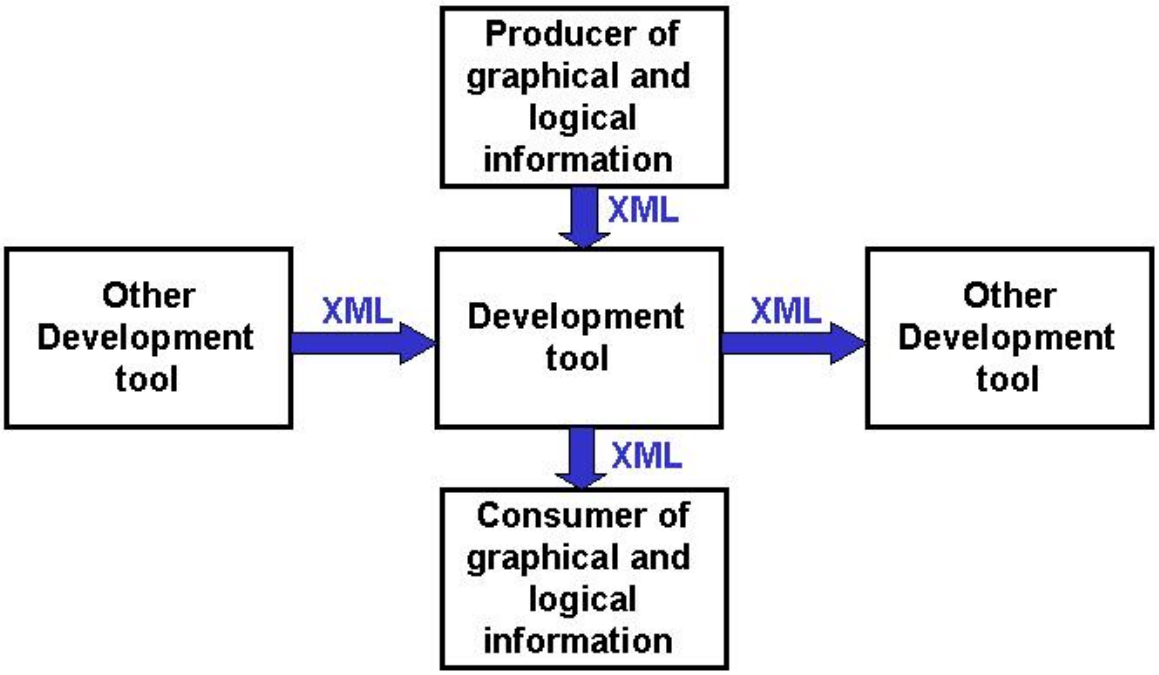
This use case provides an interface to supporting tools during the development phases, and is the counterpart of the use case above. Examples of consumers of information are validation tools, compilers, SCADA and HMI tools, as well as documentation generators; document management; source code database, version control, and document translation tools.

2.4. Use Case – Distribution format for function block libraries

This use case is focused to a format for distribution of (user derived) functions and function blocks specifically. With this, a user can create its own source library of their functions and function blocks as basis for different development systems.

2.5. Graphical overview of the Use Cases

The following picture provides an overview of these use cases. Horizontally is shown the import and export between tools of projects and POUs (use case 1 and 4). Vertically is shown use case 2 (top) and 3 (bottom).



3 Definitions, compliance, validation and transformations

3.1. Definitions

Project	A project consists of libraries and configurations. As such it contains a type part and an instance part.
Library	A library is a collection of data-types and POU types.
Element.	Any item as defined in the XML specification
Object	An element representing an (graphical) object from a PLC project. All objects will get a local number as identification by the generating system. This number is called LocalId and shall be unique within a POU code body.

3.2. Naming Conventions

Within the defined scheme, the following naming conventions are used:

- Prefixes are non-capitalized
- Identifiers start with lower case letter
- Identifiers consisting of multiple words have the first character of each word after the prefix starting with a capital letter. No underscores are used
- References to the 'xsd' elements are shown between double quotes, e.g. "element".

3.3. Compliance to IEC 61131-3 – 2nd edition

This scheme is intended to be compliant to the second edition of the IEC 61131-3 standard.

3.4. Compliance to supplier specific extensions

The goal is not to describe correct IEC 61131-3 POUs, but to represent a working state of the project, including extensions for layout and formatting.
It is possible to export syntactical incorrect projects. Such a project could be an in-between version. For instance, within FBD several unconnected blocks can be seen as a not-ready program, and as such can be exported by a system.

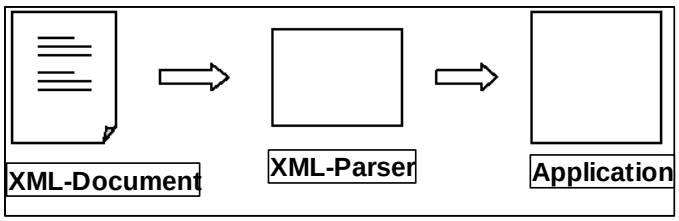
There are certain non-compliant IEC 61131-3 items added at pre-defined positions in the XML scheme, for better exchange of supplier specific extensions. The following items are defined:

- The support of pointers at datatypes
- The support for an Enum Base Type other then INT
- Persistent and non-persistent variables as LocalVars for Function Blocks
- VarInOut can have the attribute CONSTANT
- A Sub-sequence (like Macro) in SFC
- A 'JumpStep' in SFC
- A negated input (inverter) attribute at SFC transition condition input, step output, and action block output

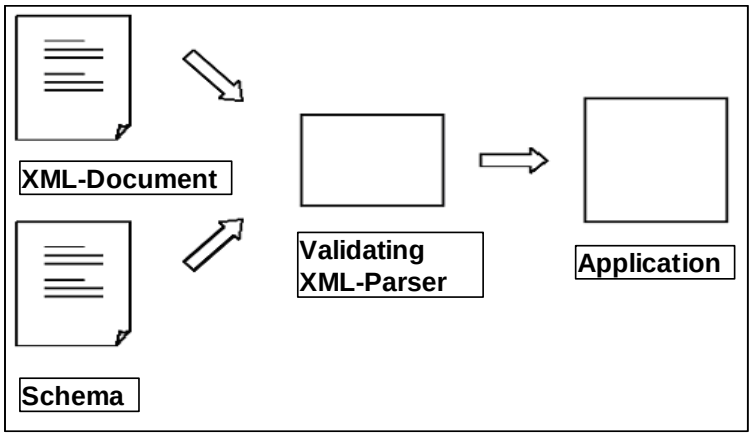
3.5. Validation, transformation and representation of XML Documents

PLCopen has defined a schema which all certified programming systems must support. Every supplier can create and publish enhanced schemas, in which the supplier specific properties are defined. These schemas can still be used within this context.
The usage of the different schemas and XML documents is explained below.

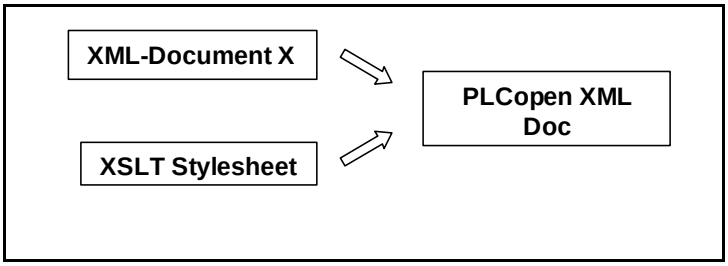
1. Flow for a well-defined document:



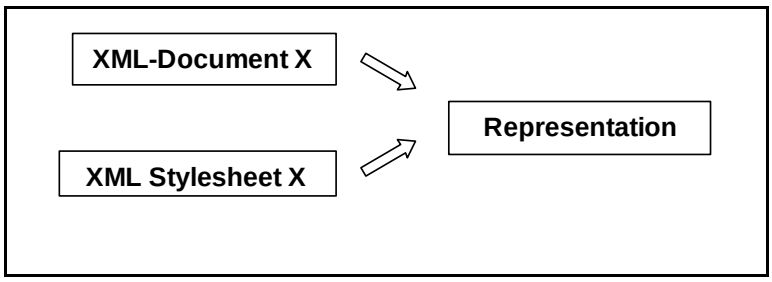
2. A valid XML-document is a well-defined XML-document of which the structure complies to a certain schema:



3. Transformation of Document of supplier X to a PLCopen XML document



4. Representation of Document of supplier X combined with an XML Style sheet

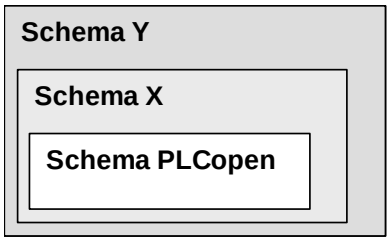


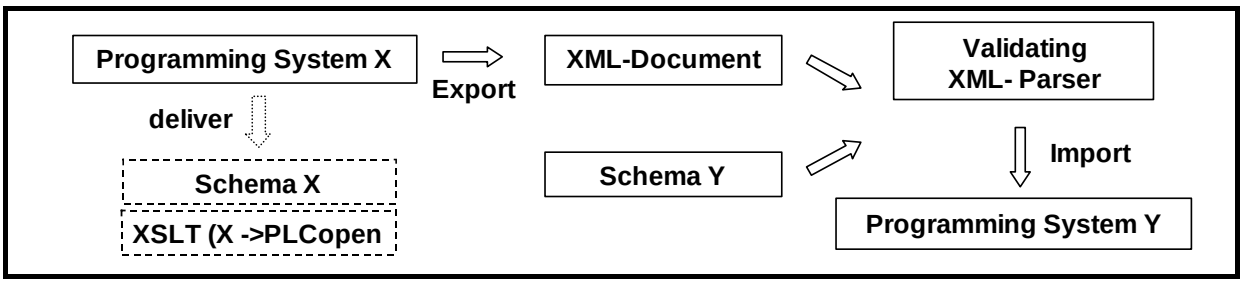
5. Extended usage of an XML Document

Each supplier, which does not directly support the PLCopen Schema, has to provide 3 files:

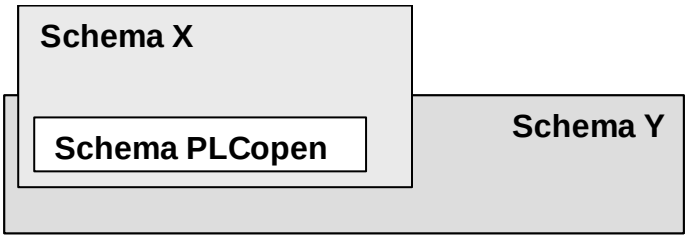
1. Their XML Scheme
2. Their transformation file to the PLCopen scheme (XML Style sheet - XSLT)
3. Their transformation from the PLCopen scheme (XSLT)

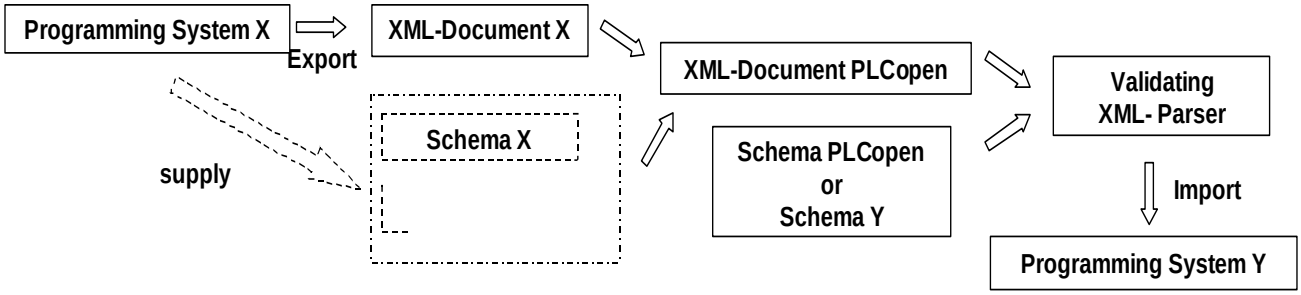
Example 1: Programming system Y can import schema X





Example 2: Programming system Y cannot import schema X




Programming System X
Export
XML-Document X
XML-Document PLCopen
Validating
XML- Parser
supply
Schema X
Schema PLCopen
or
Schema Y
Import
Programming System Y

(note: for the importing system the applicable formula is: Minimum Distance = CoordinateScaling * ApplicableCoordinate)
$X_{import} = X_{export} * (Scaling_{import} / Scaling_{export})$

Optionally, a page size is defined which lets applications map from absolute coordinates in the coordinate system to some page and relative coordinate space (in units).

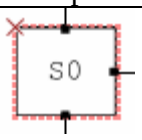
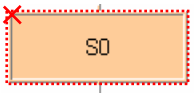
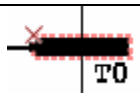
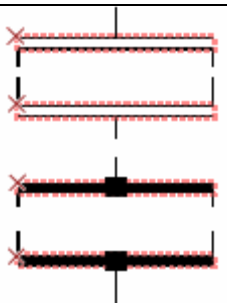
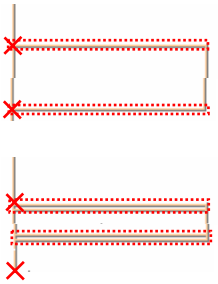
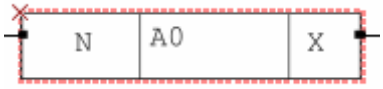
For mapping of the coordinate information to the coordinate system, the following basis is mandatory for the export:

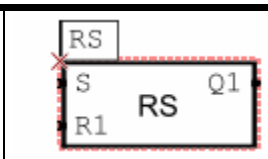
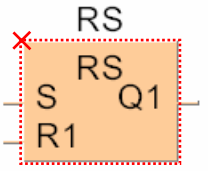
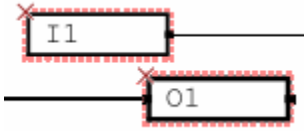
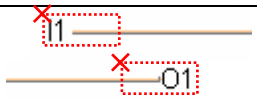
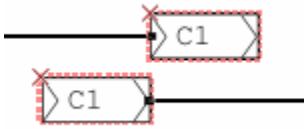
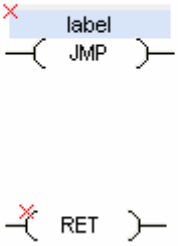
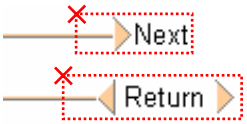
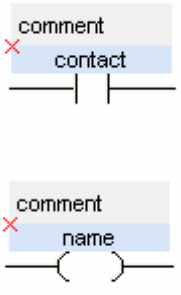
- For FBD – the relation between the applicable coordinate system used and the minimum distance between two pins, both for X and Y coordinates (CoordinateScaling)
- For LD – both the X and Y coordinates are the size of a coil (not including the variable name)
- For SFC – the size of a TRANSITION (width for X and height for Y)

See Chapter 5 - coordinateInfo on page 20 for the XML representation.

3.8. Positions

The “position” child node of an object specifies the position of the object’s anchor point. “position” has the attributes “x” and “y”.
The anchor point of an object is the upper left corner of the object rectangle.
The object rectangle contains the main body of the object. Attached elements like labels (instance name, coil name) or inverters can be outside of this rectangle.
The size of the object rectangle is specified by the “height” and “width” attributes of the object.

Object	Example 1	Example 2
Step, MacroStep, jumpStep		
transition		
selectionDivergence/Convergence, simultaneousDivergence/Convergence		
actionBlock		

block		
inVariable, outVariable		
inOutVariable		
connector, continuation		
Jump, return,		
leftPowerRail, rightPowerRail		
Contact, coil		

© PLCopen (2004, 2005)

3.9. Definition of the execution order of the graphical elements

For Function Block Diagrams, there is a possibility to explicitly document the execution order of the blocks. According to the standard this is implementation dependent.
A safer, explicit method has been used by providing an "executionId" attribute, which denotes the order of execution of all Functions and Function Blocks by unique integer numbers, and which is more flexible.

3.10. Reference of graphical elements

Every graphical element has a local ID for reference purposes.

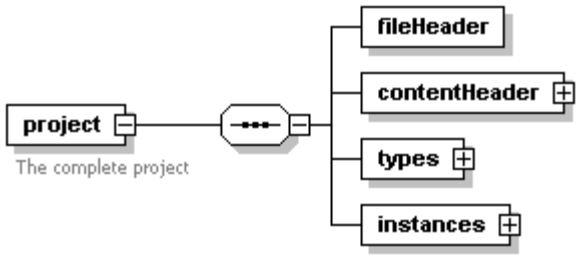
4 Overview of the scheme explanation

The following chapters explain the PLCopen schema. For this, the following structure is used:

- Project structure (chapter 5)
- Type specific part (chapter 6)
  - Datatypes
  - POUs – declaration section and code for both the graphical and textual languages
- Instance specific part (chapter 7)
  - Configuration
  - Resources
  - Tasks
  - Program instances
  - Global variables
  - Access paths

The last two chapters deal with certification and examples.

5 Project structure



5.1. Header information of an XML file

The file header and content header information originates from the publicly available specification of the IDA consortium and was specifically developed for usage in a context, where XML exchange files are generated from different tools of the same or of different vendors. It therefore contains information on the file generation and on the versioning of the content.
The content versioning information corresponds to the versioning information proposed for element types in Part 2 of the Function Block Standard IEC 61499. See the "VersionInfo" element in the enclosed proposal for Derived Data Type encoding.

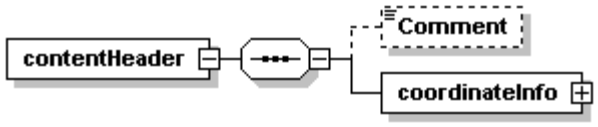
5.2. Header elements

fileHeader

Name	Type	Use	Default	Fixed
companyName	xsd:string	required		
companyURL	xsd:anyURI	optional		
productName	xsd:string	required		
productVersion	xsd:string	required		
productRelease	xsd:string	optional		
creationDateTime	xsd:dateTime	required		
contentDescription	xsd:string	optional		

The “FileHeader” element is used to provide information concerning the creation of the export / import file. Its mandatory attributes are the company name, (with optional Company URL), with product name, version, release information, and the date and time of the creation of the file. An optional description of the content is included.
Additionally the name of the company manufacturing and/or supplying the product may be included.
The format of the date and time is in conformance with the W3C consortium specification.

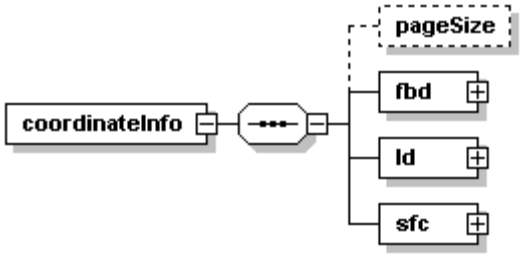
contentHeader



Name	Type	Use	Default
name	xsd:string	required	
version	xsd:string	optional	
modificationDateTime	xsd:dateTime	optional	
organization	xsd:string	optional	
author	xsd:string	optional	
language	xsd:language	optional	

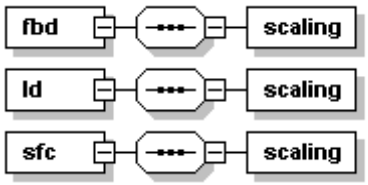
The “contentHeader” element is used to provide overview information concerning the actual content of the export / import file.
The "name" attribute is required. In case of exporting this attribute is set to the project name.
The other attributes correspond to the equally named attributes of the "VersionInfo" element as defined in IEC 61499-2. The "comment" element corresponds to the "Remarks" attribute of the "VersionInfo" element.
The attribute “language” is intended to specify the used language in the definition of the project.
The “comment’ element consists of a string.
The element “coordinateInfo” contains the information for the mapping of the coordinate system. See: 3.7 Definition of the coordinate system for graphical information:.

coordinateInfo



pageSize

Name	Type	Use	Default
x	xsd:decimal	required	
y	xsd:decimal	required	



scaling

Name	Type	Use	Default
x	xsd:decimal	required	
y	xsd:decimal	required	

6 Type specific part

6.1. Defined Datatypes

A datatype can be either an elementary type:
BOOL, BYTE, WORD, DWORD, LWORD, SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL, TIME, DATE, DT, TOD, STRING, WSTRING;

a derived type:
ARRAY, DERIVED, ENUM, SUBRANGESIGNED, SUBRANGEUNSIGNED, STRUCT;

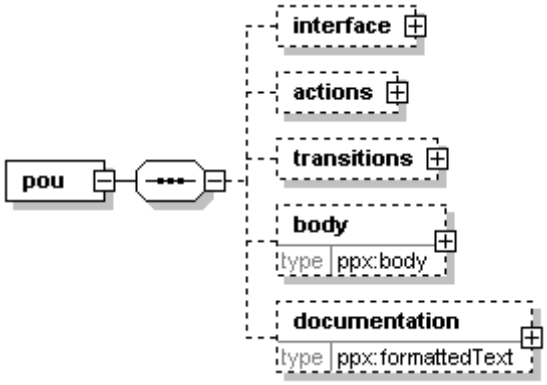
or an extended type:
POINTER.

Notes:

- The string types additionally are defined by an optional string length.
- DERIVED is a reference to a user defined data type or POU. Variable declarations use this type to declare for instance function block instances.
- ENUM is an enumerated type and is defined by a list of required values and one optional base type.
- SUBRANGESIGNED and SUBRANGEUNSIGNED are defined by a required range and a required base type. The range of SUBRANGESIGNED is ‘long’ and the range of SUBRANGEUNSIGNED is ‘unsigned long’.
- STRUCT is a structured type and is defined by a list of variables.
- In addition to the IEC 61131-3 standard, a datatype can be of the type POINTER. A pointer is defined by its required base type.

6.2. POUs

POUs consist of zero or more POU 's. A POU is defined as:



Name	Type	Use	Default
name	xsd:string	required	
pouType	ppx:pouType	required	

The "interface" element contains the declaration information (see hereunder). The code section is represented as a list of actions, transitions, or a body. Documentation can be added to a POU element.

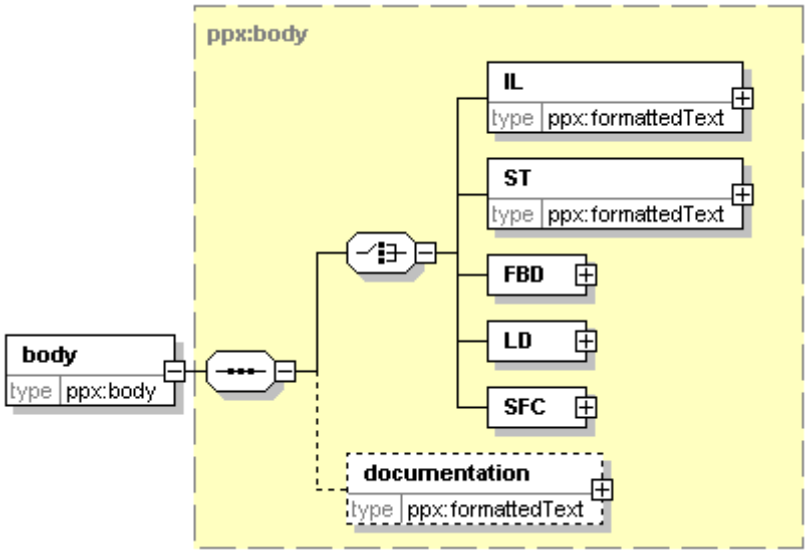
actions

The element "actions" can consist of zero or more elements of type "action". These consist of zero of more elements of "body" and optional "documentation".

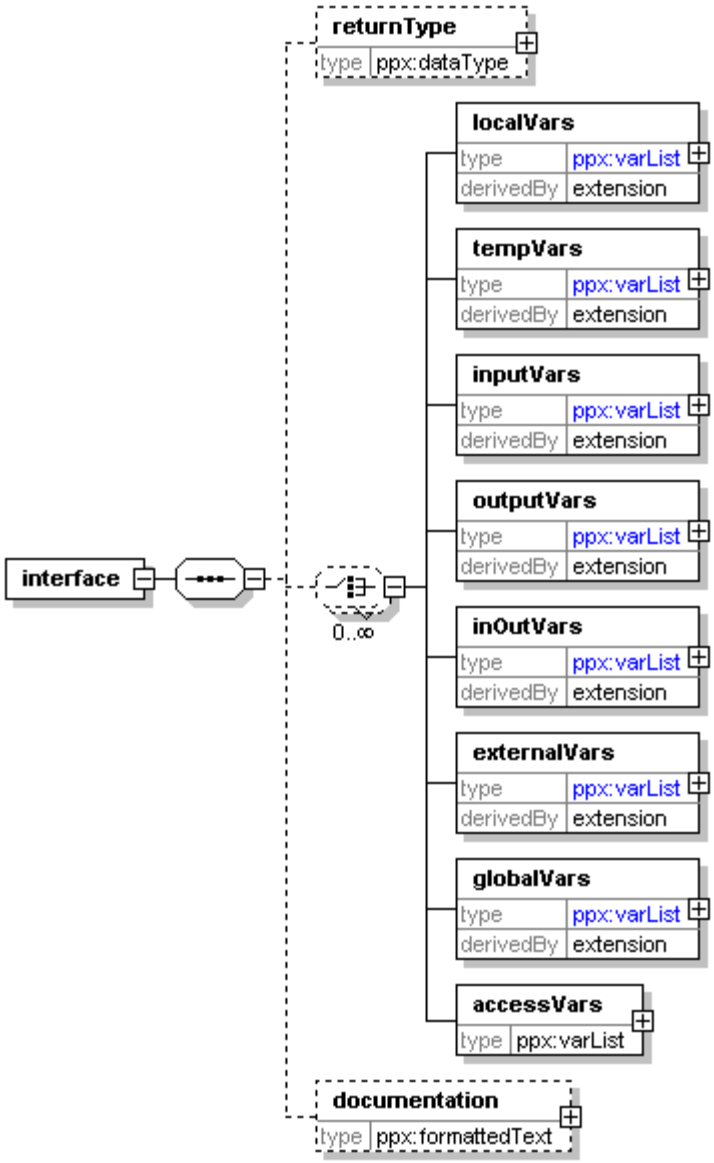
transitions

The element "transitions" can consist of zero or more elements of type "transition". These consist of zero of more elements of "body" and optional "documentation".

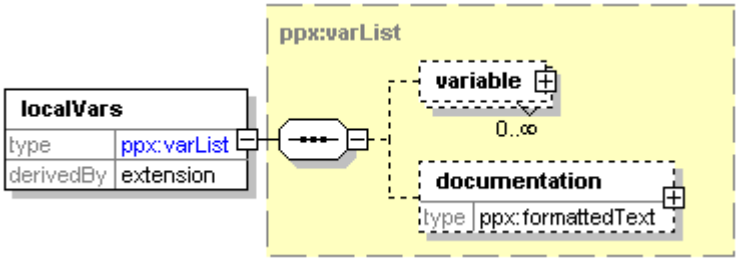
body



6.3. POUs – declaration section



The interface of a POU represents a return type (Functions), and a list of several kinds of variables: local variables, temporary variables, input variables, output variables, input/output variables, external variables, global variables and access path variables. They are defined with the same XML structure, with the same attributes. For example:



Name	Type	Use	Default
name	xsd:string	optional	
constant	xsd:boolean	optional	false
retain	xsd:boolean	optional	false
nonretain	xsd:boolean	optional	false
persistent	xsd:boolean	optional	false
nonpersistent	xsd:boolean	optional	false

A variable or variable list is defined by an optional “name”, and an optional “documentation”. Also following attributes can be defined: constant, retain, no retain, persistent and non persistent. The global variables are listed here also, and not under Chapter 7 Instance specific part.

6.4. POUs – code section

6.4.1. General

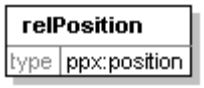
Following elements will be used to describe objects in graphical languages:

position

Name	Type	Use	Default
x	xsd:decimal	required	
y	xsd:decimal	required	

The element “position” is used to express coordinate values. Both coordinate values “x” and “y” are of type unsigned integer.

relPosition



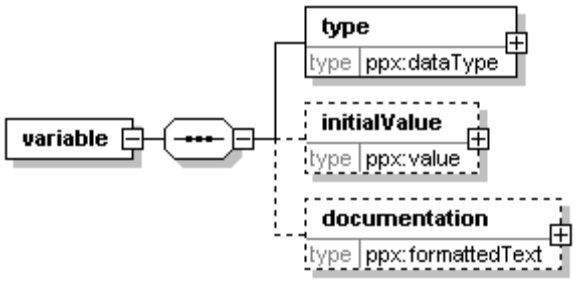
Name	Type	Use	Default
x	xsd:decimal	required	
y	xsd:decimal	required	

Relative position of the connection pin. The origin is the anchor of the block.

content

The element “content” is represented as formatted text.

variable

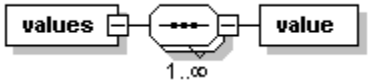


The element “variable” is a valid IEC 61131-3 variable e.g. avar[0]

expression

The operand is a valid IEC variable e.g. avar[0] or an IEC expression or multiple token text e.g. a + b (*sum*). An IEC 61131-3 parser has to be used to extract variable information.

values



A value contains a required name and an optional value, both as strings.

documentation

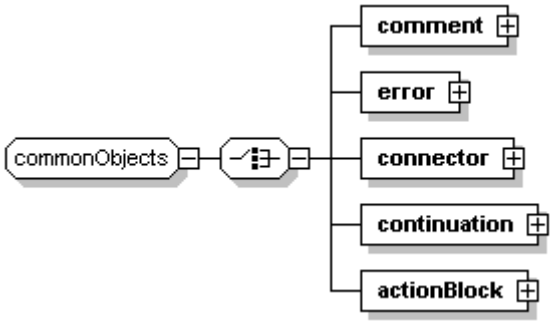
The element “Documentation” contains formatted text.

6.5. Commonalities of graphical languages

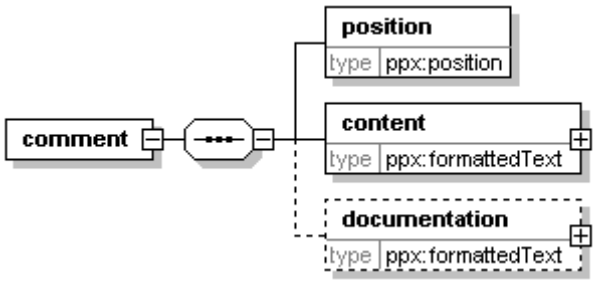
The following objects, as defined in “commonObjects”, can be used in any graphical body and have no direct IEC 61131-3 scope.

Overview Common Objects

The Common Objects are a collection of objects which have no direct IEC scope and can be used in any graphical body. The graphical representation is as follows:



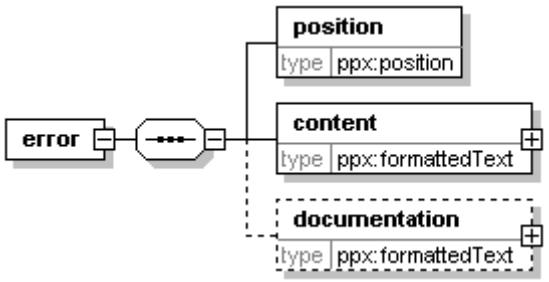
comment



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	required	
width	xsd:decimal	required	

The element “comment” is used to store arbitrary text strings, which are not associated with a graphic location. They are for example presented inside a dialog box/dialog window within the GUI. “comment” elements are sparsely used in language elements.

error



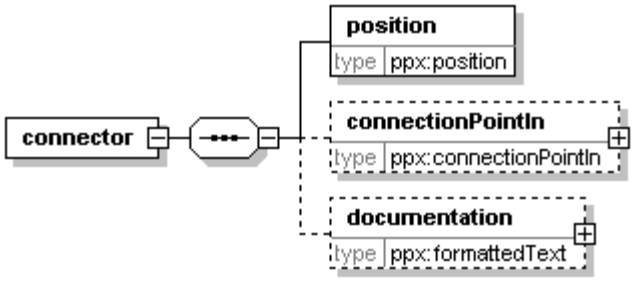
Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	required	
width	xsd:decimal	required	

Like the element “comment”, the element “error” is used to store text strings inside a rectangular frame,

- Which is associated with a graphic location and,
- Which has a defined height and width.

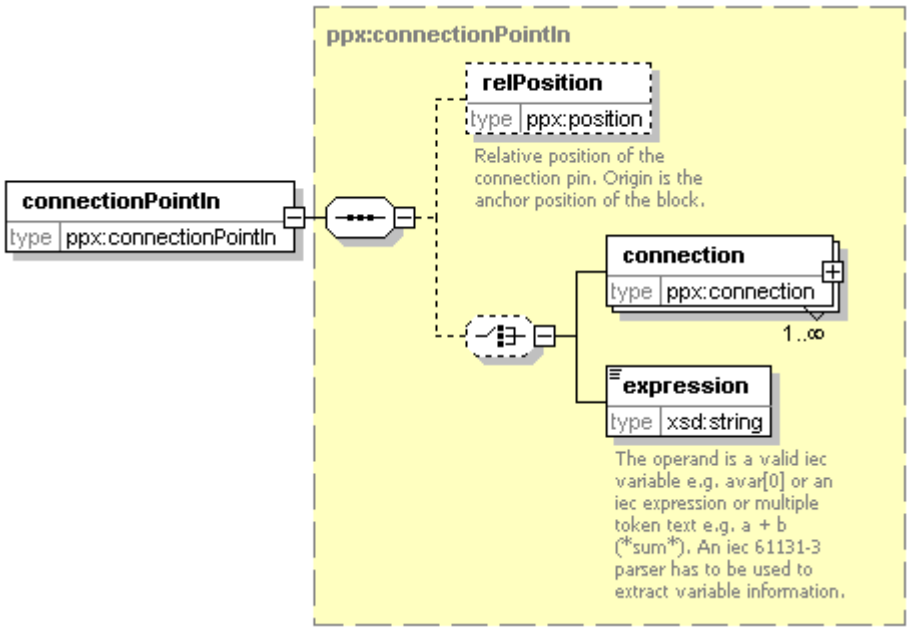
In contrast to the comment box, the error box and the text inside that box are automatically created by software utilities to indicate failures during conversion operations. Examples of those utilities are language converters for legacy languages. They create the error boxes at locations where the corresponding graphical objects would have been created in case of correct conversion.

connector

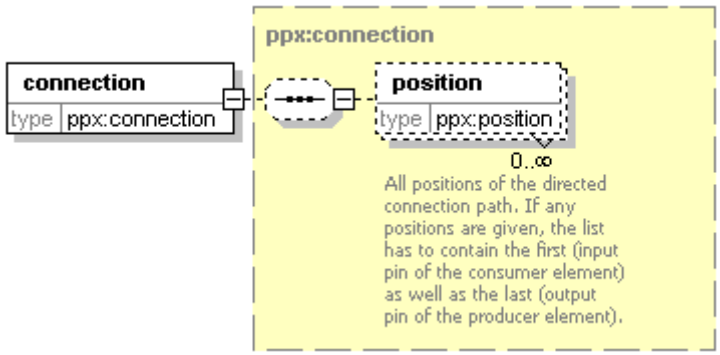


Name	Type	Use	Default
name	xsd:string	required	
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	

connectionPointIn



connection

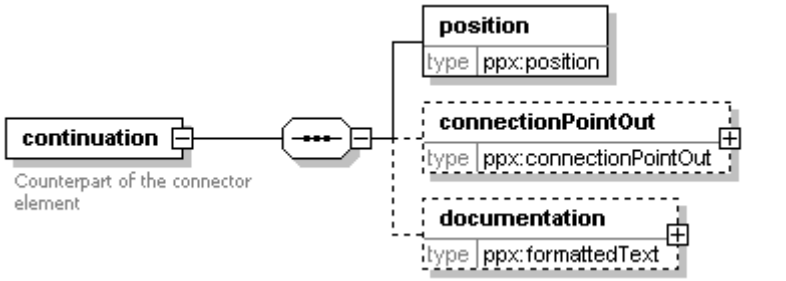


Name	Type	Use	Default
refLocalId	xsd:unsignedLong	required	
formalParameter	xsd:string	optional	

A “connection” describes a graphical coupling between a data consuming element (e.g. an input variable) and another element, which provides the data (output variable, and normally is in front of the father element, so right to left). It may contain a list of positions that describes the path or trajectory of the connection. If positions are exported, the starting point (e.g. input) and the end point (e.g. output) are part of the list (although redundant).
If no position information is provided, the link must be routed automatically.
“refLocalId” identifies the element the connection starts from.
If present, “formalParameter” specifies the element’s output the connection starts from. “formalParameter” either denotes the name of the VAR_OUTPUT / VAR_IN_OUT parameter of a POU block or refers to the “formalParameter” attribute of the corresponding “connectionPointOut”. If this “formalParameter” is not present:

- If the “refLocalId” attribute refers to a POU block, the start of the connection is the first output of this block, which is not ENO.
- If the “refLocalId” attribute refers to any other element type, the start of the connection is the elements single native output.

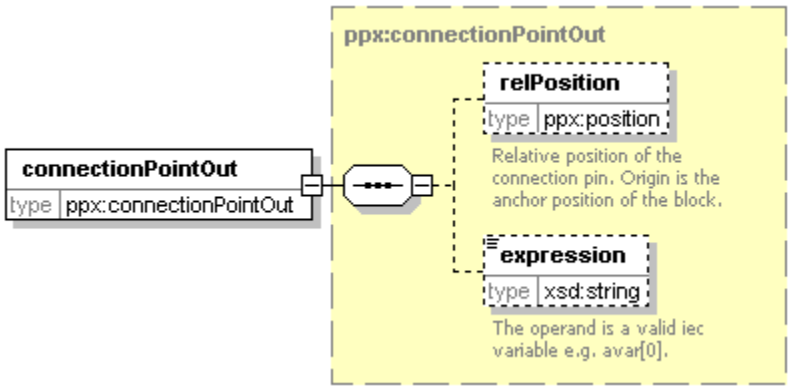
continuation



Name	Type	Use	Default
name	xsd:string	required	
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	

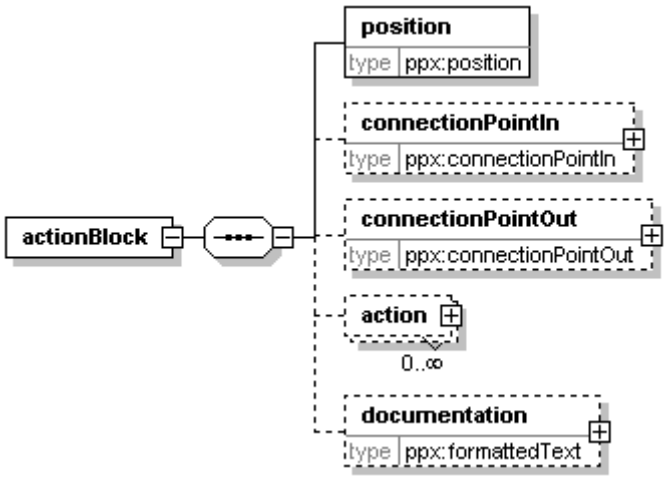
connectionPointOut

This is the counterpart of the connector element. For this reason it has a simpler “connectionPoint”:

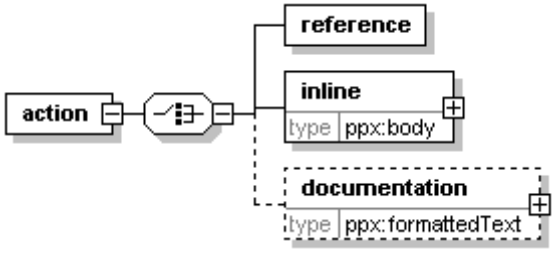


actionBlock

These are part of the common objects because there scope is beyond SFC.

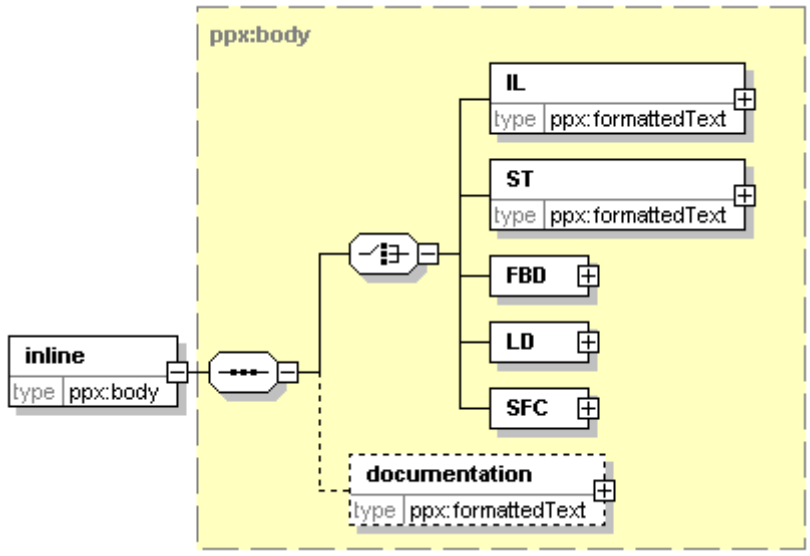


Name	Type	Use	Default
negated	xsd:boolean	optional	false



Name	Type	Use	Default
qualifier	xsd:NMTOKEN	optional	N
duration	xsd:string	optional	
indicator	xsd:string	optional	

The element “reference” is a string, containing the name of an action or Boolean variable.
The element “inline” is referencing to the in line implementation of an action body.



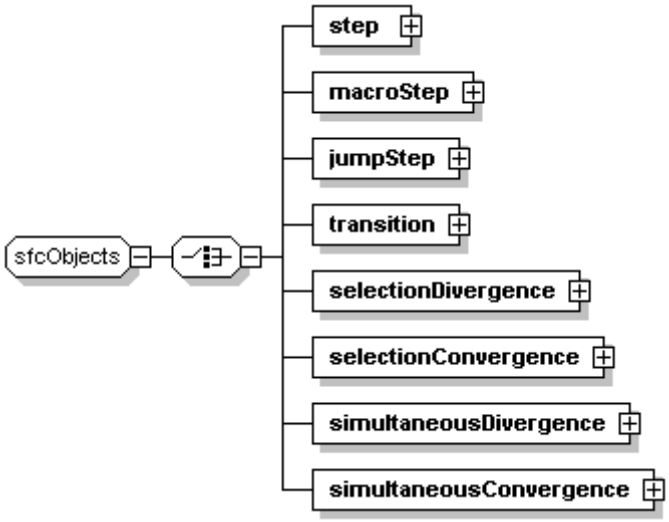
Additional information

The following objects can be used in any graphical body and have no direct IEC 61131-3 scope,

- A textual comment string not belonging to the graphical pane.
- A textual error description generated by language converters.
- Free floating lines, without any connections, will not be exported in the graphical languages, which can have an effect on POUs that are not yet readily defined.

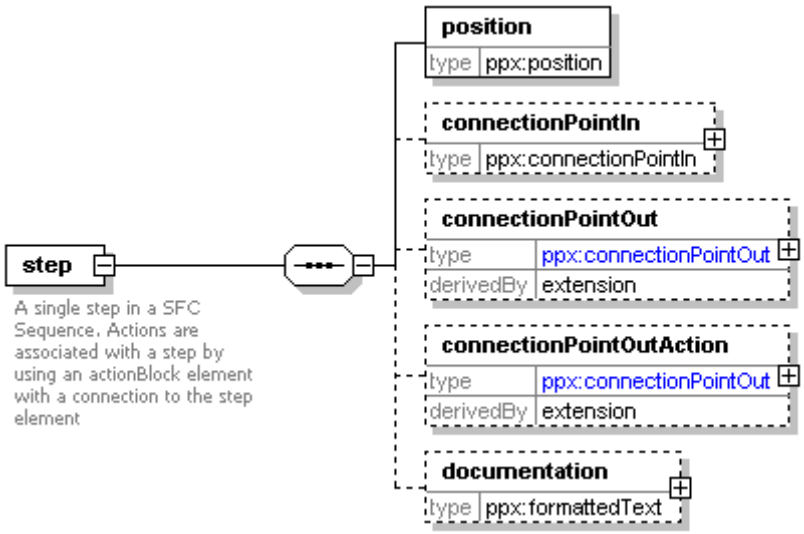
6.5.1. SFC elements

The SFC elements contain a collection of objects, which are defined in SFC. They can only be used in SFC bodies.



step

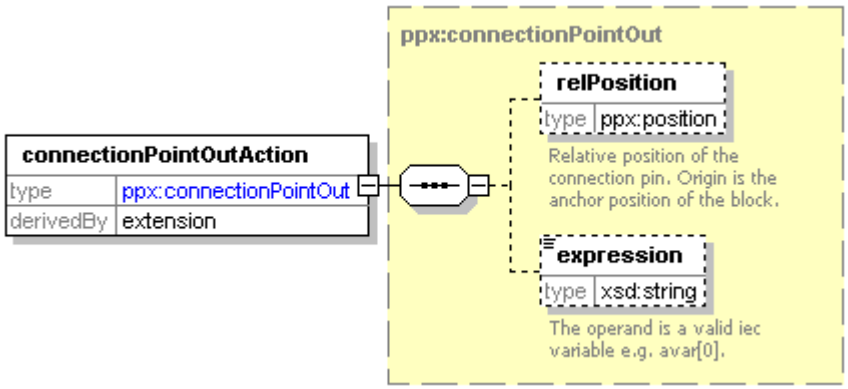
The element ‘step’ provides a single step in a SFC Sequence. Actions are associated with a step by using an actionBlock element with a connection to the step element.



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
name	xsd:string	required	
initialStep	xsd:boolean	optional	false
negated	Xsd:boolean	optional	false

Note: there is no finalStep attribute. This is either explicit connected or via an export of jumpStep.

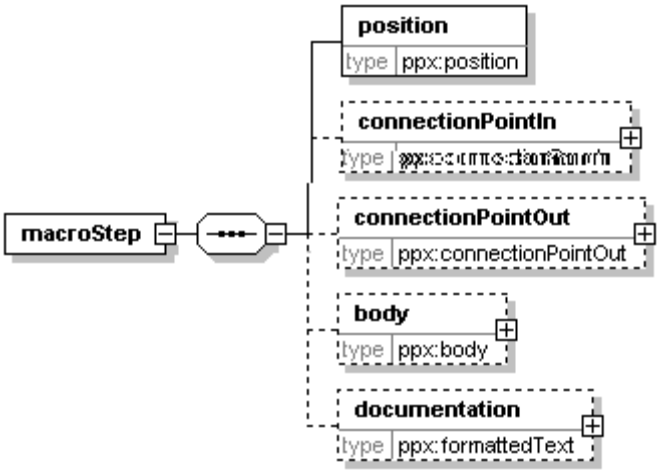
connectionPointOutAction



Name	Type	Use	Default
formalParameter	xsd:string	required	

macroStep

This element is beyond the IEC 61131-3 scope. It provides a graphical representation of several steps and transitions into one element.

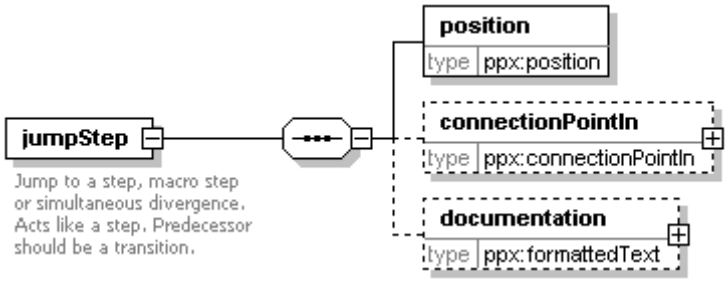


Name	Type	Use	Default	Annotation
localId	xsd:unsignedLong	required		
height	xsd:decimal	optional		
width	xsd:decimal	optional		
name	xsd:string	optional		

The macroStep body can consist of any of the IEC 61131-3 programming languages, incl. SFC. Documentation is optional to it.

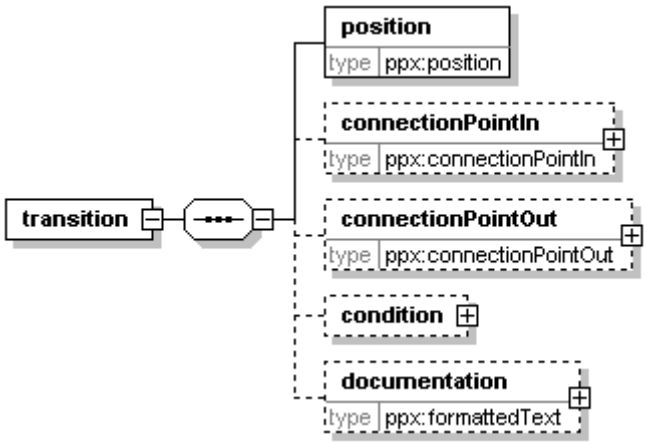
jumpStep

This element is beyond the IEC 61131-3 scope. It provides a graphical representation of jump to a label coupled to a step.



Name	Type	Use	Default	Annotation
localId	xsd:unsignedLong	required		
height	xsd:decimal	optional		
width	xsd:decimal	optional		
targetName	xsd:string	required		

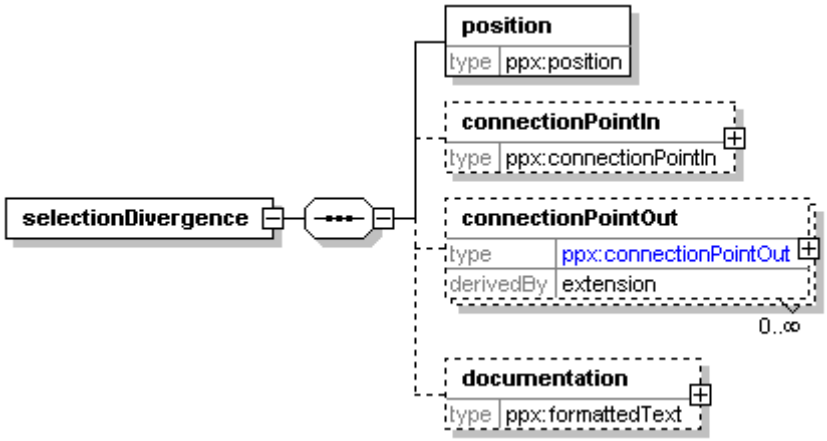
transition



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
priority	xsd:unsignedLong	optional	

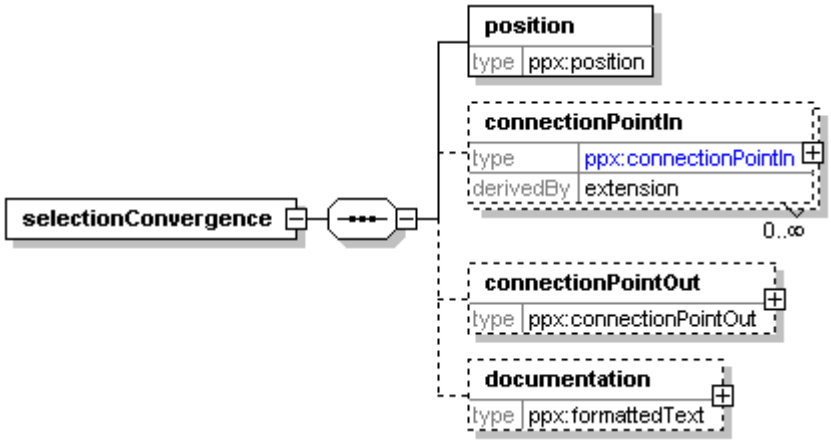
The “condition” can either contain a reference, a connection, or inline code programmed in any of the five languages.

selectionDivergence



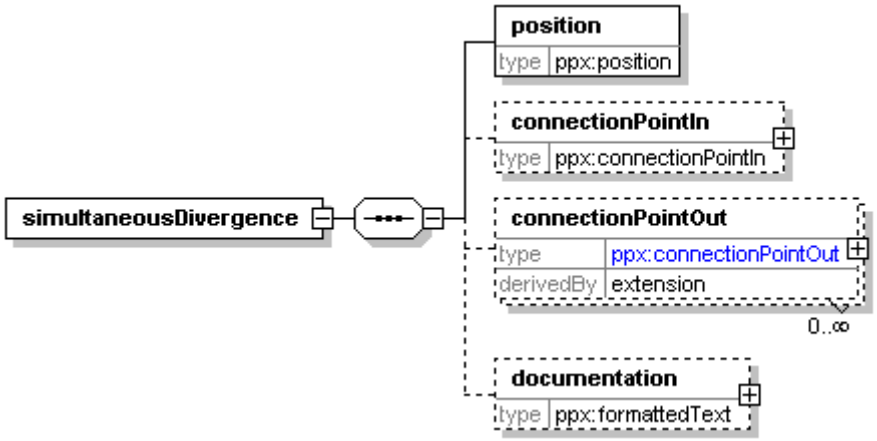
Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	

selectionConvergence



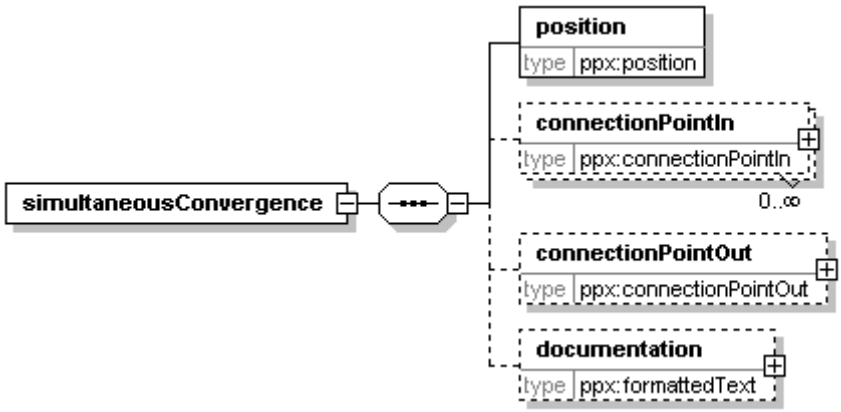
Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	

simultaneousDivergence



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
name	xsd:string	optional	

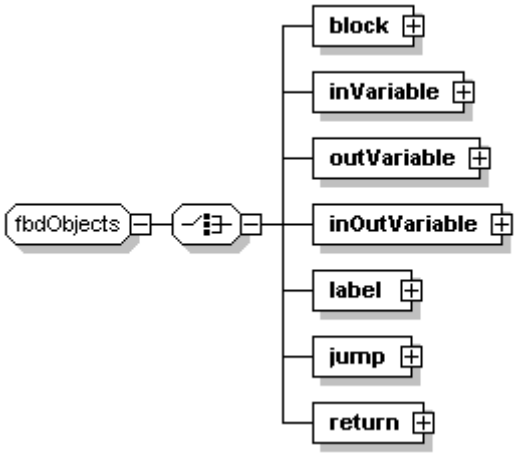
simultaneousConvergence



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	

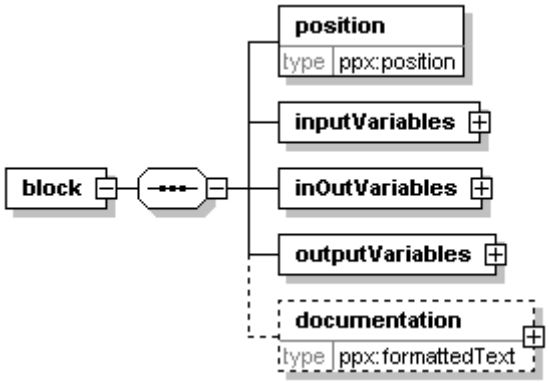
6.5.2. FBD elements

These elements are a collection of objects, which are defined in FBD. They can be used in all graphical bodies.



block

A block is a graphical representation of an operation on a function or a function block.

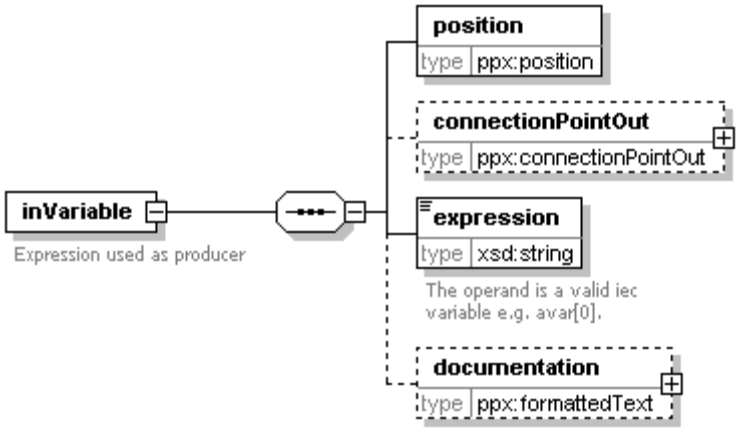


Name	Type	Use	Default
localId	xsd:unsignedLong	required	
width	xsd:decimal	optional	
height	xsd:decimal	optional	
typeName	xsd:string	required	
instanceName	xsd:string	optional	
executionOrderId	xsd:unsignedLong	optional	

The three types of variables each contain zero or more variables. The input variable can have an element “connectionPointIn”. The output variable can have an element “connectionPointOut”. The inOutVariable can have a “connectionPointIn” and a “connectionPointOut”.

inVariable

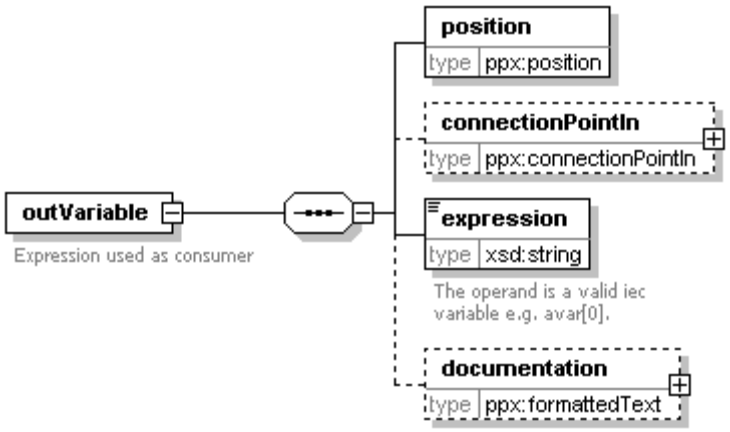
The element “inVariable” represents an input variable. The input variable can be negated; an edge modifier and a storage modifier can be entered.
The element “edge” of type “edgeModifierType” defines the edge detection behaviour (rising / falling / none) of a variable.
The “storage” element of type “storageModifierType” defines the storage mode behaviour (set / reset / none) of a variable.



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
executionOrderId	xsd:unsignedLong	optional	
negated	xsd:boolean	optional	false
edge	ppx:edgeModifierType	optional	none
storage	ppx:storageModifierType	optional	none

outVariable

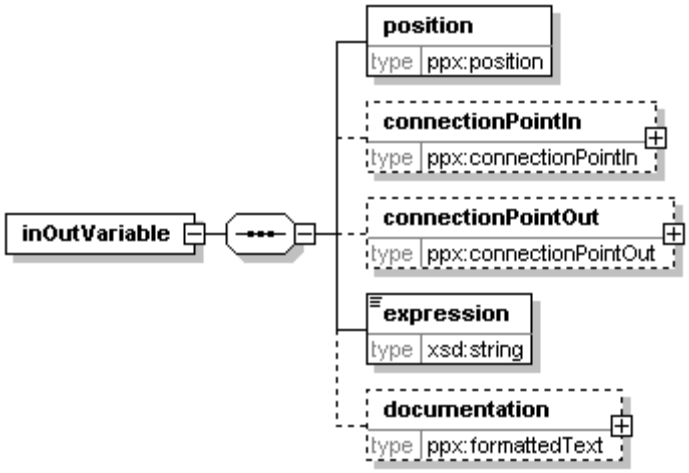
The element “outVariable” represents an output variable. The output variable can be negated; an edge modifier and a storage modifier can be entered.
The element “edge” defines the edge detection behaviour (rising / falling / none) of a variable.
The element “storage” defines the storage mode behaviour (set / reset / none) of a variable.



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
executionOrderId	xsd:unsignedLong	optional	
negated	xsd:boolean	optional	false
edge	ppx:edgeModifierType	optional	none
storage	ppx:storageModifierType	optional	none

inoutVariable

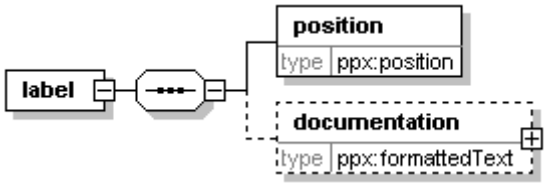
The element “inoutVariable” represents an inout variable. The input and the output can be negated; an edge modifier and a storage modifier can be entered.
The element “edge” defines the edge detection behaviour (rising / falling / none) of a variable.
The element “storage” defines the storage mode behaviour (set / reset / none) of a variable.



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
executionOrderId	xsd:unsignedLong	optional	
negatedIn	xsd:boolean	optional	false
edgeIn	ppx:edgeModifierType	optional	none
storageIn	ppx:storageModifierType	optional	none
negatedOut	xsd:boolean	optional	false
edgeOut	ppx:edgeModifierType	optional	none
storageOut	ppx:storageModifierType	optional	none

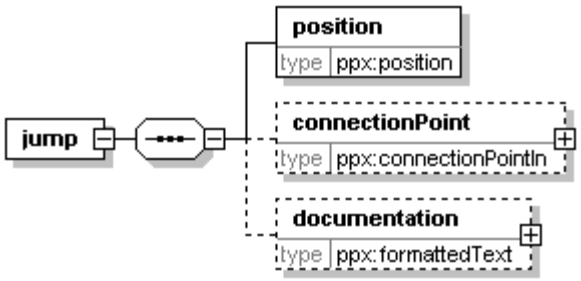
label

The element “label” is the target of a jump.



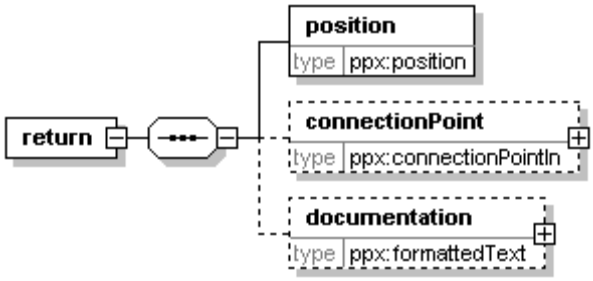
Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
label	xsd:string	required	
executionOrderId	xsd:unsignedLong	optional	

jump



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
label	xsd:string	required	
executionOrderId	xsd:unsignedLong	optional	

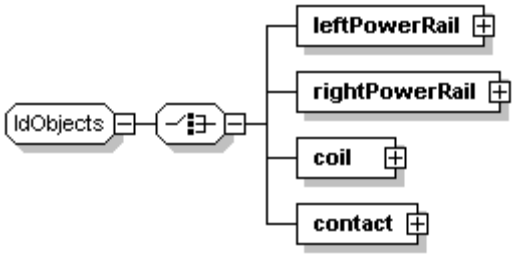
return



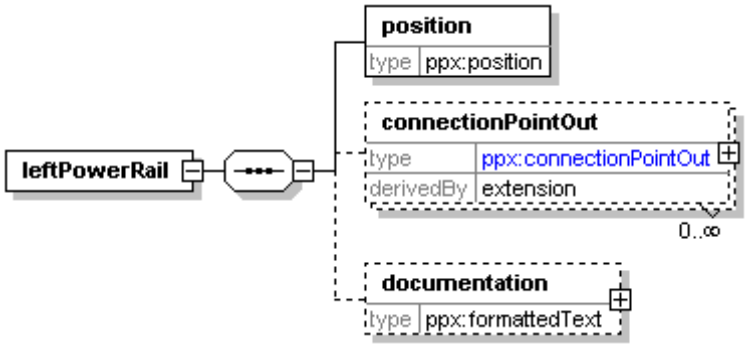
Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
executionOrderId	xsd:unsignedLong	optional	

6.5.3. LD elements

These elements describe a collection of objects, which are defined in LD, and are an extension to FBD. They can be used in LD and SFC bodies

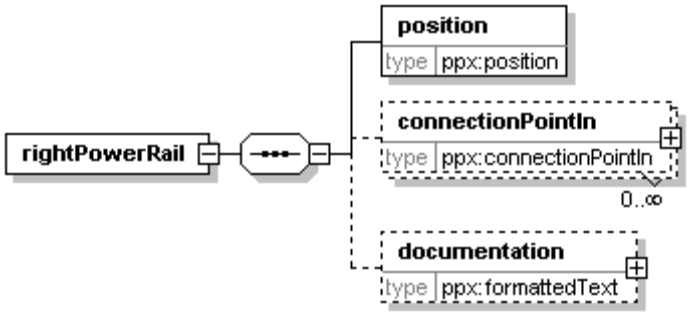


leftPowerRail



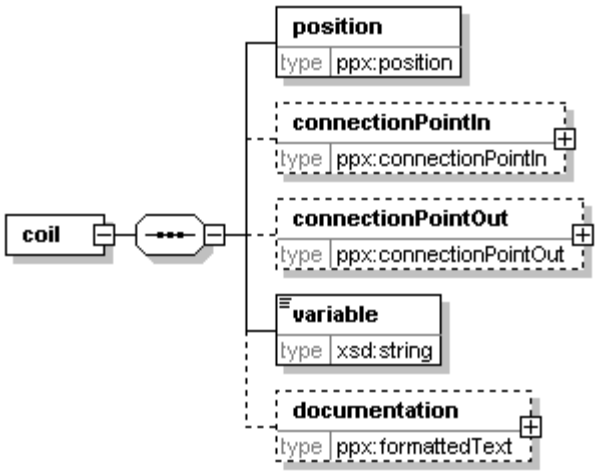
Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	

rightPowerRail



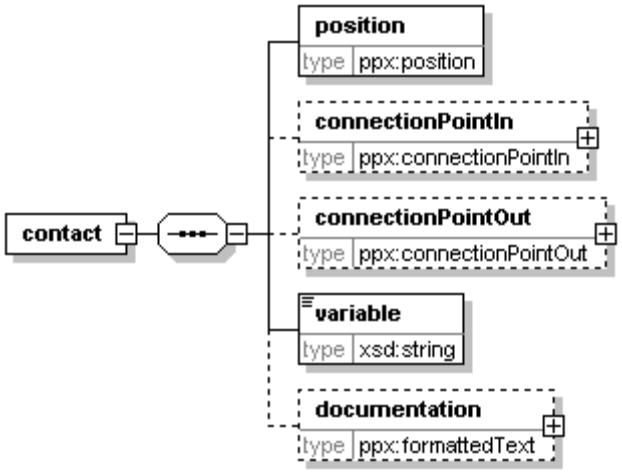
Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	

coil



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
executionOrderId	xsd:unsignedLong	optional	
negated	xsd:boolean	optional	false
edge	ppx:edgeModifierType	optional	none
storage	ppx:storageModifierType	optional	none

contact



Name	Type	Use	Default
localId	xsd:unsignedLong	required	
height	xsd:decimal	optional	
width	xsd:decimal	optional	
executionOrderId	xsd:unsignedLong	optional	
negated	xsd:boolean	optional	false
edge	ppx:edgeModifierType	optional	none
storage	ppx:storageModifierType	optional	none

6.6. Schema Textual Languages:

6.6.1. Structured Text (ST) and Instruction List (IL)

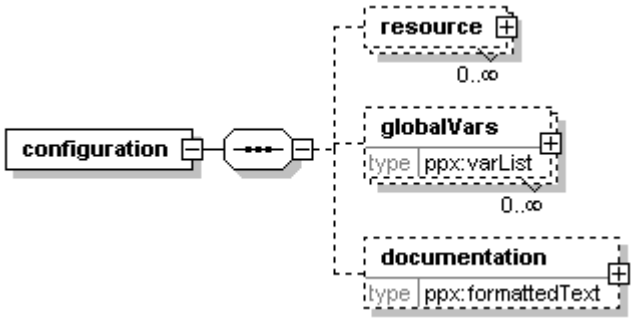
The textual languages are represented as formatted text based on XHTML.

7 Instance specific part

Instances can contain a “configurations” element, which consists of zero or more elements “configuration”.

7.1. configuration

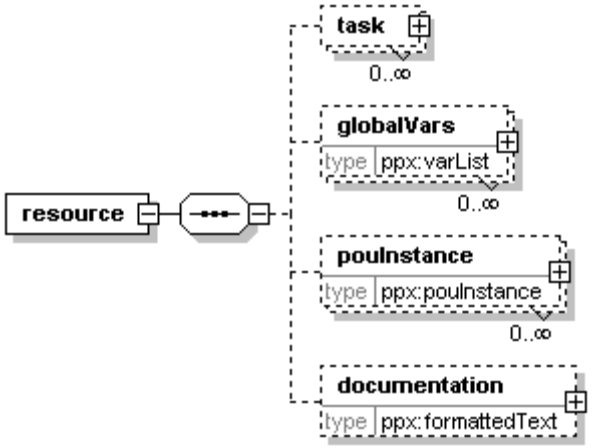
The element “configuration” represents a group of resources and global variables. It is identified by a required name .



Name	Type	Use	Default
name	xsd:string	required	

7.2. resource

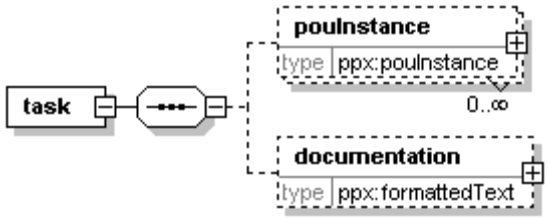
The element “resource” represents a group of programs, tasks and global variables. It is identified by a required name.



Name	Type	Use	Default
name	xsd:string	required	

7.3. task

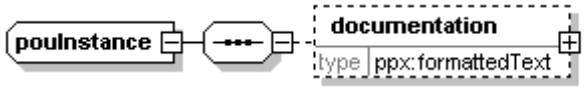
The element “task” represents a periodic or triggered task and consists of a group of program and / or function block instances. It is defined by a required priority, an optional single, and an optional interval time.



Name	Type	Use	Default
name	xsd:string	required	..
single	xsd:string	optional	
interval	xsd:time	optional	
priority	xsd:integer	required	

7.4. POU instances

The element “pouInstance” represents a program or function block instance either running with or without a task.



Name	Type	Use	Default
name	xsd:string	required	
type	ppx:pouType	required	

8 Certification

PLCopen members, which have submitted their relevant files and which are published on the PLCopen website, do fulfil the basic requirements, and can use the following logo:

This logo is owned and trademarked by PLCopen.

The mentioned relevant files can include:

1. Their XML Scheme
2. Their transformation file to the PLCopen scheme (XML Stylesheet - XSLT)
3. Their transformation from the PLCopen scheme (XSLT)

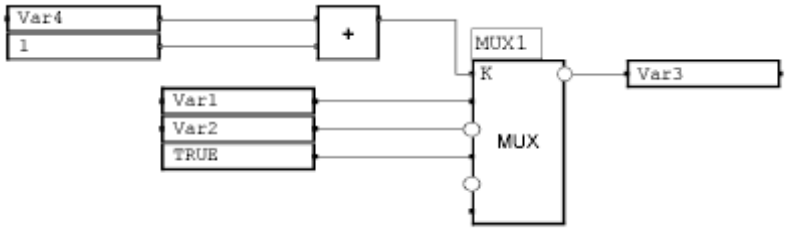
In order to use this logo free-of-charge, the relevant company has to fulfill all the following requirements:

1. The company has to be a voting member of PLCopen;
2. The company has to comply to the existing specification, as specified by the PLCopen Technical Committee 6 – XML, and as published by PLCopen, and of which this statement is a part;
3. This compliance application is provided in written form by the company to PLCopen, clearly stating the applicable software package and the supporting elements as specified in this document;
4. In case of non-fulfillment, which has to be decided by PLCopen, the company will receive a written statement concerning this from PLCopen. The company will have a one month period to either adopt their software package in such a way that it complies, represented by the issuing of a new compliance statement, or remove all reference to the specification, including the use of the logo, from all their material, be it technical or promotional;
5. The logo has to be used as is - meaning the full logo. it may be altered in size as long as the original scale and color setting is kept.
6. The logo has to be used in the context of PLCopen XML.

9 Examples

9.1. Simple example for FBD

Graphical Representation



XML example code

```
<?xml version="1.0" encoding="UTF-8"?>
<project   xmlns="http://www.plcopen.org/xml/tc6.xsd"
           xmlns:xhtml="http://www.w3.org/1999/xhtml"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.plcopen.org/xml/tc6.xsd http://www.plcopen.org/xml/tc6.xsd">
  <fileHeader companyName="plcopen"
              creationDateTime="2005-01-25T14:18:00"
              productName="NoNameNow"
              productVersion="1"/>
  <contentHeader name="prj">
    <coordinateInfo>
      <fbd>
        <scaling x="16" y="16"/>
      </fbd>
      <ld>
        <scaling x="16" y="16"/>
      </ld>
      <sfc>
        <scaling x="16" y="16"/>
      </sfc>
    </coordinateInfo>
  </contentHeader>
  <types>
    <dataTypes/>
    <pous>
      <pou name="fork1" pouType="functionBlock">
        <interface>
          <localVars>
            <variable name="Var1">
              <type>
                <BOOL/>
              </type>
            </variable>
            <variable name="Var2">
              <type>
                <BOOL/>
              </type>
```

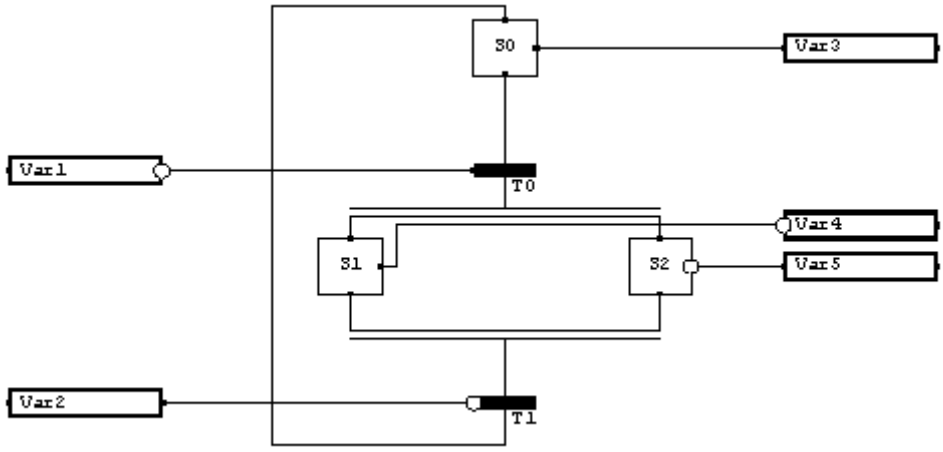
```
        </variable>
        <variable name="Var3">
          <type>
            <BOOL/>
          </type>
        </variable>
        <variable name="Var4">
          <type>
            <INT/>
          </type>
          <initialValue>
            <simpleValue value="1"/>
          </initialValue>
        </variable>
      </localVars>
    </interface>
    <body>
    <FBD>

      <inVariable height="16" localId="1" width="80">
        <position x="80" y="32"/>
        <connectionPointOut>
          <relPosition x="80" y="8"/>
        </connectionPointOut>
        <expression>Var4</expression>
      </inVariable>

      <inVariable height="16" localId="2" width="80">
        <position x="80" y="48"/>
        <connectionPointOut>
          <relPosition x="80" y="8"/>
        </connectionPointOut>
        <expression>1</expression>
      </inVariable>

      <block height="32" localId="3" typeName="ADD" instanceName="ADD" width="32">
        <position x="240" y="32"/>
        <inputVariables>
          <variable formalParameter="IN1">
            <connectionPointIn>
              <relPosition y="8" x="0"/>
              <connection refLocalId="1">
                <position x="240" y="40"/>
                <position x="160" y="40"/>
              </connection>
            </connectionPointIn>
          </variable>
          <variable formalParameter="IN2">
            <connectionPointIn>
              <relPosition y="24" x="0"/>
              <connection refLocalId="2">
                <position x="240" y="56"/>
                <position x="160" y="56"/>
              </connection>
            </connectionPointIn>
          </variable>
        </inputVariables>
        <inOutVariables/>
        <outputVariables>
          <variable formalParameter="OUT1">
```

```
          <connectionPointOut>
            <relPosition x="32" y="8"/>
          </connectionPointOut>
        </variable>
      </outputVariables>
    </block>

    <inVariable height="16" localId="4" width="80">
      <position x="160" y="80"/>
      <connectionPointOut>
        <relPosition x="80" y="8"/>
      </connectionPointOut>
      <expression>Var1</expression>
    </inVariable>

    <inVariable height="16" localId="5" width="80">
      <position x="160" y="96"/>
      <connectionPointOut>
        <relPosition x="80" y="8"/>
      </connectionPointOut>
      <expression>Var2</expression>
    </inVariable>

    <inVariable height="16" localId="6" width="80">
      <position x="160" y="112"/>
      <connectionPointOut>
        <relPosition x="80" y="8"/>
      </connectionPointOut>
      <expression>TRUE</expression>
    </inVariable>

    <block height="96" instanceName="MUX1" localId="7" typeName="MUX" width="48">
      <position x="320" y="64"/>
        <inputVariables>
          <variable formalParameter="K">
            <connectionPointIn>
              <relPosition x="0" y="8"/>
              <connection formalParameter="OUT1" refLocalId="3">
                <position x="320" y="72"/>
                <position x="312" y="72"/>
                <position x="312" y="40"/>
                <position x="272" y="40"/>
              </connection>
            </connectionPointIn>
          </variable>
          <variable formalParameter="IN1">
            <connectionPointIn>
              <relPosition x="0" y="24"/>
              <connection refLocalId="4">
                <position x="320" y="88"/>
                <position x="160" y="88"/>
              </connection>
            </connectionPointIn>
          </variable>
          <variable formalParameter="IN2" negated="true">
            <connectionPointIn>
              <relPosition x="0" y="40"/>
              <connection refLocalId="5">
                <position x="320" y="104"/>
                <position x="160" y="104"/>
```

```
                    </connection>
                  </connectionPointIn>
                </variable>
                <variable formalParameter="IN3">
                  <connectionPointIn>
                    <relPosition x="0" y="56"/>
                    <connection refLocalId="6">
                      <position x="320" y="120"/>
                      <position x="160" y="120"/>
                    </connection>
                  </connectionPointIn>
                </variable>
                <variable formalParameter="IN4" negated="true">
                  <connectionPointIn>
                    <relPosition x="0" y="72"/>
                  </connectionPointIn>
                </variable>
                <variable formalParameter="IN5">
                  <connectionPointIn>
                    <relPosition x="0" y="88"/>
                  </connectionPointIn>
                </variable>
              </inputVariables>
              <inOutVariables/>
              <outputVariables>
                <variable formalParameter="OUT1" negated="true">
                  <connectionPointOut>
                    <relPosition x="48" y="8"/>
                  </connectionPointOut>
                </variable>
              </outputVariables>
            </block>

            <outVariable height="16" localId="8" width="80">
              <position x="400" y="64"/>
              <connectionPointIn>
                <relPosition x="0" y="8"/>
                <connection formalParameter="OUT1" refLocalId="7">
                  <position x="400" y="72"/>
                  <position x="368" y="72"/>
                </connection>
              </connectionPointIn>
              <expression>Var3</expression>
            </outVariable>
          </FBD>
        </body>
      </pou>
    </pous>
  </types>
  <instances>
    <configurations/>
  </instances>
</project>
```

9.2. Simple example for SFC

Graphical Representation



XML

```
<?xml version="1.0" encoding="UTF-8"?>
<project   xmlns="http://www.plcopen.org/xml/tc6.xsd"
           xmlns:xhtml="http://www.w3.org/1999/xhtml"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.plcopen.org/xml/tc6.xsd http://www.plcopen.org/xml/tc6.xsd">
  <fileHeader companyName="plcopen"
              creationDateTime="2005-01-25T14:18:00"
              productName="NoNameNow"
              productVersion="1"/>
  <contentHeader name="prj">
    <coordinateInfo>
      <fbd>
        <scaling x="8" y="8"/>
      </fbd>
      <ld>
        <scaling x="8" y="8"/>
      </ld>
      <sfc>
        <scaling x="8" y="8"/>
      </sfc>
    </coordinateInfo>
  </contentHeader>
  <types>
    <dataTypes/>
    <pous>
      <pou name="sfc1" pouType="functionBlock">
        <interface>
          <localVars>
            <variable name="Var1">
              <type>
                <BOOL/>
              </type>
            </variable>
            <variable name="Var2">
```

```
            <type>
              <BOOL/>
            </type>
          </variable>
          <variable name="Var3">
            <type>
              <BOOL/>
            </type>
          </variable>
          <variable name="Var4">
            <type>
              <BOOL/>
            </type>
          </variable>
          <variable name="Var5">
            <type>
              <BOOL/>
            </type>
          </variable>
        </localVars>
      </interface>
      <body>
        <SFC>
          <step height="32" initialStep="true" localId="1" name="S0" width="32">
            <position x="480" y="96"/>
            <connectionPointIn>
              <relPosition x="16" y="0"/>
              <connection refLocalId="12">
                <position x="496" y="96"/>
                <position x="496" y="88"/>
                <position x="376" y="88"/>
                <position x="376" y="352"/>
                <position x="496" y="344"/>
              </connection>
            </connectionPointIn>
            <connectionPointOut formalParameter="sfc">
              <relPosition x="16" y="32"/>
            </connectionPointOut>
            <connectionPointOutAction formalParameter="x">
              <relPosition x="32" y="16"/>
            </connectionPointOutAction>
          </step>

          <outVariable height="16" localId="2" width="80">
            <position x="640" y="104"/>
            <connectionPointIn>
              <relPosition x="0" y="8"/>
              <connection formalParameter="x" refLocalId="1">
                <position x="640" y="112"/>
                <position x="496" y="112"/>
              </connection>
            </connectionPointIn>
            <expression>Var3</expression>
          </outVariable>

          <inVariable height="16" localId="3" width="80" negated="true">
            <position x="240" y="184"/>
            <connectionPointOut>
              <relPosition x="80" y="8"/>
            </connectionPointOut>
```

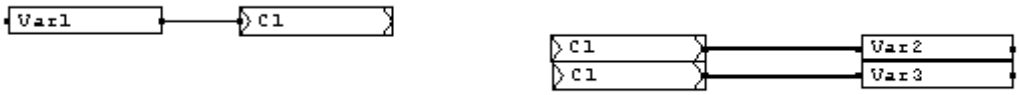
```
      <expression>Var1</expression>
    </inVariable>

    <transition height="8" localId="4" width="32">
      <position x="480" y="188"/>
      <connectionPointIn>
        <relPosition x="16" y="-12"/>
        <connection refLocalId="1" formalParameter="sfc">
          <position x="496" y="176"/>
          <position x="496" y="128"/>
        </connection>
      </connectionPointIn>
      <connectionPointOut>
        <relPosition x="16" y="24"/>
      </connectionPointOut>
      <condition>
        <connection refLocalId="3">
          <position x="480" y="192"/>
          <position x="320" y="192"/>
        </connection>
      </condition>
    </transition>

    <simultaneousDivergence height="4" localId="5" width="160">
      <position x="416" y="214"/>
      <connectionPointIn>
        <relPosition x="80" y ="0"/>
        <connection refLocalId="4">
          <position x="496" y="214"/>
          <position x="496" y="212"/>
        </connection>
      </connectionPointIn>
      <connectionPointOut formalParameter="0">
        <relPosition x="0" y="4"/>
      </connectionPointOut>
      <connectionPointOut formalParameter="160">
        <relPosition x="160" y="4"/>
      </connectionPointOut>
    </simultaneousDivergence>

    <step height="32" localId="6" name="S1" width="32">
      <position x="400" y="232"/>
      <connectionPointIn>
        <relPosition x="16" y="0"/>
        <connection refLocalId="5" formalParameter="0">
          <position x="416" y="232"/>
          <position x="416" y="218"/>
        </connection>
      </connectionPointIn>
      <connectionPointOut formalParameter="sfc">
        <relPosition x="16" y="32"/>
      </connectionPointOut>
      <connectionPointOutAction formalParameter="x">
        <relPosition x="32" y="16"/>
      </connectionPointOutAction>
    </step>

    <outVariable height="16" localId="7" negated="true" width="80">
      <position x="640" y="216"/>
      <connectionPointIn>
```

```
        <connection formalParameter="x" refLocalId="6">
          <position x="640" y="224"/>
          <position x="440" y="224"/>
          <position x="440" y="248"/>
          <position x="432" y="248"/>
        </connection>
      </connectionPointIn>
      <expression>Var4</expression>
    </outVariable>

    <step height="32" localId="8" name="S2" width="32" negated="true">
      <position x="560" y="232"/>
      <connectionPointIn>
        <relPosition x="16" y="0"/>
        <connection refLocalId="5" formalParameter="160">
          <position x="576" y="232"/>
          <position x="576" y="218"/>
        </connection>
      </connectionPointIn>
      <connectionPointOut formalParameter="sfc">
        <relPosition x="16" y="32"/>
      </connectionPointOut>
      <connectionPointOutAction formalParameter="x">
        <relPosition x="32" y="16"/>
      </connectionPointOutAction>
    </step>

    <outVariable height="16" localId="9" width="80">
      <position x="640" y="240"/>
      <connectionPointIn>
        <relPosition x="0" y="8"/>
        <connection refLocalId="8">
          <position x="640" y="248"/>
          <position x="592" y="248"/>
        </connection>
      </connectionPointIn>
      <expression>Var5</expression>
    </outVariable>

    <simultaneousConvergence height="4" localId="10" width="160">
      <position x="416" y="286"/>
      <connectionPointIn>
        <relPosition x="0" y="0"/>
        <connection refLocalId="6" formalParameter="sfc">
          <position x="416" y="286"/>
          <position x="416" y="264"/>
        </connection>
      </connectionPointIn>
      <connectionPointIn>
        <connection refLocalId="8" formalParameter="sfc">
          <position x="576" y="286"/>
          <position x="576" y="264"/>
        </connection>
      </connectionPointIn>
      <connectionPointOut>
        <relPosition x="80" y="4"/>
      </connectionPointOut>
    </simultaneousConvergence>

    <inVariable height="16" localId="11" width="80">
```

```
                <position x="240" y="320"/>
                <connectionPointOut>
                  <relPosition x="80" y="8"/>
                </connectionPointOut>
                <expression>Var2</expression>
              </inVariable>

              <transition height="8" localId="12" width="32">
                <position x="480" y="324"/>
                <connectionPointIn>
                  <relPosition x="16" y="-12"/>
                  <connection refLocalId="10">
                    <position x="496" y="312"/>
                    <position x="496" y="290"/>
                  </connection>
                </connectionPointIn>
                <connectionPointOut>
                  <relPosition x="16" y="24"/>
                </connectionPointOut>
                <condition negated="true">
                  <connection refLocalId="11">
                    <position x="480" y="328"/>
                    <position x="320" y="328"/>
                  </connection>
                </condition>
              </transition>
            </SFC>
          </body>
        </pou>
      </pous>
    </types>
    <instances>
      <configurations/>
    </instances>
</project>
```

9.3. Example connectors, connection and variables

Graphical representation



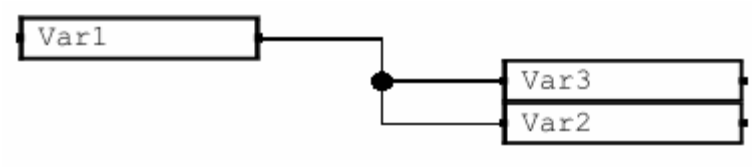
XML output example

```
<?xml version="1.0" encoding="UTF-8"?>
<project   xmlns="http://www.plcopen.org/xml/tc6.xsd" xmlns:xhtml=http://www.w3.org/1999/xhtml
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.plcopen.org/xml/tc6.xsd">
  <fileHeader companyName="plcopen"
              creationDateTime="2005-01-25T14:18:00"
              productName="NoNameNow"
              productVersion="1"/>
  <contentHeader name="prj">
    <coordinateInfo>
      <fbd>
        <scaling x="8" y="8"/>
      </fbd>
      <ld>
        <scaling x="8" y="8"/>
      </ld>
      <sfc>
        <scaling x="8" y="8"/>
      </sfc>
    </coordinateInfo>
  </contentHeader>
  <types>
    <dataTypes/>
    <pous>
      <pou name="fork1" pouType="functionBlock">
        <interface>
          <localVars>
            <variable name="Var1">
              <type>
                <BOOL/>
              </type>
            </variable>
            <variable name="Var2">
              <type>
                <BOOL/>
              </type>
            </variable>
            <variable name="Var3">
              <type>
                <BOOL/>
              </type>
            </variable>
          </localVars>
        </interface>
```

```
<body>
  <FBD>
    <inVariable height="16" localId="1" width="80">
      <position x="160" y="32"/>
      <connectionPointOut>
        <relPosition x="80" y="8"/>
      </connectionPointOut>
      <expression>Var1</expression>
    </inVariable>

    <connector name="C1" localId="10" height="16" width="80">
      <position x="240" y="32"/>
      <connectionPointIn>
        <relPosition x="0" y="8"/>
        <connection refLocalId="1">
          <position x="240" y="40"/>
          <position x="160" y="40"/>
        </connection>
      </connectionPointIn>
    </connector>

    <continuation name="C1" localId="11" height="16" width="80">
      <position x="400" y="32"/>
      <connectionPointOut>
        <relPosition x="80" y="8"/>
      </connectionPointOut>
    </continuation>

    <continuation name="C1" localId="12" height="16" width="80">
      <position x="400" y="48"/>
      <connectionPointOut>
        <relPosition x="80" y="8"/>
      </connectionPointOut>
    </continuation>

    <outVariable height="16" localId="2" negated="true" width="80">
      <position x="480" y="32"/>
      <connectionPointIn>
        <relPosition x="0" y="8"/>
        <connection refLocalId="11">
          <position x="480" y="40"/>
          <position x="400" y="40"/>
        </connection>
      </connectionPointIn>

      <expression>Var2</expression>
    </outVariable>

    <outVariable height="16" localId="3" width="80">
      <position x="480" y="48"/>
      <connectionPointIn>
        <relPosition x="80" y="8"/>
        <connection refLocalId="12">
          <position x="480" y="56"/>
          <position x="400" y="56"/>
        </connection>
      </connectionPointIn>
      <expression>Var3</expression>
    </outVariable>
  </FBD>
```

```
        </body>
      </pou>
    </pous>
  </types>
  <instances>
    <configurations/>
  </instances>
</project>
```

9.4. Example on forked connections



XML output example

```
<?xml version="1.0" encoding="UTF-8"?>
<project   xmlns="http://www.plcopen.org/xml/tc6.xsd" xmlns:xhtml="http://www.w3.org/1999/xhtml"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.plcopen.org/xml/tc6.xsd">
  <fileHeader companyName="plcopen"
              creationDateTime="2005-01-25T14:18:00"
              productName="NoNameNow"
              productVersion="1"/>
  <contentHeader name="prj">
    <coordinateInfo>
      <fbd>
        <scaling x="8" y="8"/>
      </fbd>
      <ld>
        <scaling x="8" y="8"/>
      </ld>
      <sfc>
        <scaling x="8" y="8"/>
      </sfc>
    </coordinateInfo>
  </contentHeader>
  <types>
    <dataTypes/>
    <pous>
      <pou name="fork1" pouType="functionBlock">
        <interface>
          <localVars>
            <variable name="Var1">
              <type>
                <BOOL/>
              </type>
            </variable>
            <variable name="Var2">
              <type>
                <BOOL/>
              </type>
            </variable>
```

```
            <variable name="Var3">
              <type>
                <BOOL/>
              </type>
            </variable>
          </localVars>
        </interface>
        <body>
          <FBD>
            <inVariable height="16" localId="1" width="80">
              <position x="80" y="32"/>
              <connectionPointOut>
                <relPosition x="80" y="8"/>
              </connectionPointOut>
              <expression>Var1</expression>
            </inVariable>

            <outVariable height="16" localId="2" width="80">
              <position x="240" y="48"/>
              <connectionPointIn>
                <relPosition x="0" y="8"/>
                <connection refLocalId="1">
                  <position x="240" y="56"/>
                  <position x="200" y="56"/>
                  <position x="200" y="40"/>
                  <position x="160" y="40"/>
                </connection>
              </connectionPointIn>
              <expression>Var3</expression>
            </outVariable>

            <outVariable height="16" localId="3" width="80">
              <position x="240" y="64"/>
              <connectionPointIn>
                <relPosition x="0" y="8"/>
                <connection refLocalId="1">
                  <position x="240" y="72"/>
                  <position x="200" y="72"/>
                  <position x="200" y="56"/>
                  <position x="200" y="40"/>
                  <position x="160" y="40"/>
                </connection>
              </connectionPointIn>
              <expression>Var2</expression>
            </outVariable>
          </FBD>
        </body>
      </pou>
    </pous>
  </types>
  <instances>
    <configurations/>
  </instances>
</project>
```