# Interface Control Document

## Specification

| | |
|---|---|
| **Revision:** | 1.6 |
| **Status:** | Released |
| **Project:** | ELT Core Integration Infrastructure Software |
| **File:** | ICDSpecification.docx |
| **Document ID:** | CSL-DOC-17-147262 |
| **Owner:** | Damjan Kumar |
| **Last modification:** | June 10, 2019 |
| **Created:** | August 25, 2017 |

| Prepared by | Reviewed by | Approved by |
|---|---|---|
| Damjan Kumar, CSL SWE<br>Alexander Soderqvist, CSL SWE | Aljaž Podboršek<br>Gregor Čuk | Gregor Čuk |

# Document History

| Revision | Date | Changed/reviewed | Section(s) | Modification |
|---|---|---|---|---|
| 0.1 | 2017-10-01 | Damjan Kumar | All | First version. |
| 0.2 | 2017-10-26 | Damjan Kumar | 2 | Writing storage section. |
| 1.0 | 2017-10-30 | Damjan Kumar | All | Minor changes. |
| 1.1 | 2017-11-06 | Damjan Kumar | 2.2 | Added last sentence. |
| 1.2 | 2019-01-24 | Damjan Kumar | All | Updated according comments from MAL topical review. |
| 1.3 | 2019-02-25 | Damjan Kumar | 1.1.5, Appendix C | Allowing exception element without member elements. |
| | | | All | ICD types cannot be defined outside of a package. |
| | | | 1.1.8.1. | New chapter Reserved Keywords. |
| 1.4 | 2019-04-11 | Damjan Kumar | 1.1.9.1. | Array and Blob/Timestamp mapping updated. |
| 1.5 | 2019-04-13 | Damjan Kumar | 1 | Added info about ICD storage/import/export. |
| 1.6 | 2019-06-06 | Matej Šekoranja | All | Leading double-colon when referencing nonBasicType from other package required. |

# Confidentiality

This document is classified as a confidential document. As such, it or parts thereof must not be made accessible to anyone not listed in the Audience section, neither in electronic nor in any other form.

# Scope

This document is an Interface Control Document Specification for the ELT Core Integration Infrastructure Software project.

# Audience

All Cosylab and ESO employees involved with the ELT Core Integration Infrastructure Software.

# Table of Contents

# Figures

**No table of figures entries found.**

# Tables

# Listings

# References

[1] ESO, Core Integration Infrastructure Requirements Specification, ESO-192922 Version 2

[2] ESO, E-ELT Control System Architecture Concepts, ESO-283831 Version 1

[3] Cosylab, Data Addressing Specification, CSL-DOC-17-147262 Version 1.2

[4] XML Inclusions (XInclude), Version 1.0, https://www.w3.org/TR/xinclude/

[5] XML Pointer Language (XPointer), Version 1.0, https://www.w3.org/TR/WD-xptr

[6] XPointer Framework, Version 25 March 2003, https://www.w3.org/TR/2003/REC-xptr-framework-20030325/

[7] XPointer element () Scheme, Version 25 March 2003, https://www.w3.org/TR/2003/REC-xptr-element-20030325/

[8] XML Query, https://www.w3.org/XML/Query/

# Abbreviations

**Table 1: List of Abbreviations**

| Abbreviation | Term |
|---|---|
| ICD | Interface Control Document |
| URI | Uniform Resource Identifier |
| RDBMS | A relational database management system |

# 1. Interface Control Document Specification

Interface Control Document shall be encoded in a well-formed Extensible Markup Language (XML), a format that is both human-readable and machine-readable. In addition to being a well-formed, the Interface Control Document instance needs to be valid, adhering to the XML schema definition (XSD) provided for the ICD. The XML IDL was chosen over other IDL formats (Protobuf, OMG IDL) due its ability to get out-of-box integration with Eclipse (and other XML tools) with features as-you-type document validation and intelligent code completion on the elements and attributes enabling developers to create a valid ICD document without needing to compile an IDL. XML documents can be referenced and queried with a standard XQuery language ([8]). The XML IDL files represent an ICD Model. ICD files will be stored (archived) in a software versioning and revision control system (SVN) using commands for exporting ICD files (checkout) and importing them using update/commit commands.

Interface Control Document specification defines two types of XML documents [1], each with its own XML schema definition:

1. *Type definition document*, holding structured and interface types.
2. *Topic and interface definition document*, holding association of structured and interface types with the Publish/Subscribe or Request/Response Middlelayer mapping.

## 1.1. Type definition document

### 1.1.1. Structure type and a package

A basic building block in a type definition document is a structure type. A structure member can be a primitive type (uint8_t, string …) or other structure type. The structure type is always part of a package. A package can be nested in other packages.

Appendix F lists mappings between the ICD types and implementation types for Java and C++.

Listing 1 defines a relevant (partial) XSD schema for the structure and the package element representing a structure type and a package.

Table 2, Table 3 and Table 4 describe all attributes and elements that are part of the structure and the package element.

```
<xs:element name="types">
  <xs:complexType>
    <xs:sequence>
      <xs:attribute name="package" type="packageDecl" minOccurs="0" maxOccurs="1" />
      <xs:attribute name="include" type="includeDecl" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="packageDecl">
    <xs:group ref="packageElements" />
    <xs:attribute name="name" type="unqualifiedName" use="required" />
</xs:complexType>

<!-- only one non-empty (hierachical) package per module is allowed -->
<xs:group name="packageElements">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="struct" type="structDecl" minOccurs="0" />
            <xs:element name="enum" type="enumDecl" minOccurs="0" />
            <xs:element name="union" type="unionDecl" minOccurs="0" />
            <xs:element name="interface" type="interfaceDecl" minOccurs="0" />
            <xs:element name="exception" type="exceptionDecl" minOccurs="0" />
            <xs:element name="const" type="constDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>

<xs:complexType name="structDecl">
    <xs:sequence>
        <xs:element name="member" type="memberDecl" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="unqualifiedName" use="required" />
    <xs:attribute name="baseType" type="qualifiedName" use="optional" />
</xs:complexType>

<xs:complexType name="memberDecl">
    <xs:attribute name="name" type="unqualifiedName" use="required" />
    <xs:attribute name="type" type="basicType" use="required" />
    <xs:attribute name="nonBasicTypeName" type="qualifiedName" use="optional" />
     <xs:attribute name="key" type="trueFalseKind" use="optional" default="false" />
     <xs:attribute name="arrayDimensions" type="arrayDimensionsKind" use="optional" />
     <xs:attribute name="stringMaxLength" type="xs:integer" use="optional" />
     <xs:attribute name="mime-type" type="xs:string" use="optional" />
</xs:complexType>

<xs:simpleType name="unqualifiedName">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-zA-Z]+[a-zA-Z_0-9]*" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="qualifiedName">
    <xs:restriction base="xs:string">
        <xs:pattern value="((::)?[a-zA-Z]+[a-zA-Z_0-9]*)+" />
    </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="trueFalseKind">
    <xs:restriction base="xs:string">
       <xs:enumeration value="0" />
       <xs:enumeration value="1" />
       <xs:enumeration value="true" />
       <xs:enumeration value="false" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="arrayDimensionsKind">
    <xs:restriction base="xs:string">
       <xs:pattern value="[\[,\(]+[0-9]+(,[0-9]+)*[\],\)]+" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="basicType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="boolean" />
        <xs:enumeration value="int8_t" />
        <xs:enumeration value="uint8_t" />
        <xs:enumeration value="int16_t" />
        <xs:enumeration value="uint16_t" />
        <xs:enumeration value="int32_t" />
        <xs:enumeration value="uint32_t" />
        <xs:enumeration value="int64_t" />
        <xs:enumeration value="uint64_t" />
        <xs:enumeration value="float" />
        <xs:enumeration value="double" />
        <xs:enumeration value="string" />
        <xs:enumeration value="nonBasic" />
        <xs:enumeration value="timestamp" />
        <xs:enumeration value="blob" />
        <xs:enumeration value="void" />
    </xs:restriction>
</xs:simpleType>
```

**Listing 1: Relevant (partial) schema for Structure type and a package XSD (see Appendix A for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Package name | attribute | name="P1" |

**Table 2: package element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| `name` | Structure type name | attribute | name="someType" |
| `baseType` | Structure type which members are inherited | attribute | baseType="BarType" |
| `member` | List of member elements grouping basic and structured types under the same structure | element | <member name="a1" type="float"/> |

**Table 3: structure element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Structure member name | attribute | name="member1" |
| type | Member type (primitive type or "nonBasic" for structured types) | attribute | type="nonBasic"<br>type="uint8_t" |
| nonBasicTypeName | Names a structure type if the *type* attribute is "nonBasic" | attribute | nonBasicTypeName= "::M1::someType" |
| key | Mark member as part of a key | attribute | key="true" |
| stringMaxLength | Defines a max string length if the *type* attribute is a "string" | attribute | stringMaxLength="10" |
| arrayDimensions | If set, member is an array with the dimensions defined in the attribute | attribute | arrayDimensions="[4,4]"<br>arrayDimensions="[4,4,10]" |
| mime-type | Names a MIME-TYPE if the *type* attribute is "blob" | attribute | mime-type="text/css" |

**Table 4: member element**

Listing 2 is showing examples of the structure types and packages.

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

 <package name="typedef_example">

  <struct name= "Command1Data">
   <member name="a1" type="float"/>
   <member name="a2" type="float"/>
  </struct>

  <struct name="R_CII_401_1">
   <member name="foo" type="int8_t"/>
   <member name="bar" type="int16_t"/>
   <member name="baz" type="int32_t"/>
   <member name="qux" type="int64_t"/>
   <member name="bar1" type="uint8_t"/>
   <member name="baz1" type="uint16_t"/>
   <member name="qux1" type="uint32_t"/>
   <member name="qux1" type="uint64_t"/>
  </struct>

  <struct name="R_CII_401_2">
   <member name="foo" type="float"/>
   <member name="bar" type="double"/>
  </struct>

  <struct name="R_CII_401_3">
   <member name="isValid" type="boolean"/>
  </struct>

  <struct name="R_CII_401_4">
   <member name="qux" type="string"/>
  </struct>

  <struct name="R_CII_401_5">
     <member name="matrix" type="float" arrayDimensions="[4,4]"/>
     <member name="vector" type="float" arrayDimensions="[100]"/>
  </struct>

  <struct name="R_CII_401_6">
     <member name="matrix" type="float" arrayDimensions="[536870912,536870912]"/>
     <member name="vector" type="float" arrayDimensions="[1073741824]"/>
  </struct>

  <struct name="R_CII_401_7">
     <member name="binarydata" type="blob" mime-type="text/css"/>
  </struct>

  <struct name="R_CII_401_9">
    <member name="foo" type="timestamp"/>
  </struct>

   <struct name="R_CII_402">
     <member name="foo" type="float" key="true"/>
  </struct>

  <struct name="R_CII_403">
      <member name="foo" type="string" arrayDimensions="[100,100,100]"/>
  </struct>
```

```
<struct name="S1">
    <member name="a1" type="float"/>
    <member name="a2" type="float"/>
  </struct>

  <struct name="R_CII_404">
    <member name="s1" type="nonBasic" nonBasicTypeName="S1"/>
  </struct>

  <struct name="R_CII_405">
     <member name="a1" type="float"/>
     <member name="a2" type="float"/>
  </struct>

  <struct name="R_CII_405_1" baseType="R_CII_405">
     <member name="b1" type="float"/>
      <member name="b2" type="float"/>
  </struct>

 </package>

</types>
```

**Listing 2: Structure type and package example**

### 1.1.2. Enumeration type

The type definition document shall support an enumeration type. The enumeration type is part of a package. Structure members and the unions shall support enumeration types.

Listing 3 defines a relevant (partial) XSD schema for the enum element representing an enumerator type.

Table 5 and Table 6 describe all attributes and elements that are part of the enumeration element.

```
<xs:group name="packageElements">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="enum" type="enumDecl" minOccurs="0" />
            <xs:element name="struct" type="structDecl" minOccurs="0" />
            <xs:element name="union" type="unionDecl" minOccurs="0" />
            <xs:element name="interface" type="interfaceDecl" minOccurs="0" />
            <xs:element name="exception" type="exceptionDecl" minOccurs="0" />
            <xs:element name="const" type="constDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>

<xs:complexType name="enumDecl">
    <xs:sequence>
        <xs:element name="enumerator" type="enumeratorDecl" minOccurs="1"
                    maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="unqualifiedName" use="required" />
</xs:complexType>

<xs:complexType name="enumeratorDecl">
    <xs:attribute name="name" type="unqualifiedName" use="required" />
</xs:complexType>
```

**Listing 3: Relevant (partial) enum type XSD (see Appendix A for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Enumerator type name | attribute | name="Color" |
| enumerator | List of enumerator elements | element | <enumerator name="RED"/><br><enumerator name="BLUE"/> |

**Table 5: enum element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | explicitly named constants ("enumerator") | attribute | name="RED" |

**Table 6: enumerator element**

Listing 4 is showing an example of the enum type.

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

 <package name="typedef_example">

  <enum name="CommandID">
    <enumerator name="Command1"/>
    <enumerator name="Command2"/>
  </enum>

  <struct name="Foo">
     <member name="foo" type="nonBasic" nonBasicTypeName="CommandID"/>
  </struct>

 </package>

</types>
```

**Listing 4: Enumeration example**

### 1.1.3. Union type

The type definition document shall support a union type. The union type is part of a package. Structure members shall support union types.

Listing 5 defines a relevant (partial) XSD schema for the union element representing a union type.

Table 7, Table 8, Table 9 and Table 10 describe all attributes and elements that are part of the union type.

```
<xs:group name="packageElements">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="union" type="unionDecl" minOccurs="0" />
            <xs:element name="enum" type="enumDecl" minOccurs="0" />
            <xs:element name="struct" type="structDecl" minOccurs="0" />
            <xs:element name="interface" type="interfaceDecl" minOccurs="0" />
            <xs:element name="exception" type="exceptionDecl" minOccurs="0" />
            <xs:element name="const" type="constDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>

<xs:complexType name="unionDecl">
    <xs:sequence>
        <xs:element name="discriminator" type="discriminatorDecl"/>
        <xs:element name="case" type="caseDecl" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="unqualifiedName" use="required" />
</xs:complexType>

<xs:complexType name="discriminatorDecl">
    <xs:attribute name="type" type="basicType" use="required" />
    <xs:attribute name="nonBasicTypeName" type="qualifiedName" use="optional" />
</xs:complexType>

<xs:complexType name="caseDecl">
    <xs:sequence>
        <xs:element name="caseDiscriminator" type="caseDiscriminatorDecl" />
        <xs:element name="member" type="memberDecl" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="caseDiscriminatorDecl">
    <xs:attribute name="value" type="xs:string" use="required" />
</xs:complexType>
```

**Listing 5: Relevant (partial) union type XSD (see Appendix A for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Union type name | attribute | name="someUnionType" |
| discriminator | Union discriminator | element | <discriminator type="nonBasic" nonBasicTypeName="ColorEnum" /> |
| case | List of case elements | element | <case><br><caseDiscriminator value="RED" /><br><member name="redProperties "<br>type="string" /><br></case><br><case><br><caseDiscriminator value="BLUE " /><br><member name="blueProperties"<br>type="string" /><br></case> |

**Table 7: union element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| type | Discriminator type (primitive type or “nonBasic” for enumeration types) | attribute | type="nonBasic"<br>type="uint8_t" |
| nonBasicTypeName | Names an enumeration type if the *type* attribute is “nonBasic”<br>**NOTE: only enumeration types are supported.** | attribute | nonBasicTypeName= "someEnum" |

**Table 8: discriminator element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| caseDiscriminator | Discriminator union specific case | element | <caseDiscriminator value="BLUE" /> |
| member | Table 4 | element | <member name="BlueData" type="string" /> |

**Table 9: case element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| value | Discriminator value | attribute | value="RED" |

**Table 10: caseDiscriminator element**

Listing 6 is showing an example of the union type.

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

 <package name="union_example">

   <enum name="CommandID">
       <enumerator name="Command1" />
       <enumerator name="Command2" />
   </enum>

   <union name="CommandData">
       <discriminator type="nonBasic" nonBasicTypeName="CommandID" />
       <case>
           <caseDiscriminator value="Command1" />
           <member name="command1Data" type="string" />
       </case>
       <case>
           <caseDiscriminator value="Command2" />
           <member name="command2Data" type="string" />
       </case>
       <case>
           <caseDiscriminator value="default" />
           <member name="data" type="string" />
       </case>
   </union>

   <struct name="Command">
       <member name="command" type="nonBasic" nonBasicTypeName="CommandID" />
       <member name="data" type="nonBasic" nonBasicTypeName="CommandData" />
   </struct>

 </package>
</types>
```

**Listing 6: Union type example**

### 1.1.4. Interface type

The type definition document shall support an interface type. The interface type is part of a package. Interface type can publicly extend other interface types.

Listing 7 defines a relevant (partial) XSD schema for the interface element representing an interface type.

Table 11, Table 12, Table 13 and Table 14 describe all attributes and elements that are part of the interface type.

```
<xs:group name="packageElements">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="interface" type="interfaceDecl" minOccurs="0" />
            <xs:element name="union" type="unionDecl" minOccurs="0" />
            <xs:element name="enum" type="enumDecl" minOccurs="0" />
            <xs:element name="struct" type="structDecl" minOccurs="0" />
            <xs:element name="exception" type="exceptionDecl" minOccurs="0" />
            <xs:element name="const" type="constDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>

<xs:complexType name="interfaceDecl">
    <xs:sequence>
      <xs:element name="extends" type="extendsDecl" minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="method" type="methodDecl" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="unqualifiedName" use="required" />
</xs:complexType>

<xs:complexType name="extendsDecl">
  <xs:attribute name="interfaceName" type="unqualifiedName" use="required" />
</xs:complexType>

<xs:complexType name="methodDecl">
    <xs:sequence>
        <xs:element name="argument" type="argumentDecl" minOccurs="0"
                    maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="unqualifiedName " use="optional" />
    <xs:attribute name="throws" type="qualifiedName" use="optional" />
    <xs:attribute name="returnType" type="basicTypeOrVoid" use="required" />
    <xs:attribute name="methodType" type="methodKind" use="optional" />
    <xs:attribute name="nonBasicReturnTypeName" type="qualifiedName" use="optional" />
    <xs:attribute name="arrayDimensions" type="arrayDimensionsKind" use="optional"/>
</xs:complexType>

<xs:complexType name="argumentDecl">
    <xs:attribute name="name" type="unqualifiedName" use="required" />
    <xs:attribute name="type" type="basicType" use="required" />
    <xs:attribute name="nonBasicTypeName" type="qualifiedName" use="optional" />
    <xs:attribute name="arrayDimensions" type="arrayDimensionsKind" use="optional"/>
</xs:complexType>

<xs:simpleType name="methodKind">
    <xs:restriction base="xs:string">
        <xs:enumeration value="ami" />
        <xs:enumeration value="two-way" />
    </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="basicTypeOrVoid">
    <xs:restriction base="xs:string">
        <xs:enumeration value="boolean" />
        <xs:enumeration value="int8_t" />
        <xs:enumeration value="uint8_t" />
        <xs:enumeration value="int16_t" />
        <xs:enumeration value="uint16_t" />
        <xs:enumeration value="int32_t" />
        <xs:enumeration value="uint32_t" />
        <xs:enumeration value="int64_t" />
        <xs:enumeration value="uint64_t" />
        <xs:enumeration value="float" />
        <xs:enumeration value="double" />
        <xs:enumeration value="string" />
        <xs:enumeration value="nonBasic" />
       <xs:enumeration value="timestamp" />
        <xs:enumeration value="blob" />
        <xs:enumeration value="void" />
    </xs:restriction>
</xs:simpleType>
```

**Listing 7: Relevant (partial) interface type XSD (see Appendix A for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Interface name | attribute | name="LocalControlInterface" |
| extends | List of extends interfaces from which the interface publicly inherits its methods. | element | <extends interfaceName="ControlInterface"/><br><extends interfaceName="LogInterface"/> |
| method | List of interface methods declarations | element | <method name="foo" returnType="uint8_t "<br>throws="TooFastException"<br>methodType="ami"><br><argument name="bar" type="boolean" /><br><argument name="zoo" type="uint16_t"/><br></method> |

**Table 11: interface element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Method name | attribute | Table 4 (method) |
| throws | Exception type name thrown | attribute | Table 4 (method) |
| returnType | Return type name | attribute | Table 4 (method) |
| methodType | Method type (i.e. AMI for multiple replies)<br><br>**Note: if not specified, AMI type is presumed.** | attribute | Table 4 (method) |
| nonBasicReturnTypeName | if a *returnType* is "nonBasic", this names a structured return type name | attribute | <method name="foo" returnType="nonBasic" nonBasicReturnTypeName="Bar"> </method> |
| argument | List of method arguments | element | Table 4 (method) |
| arrayDimensions | if a method is returning an array, it defines array size | | <method name="foo" returnType="uint8_t" arrayDimensions="(16)"> </method> |

**Table 12: method element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Argument name | attribute | name="arg0 " |
| type | Argument type | attribute | type="boolean" |
| nonBasicTypeName | if *type* is "nonBasic", this names the structured type name | attribute | nonBasicTypeName="Bar" |
| arrayDimensions | if parameter is an array, it defines parameter array size | | arrayDimensions="(16)" |

**Table 13: argument element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| interfaceName | Interface type name from which the methods will be inherited. | attribute | interfaceName="::Common::ControlInterface" |

**Table 14: argument element**

Listing 8 is showing an example of the interface type.

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

 <package name="typedef_example">

  <interface name="LogInterface">
    <method name="log" returnType="void">
     <argument name="category" type="string" />
    <argument name="message" type="string" />
   </method>
  </interface>

  <interface name="SomeControlSystem">
   <extends interfaceName="LogInterface"/>

   <method name="foo" returnType="string">
    <argument name="bar" type="int8_t" />
    <argument name="zoo" type="uint16_t"/>
   </method>
```

```
    <method name="fooArray" returnType="uint8_t" arrayDimensions="(16)">
     <argument name="barArrayParam" type="string " arrayDimensions="(5)" />
    </method>

   </interface>

  </package>
</types>
```

**Listing 8: Interface type example**

### 1.1.5. Exception type

The type definition document shall support an exception type. The exception type is part of a package.

Listing 9 defines a relevant (partial) XSD schema for the exception element representing an exception type.

Table 15 describes all attributes and elements that are part of the exception type.

```
<xs:group name="packageElements">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="exception" type="exceptionDecl" minOccurs="0" />
            <xs:element name="interface" type="interfaceDecl" minOccurs="0" />
            <xs:element name="union" type="unionDecl" minOccurs="0" />
            <xs:element name="enum" type="enumDecl" minOccurs="0" />
            <xs:element name="struct" type="structDecl" minOccurs="0" />
            <xs:element name="const" type="constDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>


<xs:complexType name="exceptionDecl">
    <xs:sequence>
      <xs:element name="member" type="memberDecl" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="qualifiedName" use="required" />
</xs:complexType>
```

**Listing 9: Relevant (partial) exception type XSD (see Appendix A for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Exception type name | attribute | name="SomeException" |
| member | Table 4 (optional) | element | Table 4 (member) |

**Table 15: exception element**

Listing 10 is showing an example of an exception type.

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

 <package name="typedef_example">

  <exception name="TooFastException">
   <member name="what" type="string"/>
  </exception>

  <interface name="SomeControlSystem">

   <method name="foo" returnType="nonBasic" nonBasicReturnTypeName="R_CII_405"
           throws="TooFastException"/>
  </interface>

 </package>
</types>
```

**Listing 10: Exception type example**

### 1.1.6. External type definition inclusion

It shall be possible to include other ICDs into the target ICD as a reference (a generated include) or inserting a textual fragment into an ICD using XInclude [4] and XPointer [5].

File names in the include href attribute are relative to the include paths given to the ICD generator or have absolute paths.

**NOTE:** Java parser's XInclude implementation (Apache Xerces2) supports the XPointer Framework [6] and the XPointer element () Scheme [7]. The XPointer xpointer () Scheme is currently not supported.

**NOTE:** Support for types of IDs in XPointer: For shorthand pointers and element () XPointers, currently only DTD-determined IDs are supported. Schema-determined IDs may be supported in a future release.

Listing 11 defines a relevant (partial) ICD XSD schema for the include element, XInclude schema is defined in Appendix E.

```
<xs:element name="types">
  <xs:complexType>
    <xs:sequence>
          <xs:attribute name="include" type="includeDecl" minOccurs="0"
                        maxOccurs="unbounded" />
      <xs:attribute name="package" type="packageDecl" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="includeDecl">
   <xs:attribute name="href" type="xs:string" use="required" />
</xs:complexType>
```

**Listing 11: Relevant (partial) include XSD (see Appendix A for full schema)**

Listing 12 lists an example of how to include externally defined interfaces and import text fragments using XInclude and XPointer.

**test_common.xml:**

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

 <package name="test">
   <package name="common">
     <struct name="OneForAll">
       <member name="jokeOfTheDay" type="string"/>
     </struct>
 </package>
</types>
```

**common_interface.xml:**

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

 <package name="common">

  <interface name="LogInterface">
    <method name="log" returnType="void">
     <argument name="category" type="string" />
     <argument name="message" type="string" />
   </method>
  </interface>
 </package>
</types>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd"
       xmlns:xi="http://www.w3.org/2001/XInclude">

<include href="common_interface.xml" />

<package name="typedef_example">

  <!-- from test_common.xml include struct OneForAll -->
  <xi:include href="test_common.xml" xpointer="element(/1/1/1/1)"/>

  <interface name="SomeControlSystem">
   <extends interfaceName="::common::LogInterface"/> <!-- from common_interface.xml -->
   <method name="foo" returnType="uint8_t”/>
  </interface>

 </package>
</types>
```

**Listing 12: External ICD inclusion example**

### 1.1.7. Constant

It shall be possible to define constant values of primitive types. Constant values can be used in the elements and attributes where a primitive value would be used.

Listing 13 defines a relevant (partial) XSD schema for the constant element.

Table 16 describes all attributes that are part of the constant type.

```
<xs:group name="packageElements">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="const" type="constDecl" minOccurs="0" />
            <xs:element name="exception" type="exceptionDecl" minOccurs="0" />
            <xs:element name="interface" type="interfaceDecl" minOccurs="0" />
            <xs:element name="union" type="unionDecl" minOccurs="0" />
            <xs:element name="enum" type="enumDecl" minOccurs="0" />
            <xs:element name="struct" type="structDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>

<xs:complexType name="constDecl">
    <xs:attribute name="name" type="unqualifiedName" use="required" />
    <xs:attribute name="type" type="basicType" use="required" />
    <xs:attribute name="value" type="xs:string" use="required" />
</xs:complexType>
```

**Listing 13: Relevant (partial) constant XSD (see Appendix A for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Constant name | attribute | name="USERNAME" |
| type | Constant type (only primitive types are supported) | attribute | type="string" |
| value | Constant value | attribute | value="Batman" |

**Table 16: const element**

Listing 14 lists an example of how to define constants.

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd"
       xmlns:xi="http://www.w3.org/2001/XInclude">

 <package name="m1">
  <package name="m2">

   <const name="BOOLCONSTANT" type="boolean" value="true"/>
   <const name="INT8CONSTANT" type="int8_t" value="1"/>
   <const name="FLOATSOMECONSTANT" type="float" value="6.4523424"/>
   <const name="DOUBLECONSTANT" type="float" value="6.433523424"/>
   <const name="STRINGCONSTANT" type="float" value="one"/>
  </package>
 </package>
</types>
```

**Listing 14: Constant example**

### 1.1.8. Type definition constraints

Types defined in ICD have the following constraints (Table 17):

| Type | Constraint |
|---|---|
| array | Array is bounded or fixed. Unbounded array is not allowed. |
| array | Multi-dimension array is flattened, **row-major** order. |
| array | Zero or negative dimension is not allowed. |
| array | flat array size > MAX_UINT32 is not allowed. |
| package | Only one non-empty (hierarchical) package per module is allowed. |
| structure | Extending structure means COPY from extended (including the keys). |
| structure | Only primitives and enums can be keys. |
| interface | Extending interface means COPY from extended. |
| exception | Exception type cannot publicly extend other exception types. |
| union | Only numeric primitives and enums can be union discriminators. |
| structure, interface, exception, union, enum | Duplicate type definition (case-insensitive name) in the same package is not allowed. |

**Table 17: Type constraints**

#### 1.1.8.1. Reserved Keywords

ICD must not contain any of the C++, Java and Python reserved keywords.

https://en.cppreference.com/w/cpp/keyword

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

https://docs.python.org/3/reference/lexical_analysis.html#keywords

In addition the **following keywords are not allowed** too (MAL restriction):

- getMal,
- close,
- asyncConnect,
- isConnected,
- registerConnectionListener,
- unregisterConnectionListener,
- timeout,
- typeHash,
- clazz,
- clone,
- cloneKey,
- hasKey,
- keyEquals,
- keyHash.

### 1.1.9. ICD Type mappings

ICD types will be mapped to the agnostic implementation types as follows.

#### 1.1.9.1. Primitive types mapping

ICD primitive types mappings are defined in Table 18:

| ICD TYPE | Java Type | Java Boxed Type | C++ Type |
|---|---|---|---|
| boolean | boolean | Boolean | bool |
| int8_t | byte | Byte | int8_t |
| int16_t | short | Short | int16_t |
| int32_t | int | Integer | int32_t |
| int64_t | long | Long | int64_t |
| uint8_t | byte | Byte | uint8_t |
| uint16_t | short | Short | uint16_t |
| uint32_t | int | Integer | uint32_t |
| uint64_t | long | Long | uint64_t |
| float | float | Float | float |
| double | double | Double | double |
| string | String | String | std::string |
| fixed size array | java.util.List<BoxedType> | java.util.List<BoxedType> | elt::mal::shared_vector< CppType> for primitives, std::vector<CppType> otherwise |
| array | java.util.List<BoxedType> | java.util.List<BoxedType> | elt::mal::shared_vector< CppType> for primitives, std::vector<CppType> otherwise |
| blob | elt.mal.types.Blob | elt.mal.types.Blob | elt::mal::types::Blob |
| timestamp | elt.mal.types.Timestamp | elt.mal.types.Timestamp | elt::mal::types::Timestamp |
| void | void | java.lang.Void | void |

**Table 18: ICD Primitive Type mappings**

#### 1.1.9.2. Enum mapping

ICD Enum type (Listing 20) maps to:

1. Java type: enum with ICD enumerators as java enum constants (Listing 21).
2. C++ type: unscoped C++ enumeration with ICD enumerators as enumerators (Listing 22).

#### 1.1.9.3. Union mapping

ICD Union type (Listing 28) maps to:

1. Java type: interface extending *elt.mal.ps.DataEntity* with the setters/getters for ICD caseDiscriminators and *getDiscriminator()* method for current discriminator (Listing 29).
2. C++ type: class extending elt::mal::ps::DataEntity with pure virtual methods for ICD caseDiscriminators, pure virtual method *getDiscriminator()* for current discriminator, virtual destructor and a public constant TYPE_ID for the textual name of the type (Listing 30).

#### 1.1.9.4. Interface mapping

ICD Interface type (Listing 37) maps to:

1. Java type:
    a. Interface extending *elt.mal.rr.RrEntity* with methods declared as ICD interface methods (Listing 38).
    b. Client's synchronous interface (interface name ending with postfix "Sync") extending interface from a) and *elt.mal.rr.ClientRrEntity* (Listing 39).
    c. Server's asynchronous interface (interface name prefixed with "Async") extending *elt.mal.rr.RrEntity* with methods declared as ICD interface methods returning *java.util.concurrent.CompletableFuture* for ICD interface method return type (Listing 40).
    d. Client's asynchronous interface (interface name ending with postfix "Async") extending interface from c) and *elt.mal.rr.ClientRrEntity* (Listing 41).
2. C++ type:
    a. Class extending *elt::mal::rr::RrEntity* with methods declared as ICD interface methods (Listing 42).
    b. Client's synchronous class (class name ending with postfix "Sync") extending class from a) and *elt::mal::rr::ClientRrEntity*, holding pure virtual method *timeout* for setting request timeouts (Listing 42).
    c. Server's asynchronous class (class name prefixed with "Async") extending *elt::mal::rr::RrEntity* with methods declared as ICD interface methods returning *elt::mal::future* for ICD interface method return type (Listing 42).

3. Client's asynchronous class (class name ending with postfix "Async") extending interface from 2.c) and *elt::mal::rr::ClientRrEntity* (Listing 42).

#### 1.1.9.5. Exception mapping

ICD Exception type (Listing 31) maps to:

1. Java type: class extending *java.lang.Exception* with the getters for the ICD exception members and a constructor with the ICD exception members as arguments (Listing 32).
2. C++ type: class extending *std::exception* and containing public data members for ICD exception members. (Listing 33).

#### 1.1.9.6. Package mapping

ICD Package type (Listing 34) maps to:

1. Java package (Listing 35).
2. C++ namespace (Listing 36).

#### 1.1.9.7. Structure mapping

ICD Structure type (Listing 23) maps to:

1. Java type: interface extending *elt.mal.ps.DataEntity* and the getters/setters for the ICD structure members (Listing 24, Listing 25).
2. C++ type: class extending elt::mal::ps::DataEntity with pure virtual methods for ICD Structure members, virtual destructor and a public constant TYPE_ID for the textual name of the type (Listing 26, Listing 27).

#### 1.1.9.8. Constant mapping

ICD Constant (Listing 43) maps to:

1. Java type: interface with public static data members as ICD constants (Listing 44).
2. C++ type: static constexpr data members of a namespace (Listing 45).

## 1.2. Topic and interface definition document

The topic and interface definitions shall specify (per subsystem):

- Messaging pattern:
  - Publish/Subscribe for topics
  - Request/Reply for interfaces.
- Quality of Service parameters
- Performance characteristics

- Type and interface names used (defined in the type definition document)
- Middleware mapping (**only element names for supported mappings are defined in the document**)
- Address URI ([3])

### 1.2.1. Subsystem

Topic and interface definitions are part of a subsystem. There shall be possible to define many subsystems per document.

Listing 15 defines a relevant (partial) XSD schema for the subsystem.

Table 19 describes all attributes and elements that are part of the subsystem.

```
<xs:element name="topics">
    <xs:complexType>
        <xs:group ref="subsystemElements" />
    </xs:complexType>
</xs:element>


<xs:group name="subsystemElements">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="subsystem" type="subsystemDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>


<xs:complexType name="subsystemDecl">
    <xs:group ref="topicInterfaceDefinition" />
    <xs:attribute name="name" type="identifierName" use="required" />
</xs:complexType>


<xs:group name="topicInterfaceDefinition">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="pubsub_topic" type="pubsubDecl" minOccurs="0" />
            <xs:element name="service" type="serviceDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>
```

**Listing 15: Relevant (partial) subsystem definition XSD (see Appendix D for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| `name` | Subsystem name | attribute | name="SomeSubsystem" |
| `pubsub_topic` | List of Publish/Subscribe topic definition elements | element | Table 20 |
| `service` | List of Request/Reply interface definition elements (services) | element | Table 24 |

**Table 19: subsystem element**

### 1.2.2. Topic definition

Topic and interface definition document shall define a topic definition.

Listing 16 defines a relevant (partial) XSD schema for the topic definition.

Table 20, Table 21, Table 22 and Table 23 describe all attributes and elements that are part of the topic definition.

```
<xs:group name="topicInterfaceDefinition">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="pubsub_topic" type="pubsubDecl" minOccurs="0" />
            <xs:element name="service" type="serviceDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>


<xs:complexType name="pubsubDecl">
    <xs:all>
        <xs:element name="topic_name" minOccurs="1" />
        <xs:element name="topic_type" minOccurs="1" />
        <xs:element name="address_uri" type="xs:anyURI" minOccurs="1" />
        <xs:element name="qos" type="qosPubSubDecl" minOccurs="0" />
        <xs:element name="performance" type="performanceDecl" minOccurs="0" />
        <xs:element name="mal" type="malDecl" minOccurs="0" />
    </xs:all>
</xs:complexType>


<xs:complexType name="qosPubSubDecl">
    <xs:attribute name="latency_ms" type="float" use="optional" />
    <xs:attribute name="deadline_ms" type="float" use="optional" />
</xs:complexType>


<xs:complexType name="performanceDecl">
    <xs:attribute name="rate_hz" type="xs:string" use="optional" />
    <xs:attribute name="latency_ms" type="xs:string" use="optional" />
    <xs:attribute name="synchronous" type="xs:string" use="optional" />
</xs:complexType>


<xs:complexType name="malDecl">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="dds"  minOccurs="1" />
            <xs:element name="zpb"  minOccurs="1" />
            <xs:element name="opc"  minOccurs="1" />
            <xs:element name="mudpi"  minOccurs="1" />
        </xs:choice>
    </xs:sequence>
</xs:complexType>
```

**Listing 16: Relevant (partial) topic definition XSD (see Appendix D for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| topic_name | Topic name | element | <topic_name>Az:Pos_Actual</topic_name> |
| topic_type | Topic type (structure type name defined in the type definition document) | element | <topic_type>Pos_Actual</topic_type> |
| address_uri | MAL address URI [3] | element | <address_uri><br>dds.ps://m1/Pos_Actual</address_uri> |
| qos | Quality of service | element | <qos latency_ms="0.1" deadline_ms="20"/> |
| performance | Performance meta data (performance description is in [2], 11.1.1) | element | <performance rate_hz="0.2" latency_ms="25" synchronous="false"/> |
| mal | Contains MAL provider specific elements | element | <mal><br><dds><br><!—specific elements for DDS--><br></dds><br></mal> |

**Table 20: pubsub_topic element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| latency_ms | The latency is the maximum time a sample may remain in-transit between the publisher and subscriber. That is, the arrival time - sample time cannot be greater than latency QoS. | attribute | latency_ms="0.1" |
| deadline_ms | The deadline is the maximum time between samples, i.e. 1/ (sample frequency in Hz). | attribute | deadline_ms="20" |

**Table 21: qos element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| rate_hz | Command rate in Hz | attribute | rate_hz="0.2" |
| latency_ms | Maximum age of data in seconds | attribute | latency_ms="25" |
| synchronous | Is data communication connected to absolute time or not | attribute | synchronous="false" |

**Table 22: performance element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| dds | DDS specific MAL provider settings | element | Not part of this document. |
| zpb | ZeroMQ/ProtocolBuffer specific MAL provider settings | element | Not part of this document. |
| opc | OPC/UA specific MAL provider settings | element | Not part of this document. |
| mudapi | ESO MUDAPI specific MAL provider settings | element | Not part of this document. |

**Table 23: mal element**

Listing 17 lists an example of a topic definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<topics xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="schemas/icd_topic_definition.xsd">

    <subsystem name="M1">

        <pubsub_topic>
            <topic_name>Az:Pos_Actual</topic_name>
            <topic_type>Pos_Actual</topic_type>
            <address_uri>dds.ps://m1/Pos_Actual?p1=v1&amp;amp;p2=v2</address_uri>
            <qos latency_ms="0.1" deadline_ms="20"/>
            <performance rate_hz="0.2" latency_ms="25" synchronous="false"/>
            <mal>
                <dds/>
            </mal>
        </pubsub_topic>

    </subsystem>

</topics>
```

**Listing 17: Topic definition example**

### 1.2.3. Interface definition

Topic and interface definition document shall define an interface definition (a service).

Listing 18 defines a relevant (partial) XSD schema for the interface definition.

Table 24, Table 25 and Table 26 describe all attributes and elements that are part of the interface definition (a service).

```
<xs:group name="topicInterfaceDefinition">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="service" type="serviceDecl" minOccurs="0" />
            <xs:element name="pubsub_topic" type="pubsubDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:group>


<xs:complexType name="serviceDecl">
    <xs:all>
        <xs:element name="address_uri" type="xs:anyURI" minOccurs="1" />
        <xs:element name="is_configuration" type="trueFalseKind" minOccurs="0" />
        <xs:element name="performance" type="performanceDecl" minOccurs="0" />
        <xs:element name="mal" type="malDecl" minOccurs="1" />
        <xs:element name="qos" type="qosServiceDecl" minOccurs="0" />
        <xs:element name="methods" type="methodsDecl" minOccurs="0" />
    </xs:all>
    <xs:attribute name="name" type="identifierName" use="required" />
    <xs:attribute name="interface" type="identifierName" use="required" />
</xs:complexType>


<xs:complexType name="methodsDecl">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="method" type="methodDecl" minOccurs="0" />
        </xs:choice>
    </xs:sequence>
</xs:complexType>


<xs:complexType name="methodDecl">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
            <xs:element name="qos" type="qosServiceDecl" minOccurs="0"/>
            <xs:element name="performance" type="performanceDecl" minOccurs="0" />
            <xs:element name="is_configuration" type="trueFalseKind" minOccurs="0"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="identifierName" use="optional" />
</xs:complexType>
```

**Listing 18: Relevant (partial) service definition XSD (see Appendix D for full schema)**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | Service instance name | attribute | name="SomeServiceInstanceName" |
| interface | Interface type name | attribute | interface="Common::Control" |
| address_uri | MAL address URI [3] | element | <address_uri>zpb.rr://m4lsv.pl.eso.org:5002</address_uri> |
| is_configuration | The configuration has no defined performance constraints and represents aspects of the Local Control System software which must be controllable externally from the CCS. | element | <is_configuration>true</is_configuration> |
| qos | Quality of service | element | <qos connection_timeout_sec="0.1" reply_timeout_sec="20" /> |
| performance | Performance meta data | element | <performance rate_hz="0.2" latency_ms="25" synchronous="false"/> |
| mal | Contains MAL provider specific elements | element | <mal><zpb><!—specific elements for ZeroMQ/Protocol buffers--></zpb></mal> |
| methods | List of method overrides. A method can override a | element | |

| | |
|---|---|
| | performance, a qos and a is_configuration. |

**Table 24: service element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| connection_timeout_sec | Connection timeout in seconds | attribute | connection_timeout_sec="0.1" |

**Table 25: service qos element**

| Attribute/Element | Description | XML markup | Example |
|---|---|---|---|
| name | method name | attribute | name="GetLogLevel" |
| performance | Overriding method performance | element | Table 22 |
| qos | Overriding method qos | element | Table 21 |
| is_configuration | Overriding configuration method marker | element | <is_configuration>true </is_configuration> |

**Table 26: method element**

Listing 19 lists an example of an interface definition (a service).

```
<?xml version="1.0" encoding="UTF-8"?>
<topics xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="schemas/icd_topic_definition.xsd">

    <subsystem name="M1">

        <service name="MSA1" interface="MSAzimuth" >
            <address_uri>zpb.rr://m4lsv.pl.eso.org:5002</address_uri>
            <performance rate_hz="0.2" latency_ms="25" synchronous="false"/>
            <qos connection_timeout_sec="0.1" reply_timeout_sec="20" />
            <mal>
               <zpb/>
            </mal>
            <methods>
               <method name="SomeMethodThatIsPartOfMSAzimuth">
                  <performance rate_hz="10" latency_ms="2" synchronous="false"/>
                  <qos connection_timeout_sec="0.1" reply_timeout_sec="20" />
               </method>
               <method name="GetLogLevel">
                   <is_configuration>true</is_configuration>
               </method>
            </methods>
        </service>

   </subsystem>

</topics>
```

**Listing 19: Interface definition (a service) example**

# Appendix A. Examples of Type Definitions

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd"
       xmlns:xi="http://www.w3.org/2001/XInclude">


 <include href="common_interface.xml"/>

 <package name="typedef_example">

  <!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
                   Enumerator -->

  <enum name="CommandID">
    <enumerator name="Command1"/>
    <enumerator name="Command2"/>
  </enum>

  <!-- [R-CII-400] Type definitions shall express structured types.  -->

  <struct name= "Command1Data">
   <member name="a1" type="float"/>
   <member name="a2" type="float"/>
  </struct>

  <struct name= "Command2Data">
   <member name="b1" type="float"/>
   <member name="b2" type="float"/>
  </struct>
```

```
<!-- [R-CII-1140] Type definitions shall ideally express union types. -->

<struct name= "DateStructure">
 <member name="day" type="int16_t"/>
 <member name="month" type="int16_t"/>
 <member name="year" type="int16_t"/>
</struct>

<union name="Date">
 <discriminator type="int16_t"/>
 <case>
  <caseDiscriminator value="1"/>
  <member name="stringFormat" type="string"/>
 </case>
 <case>
  <caseDiscriminator value="2"/>
  <member name="digitalFormat" type="int32_t"/>
 </case>
 <case>
  <caseDiscriminator value="default"/>
  <member name="structFormat" type="nonBasic"
          nonBasicTypeName= "typedef_example::DateStructure"/>
 </case>
</union>

<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
       1-byte, 2-byte, 4-byte and 8-byte signed or unsigned Integer  -->

<struct name="R_CII_401_1">
 <member name="foo" type="int8_t"/>
 <member name="bar" type="int16_t"/>
 <member name="baz" type="int32_t"/>
 <member name="qux" type="int64_t"/>
 <member name="bar1" type="uint8_t"/>
 <member name="baz1" type="uint16_t"/>
 <member name="qux1" type="uint32_t"/>
 <member name="qux1" type="uint64_t"/>
</struct>

<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
                 Single or double precision floating point number. -->

<struct name="R_CII_401_2">
 <member name="foo" type="float"/>
 <member name="bar" type="double"/>
</struct>

<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
                 Boolean -->

<struct name="R_CII_401_3">
 <member name="isValid" type="boolean"/>
</struct>
```

```
<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
                 Character String -->

<struct name="R_CII_401_4">
 <member name="qux" type="string"/>
</struct>


<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
       Multidimensional Vector or 2-D Matrix The Vector or Matrix shall be
       accompanied with associated dimension information. -->

<struct name="R_CII_401_5">
  <member name="matrix" type="float" arrayDimensions="[4,4]"/>
  <member name="vector" type="float" arrayDimensions="[100]"/>
</struct>


<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
      Vectors and Matrices up to 4 GB in size(uncompressed) shall be
      handled. -->

<struct name="R_CII_401_6">
 <member name="matrix" type="float"
         arrayDimensions="[536870912,536870912]"/>
  <member name="vector" type="float" arrayDimensions="[1073741824]"/>
</struct>

<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
      Binary (e.g. LONGBLOB) Binary data will be accompanied with associated
      MIME type info. -->

<struct name="R_CII_401_7">
 <member name="binarydata" type="blob" mime-type="text/css"/>
</struct>


<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
       Binary data up to 4 GB in size shall be handled -->

<struct name="R_CII_401_8">
 <member name="binarydata" type="blob"
         mime-type="application/octet-stream"/>
</struct>

<!-- [R-CII-401] Type definitions shall express the CII Basic Data Type Set.
       Time Date/Timestamp Microsecond precision shall be supported. -->

<struct name="R_CII_401_9">
  <member name="foo" type="timestamp"/>
</struct>
```

```
<!-- [R-CII-402] Type definitions shall support declaration of keys. -->

<struct name="R_CII_402">
   <member name="foo" type="float" key="true"/>
</struct>

<!-- [R-CII-403] Type definitions shall support multidimensional types. -->

<struct name="R_CII_403">
 <member name="foo" type="string" arrayDimensions="[100,100,100]"/>
</struct>

<!-- [R-CII-404] Type definitions shall support inclusion of other type
                 definitions as members. -->

<struct name="S1">
  <member name="a1" type="float"/>
  <member name="a2" type="float"/>
</struct>

<struct name="S2">
  <member name="b1" type="float"/>
  <member name="b2" type="float"/>
</struct>

<struct name="R_CII_404">
 <member name="s1" type="nonBasic" nonBasicTypeName="S1"/>
 <member name="s2" type="nonBasic" nonBasicTypeName="S2"/>
</struct>

<!-- [R-CII-405] Type definitions shall be able to extend other type
                 definitions. -->

<struct name="R_CII_405">
 <member name="a1" type="float"/>
 <member name="a2" type="float"/>
</struct>

<!-- extending only structures -->

<struct name="R_CII_405_1" baseType="typedef_example::R_CII_405">
 <member name="b1" type="float"/>
 <member name="b2" type="float"/>
</struct>

<interface name="MSAzimuth">

 <method name="Move_RefPos" returnType="int32_t" methodType="two-way">
  <argument name="requestedPos" type="double"/>
 </method>
```

```
   <method name="foo" returnType="nonBasic"
           nonBasicReturnTypeName="R_CII_405" methodType="ami">
    <argument name="bar" type="nonBasic" nonBasicTypeName="R_CII_404" />
    <argument name="zoo" type="uint16_t"/>
   </method>

  </interface>

   <!-- Exception -->

  <exception name="TooFastException">
   <member name="what" type="string"/>
  </exception>


  <!-- Not allowing inheritance on exceptions -->
  <!-- <exception name="SuperFastException" baseType="TooFastException"/>  -->


  <!-- Inclusion of types from external XMLs -->

  <struct name="M1ComposedType">
   <member name="b1" type="nonBasic" nonBasicTypeName= "M1::SomeM1Type"/>
   <member name="b2" type="float"/>
  </struct>

  <!-- Inclusion of multiple interfaces -->

  <interface name="SomeControlSystem">

   <extends interfaceName="Common::ControlInterface"/>
   <extends interfaceName="Common::LogInterface"/>

   <method name="foo" returnType="nonBasic" nonBasicReturnTypeName="R_CII_405"
           throws="TooFastException" methodType="ami">
    <argument name="bar" type="nonBasic" nonBasicTypeName="R_CII_404" />
    <argument name="zoo" type="uint16_t"/>
   </method>

  </interface>

 </package>
</types>
```

# Appendix B. Examples of Topic Definitions

```
<?xml version="1.0" encoding="UTF-8"?>
<topics xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="schemas/icd_topic_definition.xsd">

 <!-- [R-CII-393] ICDs shall be represented in a human readable formal syntax,
                  ideally an existing standards. -->

 <!-- [R-CII-394] ICD shall be flexible in order to define and express
      interfaces not necessarily built using the ICD and MAL language bindings
      (e.g. an industrial PC running OPC/UA Data Access server). -->

 <!-- [R-CII-395] ICDs shall contain artifacts defining the following:
       - Type definitions (structured types built from primitives).
       - Topic definitions (association of type definitions with Pub/Sub
                            topics and QoS parameters).
       - Interface definitions (association of type definitions with Req/Rep
                                interfaces and QoS parameters) -->

<!-- [R-CII-407] Topic and interface definitions shall specify the name of an
                 interface or topic, its messaging pattern (Publish/Subscribe
                 for Topics and Request/Reply for Interfaces), the
                 corresponding Quality of Service parameters and the types
                 involved in the messaging pattern.  -->

<!-- [R-CII-408] Specifically the topic and interface definitions shall
                 specify the following:
             - Middleware Mapping: Middleware scheme identifier (see 9.6)
             - Message Pattern Pub/Sub or Req/Rep (see 9.3)
             - Topic or Interface Name Pub/Sub:
                    Topic Name (per 9.3.1)
                    Req/Reply: Interface Name (per 9.3.2)
             - Address Parameters: Attributes associated with Middleware
                                   addressing
             - Associated Types Pub/Sub: Type associated with topic,
                                Req/Rep: Request Type and Reply Type.
             - QoS Parameters Message Pattern specific QoS parameters:
                  Publish/Subscribe: see 9.3.1.4, Request/Reply: see 9.3.2.7
              -->
```

```
<subsystem name="EdgeSensors">

 <!-- Synchronous Measurement -->

 <!-- [R-CII-408] - Message Pattern Pub/Sub -->

 <pubsub_topic>
  <!-- [R-CII-408]  - Topic Name (per 9.3.1)  -->
  <topic_name>Az:Pos_Actual</topic_name>
  <!-- [R-CII-408]  - Type associated with topic -->
  <topic_type>Pos_Actual</topic_type>
  <!-- [R-CII-408]  - Middleware Mapping: Middleware scheme identifier (see
                      9.6)  -->
  <!-- [R-CII-408]  - Address Parameters: Attributes associated with
                      Middleware addressing -->
  <address_uri>dds.ps://m1/Pos_Actual?p1=v1&amp;amp;p2=v2</address_uri>
  <!-- [R-CII-408]  - QoS Parameters Message Pattern specific QoS parameters:
                      Publish/Subscribe: see 9.3.1.4-->
  <qos latency_ms="0.1" deadline_ms="20"/>
  <performance rate_hz="20" latency_ms="25" synchronous="true"/>
  <mal>
   <dds>
    <!-- specific settings for DDS -->
   </dds>
  </mal>
 </pubsub_topic>

 <!-- Asynchronous Measurements  -->

 <pubsub_topic>
  <topic_name>Az:ServoStatus</topic_name>
  <topic_type>ServoStatus</topic_type>
  <address_uri>dds.ps://m1/ServoStatus</address_uri>
  <qos latency_ms="0.1" deadline_ms="20"/>
  <performance rate_hz="20" latency_ms="25" synchronous="false"/>
  <mal>
   <dds/>
  </mal>
 </pubsub_topic>

 <!-- Synchronous command -->

 <!-- [R-CII-408] - Message Pattern Req/Rep (see 9.3) -->

  <!-- [R-CII-408]  - Req/Reply: Interface Name (per 9.3.2) -->
  <service name="Robot1" interface="RobotControl" >
  <address_uri>zpb.rr://m4lsv.pl.eso.org:5001</address_uri>
  <performance rate_hz="20" latency_ms="25" synchronous="true"/>
  <!-- [R-CII-408]  - QoS Parameters Message Pattern specific QoS parameters:
                      Request/Reply: see 9.3.2.7 -->
  <qos connection_timeout_sec="0.1" reply_timeout_sec="20"/>
  <mal>
   <zpb>
        <!-- specific settings for ZeroMQ specific MAL -->
   </zpb>
  </mal>
```

```
  <methods>
      <method name="MoveToPark">
           <performance rate_hz="120" latency_ms="5" synchronous="true"/>
           <qos connection_timeout_sec="0.1" reply_timeout_sec="20"/>
      </method>
      <method name="GetLogLevel">
          <is_configuration>true</is_configuration>
      </method>
  </methods>

 </service>

</subsystem>

</topics>
```

# Appendix C. ICD Type Definition Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <xs:element name="types">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="include" type="includeDecl" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="package" type="packageDecl" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- only one non-empty (hierachical) package per module is allowed -->
  <xs:group name="packageElements">
    <xs:choice>
        <xs:element name="package" type="packageDecl"/>
           <xs:sequence>
             <xs:choice maxOccurs="unbounded">
               <xs:element name="struct" type="structDecl" minOccurs="0"/>
               <xs:element name="enum" type="enumDecl" minOccurs="0"/>
               <xs:element name="union" type="unionDecl" minOccurs="0" />
               <xs:element name="interface" type="interfaceDecl" minOccurs="0"/>
               <xs:element name="exception" type="exceptionDecl" minOccurs="0"/>
               <xs:element name="const" type="constDecl" minOccurs="0"/>
             </xs:choice>
           </xs:sequence>
    </xs:choice>
  </xs:group>

  <xs:complexType name="packageDecl">
    <xs:group ref="packageElements"/>
    <xs:attribute name="name" type="unqualifiedName" use="required"/>
  </xs:complexType>

  <xs:complexType name="structDecl">
    <xs:sequence>
      <xs:element name="member" type="memberDecl" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="unqualifiedName" use="required"/>
    <xs:attribute name="baseType" type="qualifiedName" use="optional"/>
  </xs:complexType>

  <xs:complexType name="memberDecl">
    <xs:attribute name="name" type="unqualifiedName" use="required"/>
    <xs:attribute name="type" type="basicType" use="required"/>
    <xs:attribute name="nonBasicTypeName" type="qualifiedName" use="optional"/>
    <xs:attribute name="key" type="trueFalseKind" use="optional" default="false"/>
    <xs:attribute name="arrayDimensions" type="arrayDimensionsKind" use="optional"/>
    <xs:attribute name="stringMaxLength" type="xs:integer" use="optional"/>
    <xs:attribute name="mime-type" type="xs:string" use="optional"/>
  </xs:complexType>

   <xs:complexType name="enumDecl">
    <xs:sequence>
      <xs:element name="enumerator" type="enumeratorDecl" minOccurs="1"
                  maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="unqualifiedName" use="required"/>
  </xs:complexType>
```

```
<xs:complexType name="enumeratorDecl">
  <xs:attribute name="name" type="unqualifiedName" use="required"/>
</xs:complexType>

<xs:complexType name="unionDecl">
  <xs:sequence>
    <xs:element name="discriminator" type="discriminatorDecl"/>
    <xs:element name="case" type="caseDecl" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="unqualifiedName" use="required"/>
</xs:complexType>

<xs:complexType name="discriminatorDecl">
  <xs:attribute name="type" type="basicType" use="required"/>
  <xs:attribute name="nonBasicTypeName" type="qualifiedName" use="optional"/>
</xs:complexType>

<xs:complexType name="caseDecl">
  <xs:sequence>
      <xs:element name="caseDiscriminator" type="caseDiscriminatorDecl"/>
      <xs:element name="member" type="memberDecl"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="caseDiscriminatorDecl">
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>

 <xs:complexType name="interfaceDecl">
  <xs:sequence>
    <xs:element name="extends" type="extendsDecl" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="method" type="methodDecl" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="unqualifiedName" use="required"/>
</xs:complexType>

<xs:complexType name="extendsDecl">
  <xs:attribute name="interfaceName" type="qualifiedName" use="required"/>
</xs:complexType>

<xs:complexType name="methodDecl">
  <xs:sequence>
    <xs:element name="argument" type="argumentDecl" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="unqualifiedName" use="required"/>
  <xs:attribute name="throws" type="qualifiedName" use="optional"/>
  <xs:attribute name="returnType" type="basicTypeOrVoid" use="required"/>
  <xs:attribute name="methodType" type="methodKind" use="optional"/>
  <xs:attribute name="nonBasicReturnTypeName" type="qualifiedName" use="optional"/>
  <xs:attribute name="arrayDimensions" type="arrayDimensionsKind" use="optional"/>
</xs:complexType>

<xs:complexType name="argumentDecl">
  <xs:attribute name="name" type="unqualifiedName" use="required"/>
  <xs:attribute name="type" type="basicType" use="required"/>
  <xs:attribute name="nonBasicTypeName" type="qualifiedName" use="optional"/>
  <xs:attribute name="arrayDimensions" type="arrayDimensionsKind" use="optional"/>
</xs:complexType>

<xs:simpleType name="methodKind">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ami"/>
    <xs:enumeration value="two-way"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:complexType name="exceptionDecl">
  <xs:sequence>
    <xs:element name="member" type="memberDecl" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="qualifiedName" use="required"/>
</xs:complexType>

<xs:complexType name="constDecl">
  <xs:attribute name="name" type="unqualifiedName" use="required"/>
  <xs:attribute name="type" type="basicType" use="required"/>
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="includeDecl">
  <xs:attribute name="href" type="xs:string" use="required"/>
</xs:complexType>

 <xs:simpleType name="unqualifiedName">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z]+[a-zA-Z_0-9]*"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="qualifiedName">
  <xs:restriction base="xs:string">
    <xs:pattern value="((::)?[a-zA-Z]+[a-zA-Z_0-9]*)+"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="basicType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="int8_t"/>
    <xs:enumeration value="uint8_t"/>
    <xs:enumeration value="int16_t"/>
    <xs:enumeration value="uint16_t"/>
    <xs:enumeration value="int32_t"/>
    <xs:enumeration value="uint32_t"/>
    <xs:enumeration value="int64_t"/>
    <xs:enumeration value="uint64_t"/>
    <xs:enumeration value="float"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="nonBasic"/>
    <xs:enumeration value="timestamp"/>
    <xs:enumeration value="blob"/>
    <xs:enumeration value="void"/>
  </xs:restriction>
</xs:simpleType>
```

```
  <xs:simpleType name="basicTypeOrVoid">
    <xs:restriction base="xs:string">
      <xs:enumeration value="boolean"/>
      <xs:enumeration value="int8_t"/>
      <xs:enumeration value="uint8_t"/>
      <xs:enumeration value="int16_t"/>
      <xs:enumeration value="uint16_t"/>
      <xs:enumeration value="int32_t"/>
      <xs:enumeration value="uint32_t"/>
      <xs:enumeration value="int64_t"/>
      <xs:enumeration value="uint64_t"/>
      <xs:enumeration value="float"/>
      <xs:enumeration value="double"/>
      <xs:enumeration value="string"/>
      <xs:enumeration value="nonBasic"/>
      <xs:enumeration value="timestamp"/>
      <xs:enumeration value="blob"/>
      <xs:enumeration value="void"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="trueFalseKind">
    <xs:restriction base="xs:string">
      <xs:enumeration value="0"/>
      <xs:enumeration value="1"/>
      <xs:enumeration value="true"/>
      <xs:enumeration value="false"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="arrayDimensionsKind">
    <xs:restriction base="xs:string">
      <xs:pattern value="[\[,\(]+[0-9]+(,[0-9]+)*[\],\)]+"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="memberId">
    <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="268435455"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

# Appendix D. ICD Topic Definitions Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <xs:element name="topics">
    <xs:complexType>
      <xs:group ref="subsystemElements" />
    </xs:complexType>
  </xs:element>

  <xs:group name="subsystemElements">
    <xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="include" type="includeDecl" minOccurs="0" />
        <xs:element name="subsystem" type="subsystemDecl" minOccurs="0" />
      </xs:choice>
    </xs:sequence>
  </xs:group>

  <xs:complexType name="includeDecl">
    <xs:attribute name="href" type="xs:string" use="required"/>
  </xs:complexType>

  <xs:complexType name="subsystemDecl">
    <xs:group ref="topicInterfaceDefinition" />
    <xs:attribute name="name" type="identifierName" use="required" />
  </xs:complexType>

  <xs:group name="topicInterfaceDefinition">
    <xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="pubsub_topic" type="pubsubDecl" minOccurs="0" />
        <xs:element name="service" type="serviceDecl" minOccurs="0" />
      </xs:choice>
    </xs:sequence>
  </xs:group>

  <!-- PubSub definition -->
  <xs:complexType name="pubsubDecl">
    <xs:all>
      <xs:element name="topic_name" minOccurs="1" />
      <xs:element name="topic_type" minOccurs="1" />
      <xs:element name="address_uri" type="xs:anyURI" minOccurs="1" />
      <xs:element name="qos" type="qosPubSubDecl" minOccurs="0" />
      <xs:element name="performance" type="performanceDecl" minOccurs="0" />
      <xs:element name="mal" type="malDecl" minOccurs="0" />
    </xs:all>
  </xs:complexType>

  <xs:complexType name="qosPubSubDecl">
    <xs:attribute name="latency_ms" type="xs:float" use="required" />
    <xs:attribute name="deadline_ms" type="xs:float" use="required" />
  </xs:complexType>

  <xs:complexType name="performanceDecl">
    <xs:attribute name="rate_hz" type="xs:float" use="required" />
    <xs:attribute name="latency_ms" type="xs:float" use="required" />
    <xs:attribute name="synchronous" type="trueFalseKind" use="required" />
  </xs:complexType>
```

```
  <xs:complexType name="malDecl">
    <xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="dds" maxOccurs="1" />
        <xs:element name="zpb" maxOccurs="1" />
        <xs:element name="opc" maxOccurs="1" />
        <xs:element name="mudapi" maxOccurs="1" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>

  <!-- Service definition -->
  <xs:complexType name="serviceDecl">
    <xs:all>
      <xs:element name="address_uri" type="xs:anyURI" minOccurs="1" />
      <xs:element name="is_configuration" type="trueFalseKind" minOccurs="0" />
      <xs:element name="performance" type="performanceDecl" minOccurs="0" />
      <xs:element name="mal" type="malDecl" minOccurs="1" />
      <xs:element name="qos" type="qosServiceDecl" minOccurs="0" />
      <xs:element name="methods" type="methodsDecl" minOccurs="0" />
    </xs:all>
    <xs:attribute name="name" type="identifierName" use="required" />
    <xs:attribute name="interface" type="identifierName" use="required" />
  </xs:complexType>

  <xs:complexType name="methodsDecl">
    <xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="method" type="methodDecl" minOccurs="0" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="methodDecl">
    <xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="qos" type="qosServiceDecl" minOccurs="0" />
        <xs:element name="performance" type="performanceDecl" minOccurs="0" />
        <xs:element name="is_configuration" type="trueFalseKind" minOccurs="0" />
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="identifierName" use="required" />
  </xs:complexType>

  <xs:complexType name="qosServiceDecl">
    <xs:attribute name="connection_timeout_sec" type="xs:float" use="required" />
    <xs:attribute name="reply_timeout_sec" type="xs:float" use="required" />
  </xs:complexType>

  <!-- types -->

  <xs:simpleType name="identifierName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z]+[a-zA-Z_0-9]*" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="trueFalseKind">
    <xs:restriction base="xs:string">
      <xs:pattern value="0|1" />
      <xs:pattern value="[Tt][Rr][Uu][Ee]" />
      <xs:pattern value="[Ff][Aa][Ll][Ss][Ee]"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

# Appendix E. XInclude XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xi="http://www.w3.org/2001/XInclude" targetNamespace="http://www.w3.org/2001/XInclude"
finalDefault="extension">
	<xs:annotation>
		<xs:documentation>
			Not normative, but may be useful. See the REC
			http://www.w3.org/TR/XInclude for definitive
			information about this
			namespace.
		</xs:documentation>
	</xs:annotation>
	<xs:element name="include" type="xi:includeType"/>
	<xs:complexType name="includeType" mixed="true">
		<xs:choice minOccurs="0" maxOccurs="unbounded">
			<xs:element ref="xi:fallback"/>
			<xs:any namespace="##other" processContents="lax"/>
			<xs:any namespace="##local" processContents="lax"/>
		</xs:choice>
		<xs:attribute name="href" type="xs:anyURI" use="optional"/>
		<xs:attribute name="parse" type="xi:parseType" use="optional"
		      default="xml"/>
		<xs:attribute name="xpointer" type="xs:string" use="optional"/>
		<xs:attribute name="encoding" type="xs:string" use="optional"/>
		<xs:attribute name="accept" type="xs:string" use="optional"/>
		<xs:attribute name="accept-language" type="xs:string"
		              use="optional"/>
		<xs:anyAttribute namespace="##other" processContents="lax"/>
	</xs:complexType>
	<xs:simpleType name="parseType">
		<xs:restriction base="xs:token">
			<xs:enumeration value="xml"/>
			<xs:enumeration value="text"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:element name="fallback" type="xi:fallbackType"/>
	<xs:complexType name="fallbackType" mixed="true">
		<xs:choice minOccurs="0" maxOccurs="unbounded">
			<xs:element ref="xi:include"/>
			<xs:any namespace="##other" processContents="lax"/>
			<xs:any namespace="##local" processContents="lax"/>
		</xs:choice>
		<xs:anyAttribute namespace="##other" processContents="lax"/>
	</xs:complexType>
</xs:schema>
```

# Appendix F. ICD Type mapping example

## 1.3. ICD mapping example

### 1.3.1. Enum mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

  <package name="icdSpecification">

    <enum name="EnumTest">
      <enumerator name="enum0"/>
      <enumerator name="enum1"/>
      <enumerator name="enum2"/>
    </enum>

  </package>
</types>
```

**Listing 20: ICD Enum mapping example**

#### 1.3.1.1. Enum Java mapping

```
package icdSpecification;

public enum EnumTest {
  enum0,
  enum1,
  enum2;

  // added for MatLab
  static Class<?> clazz() {
    return EnumTest.class;
  }
}
```

**Listing 21: Enum Java mapping**

#### 1.3.1.2. Enum C++ mapping

```
namespace icdSpecification {
enum EnumTest {
  enum0,
  enum1,
  enum2};

}  // namespace mal
}  // namespace elt
```

**Listing 22: Enum C++ mapping**

### 1.3.2. Structure mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

 <package name="icdSpecification">

  <struct name="PrimitiveStruct">
   <member name="boolval" type="boolean" key="true"/>
   <member name="int8val" type="int8_t"/>
   <member name="stringval" type="string"/>
  </struct>

  <struct name="PrimitiveArrayStruct">
   <member name="boolvalArray" type="boolean" arrayDimensions="(16)"/>
   <member name="int8valArray" type="int8_t" arrayDimensions="(16)"/>
   <member name="stringvalArray" type="string" arrayDimensions="(16)"/>
  </struct>


/package>
</types>
```

**Listing 23: ICD Structure mapping example**

#### 1.3.2.1. Structure Java Mapping

```
package icdSpecification;

// generated
public interface PrimitiveStruct extends elt.mal.ps.DataEntity<PrimitiveStruct> {

  boolean getBoolval();
  void setBoolval(boolean boolval);

  byte getInt8val();
  void setInt8val(byte int8val);

  String getStringval();
  void setStringval(String stringval);

  // added for MatLab
  static Class<?> clazz() {
    return PrimitiveStruct.class;
  }
}
```

**Listing 24: Structure with primitive types Java mapping**

```java
package icdSpecification;

// generated
public interface PrimitiveArrayStruct extends elt.mal.ps.DataEntity<PrimitiveArrayStruct> {

  java.util.List<Boolean> getBoolvalArray();
  void setBoolvalArray(java.util.List<Boolean> boolvalArray);

  java.util.List<Byte> getInt8valArray();
  void setInt8valArray(java.util.List<Byte> int8valArray);

  java.util.List<String> getStringvalArray();
  void setStringvalArray(java.util.List<String> stringvalArray);

  // added for MatLab
  static Class<?> clazz() {
    return PrimitiveArrayStruct.class;
  }
}
```

**Listing 25: Structure with array types Java mapping**

### 1.3.2.2. Structure C++ Mapping

```cpp
namespace icdSpecification {


// generated
class PrimitiveStruct : public ::elt::mal::ps::DataEntity<PrimitiveStruct> {
 public:
  static constexpr const char* TYPE_ID = "PrimitiveStruct";


  virtual bool getBoolval() const = 0;
  virtual void setBoolval(bool boolval) = 0;

  virtual int8_t getInt8val() const = 0;
  virtual void setInt8val(int8_t int8val) = 0;

  virtual std::string getStringval() const = 0;
  virtual void setStringval(const std::string& stringval) = 0;
  virtual ~PrimitiveStruct() = default;
};

} // icdSpecification
```

**Listing 26: Structure with primitive types C++ mapping**

```cpp
namespace icdSpecification {

// generated
class PrimitiveArrayStruct : public ::elt::mal::ps::DataEntity<PrimitiveArrayStruct> {
 public:
  static constexpr const char* TYPE_ID = "PrimitiveArrayStruct";


  virtual std::vector<bool> getBoolvalArray() const = 0;
  virtual void setBoolvalArray(const std::vector<bool>& boolvalArray) = 0;

  virtual std::vector<int8_t> getInt8valArray() const = 0;
  virtual void setInt8valArray(const std::vector<int8_t>& int8valArray) = 0;

  virtual std::vector<std::string> getStringvalArray() const = 0;
  virtual void setStringvalArray(const std::vector<std::string>& stringvalArray) = 0;
  virtual ~PrimitiveArrayStruct() = default;
};



} // icdSpecification
```

**Listing 27: Structure with array types C++ mapping**

### 1.3.3. Union Mapping

```xml
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

  <package name="icdSpecification">

    <enum name="EnumTest">
      <enumerator name="enum0"/>
      <enumerator name="enum1"/>
      <enumerator name="enum2"/>
    </enum>

    <union name="PrimitiveUnion">
      <discriminator type="nonBasic" nonBasicTypeName="EnumTest"/>
      <case>
        <caseDiscriminator value="enum0"/>
        <member name="enum0value" type="boolean"/>
      </case>
      <case>
        <caseDiscriminator value="enum1"/>
        <member name="enum1value" type="int8_t"/>
      </case>
      <case>
        <caseDiscriminator value="enum2"/>
        <member name="enum2value" type="int16_t"/>
      </case>
    </union>
  </package>
</types>
```

**Listing 28: ICD Union mapping example**

#### 1.3.3.1. Union Java Mapping

```
package icdSpecification;

// generated
public interface PrimitiveUnion extends elt.mal.ps.DataEntity<PrimitiveUnion> {

  icdSpecification.EnumTest getDiscriminator();

  boolean getEnum0value();
  void setEnum0value(boolean enum0value);

  byte getEnum1value();
  void setEnum1value(byte enum1value);

  short getEnum2value();
  void setEnum2value(short enum2value);

  // added for MatLab
  static Class<?> clazz() {
    return PrimitiveUnion.class;
  }
}
```

**Listing 29: Union Java mapping**

#### 1.3.3.2. Union C++ Mapping

```
namespace icdSpecification {


enum EnumTest {
  enum0,
  enum1,
  enum2};

// generated
class PrimitiveUnion : public ::elt::mal::ps::DataEntity<PrimitiveUnion> {
 public:
  static constexpr const char* TYPE_ID = "PrimitiveUnion";

  virtual EnumTest getDiscriminator() const = 0;
  virtual void setDiscriminator(const EnumTest& value) = 0;

  virtual bool getEnum0value() const = 0;
  virtual void setEnum0value(bool enum0value) = 0;

  virtual int8_t getEnum1value() const = 0;
  virtual void setEnum1value(int8_t enum1value) = 0;

  virtual int16_t getEnum2value() const = 0;
  virtual void setEnum2value(int16_t enum2value) = 0;
  virtual ~PrimitiveUnion() = default;
};


} // icdSpecification
```

**Listing 30: Union C++ mapping**

### 1.3.4. Exception Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

  <package name="icdSpecification">
    <exception name="ExceptionTest">
      <member name="param1" type="string"/>
      <member name="param2" type="int8_t"/>
    </exception>
  </package>
</types>
```

**Listing 31: ICD exception mapping example**

#### 1.3.4.1. Exception Java Mapping

```
package icdSpecification;

// generated
@SuppressWarnings("serial")
public class ExceptionTest extends Exception {

  public ExceptionTest(String param1, byte param2) {
    this.param1 = param1;
    this.param2 = param2;
  }

  protected final String param1;
  public String getParam1() {
    return param1;
  }

  protected final byte param2;
  public byte getParam2() {
    return param2;
  }
}
```

**Listing 32: Exception Java mapping**

#### 1.3.4.2. Exception C++ Mapping

```
namespace icdSpecification {

// generated
class ExceptionTest : public std::exception {
 public:
  std::string param1;
  int8_t param2;
};


} // icdSpecification
```

**Listing 33: Exception C++ mapping**

### 1.3.5. Package Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

  <package name="icdSpecification">
  </package>
</types>
```

**Listing 34: ICD package mapping example**

#### 1.3.5.1. Package Java Mapping

```
package icdSpecification;
```

**Listing 35: Package Java mapping**

#### 1.3.5.2. Package C++ Mapping

```
namespace icdSpecification {

} // icdSpecification
```

**Listing 36: Package C++ mapping**

### 1.3.6. Interface Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

  <package name="icdSpecification">

    <interface name="InterfaceTypeTest">
      <method name="primitiveMethod" returnType="uint8_t">
        <argument name="boolval" type="boolean"/>
        <argument name="int8val" type="int8_t"/>
        <argument name="stringval" type="string"/>
      </method>
      <method name="primitiveArrayMethod" returnType="uint8_t"
              arrayDimensions="(16)">
        <argument name="boolvalArray" type="boolean" arrayDimensions="(16)"/>
        <argument name="int8valArray" type="int8_t" arrayDimensions="(16)"/>
        <argument name="stringvalArray" type="string" arrayDimensions="(16)"/>
      </method>

    </interface>
  </package>
</types>
```

**Listing 37: ICD interface mapping example**

### 1.3.6.1. Interface Java Mapping

```
package icdSpecification;

public interface InterfaceTypeTest extends elt.mal.rr.RrEntity {

  byte primitiveMethod(boolean boolval, byte int8val, String stringval);

  java.util.List<Byte> primitiveArrayMethod(java.util.List<Boolean> boolvalArray,
                                            java.util.List<Byte> int8valArray,
                                            java.util.List<String> stringvalArray);

  static int typeHash() {
    return -742704276;
  }

  // added for MatLab
  static Class<?> clazz() {
    return InterfaceTypeTest.class;
  }
}
```

**Listing 38: Inteface Java mapping**

```
package icdSpecification;

public interface InterfaceTypeTestSync extends InterfaceTypeTest, elt.mal.rr.ClientRrEntity
{
  InterfaceTypeTestSync timeout(long timeoutDuration, java.util.concurrent.TimeUnit
                                timeoutUnit);

  // added for MatLab
  static Class<?> clazz() {
    return InterfaceTypeTestSync.class;
  }
}
```

**Listing 39: Client's Synchronous Interface Java Mapping**

```
package icdSpecification;

public interface AsyncInterfaceTypeTest extends elt.mal.rr.RrEntity {

  CompletableFuture<Byte> primitiveMethod(boolean boolval, byte int8val, String stringval);

  CompletableFuture<java.util.List<Byte>> primitiveArrayMethod(java.util.List<Boolean>
boolvalArray, java.util.List<Byte> int8valArray, java.util.List<String> stringvalArray);

  CompletableFuture<Void> enumTest(icdSpecification.EnumTest enumValue);
  CompletableFuture<Float> setValue(float value);

  static int typeHash() {
    return -742704276;
  }

  // added for MatLab
  static Class<?> clazz() {
    return AsyncInterfaceTypeTest.class;
  }
}
```

**Listing 40: Server's Asynchronous Interface Java Mapping**

```
package icdSpecification;

public interface InterfaceTypeTestAsync extends AsyncInterfaceTypeTest,
elt.mal.rr.ClientRrEntity {

  // added for MatLab
  static Class<?> clazz() {
    return InterfaceTypeTestAsync.class;
  }
}
```

**Listing 41: Client's Asynchronous Interface Java Mapping**

### 1.3.6.2. Interface C++ Mapping

```
namespace icdSpecification {

class InterfaceTypeTest : public virtual ::elt::mal::rr::RrEntity {
 public:
  virtual uint8_t primitiveMethod(bool boolval, int8_t int8val,
                                  const std::string& stringval) = 0;

  virtual std::vector<uint8_t> primitiveArrayMethod(const std::vector<bool>& boolvalArray,
                                                    const std::vector<int8_t>& int8valArray,
                                                    const std::vector<std::string>&
                                                       stringvalArray) = 0;

  static int typeHash() {
    return -742704276;
  }

  virtual ~InterfaceTypeTest() = default;
};


class AsyncInterfaceTypeTest : public virtual ::elt::mal::rr::RrEntity {
 public:
  virtual ::elt::mal::future<uint8_t> primitiveMethod(bool boolval, int8_t int8val,
                                                      const std::string& stringval) = 0;
  virtual ::elt::mal::future<std::vector<uint8_t>> primitiveArrayMethod(const
                std::vector<bool>& boolvalArray, const std::vector<int8_t>& int8valArray,
                const std::vector<std::string>& stringvalArray) = 0;

  static int typeHash() {
    return -742704276;
  }

  virtual ~AsyncInterfaceTypeTest() = default;
};

class InterfaceTypeTestSync :
  public virtual ::icdSpecification::InterfaceTypeTest,
  public virtual ::elt::mal::rr::ClientRrEntity {
    virtual InterfaceTypeTestSync timeout(std::chrono::milliseconds timeoutDuration) = 0;
};

class InterfaceTypeTestAsync :
  public virtual AsyncInterfaceTypeTest,
  public virtual ::elt::mal::rr::ClientRrEntity {
};

class AsyncInterfaceTypeTestImplWrapper :
  public virtual AsyncInterfaceTypeTest {
 public:
```

```
explicit AsyncInterfaceTypeTestImplWrapper(
    const std::shared_ptr<InterfaceTypeTest> &entity)
    : m_entity(entity) {}

  virtual ::elt::mal::future<uint8_t> primitiveMethod(bool boolval, int8_t int8val,
                                                      const std::string& stringval) {
    ::elt::mal::promise<uint8_t> p;

    p.set_value(m_entity->primitiveMethod(boolval, int8val, stringval));
    return p.get_future();
  }

  virtual ::elt::mal::future<std::vector<uint8_t>> primitiveArrayMethod(
            const std::vector<bool>& boolvalArray,
            const std::vector<int8_t>& int8valArray,
            const std::vector<std::string>& stringvalArray) {
    ::elt::mal::promise<std::vector<uint8_t>> p;

    p.set_value(m_entity->primitiveArrayMethod(boolvalArray, int8valArray, stringvalArray));
    return p.get_future();
  }

  virtual ~AsyncInterfaceTypeTestImplWrapper() = default;

 private:
    std::shared_ptr<InterfaceTypeTest> m_entity;
};


} // icdSpecification
```

**Listing 42: Interface C++ Mapping**

## 1.3.7. Constant Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schemas/icd_type_definition.xsd">

  <package name="icdSpecification">

    <const name="boolval" type="boolean" value="true"/>
    <const name="int8val" type="int8_t" value="1"/>
  </package>
</types>
```

**Listing 43: ICD constant mapping example**

### 1.3.7.1. Constant Java Mapping

```
package icdSpecification;

// generated
public interface Constants {
  public static boolean boolval = true;
  public static byte int8val = 1;
}
```

**Listing 44: Constant Java Mapping**

### 1.3.7.2. Constant C++ Mapping

```
namespace icdSpecification {

static constexpr bool boolval = true;
static constexpr int8_t int8val = 1;

} // icdSpecification
```

**Listing 45: Constant C++ Mapping**