

DIN EN 61131-10

ICS 25.040.40; 35.060; 35.080; 35.240.50

Einsprüche bis 2017-12-20

Entwurf

**Speicherprogrammierbare Steuerungen –
Teil 10: XML-Austauschformat für IEC 61131-3
(IEC 65B/1070/CD:2017);
Text English**

Programmable controllers –
Part 10: XML Exchange Formats for Programs according to IEC 61131-3
(IEC 65B/1070/CD:2017);
Text English

Anwendungswarnvermerk

Dieser Norm-Entwurf mit Erscheinungsdatum 2017-10-20 wird der Öffentlichkeit zur Prüfung und Stellungnahme vorgelegt.

Weil die beabsichtigte Norm von der vorliegenden Fassung abweichen kann, ist die Anwendung dieses Entwurfs besonders zu vereinbaren.

Stellungnahmen werden erbeten

- vorzugsweise online im Norm-Entwurfs-Portal von DIN unter www.din.de/go/entwuerfe bzw. für Norm-Entwürfe der DKE auch im Norm-Entwurfs-Portal der DKE unter www.entwuerfe.normenbibliothek.de, sofern dort wiedergegeben;
- oder als Datei per E-Mail an dke@vde.com möglichst in Form einer Tabelle. Die Vorlage dieser Tabelle kann im Internet unter www.din.de/go/stellungnahmen-norm-entwuerfe oder für Stellungnahmen zu Norm-Entwürfen der DKE unter www.dke.de/stellungnahme abgerufen werden;
- oder in Papierform an die DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE, Stresemannallee 15, 60596 Frankfurt am Main.

Die Empfänger dieses Norm-Entwurfs werden gebeten, mit ihren Kommentaren jegliche relevanten Patentrechte, die sie kennen, mitzuteilen und unterstützende Dokumentationen zur Verfügung zu stellen.

Gesamtumfang 153 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

Nationales Vorwort

Die englische Originalfassung des internationalen Dokuments IEC 65B/1070/CD:2017 „Programmable controllers – Part 10: XML Exchange Formats for Programs according to IEC 61131-3“ (CD, en: Committee Draft) ist unverändert in diesen Norm-Entwurf übernommen worden. Dieser Norm-Entwurf enthält eine autorisierte deutsche Übersetzung.

Das internationale Dokument wurde vom SC 65B „Measurement and control devices“ der Internationalen Elektrotechnischen Kommission (IEC) erarbeitet und den nationalen Komitees zur Stellungnahme vorgelegt.

Die IEC und das Europäische Komitee für Elektrotechnische Normung (CENELEC) haben vereinbart, dass ein auf IEC-Ebene erarbeiteter Entwurf für eine Internationale Norm zeitgleich (parallel) bei IEC und CENELEC zur Umfrage (CDV-Stadium) und Abstimmung als FDIS (en: Final Draft International Standard) bzw. Schluss-Entwurf für eine Europäische Norm gestellt wird, um eine Beschleunigung und Straffung der Normungsarbeit zu erreichen. Dokumente, die bei CENELEC als Europäische Norm angenommen und ratifiziert werden, sind unverändert als Deutsche Normen zu übernehmen.

Da der Abstimmungszeitraum für einen FDIS bzw. Schluss-Entwurf prEN nur 2 Monate beträgt, und dann keine sachlichen Stellungnahmen mehr abgegeben werden können, sondern nur noch eine „JA/NEIN“-Entscheidung möglich ist, wobei eine „NEIN“-Entscheidung fundiert begründet werden muss, wird bereits der CD als DIN-Norm-Entwurf veröffentlicht, um die Stellungnahmen aus der Öffentlichkeit frühzeitig berücksichtigen zu können.

Für dieses Dokument ist das nationale Arbeitsgremium K 962 „SPS“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE (www.dke.de) zuständig.

Für den Fall einer undatierten Verweisung im normativen Text (Verweisung auf ein Dokument ohne Angabe des Ausgabedatums und ohne Hinweis auf eine Abschnittsnummer, eine Tabelle, ein Bild usw.) bezieht sich die Verweisung auf die jeweils aktuellste Ausgabe des in Bezug genommenen Dokuments.

Für den Fall einer datierten Verweisung im normativen Text bezieht sich die Verweisung immer auf die in Bezug genommene Ausgabe des Dokuments.

Der Zusammenhang der zitierten Dokumente mit den entsprechenden Deutschen Dokumenten ergibt sich, soweit ein Zusammenhang besteht, grundsätzlich über die Nummer der entsprechenden IEC-Publikation. Beispiel: IEC 60068 ist als EN 60068 als Europäische Norm durch CENELEC übernommen und als DIN EN 60068 ins Deutsche Normenwerk aufgenommen.

Das Präsidium des DIN hat mit Präsidialbeschluss 1/2004 festgelegt, dass DIN-Normen, deren Inhalt sich auf internationale Arbeitsergebnisse der Informationsverarbeitung gründet, unter bestimmten Bedingungen allein in englischer Sprache veröffentlicht werden dürfen. Diese Bedingungen sind für den vorliegenden Normentwurf erfüllt.

Da sich die Benutzer des vorliegenden Normentwurfs der englischen Sprache als Fachsprache bedienen, wird die Englische Fassung der EN 61131-10 veröffentlicht. Zu deren Abschnitt 1 und Abschnitt 3, der die Begriffe festlegt, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA des vorliegenden Normentwurfs hinzugefügt. Für viele der verwendeten Begriffe existieren keine gebräuchlichen deutschen Benennungen, da sich die deutschen Anwender in der Regel ebenfalls der englischen Benennungen bedienen. Dieser Normentwurf steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

Es wird auf die Möglichkeit hingewiesen, dass einige Elemente dieses Dokuments Patentrechte berühren können. ISO [und/oder] IEC sind nicht dafür verantwortlich, einige oder alle diesbezüglichen Patentrechte zu identifizieren.

Nationaler Anhang NA (informativ)

Auszugweise Übersetzung^{N1)}

1 Anwendungsbereich

1.1 Allgemeines

Die vorliegende Norm legt ein XML-basiertes Austauschformat für den Export und den Import von IEC 61131-3-Projekten fest.

Ein vollständiges IEC 61131-3-Projekt, das in einer IEC 61131-3-Umgebung implementiert wird, kann zwischen verschiedenen Programmierumgebungen übertragen werden. Damit ist der Austausch von Konfigurationselementen, Datentypen und Programm-Organisationseinheiten (POE) möglich, die geschrieben sind in:

- der Textsprache: Anweisungsliste (AWL);
- der Textsprache: Strukturierter Text (ST);
- der grafischen Sprache: Kontaktplan (KOP);
- der grafischen Sprache: Funktionsbausteinsprache (FBS); und
- in der Ablaufsprache (AS).

Das Austauschformat ist als ein entsprechendes XML-Schema festgelegt. Das XML-Schema ist eine unabhängige Datei mit der Erweiterung .xsd und als solche Teil dieser Spezifikation. Diese Norm enthält eine Beschreibung dieses Schemas. Es wird davon ausgegangen, dass der Leser dieser Norm mit der XML-Technologie vertraut ist.

Bild 1 enthält einen exemplarischen Überblick über die Anwendung des XML-Austauschformates. Verschiedene Werkzeuge erzeugen und verarbeiten XML-basierte IEC 61131-3-Informationen.

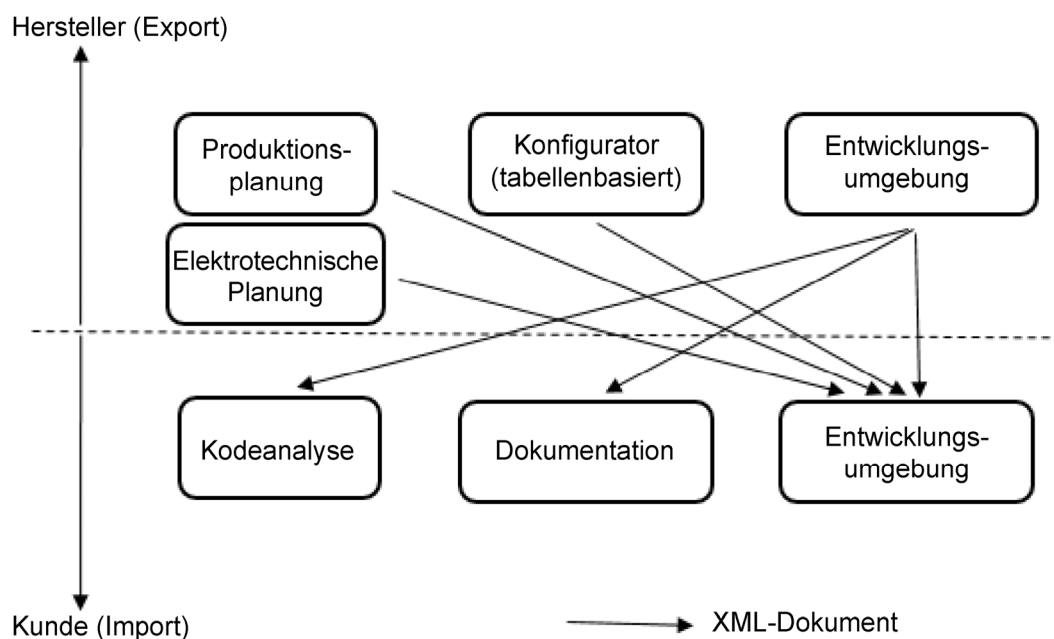


Bild 1 – Überblick über die Hauptanwendung des XML-Austauschformates (Beispiel)

^{N1)} Nationale Fußnote: Die Nummerierung von 65B/1070/CD:2017-01 wurde beibehalten

Die Anwendung des XML-Austauschformates sollte mehr bieten als einen einfachen Export/Import von einer Entwicklungsumgebung in eine andere. Es sollten alle einschlägigen Informationen exportiert werden. Dazu können Informationen über Koordinaten für grafische Werkzeuge gehören. Das Importwerkzeug sollte filtern können, welche Teile der Information in seine Zielumgebung importiert werden müssen. Lieferantenspezifische Informationen und Attribute dürfen in die Exportdatei eingebunden werden und gegebenenfalls selektiv importiert werden. Die lieferantenspezifischen Informationen dürfen den Logikteil des Programmes nicht beeinflussen.

Eine Filterung sollte beim Import erfolgen – auf diese Weise müssen Lieferanten dafür sorgen, dass ihre Erweiterungen des XML-Schemas so erfolgen, dass die Vernachlässigung von Informationen beim Import nicht die Funktionalität des IEC 61131-3-Projektes beeinflusst. Lieferantenspezifische Attribute und Informationen dürfen durch ein lieferantenspezifisches XML-Schema ergänzt werden – neben dem in der vorliegenden Norm definierten Austauschformat.

Die beschriebenen Formate sind für den Import und den Export von IEC 61131-3-Projekten vorgesehen. Ein solches IEC 61131-3-Projekt kann sich in der Entwicklung befinden und damit unvollständig sein.

Beim Austausch von Konstrukten der grafischen Sprache zwischen verschiedenen Programmiersystemen liegt der Schwerpunkt auf der logischen Information, wobei die grafische Darstellung ausdrücklich wahlfrei erfolgt.

1.2 Implementierungsspezifische Parameter

Diese Norm enthält keine Mittel oder Anforderungen für eine konforme Funktionalität (z. B. eine Untermenge von Funktionen, die von allen Programmier- und Testeinrichtungen (PUTE) unterstützt werden müssen). Sie ermöglicht den Austausch aller möglichen Eigenschaften, die in IEC 61131-3 definiert sind.

Darüber hinaus können viele implementierungsspezifische Eigenschaften mit AddData-Mechanismen ausgedrückt werden.

Bei einigen Anwendungsfällen werden Programme entweder von einer PUTE zu einer anderen übertragen oder für die Anwendung in verschiedenen PUTE generiert. In beiden Fällen dürfen die Funktionsmengen dieser PUTE sowie die Einstellung ihrer implementierungsabhängigen Parameter verschieden sein. Wenn mehrere PUTE zu unterstützen/berücksichtigen sind, muss die Funktionalität des Programmes auf die Untermenge begrenzt werden, die von allen betreffenden PUTE unterstützt wird. Einige dieser Funktionen können aus den Eigenschaftentabellen der entsprechenden PUTE in IEC 61131-3 bestimmt werden, z. B.:

- unterstützte Datentypen und Standardfunktionen;
- bevorrechtigte oder nicht-bevorrechtigte Aufrufe;
- AS mit einem oder ohne einen finalen Durchlauf usw.

Weitere Funktionen und Einstellungen implementierungsabhängiger Parameter können mehr Aufwand bei der Bestimmung erfordern, z. B.:

- der größte Umfang des Codes oder die größte Menge von Variablen je POE;
- die größte Länge der Bezeichner (Länge des Variablennamens);
- die Größe der Variablen STRING und WSTRING mit der voreingestellten oder der größten Länge;
- AS zur Bewertung aller Transitionsbedingungen oder nur solcher mit aktiven Schritten als Vorgänger;
- Bereich und Genauigkeit der Datentypen TIME, DATE, TOD, DT;
- das Laufzeitverhalten der (POE in der) SPS;
- die Ausführungsreihenfolge innerhalb eines grafischen Netzwerkes usw.

Diese Unterschiede sind bei Anwendungsfällen mit mehr als einer PUTE zu berücksichtigen. In bestimmten Fällen kann es angebracht sein, nur die von allen betreffenden PUTE unterstützte Funktionalität zu nutzen, in anderen Fällen kann es notwendig sein, das Programm nach dem Import in die PUTE von Hand zu ändern und zu prüfen.

Die vorliegende Norm enthält keine Anforderungen an die übereinstimmenden Funktionen der PUTE. Sie legt ein Austauschformat für den Austausch von Programmen fest, die mit IEC 61131-3 übereinstimmen.

3 Begriffe und Abkürzungen

3.1 Allgemeine Begriffe

Für die Anwendung dieses Dokuments gelten die in IEC 61131-3 angegebenen und die folgenden Begriffe.

3.1.1

IEC 61131-3-Projekt

Informationsmenge nach IEC 61131-3, die POE-Typen und -Instanzen, Datentypen sowie Instanzen, Konfigurationen und Ressourcen globaler Variablen enthält

3.2 Abkürzungen

FBS	Funktionsbausteinsprache
AWL	Anweisungsliste
KOP	Kontaktplan
SPS	Speicherprogrammierbare Steuerung
POE	Programm-Organisationseinheit
PUTE	Programmier- und Testeinrichtung
AS	Ablaufsprache
ST	Strukturierter Text
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Erweiterbare Auszeichnungssprache (en: Extensible Markup Language)

CONTENTS

FOREWORD.....	8
INTRODUCTION.....	10
1 Scope.....	11
1.1 General.....	11
1.2 Implementation Specific Parameters	12
2 Normative references	12
3 Terms, definitions, abbreviated terms and acronyms	13
3.1 General terms and definitions	13
3.2 Abbreviated terms and acronyms	13
4 Overview of schema concepts	13
4.1 Naming conventions.....	13
4.2 Coordinate system of graphical languages	13
4.3 Schema extension concepts.....	15
5 Compliance	15
5.1 General.....	15
5.2 Feature tables.....	16
5.3 Vendor's compliance statement	16
6 Main schema element "Project"	16
6.1 General.....	16
6.2 "FileHeader".....	17
6.3 "ContentHeader"	17
6.4 "Types"	18
6.5 "Instances".....	19
6.5.1 General ("Configuration").....	19
6.5.2 "Resource"	20
6.5.3 "AccessVars"	20
6.5.4 "ConfigVars"	21
7 Abstract types	22
7.1 Purpose of abstract types	22
7.2 Abstract types for data type specifications	23
7.2.1 General	23
7.2.2 "TypeSpecBase"	23
7.2.3 "InstantlyDefinableTypeSpecBase"	23
7.3 Abstract types for behaviour representations	23
7.3.1 General	23
7.3.2 "BehaviorRepresentationBase"	24
7.3.3 "ProgrammingLanguageBase".....	24
7.4 Abstract types for graphical objects	24
7.4.1 General	24
7.4.2 "IdentifiedObjectBase"	25
7.4.3 "GraphicalObjectBase".....	26
7.4.4 "CommonObjectBase".....	26
7.4.5 "FbdObjectBase"	27
7.4.6 "LdObjectBase".....	27
7.4.7 "SfcObjectBase"	27

7.4.8	“NetworkBase”	28
7.5	Abstract types for textual constructs	29
7.5.1	General	29
7.5.2	“TextualObjectBase”	29
7.5.3	“NamespaceContentBase”	30
7.5.4	“TaskBase”	31
8	“NamespaceDecl”	31
9	User defined data type declaration	32
9.1	“UserDefinedTypeDecl”	32
9.2	“ArrayTypeSpec”	32
9.3	“DirectlyDerivedTypeSpec”	33
9.4	“EnumTypeSpec”	33
9.5	“EnumTypeWithNamedValueSpec”	33
9.6	“StructTypeSpec”	34
9.7	“SubrangeTypeSpec”	34
9.8	“ReferenceTypeSpec”	35
9.9	“ElementaryType”	35
10	POU declaration	35
10.1	“PouDecl”	35
10.2	“Program”	36
10.3	“FunctionBlock”	37
10.4	“Class”	38
10.5	“Function”	39
10.6	“Interface”	39
10.7	“Action”	40
10.8	“NamedTransition”	40
10.9	“MethodPrototype”	41
10.10	“Method”	42
10.11	“ParameterSet”	42
10.12	“VarListWithAccessSpec”	44
10.13	“AccessSpecifiers”	45
10.14	“Body”	45
10.15	“BodyWithoutSFC”	45
10.16	“Predicate”	46
11	Variable declaration	46
11.1	“VarList”	46
11.2	“ExternalVarList”	47
11.3	“VariableDecl”	48
11.4	“VariableDeclPlain”	48
11.5	“TypeRef”	49
11.6	“Value”	49
11.7	“AddressExpression”	49
11.8	“FixedAddressExpression”	50
12	Behaviour representation	50
12.1	“IL”	50
12.2	“ST”	51
12.3	“LD”	51
12.4	“FBD”	51

12.5	“SFC”	52
13	Graphical behaviour representation	52
13.1	Common	52
13.1.1	“Comment”	52
13.1.2	“Connector”	53
13.1.3	“Continuation”	53
13.1.4	“ActionBlocks”	54
13.2	Only for LD and FBD	56
13.2.1	“Block”	56
13.2.2	“graphicalFormalParameterCommon”	58
13.2.3	“DataSource”	59
13.2.4	“DataSink”	59
13.2.5	“Unconnected”	60
13.2.6	“Jump”	61
13.2.7	“Return”	61
13.3	Only for LD	62
13.3.1	“LeftPowerRail”	62
13.3.2	“RightPowerRail”	63
13.3.3	“Coil”	63
13.3.4	“Contact”	64
13.3.5	“CompareContact”	65
13.4	Only for SFC	66
13.4.1	“Step”	66
13.4.2	“Transition”	67
13.4.3	“SelectionDivergence”	68
13.4.4	“SelectionConvergence”	69
13.4.5	“SimultaneousDivergence”	70
13.4.6	“SimultaneousConvergence”	70
13.5	Connections	71
13.5.1	General	71
13.5.2	“ConnectionPointIn”	71
13.5.3	“Connection”	72
13.5.4	“FeedbackConnection”	72
13.5.5	“ConnectionPointOut”	73
14	Resource declaration	73
14.1	“ProgramInstance”	73
14.2	“StandardTask”	75
14.3	“ParameterAssignment”	75
15	Miscellaneous	75
15.1	“XyDecimalValue”	75
15.2	“AddData”	76
15.3	“FormattedText”	76
15.4	“EdgeModifierType”	76
15.5	“ComplexIdentifier”	76
15.6	“SimpleIdentifier”	76
Annex A	(normative) Formal XML exchange format schema definition	78
Annex B	(informative) Recommended schemata	142
B.1	Recommended schemata to be used by “AddData”	142

B.2 Recommended schema to be used by abstract type	146
Bibliography.....	149

Figure 1 – Main overview of XML exchange format usage (example)	11
Figure 2 – Mapping coordinate information to the coordinate system	14
Figure 3 – Transforming coordinate position using the scaling information	15
Figure 4 – Main schema element “Project”	17
Figure 5 – Element “FileHeader”	17
Figure 6 – Element “ContentHeader”	18
Figure 7 – Element “Types”	19
Figure 8 – Element “Instances”	19
Figure 9 – Element “Resource”	20
Figure 10 – Element “AccessVars”	21
Figure 11 – Element “ConfigVars”	22
Figure 12 – Extension relationship among complex types for data type specifications	23
Figure 13 – Extension relationship among complex types for behaviour representations	24
Figure 14 – Extension relationship among complex types for graphical objects	25
Figure 15 – Complex type “IdentifiedObjectBase”	26
Figure 16 – Complex type “GraphicalObjectBase”	26
Figure 17 – Complex type “CommonObjectBase”	27
Figure 18 – Complex type “FbdObjectBase”	27
Figure 19 – Complex type “LdObjectBase”	27
Figure 20 – Complex type “SfcObjectBase”	28
Figure 21 – Complex type “NetworkBase”	28
Figure 22 – Extension relationship among complex types for textual objects	29
Figure 23 – Complex type “TextualObjectBase”	30
Figure 24 – Complex type “NamespaceContentBase”	30
Figure 25 – Complex type “TaskBase”	31
Figure 26 – Complex type “NamespaceDecl”	31
Figure 27 – Complex type “UserDefinedTypeDecl”	32
Figure 28 – Complex type “ArrayTypeSpec”	32
Figure 29 – Complex type “DirectlyDerivedTypeSpec”	33
Figure 30 – Complex type “EnumTypeSpec”	33
Figure 31 – Complex type “EnumTypeWithNamedValueSpec”	34
Figure 32 – Complex type “StructTypeSpec”	34
Figure 33 – Complex type “SubrangeTypeSpec”	35
Figure 34 – Complex type “ReferenceTypeSpec”	35
Figure 35 – Complex type “PouDecl”	35
Figure 36 – Complex type “Program”	36
Figure 37 – Complex type “FunctionBlock”	37
Figure 38 – Complex type “Class”	38
Figure 39 – Complex type “Function”	39
Figure 40 – Complex type “Interface”	40

Figure 41 – Complex type “Action”	40
Figure 42 – Complex type “NamedTransition”	41
Figure 43 – Complex type “MethodPrototype”	41
Figure 44 – Complex type “Method”	42
Figure 45 – Complex type “ParameterSet”	44
Figure 46 – Complex type “VarListWithAccessSpec”	45
Figure 47 – Complex type “Body”	45
Figure 48 – Complex type “BodyWithoutSFC”	46
Figure 49 – Complex type “Predicate”	46
Figure 50 – Complex type “VarList”	47
Figure 51 – Complex type “ExternalVarList”	47
Figure 52 – Complex type “VariableDecl”	48
Figure 53 – Complex type “VariableDeclPlain”	48
Figure 54 – Complex type “TypeRef”	49
Figure 55 – Complex type “Value”	49
Figure 56 – Complex type “AddressExpression”	50
Figure 57 – Complex type “FixedAddressExpression”	50
Figure 58 – Complex type “IL”	51
Figure 59 – Complex type “ST”	51
Figure 60 – Complex type “LD”	51
Figure 61 – Complex type “FBD”	52
Figure 62 – Complex type “SFC”	52
Figure 63 – Complex type “Comment”	53
Figure 64 – Complex type “Connector”	53
Figure 65 – Complex type “Continuation”	54
Figure 66 – Complex type “ActionBlocks”	55
Figure 67 – Complex type “Block”	57
Figure 68 – Complex type “DataSource”	59
Figure 69 – Complex type “DataSink”	60
Figure 70 – Complex type “Unconnected”	60
Figure 71 – Complex type “Jump”	61
Figure 72 – Complex type “Return”	62
Figure 73 – Complex type “LeftPowerRail”	62
Figure 74 – Complex type “RightPowerRail”	63
Figure 75 – Complex type “Coil”	64
Figure 76 – Complex type “Contact”	65
Figure 77 – Complex type “CompareContact”	66
Figure 78 – Complex type “Step”	67
Figure 79 – Complex type “Transition”	68
Figure 80 – Complex type “SelectionDivergence”	69
Figure 81 – Complex type “SelectionConvergence”	70
Figure 82 – Complex type “SimultaneousDivergence”	70
Figure 83 – Complex type “SimultaneousConvergence”	71

Figure 84 – Complex type “ConnectionPointIn”	71
Figure 85 – Complex type “Connection”	72
Figure 86 – Complex type “FeedbackConnection”	72
Figure 87 – Complex type “ConnectionPointOut”	73
Figure 88 – Element “ProgramInstance”	74
Figure 89 – Complex type “StandardTask”	75
Figure 90 – Complex type “ParameterAssignment”	75
Figure 91 – Complex type “XyDecimalValue”	76
Figure 92 – Complex type “AddData”	76
Figure 93 – Complex type “FormattedText”	76

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PROGRAMMABLE CONTROLLERS –

Part 10: XML Exchange Format according to IEC 61131-3

FOREWORD

- 1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
- 2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
- 3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
- 5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
- 6) All users should ensure that they have the latest edition of this publication.
- 7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
- 8) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
- 9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61131 10 has been prepared by subcommittee 65B: Measurement and control devices, of IEC technical committee 65: Industrial-process measurement, control and automation.

The text of this standard is based on the following documents:

PNW	Result on voting
65B/912/NP	65B/937/RVN

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on IEC web site under "<http://webstore.iec.ch>" in the data related to the specific publication. At this date, the publication will be

- 54 • reconfirmed,
- 55 • withdrawn,
- 56 • replaced by a revised edition, or
- 57 • amended.

58

59 The National Committees are requested to note that for this publication the stability date
60 2023.

61 Editor's note: THIS TEXT IS INCLUDED FOR THE INFORMATION OF THE NATIONAL
62 COMMITTEES AND WILL BE DELETED AT THE PUBLICATION STAGE.

63

64 **IMPORTANT** – The 'colour inside' logo on the page of this publication indicates that it
65 contains colours which are considered to be useful for the correct understanding.
66 Users should therefore print this document using a colour printer.

67

INTRODUCTION

The International Standard IEC 61131 describes Programmable Logic Controllers (PLCs).

IEC 61131-3 defines programming languages. Users want to have standardized programming languages and be able to exchange a complete program or parts of it between different development environments, i.e. from an exporting environment to an importing environment. IEC 61131-3 standard defines program organization units (POUs). But an entire program also consists of user defined data types, global and external declarations and other elements besides the POUs. In this standard the term IEC 61131-3 project is used. It contains all above mentioned language elements, required for an exchange, in order to get a consistent program in the importing environment.

The exchange of POUs developed in one of the textual languages, i.e. Instruction List (IL) and Structured Text (ST) or Sequential Function Chart (SFC) is possible, because a syntax description of these languages is part of the IEC 61131-3 standard. The objective of this standard is to extend the reuse of programmed solutions both for textual languages and graphical languages. Furthermore the completeness of exchange between the different environments depends on the supported features that are listed in the compliance list defined in IEC 61131-3.

This standard defines a solution-independent eXtensible Markup Language (XML) based exchange format, to be supported by interfaces of different kinds of software tools. Beside textual and program logic information, it also provides the ability to transfer graphical-representation information, e.g. the position and size of function blocks and how they are connected. The design of the 'transferred' parts shall represent the same program logic, however it may be altered in look and feel.

This standard's XML exchange format enables a transfer of IEC 61131-3 projects, from an exporting environment to an importing environment, including extensions for layout and formatting.

This standard's XML exchange format can not only describe correct IEC 61131-3 POUs, but it can represent a working state of the IEC 61131-3 project. For example, even if the IEC 61131-3 source project is incomplete, e.g. contains compile errors, it can be represented.

Syntactically incorrect IEC 61131-3 projects can be represented. Such a project could be an in-between version, e.g. within a Function Block Diagram (FBD) containing several unconnected blocks, this too can be represented.

This standard's XML exchange format provides for life cycle management of automation systems, e.g. in case of redesign, maintenance or device replacement. If an IEC 61131-3 project is stored in this standard's XML exchange format, it could be reused independent of a special development environment. And thus, it could be modified and maintained by any other development environment supporting this standard's XML exchange format.

This International Standard was developed using material from PLCopen^{®1}.

¹ PLCopen[®] is the registered trademark of PLCopen. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

PROGRAMMABLE CONTROLLERS –

Part 10: XML Exchange Format according to IEC 61131-3

1 Scope

1.1 General

This standard specifies an XML-based exchange format for the export and import of IEC 61131-3 projects. A complete IEC 61131-3 project implemented in an IEC 61131-3 environment can be transferred between different programming environments. It allows for the exchange of configuration elements, data types, and POU's written in:

- the textual language, Instruction List (IL),
- the textual language, Structured Text (ST),
- the graphical language, Ladder Diagram (LD),
- the graphical language, Function Block Diagram (FBD), and
- Sequential Function Chart (SFC).

The exchange format is specified as a corresponding XML schema. The XML schema is an independent file with the .xsd extension and as such part of this specification. A description of this schema is contained in this standard. It is assumed that the reader of this standard is familiar with XML technology.

Figure 1 provides an example overview of the usage of the XML exchange format. Different tools are producing and consuming XML based IEC 61131-3 information.

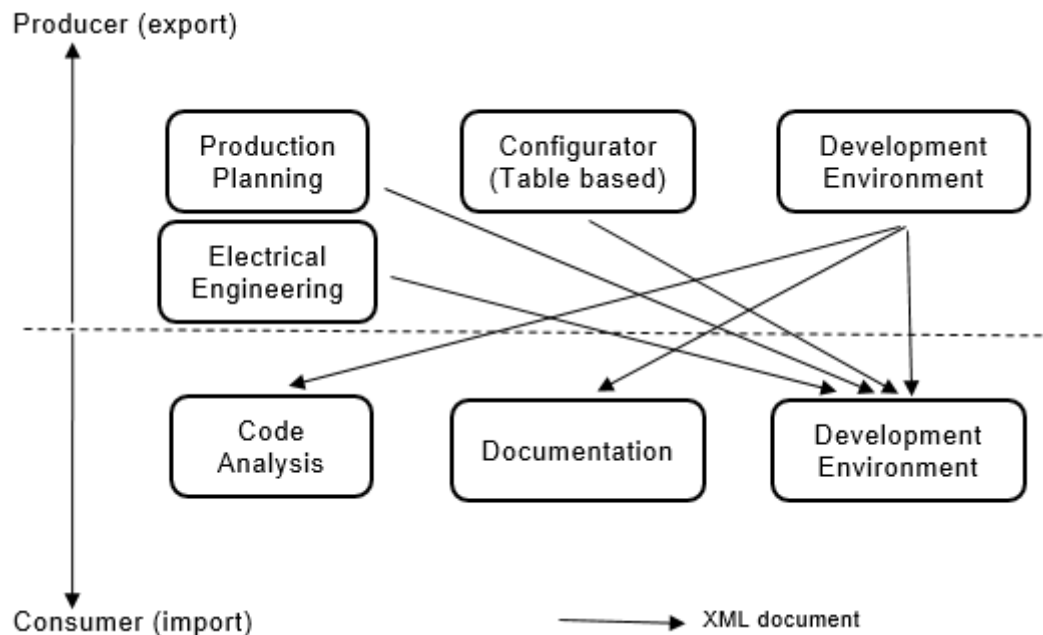


Figure 1 – Main overview of XML exchange format usage (example)

The usage of the XML exchange format should provide more than a simple export / import from one development environment to another. All relevant information should be exported. This may include coordinate information for graphical tools. The importing tool should be able to filter which parts of this information need to be imported into its destination environment. Vendor specific information and attributes may be included in the export file and selectively imported, if applicable. The vendor specific information shall not influence the logic part of the program. Filtering should be done on the import – thus vendors shall take care that their extensions of the XML schema shall be done in such a way that neglecting the information

138 during import does not affect the functionality of the IEC 61131-3 project. Vendor specific
139 attributes and information may be added by vendor specific XML schema – besides the XML
140 exchange format defined in this standard.

141 The described formats are designed for the import and export of IEC 61131-3 projects. Such
142 an IEC 61131-3 project can be under development and as a consequence incomplete.

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

145 1.2 Implementation Specific Parameters

146 This standard does not provide means or requirements for compliant functionality (e.g.
147 functional subset which has to be supported by all Programming and Debugging Tools
148 (PADTs)). This standard enables the exchange of all possible features defined in IEC 61131-
149 3. Moreover, many implementation specific features can be expressed using the AddData
150 mechanism.

151 In some use cases programs are either transferred from one PADT to another or generated
152 for the use in different PADT. In both cases the function set of these PADT may be different
153 as well as their settings of implementation dependent parameters. If several PADT have to be
154 supported/considered, the functionality of the program has to be restricted to the subset
155 supported by all PADTs in question. Some of these functions can be determined from the IEC
156 61131-3 feature tables of the concerned PADT, for example:

- 157 • supported data types and standard functions,
- 158 • pre-emptive or non-pre-emptive scheduling,
- 159 • SFC with or without a final scan, etc.

160 Other functions and settings of implementation dependent parameters may require more effort
161 to determine, for example:

- 162 • maximum amounts of code or variables per POU,
- 163 • maximum length of identifiers (variable name length),
- 164 • size of STRING and WSTRING variables with default length or maximum length,
- 165 • SFC to evaluate all transition conditions or only those with active steps as predecessors,
- 166 • range and precision of data types TIME, DATE, TOD, DT,
- 167 • runtime performance of (the POU in) the PLC,
- 168 • execution order within a graphical network, etc.

169 These differences have to be considered for use cases with more than one PADT. In some
170 cases it may be appropriate to use only functionality supported by all concerned PADTs, in
171 other cases it may be necessary to manually change and test the program after importing into
172 the PADT.

173 This standard does not state requirements regarding compliant functions of the PADT. It
174 defines an exchange format to exchange programs which are compliant to IEC 61131-3.

175 2 Normative references

176 The following documents, in whole or in part, are normatively referenced in this standard and
177 are indispensable for its application. For dated references, only the edition cited applies. For
178 undated references, the latest edition of the referenced document (including any
179 amendments) applies.

180 IEC 61131-1, *Programmable controllers – Part 1: General information*

181 IEC 61131-3, *Programmable controllers – Part 3: Programming Languages*

3 Terms, definitions, abbreviated terms and acronyms

3.1 General terms and definitions

For the purposes of this document, the terms and definitions given in IEC 61131-3 and the following apply.

3.1.1

IEC 61131-3 project

set of information according to IEC 61131-3 comprising POU types and instances, data types, global variables instances, configurations and resources

3.2 Abbreviated terms and acronyms

FBD	Function Block Diagram
IL	Instruction List
LD	Ladder Diagram
PLC	Programmable Logic Controller
POU	Program Organization Unit
PADT	Programming and Debugging Tool
SFC	Sequential Function Chart
ST	Structured Text
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

4 Overview of schema concepts

4.1 Naming conventions

Within all defined schemata, including the recommendations, the following naming conventions are used:

- Attribute names shall start with lower case letter.
- Type and element names shall start with upper case letter.
- Names consisting of multiple words shall have the first character of each word, except the first, starting with a capital letter. No underscores shall be used.

4.2 Coordinate system of graphical languages

This standard supports the exchange of IEC 61131-3 projects including POUs programmed in one of the graphical languages, i.e. Function Block Diagram and Ladder Diagram, and Sequential Function Chart. This standard defines a virtual coordinate system in order to optionally express the arrangement of graphical elements (e.g. function block invocation, contact, coil, and step). This enables (but does not require) the importing system to arrange the graphical representation of the imported code as similar as possible to its representation in the exporting system. Therefore each element in a graphical language can store its position and size as an XML attribute. Additionally, a sub-element (e.g. formal parameters of a function block invocation) can store a position relative to its parent element. Furthermore a connection between two elements can store a set of positions to express the routing layout. Connections are described in the sub-clause 13.5.

The graphical position and size are optional attributes and shall be used for graphical representation only. The evaluation order of the graphical elements is given by the

“evaluationOrder” of the “FbdNetwork” and “Rung” (LD) elements. The elements contained in such network or rung are connected to each other by connections. With these connections the rules of IEC 61131-3 apply for the execution order within a network or rung. Similarly within SFC the steps, transitions and action blocks are connected to each other.

The virtual coordinate system has two axes X and Y. The origin (0, 0) of the coordinate system shall be the top-left corner of the graphical body. Positive X is to the right, and positive Y is down. A position of a graphical element identifies the upper left corner of this element. Consequently the position of an element shall always have positive coordinates; while a relative position between a sub-element and its parent may have positive or negative values.

In order to help the importing system to scale the graphics, the exporting system can add information about its usage of the virtual coordinate system in the “CoordinateInfo” of the “ContentHeader” as shown in Figure 2.

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

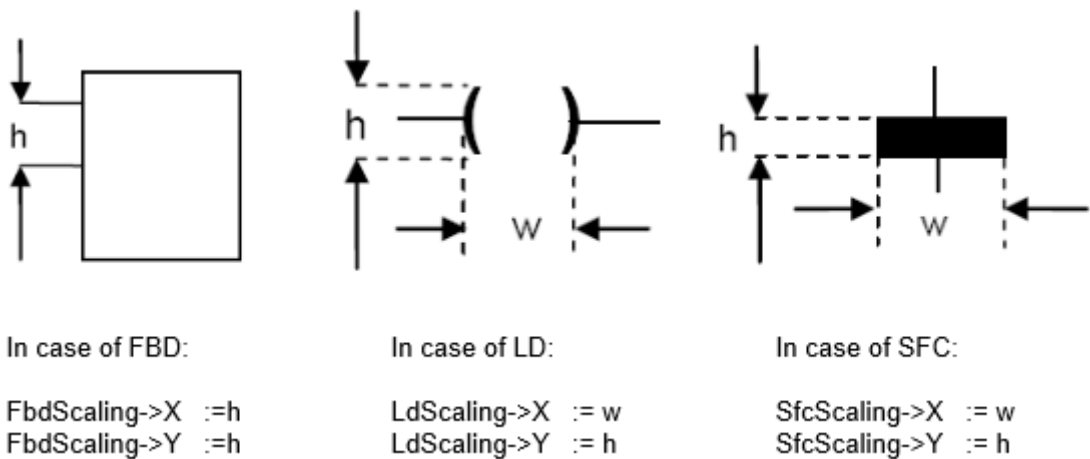


Figure 2 – Mapping coordinate information to the coordinate system

With this given information a coordinate transformation can be performed. For example if a system A with FbdScaling = 10 exports a function block invocation at the position P(30, 20) then during import a system B with FbdScaling = 5 may transform the position P to (15, 10) as shown in Figure 3.

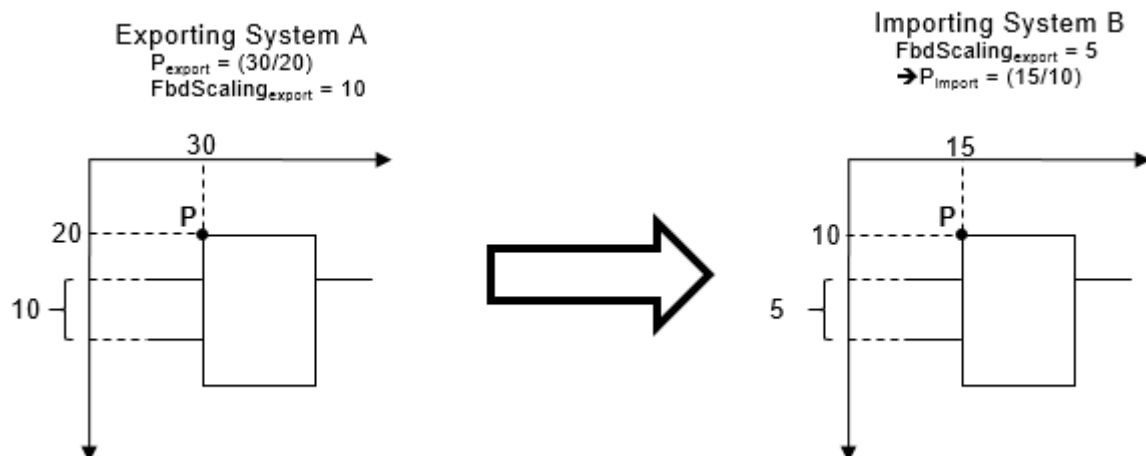


Figure 3 – Transforming coordinate position using the scaling information

4.3 Schema extension concepts

The schema is created in accordance to the language elements and features defined in IEC 61131-3. Nevertheless a PADT using the schema might support additional features or might need to store additional information in the exported file. The schema supports two extension mechanisms allowing incorporation of information in the exchange format, which is outside the scope of IEC 61131-3.

The first mechanism is the “AddData” element, which can be nested below various other elements. Its usage is intended for the extension of other elements, by which they are enriched with information, but which is not crucial for the element’s exchange between tools. Annex B.1 contains recommendations for possible common extensions which can be done by using “AddData”. They should be seen as a guideline for designing new extensions.

A PADT making use of the “AddData” mechanism should also provide the schemas defining those extensions. All additional schemas used shall be listed in the “AddDataInfo” element described in the sub-clause 6.3. Details about the contents of “AddData” are contained in the sub-clause 15.2.

The second mechanism usable for extensions is abstract types. All abstract types used in the schema are listed in the clause 7. Abstract types allow complex extensions of the schema which can be standalone and do not rely on extending already existing elements. Instead a new element type can be derived from the abstract type and used as a substitution for the element type where the abstract type is specified. Possible examples for such extensions are vendor specific programming languages or task types. As with “AddData” the schemas for this extensions should also be made available for other tools required to understand the extensions. Annex B.2 contains a recommendation for possible common extensions which can be done by using abstract types. It should be seen as a guideline for designing new extensions.

5 Compliance

5.1 General

A PLC programming and debugging tool (PADT), as defined in IEC 61131-1, which claims to comply, wholly or partially, with the requirements of this standard shall:

- provide a subset of the features and provide the corresponding vendor’s compliance statement as defined below,
- not require the inclusion of substitute or additional schema elements in order to accomplish any of the features,
- provide a document that specifies all vendor specific dependencies. This includes the implementation dependencies explicitly designated in this standard and the limiting

parameters like maximum length, number, size and range of value which are not explicitly defined in this standard,

d) provide a document that specifies all errors that are detectable and reported by the implementation,

e) not use any of the schema element names defined in this standard for implementation-defined features whose functionality differs from that described in this standard.

5.2 Feature tables

This part re-uses the feature tables defined in IEC 61131-3. All tables in IEC 61131-3 are used for a special purpose in a common way. The first column contains the “feature number”, the second column gives the “feature description”, and the following columns may contain examples or further information. This table structure is used in the vendor’s compliance statement.

There are 2 possibilities for a vendor in order to re-use the IEC 61131-3 feature tables for the purpose of this standard:

1. adding 2 columns (XML export, XML import) to the feature tables of the PADT, or
2. creating a separate list of feature tables for the export / import functionality.

5.3 Vendor’s compliance statement

The vendor may define any consistent subset of the features listed in the feature tables of IEC 61131-3 and shall declare the provided subset in the “Vendor’s compliant statement”.

The vendor’s compliance statement shall be included in the documentation accompanying the system, or shall be produced by the system itself.

The format of the vendor’s compliance statement shall provide the following information:

- The general information including the vendor name and address, the product name and version, and the date of issue.
- For each implemented feature the number of the corresponding feature table, the feature number and the applicable programming language.
Optional is the title and subtitle of the feature table, the feature description, examples, vendor’s note etc.

Not implemented tables and features may be omitted.

Figure 4 of IEC 61131-3 provides an example of a compliance statement, which could be modified for use with this standard.

6 Main schema element “Project”

6.1 General

The main schema element “Project” represents an IEC 61131-3 project. Its content is shown in Figure 4.

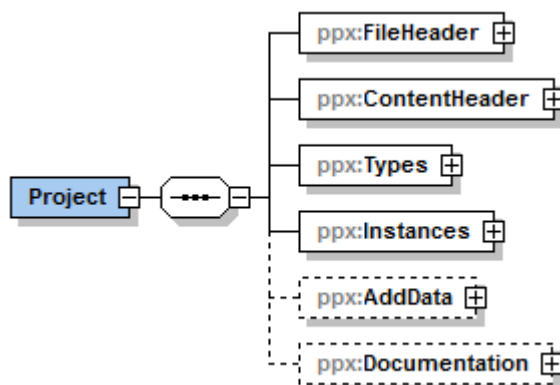


Figure 4 – Main schema element “Project”

“FileHeader”, “ContentHeader”, “Types”, and “Instances” are explained in the sub-clauses 6.2, 6.3, 6.4, and 6.5.

“Documentation” is of type “FormattedText” which is explained in the sub-clause 15.3.

6.2 “FileHeader”

“FileHeader” represents information concerning the exporting tool of the XML file. Its content is shown in Figure 5.

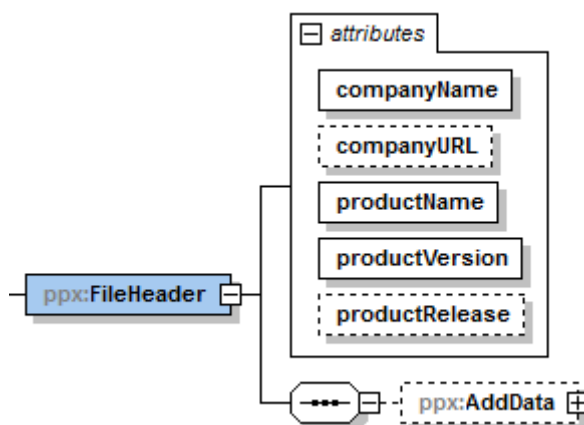


Figure 5 – Element “FileHeader”

The attributes “companyName” and “companyURL” are describing the vendor of the exporting tool, “productName”, “productVersion”, and “productRelease” the tool itself.

6.3 “ContentHeader”

“ContentHeader” represents overview information concerning the content of the XML file. Its content is shown in Figure 6.

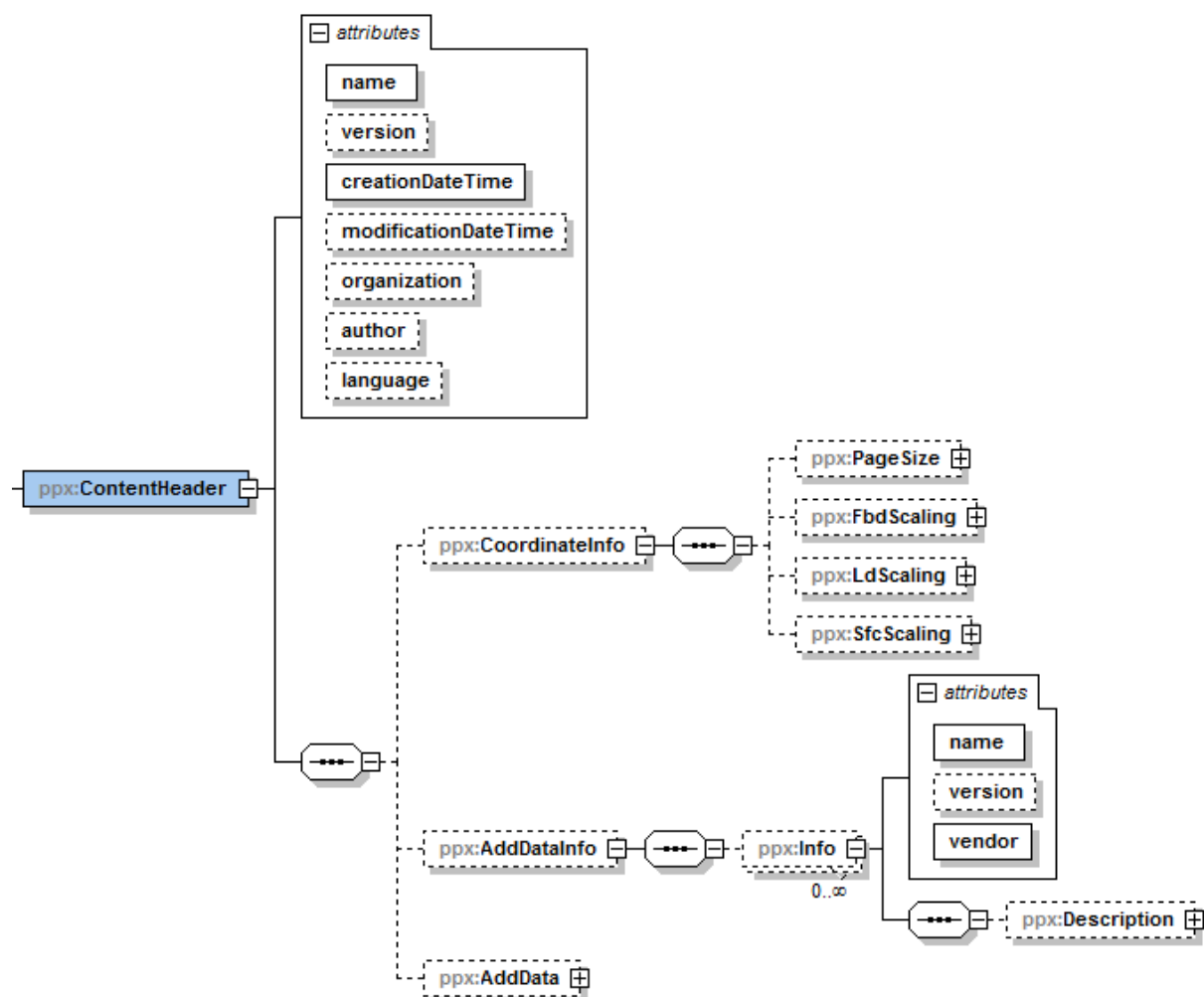


Figure 6 – Element “ContentHeader”

The attributes “name” (at export this is set to the project name), “version”, “creationDateTime”, “modificationDateTime”, “organization”, “author”, and “language” (language used in the definition of the project) are available.

“PageSize”, “FbdScaling”, “LdScaling”, and “SfcScaling” are of type “XyDecimalValue” which is explained in the sub-clause 15.1. The scaling concept is explained in the sub-clause 4.2.

The “AddDataInfo” represents an unique identification of the corresponding “AddData” content. The “name” attribute shall include the vendor domain (URI) to ensure unique names. The attribute “name” shall be used further in the produced XML document to refer to the corresponding AddData schema. Therefore “name” shall be unique in the produced XML document. The attribute “vendor” shall include the domain URL. Using this attribute “name” the importing tool may process the “AddData”. The vendor shall specify the behaviour of the importing tool in case the name is not known by the importing tool especially regarding a later export from this tool. The attribute “version” indicates a schema version. “Description” is a formatted text according to <http://www.w3.org/1999/xhtml> and contains a brief description of each schema used in the file.

6.4 “Types”

“Types” represents all type related elements, as defined in IEC 61131-3. Its content is shown in Figure 7.

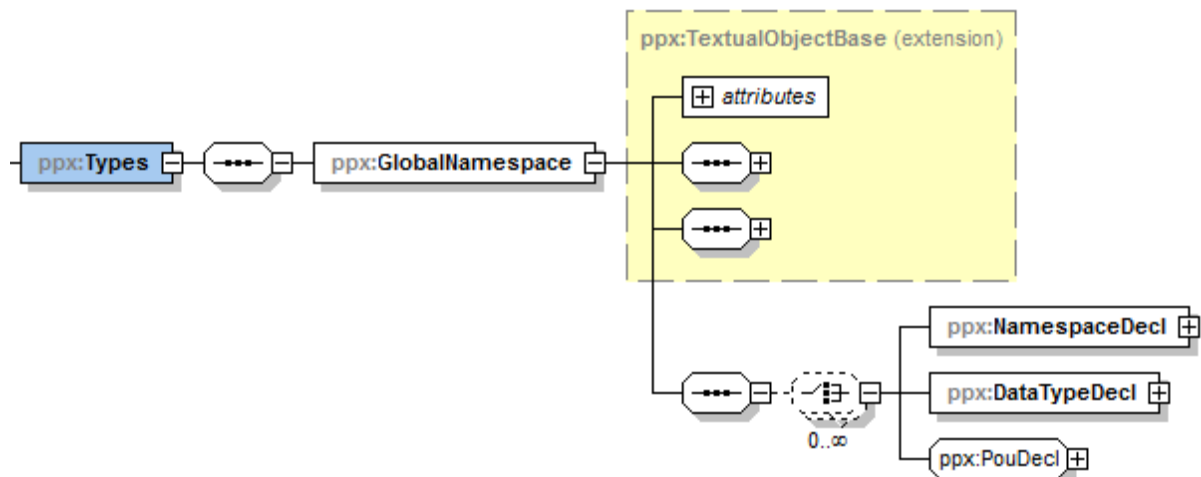


Figure 7 – Element “Types”

A “GlobalNamespace” is based on the abstract type “TextualObjectBase” which is explained in the sub-clause 7.5.2.

“NameSpaceDecl” of type “NameSpaceDecl” is explained in the clause 8. “DataTypeDecl” of type “UserDefinedTypeDecl” is explained in the sub-clause 9.1. The contents of the group “PouDecl” is explained in the sub-clause 10.1.

6.5 “Instances”

6.5.1 General (“Configuration”)

“Instances” represents all instance related elements, as defined in IEC 61131-3. Its content is shown in Figure 8.

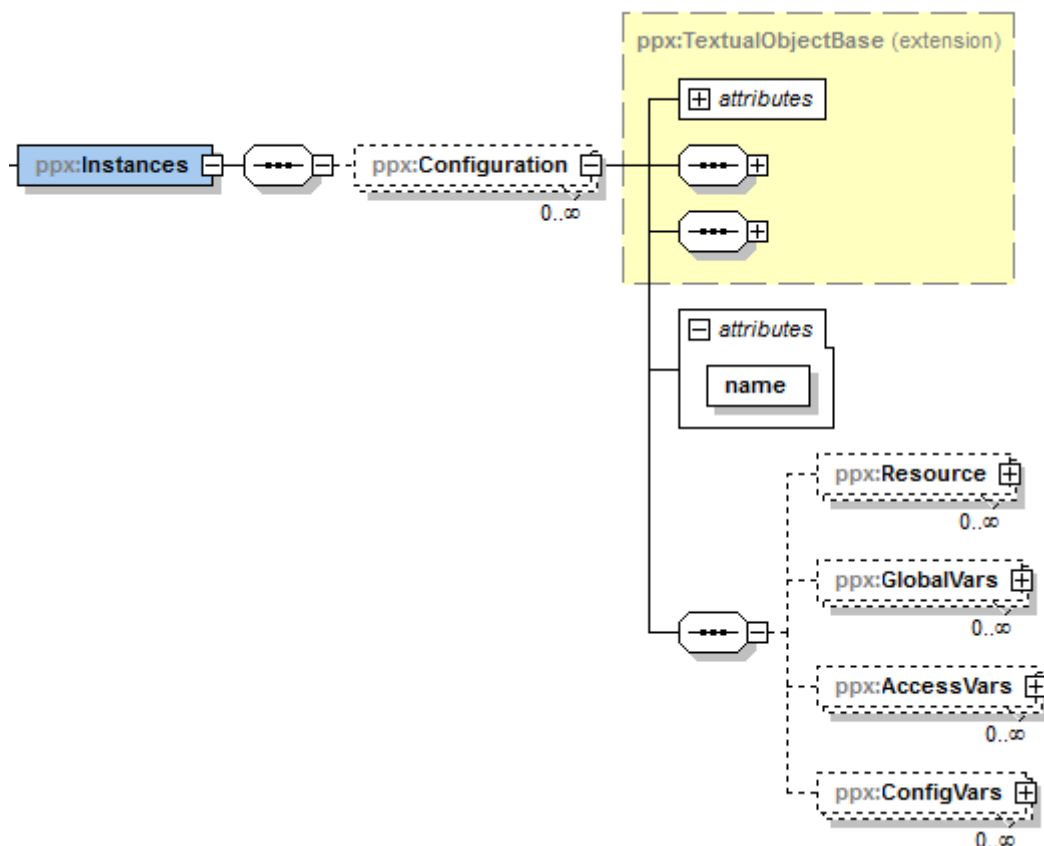


Figure 8 – Element “Instances”

368 A “Configuration” is based on the abstract type “TextualObjectBase” which is explained in the
 369 sub-clause 7.5.2. A “Configuration” is identified by the attribute “name”.

370 “Resource”, “GlobalVars” (IEC 61131-3: global variables) of type “VarList”, “AccessVars”, and
 371 “ConfigVars” are explained in the sub-clauses 6.5.2, 11.1, 6.5.3, and 6.5.4.

372 6.5.2 “Resource”

373 “Resource” represents resource, as defined in IEC 61131-3. Its content is shown in Figure 9.

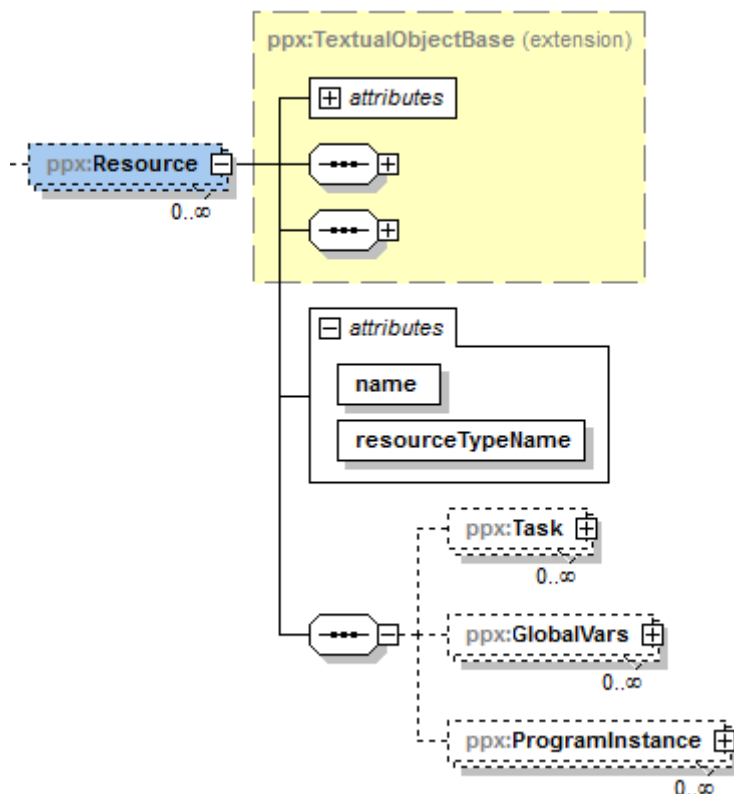


Figure 9 – Element “Resource”

376 A “Resource” is based on the abstract type “TextualObjectBase” which is explained in the
 377 sub-clause 7.5.2. A “Resource” is identified by the attribute “name”. Additionally, the
 378 “resourceTypeName” is available.

379 A “Resource” contain “Task” of type “TaskBase” which is explained in the sub-clause 7.5.4,
 380 “GlobalVars” (IEC 61131-3: global variables) of type “VarList” which is explained in the sub-
 381 clause 11.1, and “ProgramInstance” which is explained in the sub-clause 14.1.

382 6.5.3 “AccessVars”

383 “AccessVars” represents access variables, as defined in IEC 61131-3. Its content is shown in
 384 Figure 10.

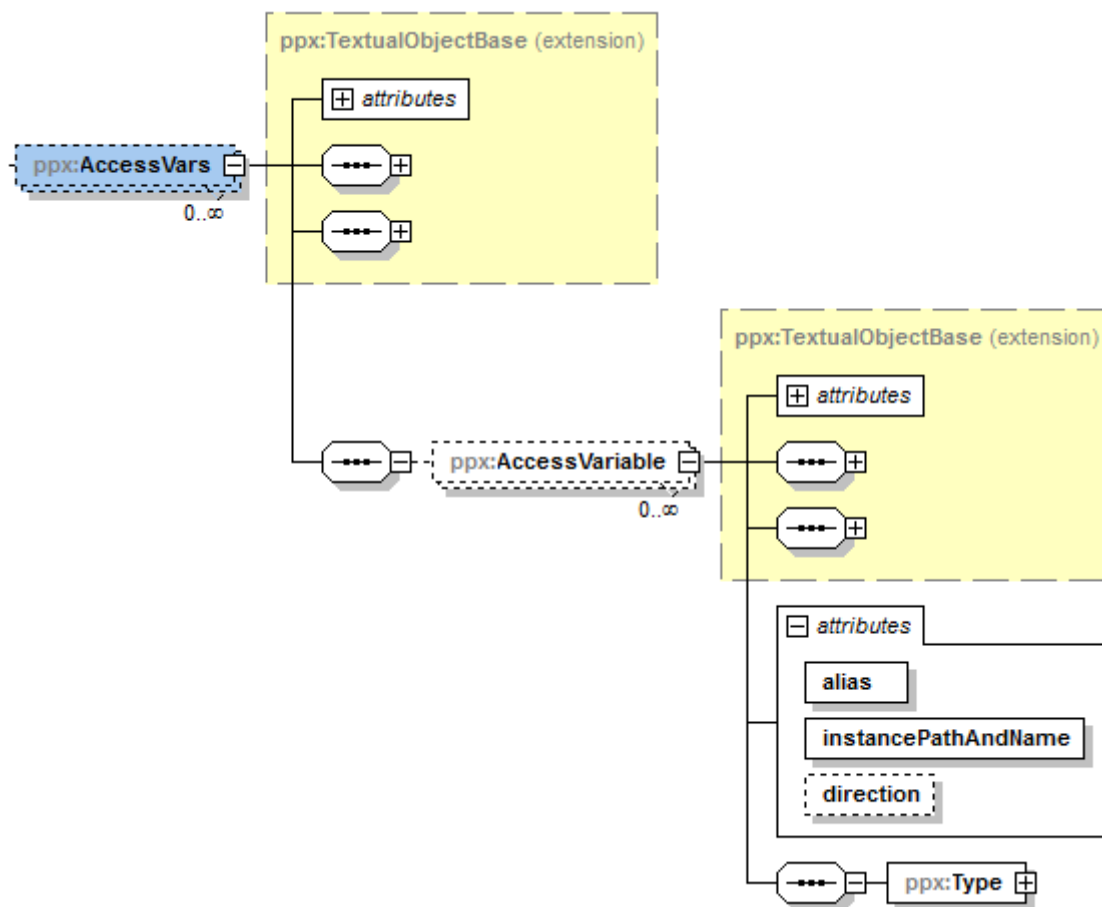


Figure 10 – Element “AccessVars”

“AccessVars” is based on the abstract type “TextualObjectBase” which is explained in the sub-clause 7.5.2. It consists of “AccessVariable”, again based on “TextualObjectBase”.

The attributes “alias”, “instancePathAndName”, and “direction” are available.

“Type” of type “TypeRef” is explained in the sub-clause 11.5.

6.5.4 “ConfigVars”

“ConfigVars” represents config variables, as defined in IEC 61131-3. Its content is shown in Figure 11.

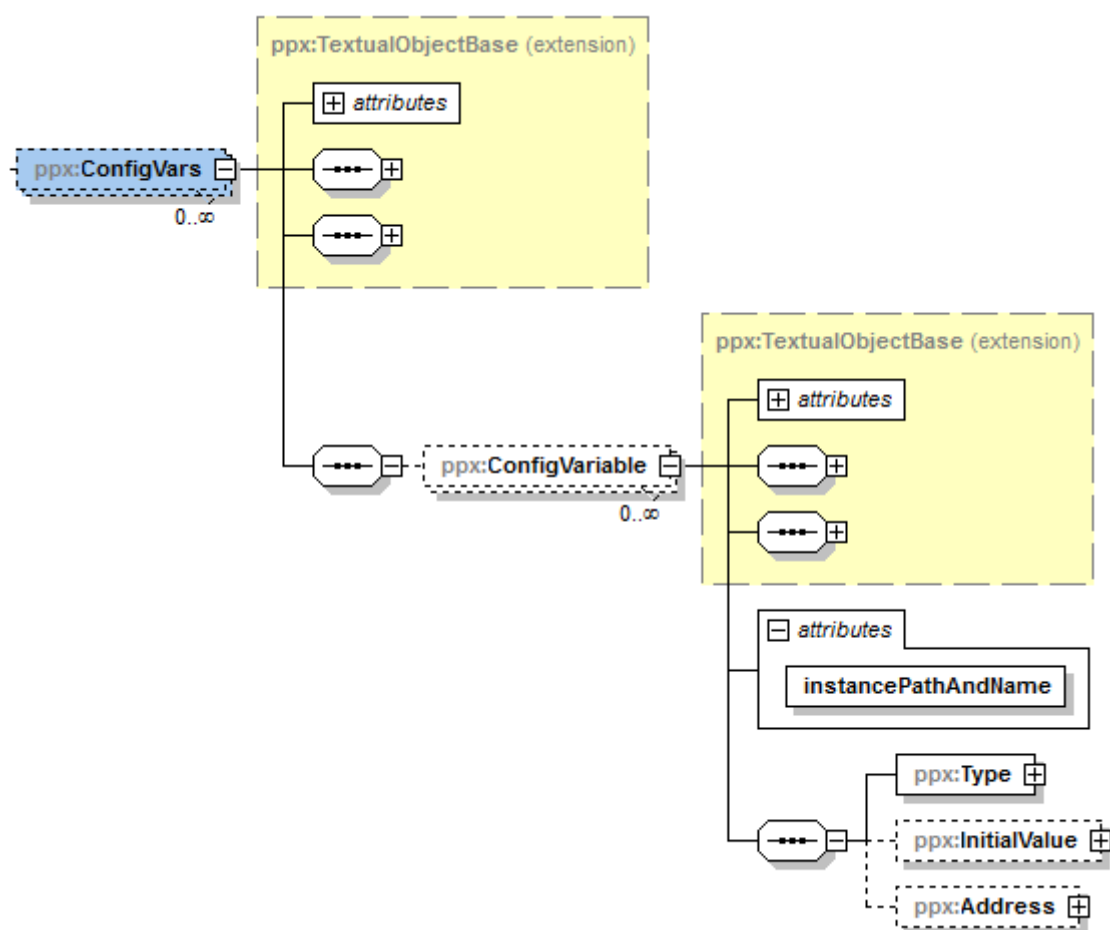


Figure 11 – Element “ConfigVars”

“ConfigVars” is based on the abstract type “TextualObjectBase” which is explained in the sub-clause 7.5.2. It consists of “ConfigVariable”, again based on “TextualObjectBase”.

The attribute “instancePathAndName” is available.

“Type” of type “TypeRef”, “InitialValue” of type “Value”, and “Address” of type “FixedAddressExpression” are explained in the sub-clauses 11.5, 11.6, and 11.8.

7 Abstract types

7.1 Purpose of abstract types

Abstract complex types are provided in the schema for either of the following purposes:

- A) Simply to avoid defining similar attributes and sub-elements in the specific complex type multiple times. For this purpose, common attributes and sub-elements, among specific complex types, are defined in an abstract complex type. This abstract type is extended by each related concrete type.
- B) For vendor-specific extension. For this purpose, the abstract complex type is specified as the type of the element in the standard schema. This allows any concrete complex type which extends the abstract type to be applied for the type of the element. In this case, the abstract type itself may even contain no attributes nor sub-elements. By adding extended complex type in vendor-specific extension schema, it becomes available as the type for the element.

7.2 Abstract types for data type specifications

7.2.1 General

Extension relationships among this kind of abstract types and concrete types are shown in Figure 12.

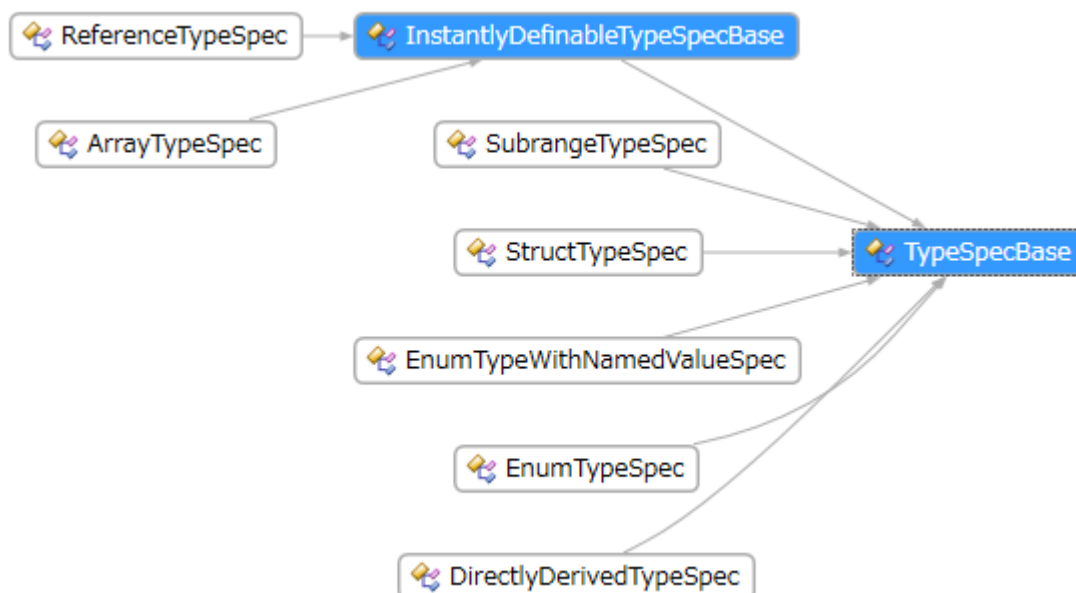


Figure 12 – Extension relationship among complex types for data type specifications

7.2.2 “TypeSpecBase”

The abstract complex type “TypeSpecBase” is provided for vendor-specific extension of user defined data types. It shall be specified as the extension base of the complex type which represents user defined data type specification. It has no specific contents.

7.2.3 “InstantlyDefinableTypeSpecBase”

The abstract complex type “InstantlyDefinableTypeSpecBase” is also provided for vendor-specific extension of user defined data types. It shall be specified as the extension base of the complex type which represents user defined data type specification which can be instantly defined within variable declarations. It is based on the abstract type “TypeSpecBase” which is explained in the sub-clause 7.2.2, but has no specific contents.

7.3 Abstract types for behaviour representations

7.3.1 General

Extension relationships among this kind of abstract types and concrete types are shown in Figure 13

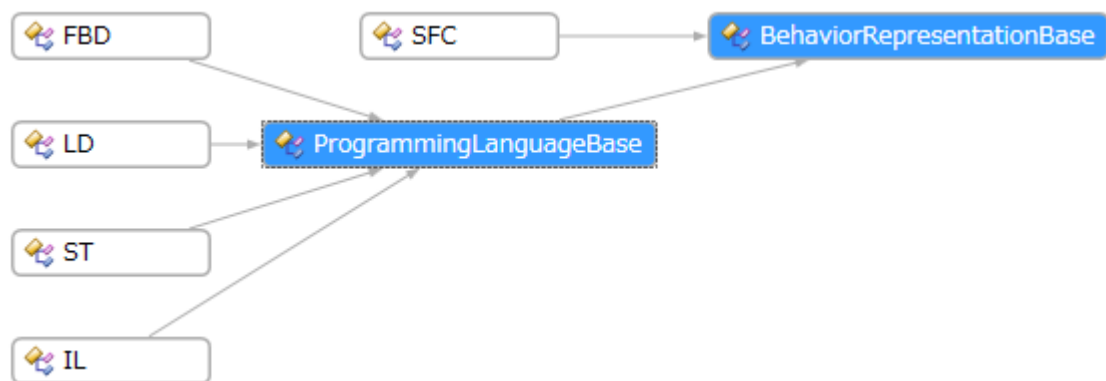


Figure 13 – Extension relationship among complex types for behaviour representations

7.3.2 “BehaviorRepresentationBase”

The abstract complex type “BehaviorRepresentationBase” is provided for vendor-specific extension of behaviour description. It shall be specified as the extension base of the complex type which represents high-level behaviour representation which needs to have some internal states such as SFC. It has no contents. Behaviour representations derived from this type is not available within function nor method.

7.3.3 “ProgrammingLanguageBase”

The abstract complex type “ProgrammingLanguageBase” is also provided for vendor-specific extension of behaviour description. It shall be specified as the extension base of the complex type which represents a graphical or textual programming language, such as FDB, LD, IL, or ST. It extends “BehaviorRepresentationBase” which is explained in the sub-clause 7.3.2 but has no additional contents. “ProgrammingLanguageBase” is available for function and method.

7.4 Abstract types for graphical objects

7.4.1 General

Extension relationships among this kind of abstract types and concrete types are shown in Figure 14.

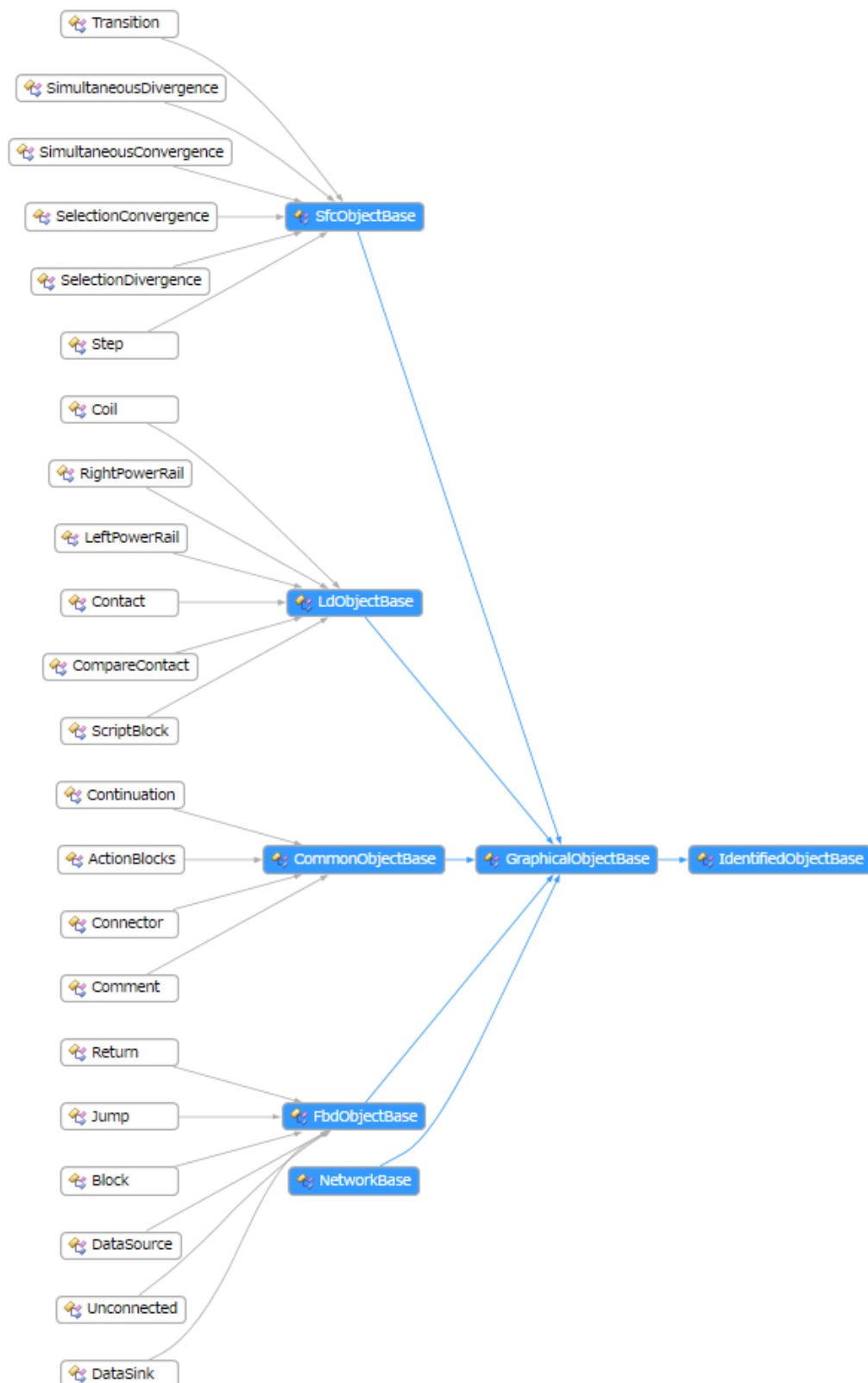


Figure 14 – Extension relationship among complex types for graphical objects

7.4.2 “IdentifiedObjectBase”

The complex type “IdentifiedObjectBase” is an abstract type which contains common data for objects to be identified with “globalId” attribute. Its content is shown in Figure 15.

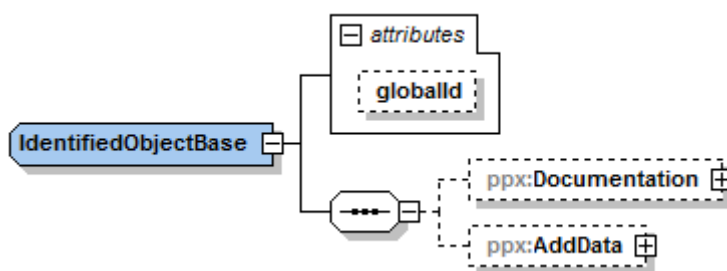


Figure 15 – Complex type “IdentifiedObjectBase”

The main purpose of the attribute “globalId” is the capability to unambiguously identify an element within an XML document from the outside.

The element “Documentation” of complex type “FormattedText” represents optional textual documentation for the object. “FormattedText” is explained in the sub-clause 15.3.

7.4.3 “GraphicalObjectBase”

The complex type “GraphicalObjectBase” is an abstract type which contains common data for graphical objects such as position and size. Its content is shown in Figure 16.

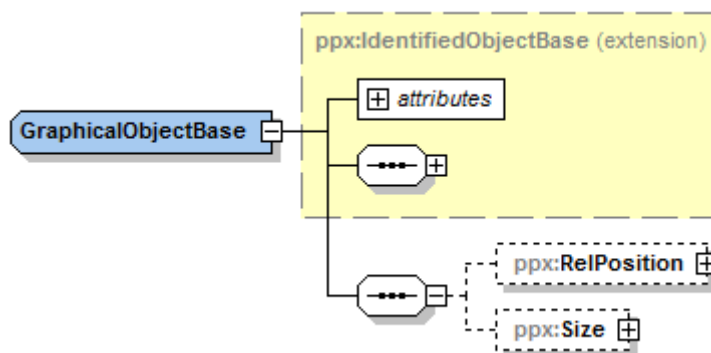


Figure 16 – Complex type “GraphicalObjectBase”

“GraphicalObjectBase” is based on the abstract type “IdentifiedObjectBase”.

The element “RelPosition” of type complex type “XyDecimalValue”, which is explained in the sub-clause 15.1, represents the relative position of the anchor point of the graphical object within the parent graphical object.

The element “Size” of type complex type “XyDecimalValue”, which is explained in the sub-clause 15.1, represents the size of the graphical object.

7.4.4 “CommonObjectBase”

The complex type “CommonObjectBase” is an abstract type to be the extension base of complex types which represent graphical objects commonly available in FBD, LD and SFC. This complex type may also be used for vendor-specific extension of graphical objects. By adding complex type extended from this abstract type, the vendor can make their specific graphical object commonly available in FBD, LD and SFC, in addition to the standard graphical objects such as “Connector”, “Continuation”, etc. Its content is shown in Figure 17.

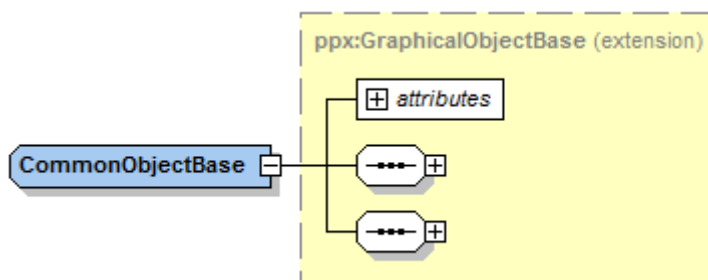


Figure 17 – Complex type “CommonObjectBase”

This complex type extends the content of the base complex type “GraphicalObjectBase” which is explained in the sub-clause 7.4.3 but itself has no specific contents.

7.4.5 “FbdObjectBase”

The complex type “FbdObjectBase” is an abstract type to be the extension base of complex types which represent graphical objects commonly available in FBD and LD. This complex type may also be used for vendor-specific extension of graphical objects. By adding complex type extended from this abstract type, the vendor can make their specific graphical object commonly available in FBD and LD in addition to the standard graphical objects such as “Block”, “DataSource”, “DataSink”, etc. Its content is shown in Figure 18.

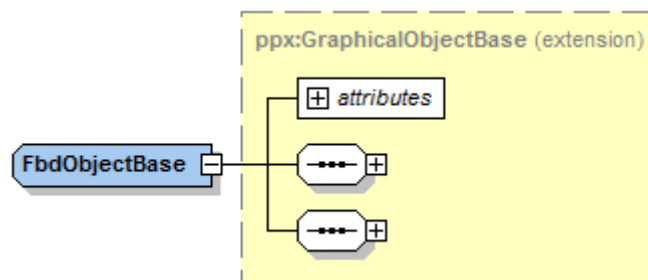


Figure 18 – Complex type “FbdObjectBase”

This complex type extends the content of the base complex type “GraphicalObjectBase” which is explained in the sub-clause 7.4.3 but itself have no specific contents.

7.4.6 “LdObjectBase”

The complex type “LdObjectBase” is an abstract type to be the extension base of complex types which represent graphical objects only available in LD. This complex type may also be used for vendor-specific extension of graphical objects. By adding complex type extended from this abstract type, the vendor can make their specific graphical object only available in LD in addition to the standard graphical objects such as “Coil”, “Contact”, etc. Its content is shown in Figure 19.

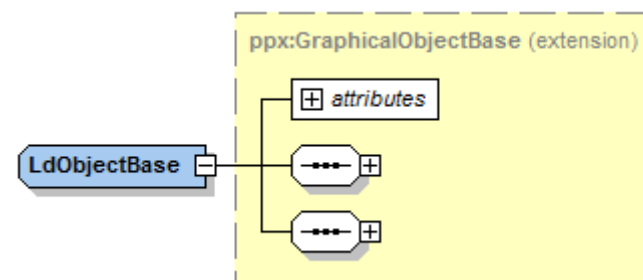
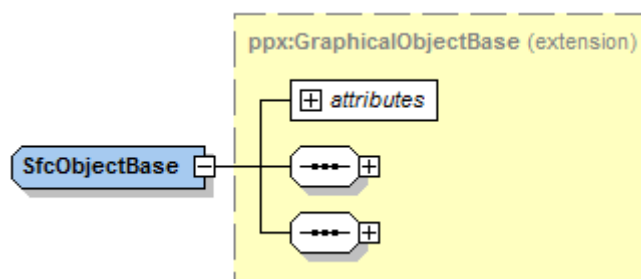


Figure 19 – Complex type “LdObjectBase”

505 This complex type extends the content of the base complex type “GraphicalObjectBase” which
 506 is explained in the sub-clause 7.4.3 but itself have no specific contents.

507 7.4.7 “SfcObjectBase”

508 The complex type “SfcObjectBase” is an abstract type to be the extension base of complex
 509 types which represent graphical objects only available in SFC. It may also be used for vendor-
 510 specific extension of graphical objects. By adding complex type extended from this abstract
 511 type, the vendor can make their specific graphical object only available in SFC in addition to
 512 the standard graphical objects such as “Step”, “Transition”, etc. Its content is shown in Figure
 513 20.

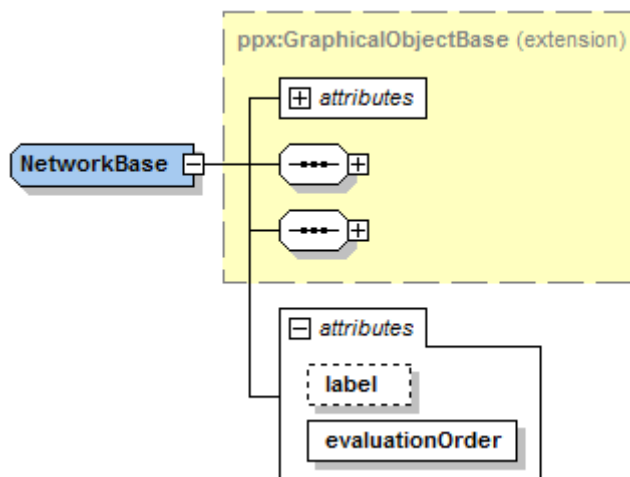


514
 515 **Figure 20 – Complex type “SfcObjectBase”**

516 This complex type extends the content of the base complex type “GraphicalObjectBase” which
 517 is explained in the sub-clause 7.4.3 but itself have no specific contents.

518 7.4.8 “NetworkBase”

519 The complex type “NetworkBase” is an abstract type which contains common attributes and
 520 sub-element for the network of LD and FBD. It is specified as the extension base of the
 521 complex type of “Rung” element of the complex type “LD”, and “FbdNetwork” element of the
 522 complex type “FBD”. Its content is shown in Figure 21.



523
 524 **Figure 21 – Complex type “NetworkBase”**

525 “NetworkBase” extends the content of the base complex type “GraphicalObjectBase”.

526 The optional attribute “label” represents the label of the network which can be specified as the
 527 target of the execution control element “jump”.

528 The required attribute “evaluationOrder” represents evaluation order of each network within
 529 LD or FBD, which have to be significant from the view point of IEC 61131-3. The network with
 530 a smaller number of “evaluationOrder” shall be evaluated earlier. It shall not have negative
 531 value. This attribute is provided because XML 1.0 specification of W3C does not ensure that
 532 appearance order of sibling elements is significant. It is strongly recommended that the

exporter should give ascending incremental number to the attribute of each “Rung” element within a LD, and each “FbdNetwork” elements within a LD or FBD.

7.5 Abstract types for textual constructs

7.5.1 General

Extension relationships among this kind of abstract types and concrete types are shown in Figure 22.

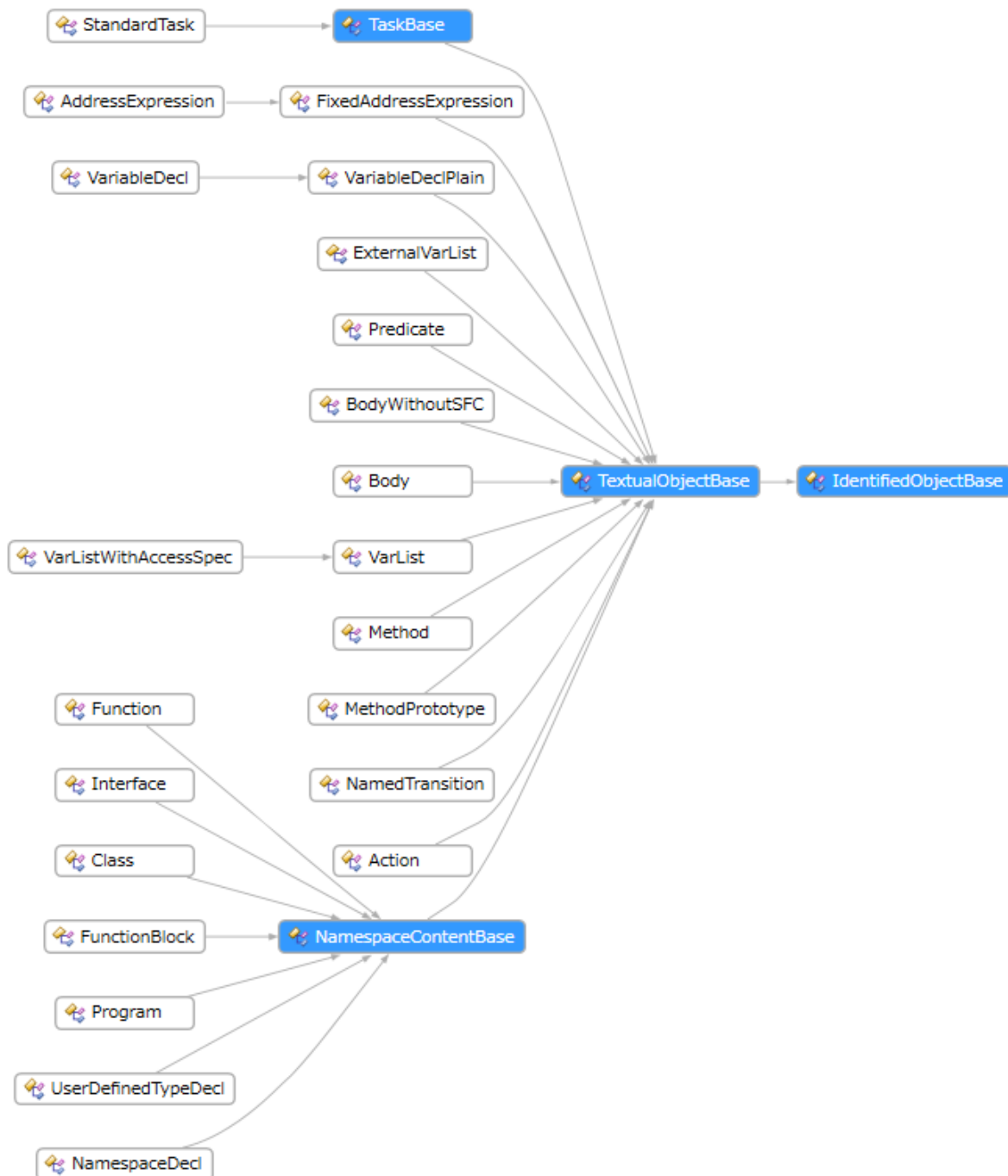


Figure 22 – Extension relationship among complex types for textual objects

7.5.2 “TextualObjectBase”

The complex type “TextualObjectBase” is an abstract type which contains common data for all textual objects declared by some textual construct. Its content is shown in Figure 23.

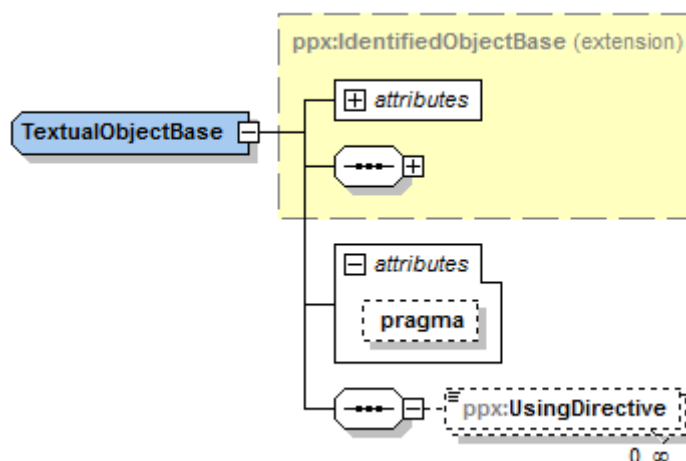


Figure 23 – Complex type “TextualObjectBase”

“TextualObjectBase” is based on the abstract type “IdentifiedObjectBase” which is explained in the sub-clause 7.4.2.

The attribute “pragma” is of type string. In order to hold several pragmas within the same string each pragma shall be embedded in curly brackets, as the syntax is defined in IEC 61131-3.

The element “UsingDirective” of type “SimpleIdentifier” which is explained in the sub-clause 15.6 represents using namespace directive with “USING” keyword, as defined in IEC 61131-3.

7.5.3 “NamespaceContentBase”

The complex type “NamespaceContentBase” is an abstract type which contains common data for the contents of namespace such as data types and POU. Its content is shown in Figure 24.

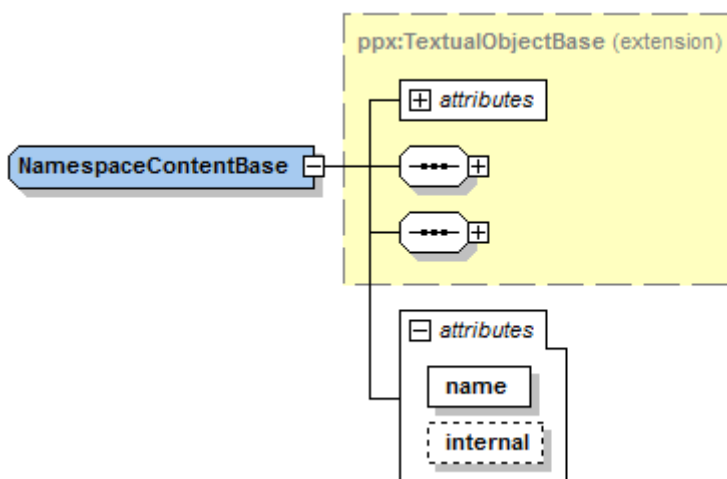


Figure 24 – Complex type “NamespaceContentBase”

“NamespaceContentBase” is based on the abstract type “IdentifiedObjectBase” which is explained in the sub-clause 7.4.2.

The required attribute “name” represents the name of the object, such as user-defined data type or POU definition. The optional attribute “internal” represents internal access specifier with keyword “INTERNAL”, as defined in IEC 61131-3.

7.5.4 “TaskBase”

The complex type “TaskBase” is an abstract type to be the extension base of complex types which represent task as defined in IEC 61131-3. It is provided for vendor-specific extension of task. By adding complex type extending this abstract type, the vendor can make their specific task available in addition to the “StandardTask” which also extends this abstract type. Its content is shown in Figure 25.

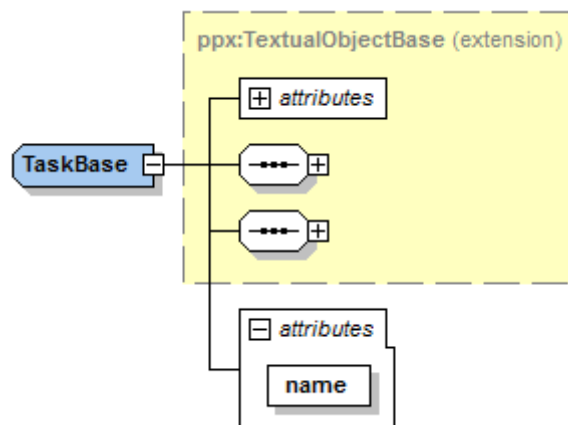


Figure 25 – Complex type “TaskBase”

“TaskBase” is based on the abstract type “IdentifiedObjectBase” which is explained in the sub-clause 7.4.2.

The required attribute “name” represents name of the task, which can be referenced by program instance declaration, as defined in IEC 61131-3.

8 “NamespaceDecl”

The complex type “NamespaceDecl” represents namespace, as defined in IEC 61131-3. Its content is shown in Figure 26.

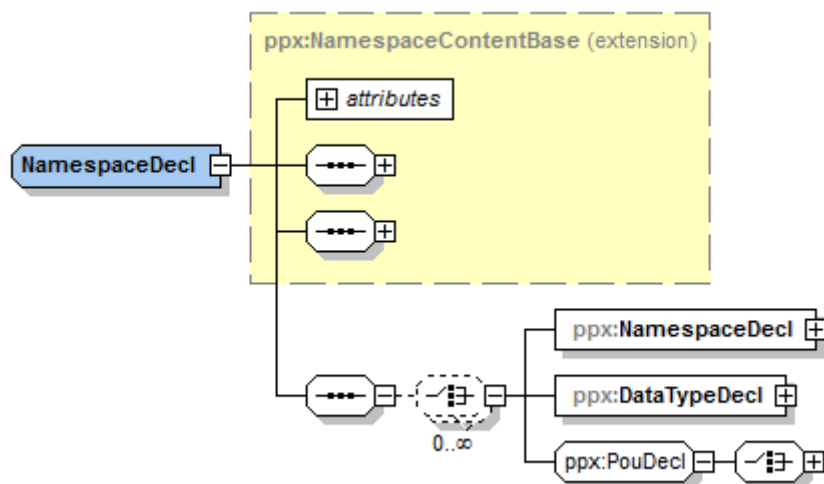


Figure 26 – Complex type “NamespaceDecl”

A “NamespaceDecl” is based on the abstract type “NamespaceContentBase” which is explained in the sub-clause 7.5.3.

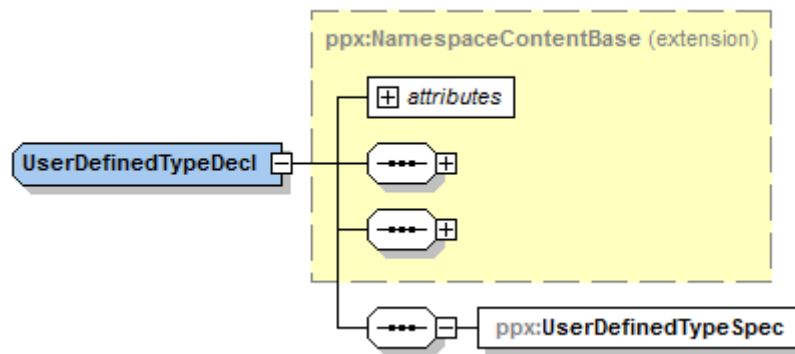
Namespaces may be defined recursively, so the element “NamespaceDecl” can be contained in itself.

585 “DataTypeDecl” of type “UserDefinedTypeDecl” is explained in the sub-clause 9.1. The
 586 contents of the group “PouDecl” is explained in the sub-clause 10.1.

587 9 User defined data type declaration

588 9.1 “UserDefinedTypeDecl”

589 The complex type “UserDefinedTypeDecl” represents all user-defined data type related
 590 elements, as defined in IEC 61131-3. Its content is shown in Figure 27.



591
 592 **Figure 27 – Complex type “UserDefinedTypeDecl”**

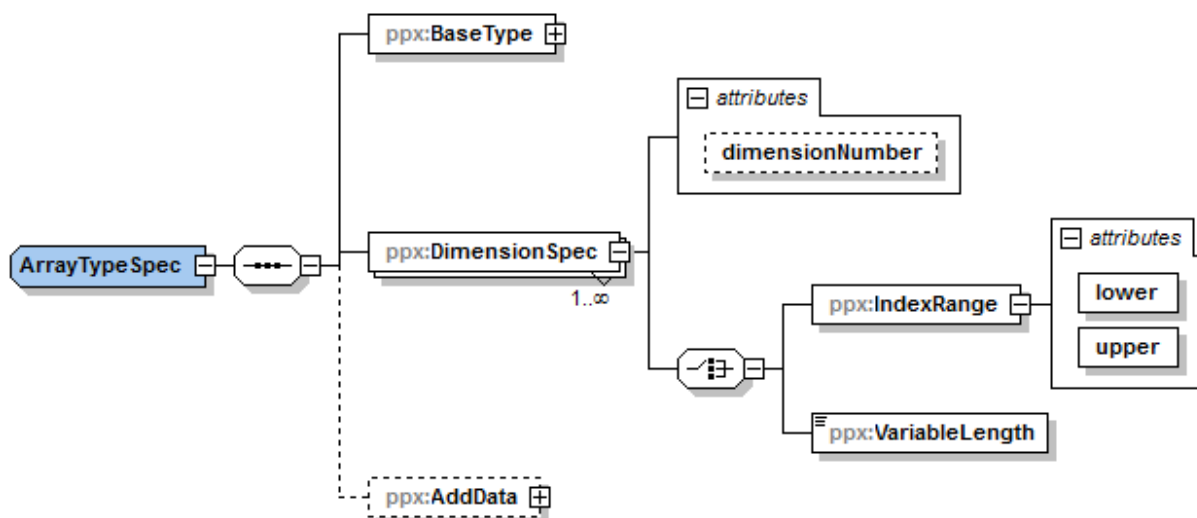
593 A “UserDefinedTypeDecl” inherits all elements of the type “NamespaceContentBase” which is
 594 explained in the sub-clause 7.4.2.

595 “UserDefinedTypeSpec” of abstract type “TypeSpecBase” is explained in the sub-clause
 596 7.2.2. This element shall contain a derived concrete user-defined data type. Sub-clause 4.3
 597 explains how abstract types are to be used.

598 The possible derived types as defined in IEC 61131-3 are explained in the sub-clauses 9.2,
 599 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, and 9.9.

600 9.2 “ArrayTypeSpec”

601 The complex type “ArrayTypeSpec” represents array type, as defined in IEC 61131-3. Its
 602 content is shown in Figure 28.



603
 604 **Figure 28 – Complex type “ArrayTypeSpec”**

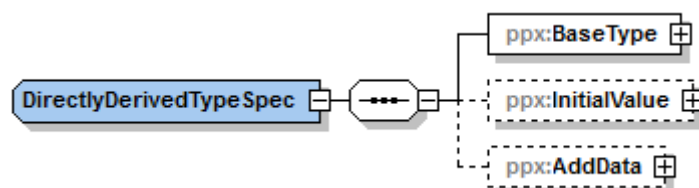
605 “BaseType” represents the base type name of the array. “BaseType” is of type “TypeRef”,
 606 which is explained in the sub-clause 11.5.

607 The "DimensionSpec" shall be used to specify one or more dimensions for the array type. The
608 actual number of dimensions is set as value of the attribute "dimensionNumber".

609 The "IndexRange" element shall be used to specify the lower and upper border of the
610 dimension as attribute values. If it is a variable length array as defined in IEC 61131-3 then
611 the "VariableLength" element shall be used instead.

612 9.3 "DirectlyDerivedTypeSpec"

613 The complex type "DirectlyDerivedTypeSpec" represents directly derived data types, as defined
614 in IEC 61131-3. Its content is shown in Figure 29.



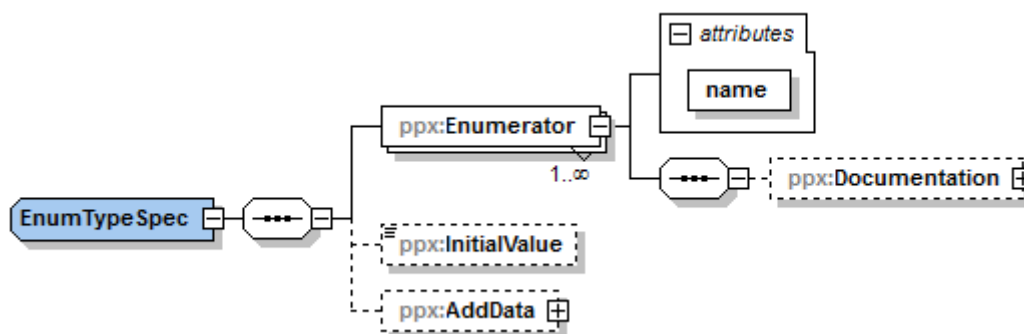
615

616 **Figure 29 – Complex type "DirectlyDerivedTypeSpec"**

617 "BaseType" of type "TypeRef" and "InitialValue" of type "Value" are explained in the sub-
618 clauses 11.5 and 11.6.

619 9.4 "EnumTypeSpec"

620 The complex type "EnumTypeSpec" represents enumerated type, as defined in IEC 61131-3.
621 Its content is shown in Figure 30.



622

623 **Figure 30 – Complex type "EnumTypeSpec"**

624 For each element of the enumeration list an element "Enumerator" is required. The attribute
625 "name" stores the associated identifier. "Documentation" of type "FormattedText" is explained
626 in the sub-clause 15.3.

627 "InitialValue" of type "SimpleIdentifier", which is explained in the sub-clause 15.6, represents
628 the initial value of the enumerated type, which should be the name of one of the enumerators
629 of this type.

630 9.5 "EnumTypeWithNamedValueSpec"

631 The complex type "EnumTypeWithNamedValueSpec" represents data type with named values,
632 as defined in IEC 61131-3. Its content is shown in Figure 31.

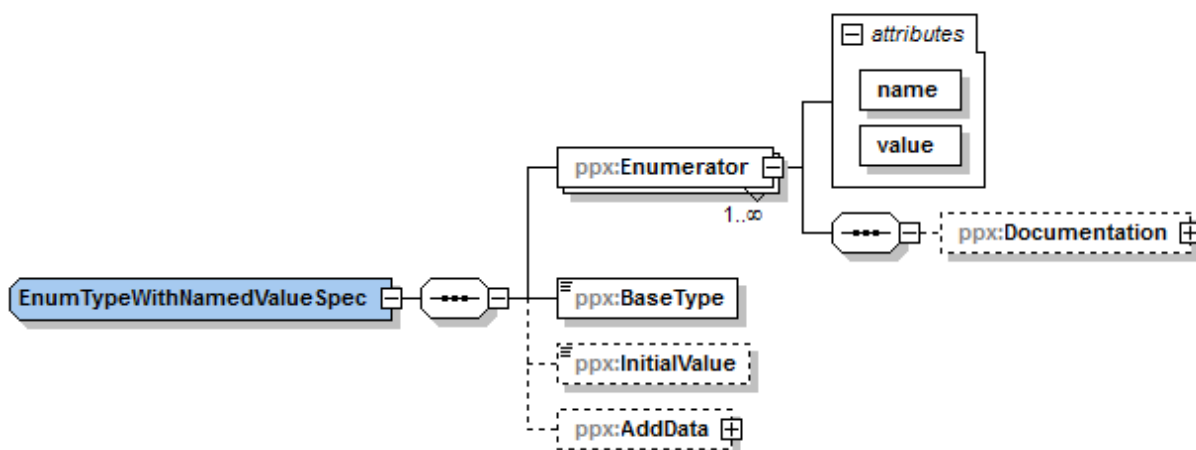


Figure 31 – Complex type “EnumTypeWithNamedValueSpec”

For each element of the enumeration list an element "Enumerator" is required. The attribute "name" stores the associated identifier. Additionally, there is an attribute "value" for storage of the enumeration value associated with the identifier. "Documentation" of type "FormattedText" is explained in the sub-clause 15.3.

"BaseType" of type "ElementaryType" and "InitialValue" of type "SimpleIdentifier" are explained in the sub-clauses 9.9 and 15.6.

9.6 “StructTypeSpec”

The complex type “StructTypeSpec” represents structured type, as defined in IEC 61131-3. Its content is shown in Figure 32.

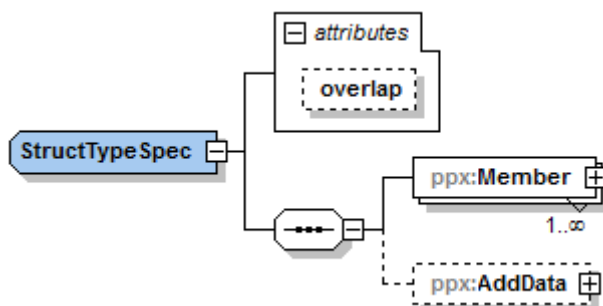


Figure 32 – Complex type “StructTypeSpec”

The attribute "overlap" maps to the OVERLAP keyword in structured data type declarations. "Member" of type "VariableDecl" is explained in the sub-clause 11.3.

9.7 “SubrangeTypeSpec”

The complex type “SubrangeTypeSpec” represents subrange data types, as defined in IEC 61131-3. Its content is shown in Figure 33.

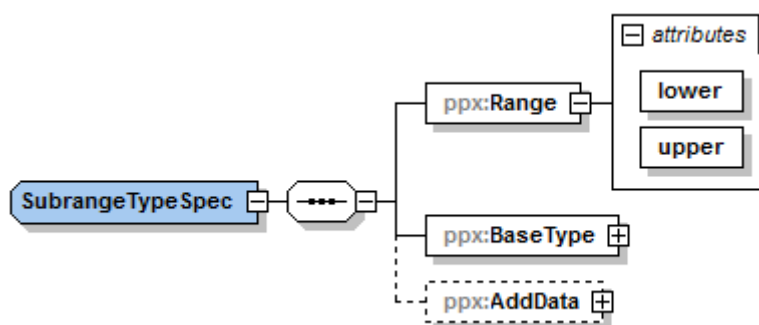


Figure 33 – Complex type “SubrangeTypeSpec”

The limits of the subrange type shall be given in the attributes "lower" and "upper" of the element "Range".

"BaseType" of type "TypeRef" is explained in the sub-clause 11.5.

9.8 “ReferenceTypeSpec”

The complex type “ReferenceTypeSpec” represents references, as defined in IEC 61131-3. Its content is shown in Figure 34.

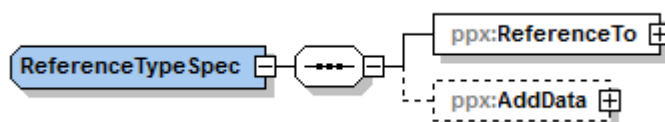


Figure 34 – Complex type “ReferenceTypeSpec”

“ReferenceTo” of type “TypeRef” is explained in the sub-clause 11.5.

9.9 “ElementaryType”

The simple type “ElementaryType” represents all elementary data types, as defined in IEC 61131-3. It is of type string and the possible values are restricted to the names used in IEC 61131-3 for the elementary data types.

10 POU declaration

10.1 “PouDecl”

The group “PouDecl” represents a choice between the different POUs (program organisation units), as defined in IEC 61131-3. Its content is shown in Figure 35.

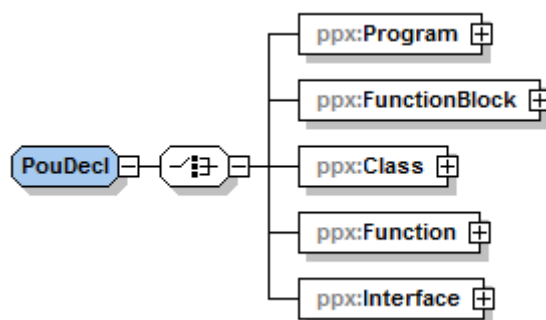
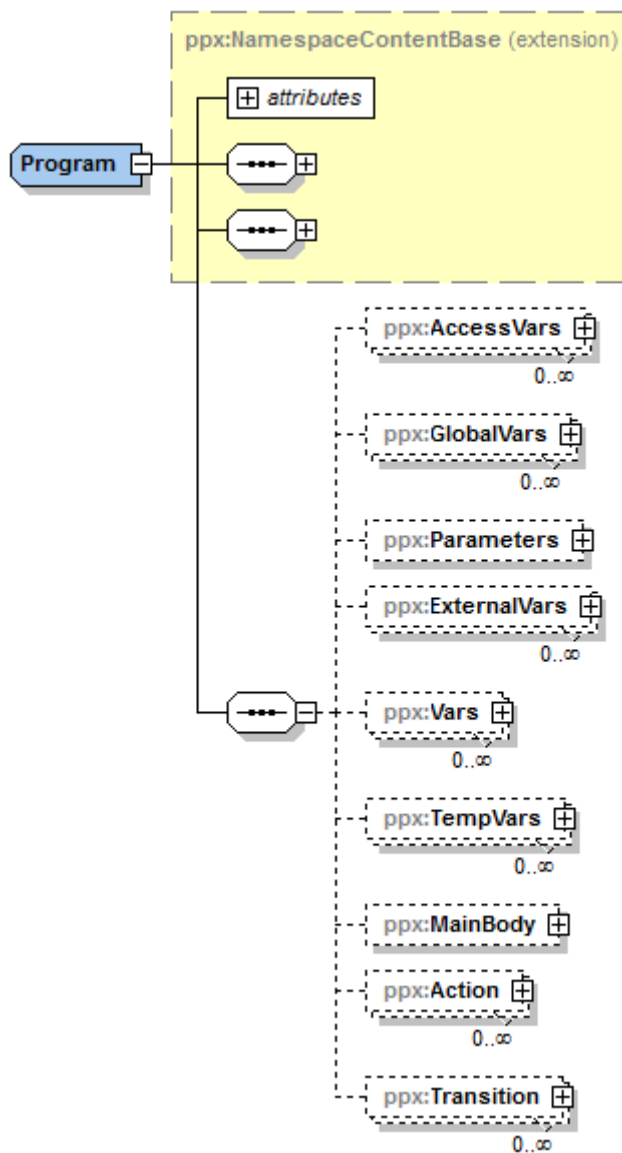


Figure 35 – Complex type “PouDecl”

672 “Program” of type “Program”, “FunctionBlock” of type “FunctionBlock”, “Class” of type “Class”,
 673 “Function” of type “Function”, and “Interface” of type “Interface” are explained in the sub-
 674 clauses 10.2, 10.3, 10.4, 10.5, and 10.6.

675 10.2 “Program”

676 The complex type “Program” represents a program, as defined in IEC 61131-3. Its content is
 677 shown in Figure 36.



678
 679 **Figure 36 – Complex type “Program”**

680 “AccessVars” and “GlobalVars” are of type “VarList” which is explained in the sub-clause 11.1.
 681 “Parameters” of type “ParameterSet” is explained in the sub-clause 10.11. “ExternalVars” of
 682 type “ExternalVarList” is explained in the sub-clause 11.2. “Vars” of type
 683 “VarListWithAccessSpec” is explained in the sub-clause 11.2. “TempVars” of type “VarList” is
 684 explained in the sub-clause 11.1.

685 “MainBody” of type “Body” is explained in the sub-clause 10.14.

686 “Action” of type “Action” and “Transition” of type “NamedTransition” are explained in the sub-
 687 clauses 10.7 and 10.8.

10.3 “FunctionBlock”

The complex type “FunctionBlock” represents a function block, as defined in IEC 61131-3. Its content is shown in Figure 37.

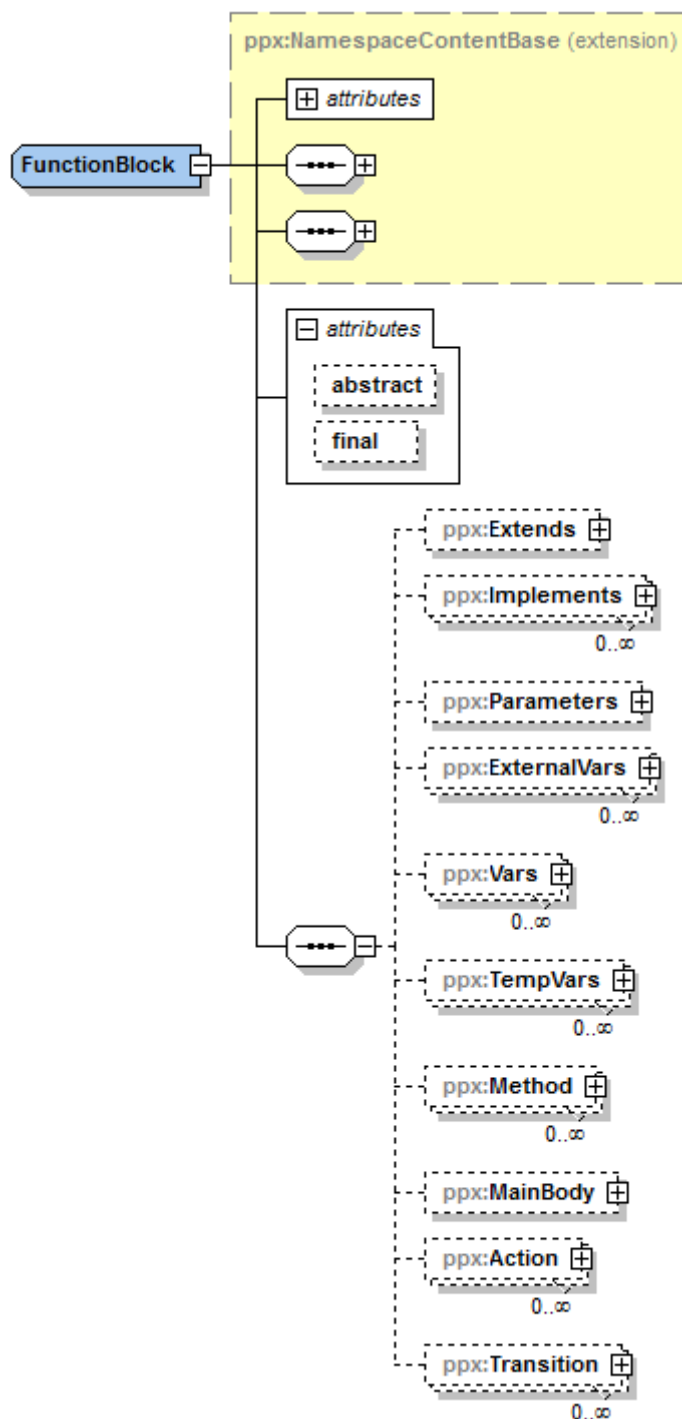


Figure 37 – Complex type “FunctionBlock”

The attributes “abstract” and “final” represent whether the function block has been defined using the keywords ABSTRACT and FINAL, as defined in IEC 61131-3.

“Extends” contains the name of base class or function block if applicable. “Implements” contains the implemented interfaces if applicable. Both are of type “typeRef” which is explained in the sub-clause 11.5.

698 “Parameters” of type “ParameterSet” is explained in the sub-clause 10.11. “ExternalVars” of
 699 type “ExternalVarList” is explained in the sub-clause 11.2. “Vars” of type
 700 “VarListWithAccessSpec” is explained in the sub-clause 10.12. “TempVars” of type “VarList” is
 701 explained in the sub-clause 11.1.

702 “Method” of type “Method” is explained in the sub-clause 10.10.

703 “MainBody” of type “Body” is explained in the sub-clause 10.14.

704 “Action” of type “Action” and “Transition” of type “NamedTransition” are explained in the sub-
 705 clauses 10.7 and 10.8.

706 10.4 “Class”

707 The complex type “Class” represents a class, as defined in IEC 61131-3. Its content is shown
 708 in Figure 38.

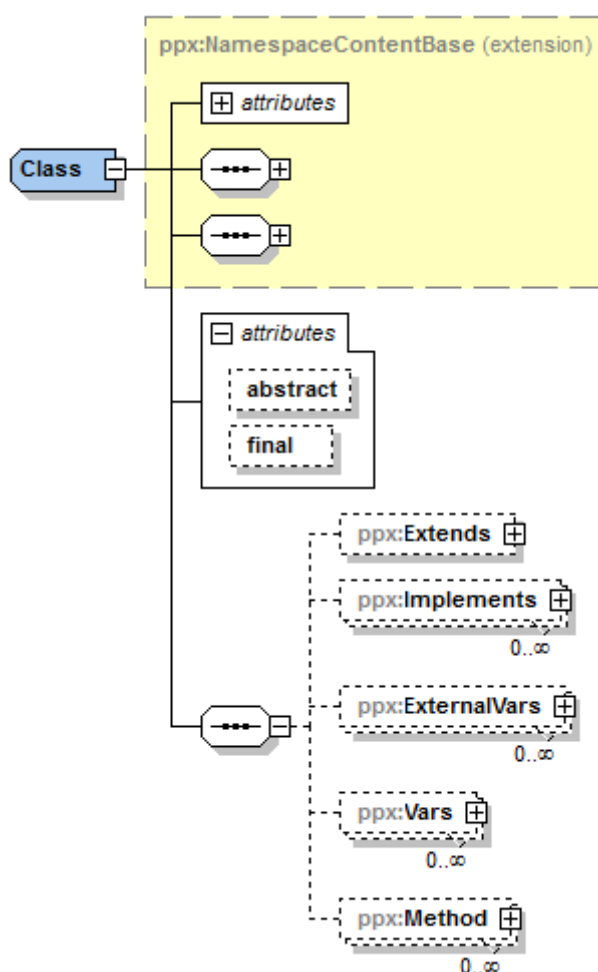


Figure 38 – Complex type “Class”

711 The attributes “abstract” and “final” represent whether the class has been defined using the
 712 keywords ABSTRACT and FINAL, as defined in IEC 61131-3.

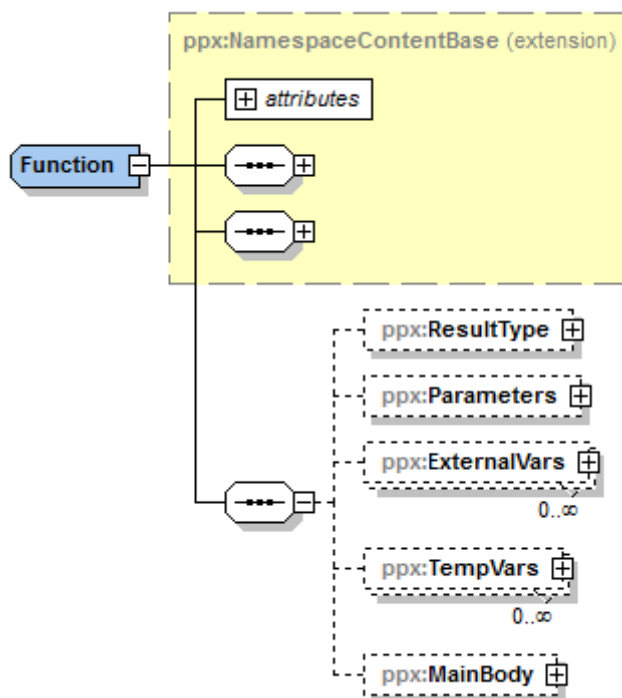
713 “Extends” contains the name of base class if applicable. “Implements” contains the
 714 implemented interfaces if applicable. Both are of type “typeRef” which is explained in the sub-
 715 clause 11.5.

716 “ExternalVars” of type “ExternalVarList” is explained in the sub-clause 11.2. “Vars” of type
 717 “VarListWithAccessSpec” is explained in the sub-clause 10.12.

718 “Method” of type “Method” is explained in the sub-clause 10.10.

719 **10.5 “Function”**

720 The complex type “Function” represents a function, as defined in IEC 61131-3. Its content is
 721 shown in Figure 39.



722
 723 **Figure 39 – Complex type “Function”**

724 “ResultType” contains the name of the data type being delivered by the function. “ResultType”
 725 of type “TypeRef” is explained in the sub-clause 11.5.

726 “Parameters” of type “ParameterSet” is explained in the sub-clause 10.11. “ExternalVars” of
 727 type “ExternalVarList” is explained in the sub-clause 11.2. “TempVars” of type “VarList” is
 728 explained in the sub-clause 11.1.

729 “MainBody” of type “Body” is explained in the sub-clause 10.14.

730 **10.6 “Interface”**

731 The complex type “Interface” represents an interface, as defined in IEC 61131-3. Its content is
 732 shown in Figure 40.

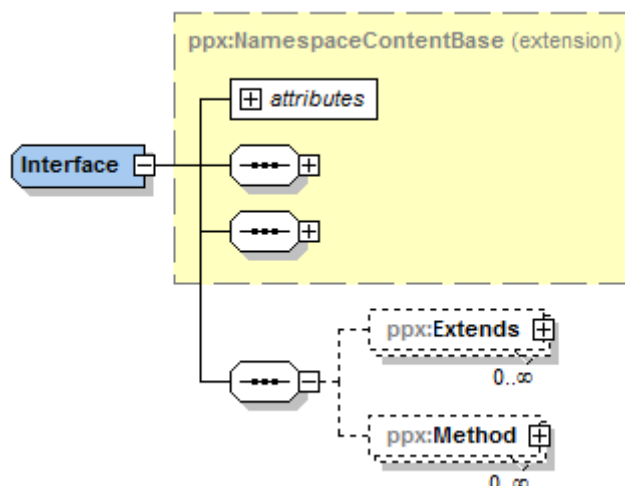


Figure 40 – Complex type “Interface”

“Extends” contains the name of base interfaces if applicable and is of type “typeRef” which is explained in the sub-clause 11.5.

“Method” of type “MethodPrototype” is explained in the sub-clause 10.9.

10.7 “Action”

The complex type “Action” represents an action, as defined in IEC 61131-3. Its content is shown in Figure 41.

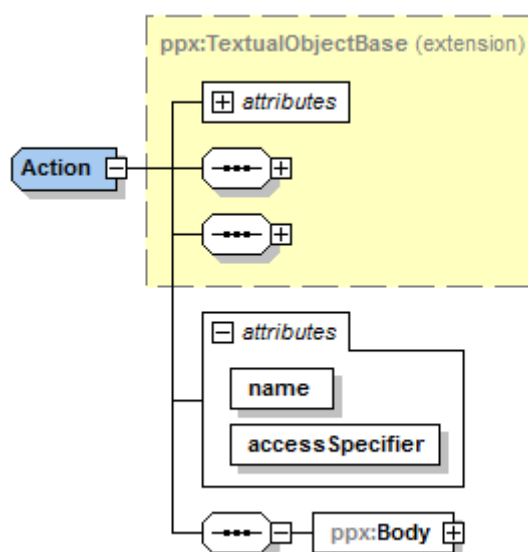


Figure 41 – Complex type “Action”

The attribute “name” of type “SimpleIdentifier” is explained in the sub-clause 15.6.

The attribute “accessSpecifier” of type “AccessSpecifiers” is explained in the sub-clause 10.13.

“Body” of type “Body” is explained in the sub-clause 10.14.

10.8 “NamedTransition”

The complex type “NamedTransition” represents a transition with a transition name, as defined in IEC 61131-3. Its content is shown in Figure 42.

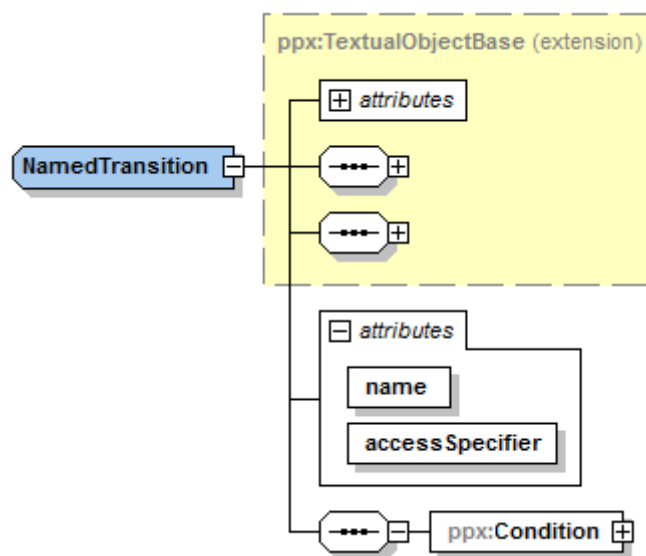


Figure 42 – Complex type “NamedTransition”

The attribute “name” of type “SimpleIdentifier” is explained in the sub-clause 15.6.

The attribute “accessSpecifier” of type “AccessSpecifiers” is explained in the sub-clause 10.13.

“Condition” of type “Predicate” is explained in the sub-clause 10.16.

10.9 “MethodPrototype”

The complex type “MethodPrototype” represents method prototype, as defined in IEC 61131-3. Its content is shown in Figure 43.

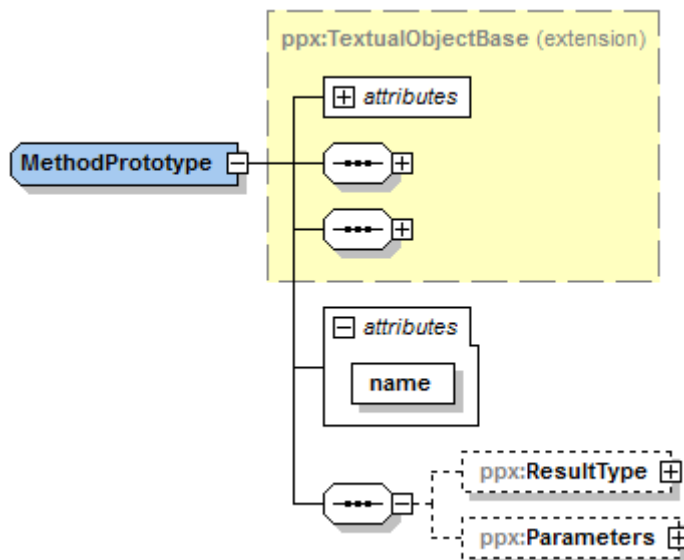


Figure 43 – Complex type “MethodPrototype”

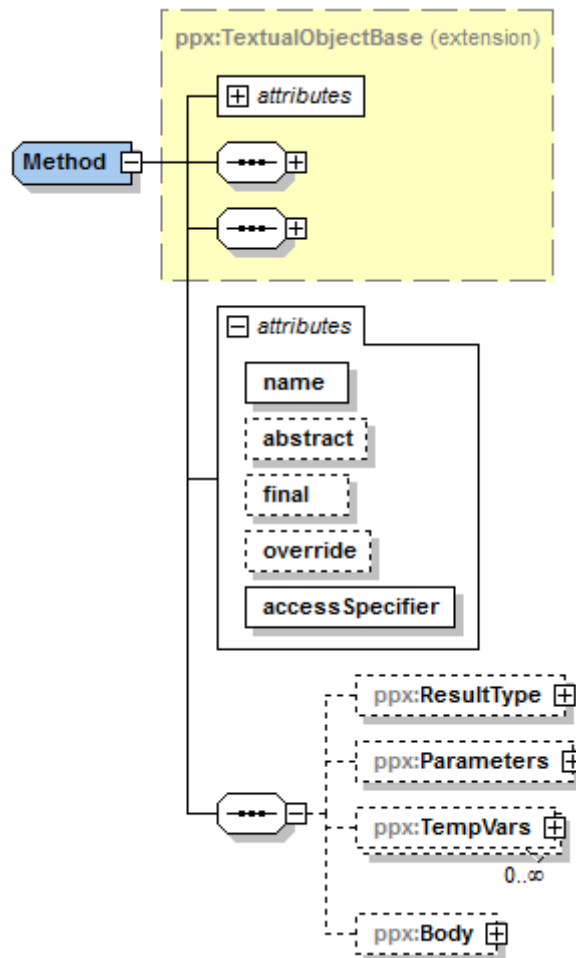
The attribute “name” of type “SimpleIdentifier” is explained in the sub-clause 15.6.

“ResultType” contains the name of the data type being delivered by the method. “ResultType” of type “TypeRef” is explained in the sub-clause 11.5.

“Parameters” of type “ParameterSet” is explained in the sub-clause 10.11.

765 **10.10 “Method”**

766 The complex type “Method” represents a method, as defined in IEC 61131-3. Its content is
 767 shown in Figure 44.



768
 769 **Figure 44 – Complex type “Method”**

770 The attribute “name” of type “SimpleIdentifier” is explained in the sub-clause 15.6.

771 The attributes “abstract”, “final” and “override” represent whether the method has been
 772 defined using the keywords ABSTRACT, FINAL and OVERRIDE, as defined in IEC 61131-3. If
 773 “abstract” is set to “true” then “Body” shall be omitted.

774 The attribute “accessSpecifier” of type “AccessSpecifiers” is explained in the sub-clause
 775 10.13.

776 “ResultType” contains the name of the data type being delivered by the method. “ResultType”
 777 of type “TypeRef” is explained in the sub-clause 11.5.

778 “Parameters” of type “ParameterSet” is explained in the sub-clause 10.11.

779 TempVars” of type “VarList” is explained in the sub-clause 11.1.

780 “Body” of type “Body” is explained in the sub-clause 10.14.

781 **10.11 “ParameterSet”**

782 The complex type “ParameterSet” represents parameters used for a function, function block,
 783 program, method prototype or method, as defined in IEC 61131-3, using the keywords
 784 VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT. Its content is shown in Figure 45.