S-Series ILS specifications XML schema implementation guidance

SX005G-B6865-05000-00

Issue No. 1.0

Publishers:

AeroSpace and Defence Industries Association of Europe

Aerospace Industries Association

Copyright and user agreement

1 Copyright

All correspondence and queries should be directed to:

ASD
10 Rue Montoyer
B-1000 Brussels
Belgium

2 Agreement for use of the specification SX005G™ suite of information

2.1 Definitions

SX005G™ suite of information means, but is not limited to:

- S-series ILS specification XML schema implementation guidance
- examples (eg, XML instances, UML models, pdf files, style sheets) and schemas
- any other software or information under the heading "**SX005G™ suite of information**", available for download from www.sX000i.org

Copyright holder means AeroSpace and Defense Industries Association of Europe (ASD).

2.2 Notice to user

By using all or any portion of **SX005G™ suite of information,** you accept the terms and conditions of this user agreement.

This user agreement is enforceable against you and any legal entity that has obtained **SX0005G™ suite of information** or any portion thereof and on whose behalf it is used.

2.3 License to use

As long as you comply with the terms of this user agreement, the copyright holders grant to you a non-exclusive license to use **SX005G™ suite of information**.

2.4 Intellectual property rights

SX005G™ suite of information is the intellectual property of and is owned by the copyright holder. Except as expressly stated herein, this user agreement does not grant you any intellectual property right in the **SX005G™ suite of information** and all rights not expressly granted are reserved by the copyright holder.

2.5 No modifications

You must not modify, adapt or translate, in whole or in part, **SX005G™ suite of information**.

2.6 No warranty

SX005G™ suite of information is being delivered to you "as is". The copyright holder does not warrant the performance or result you may obtain by using **SX005G™ suite of information**. The copyright holder makes no warranties, representations or indemnities, express or implied,

whether by statute, common law, custom, usage or otherwise as to any matter including without limitation merchantability, integration, satisfactory quality, fitness for any particular purpose, or non-infringement of third parties rights.

2.7 Limitation of liability

In no event will the copyright holder be liable to you for any damages, claims or costs whatsoever or any consequential, indirect or incidental damages, or any lost profits or lost savings or for any claim by a third party, even if the copyright holder has been advised of the possibility of such damages, claims, costs, lost profits or lost savings.

2.8 Indemnity

You agree to defend, indemnify, and hold harmless the copyright holder and its parents and affiliates and all of their employees, agents, directors, officers, proprietors, partners, representatives, shareholders, servants, attorneys, predecessors, successors, assigns, and those who have worked on the preparation, publication or distribution of the **SX005G™ suite of information** from and against any and all claims, proceedings, damages, injuries, liabilities, losses, costs, and expenses (including reasonable attorneys' fees and litigation expenses), relating to or arising from your use of the **SX005G™ suite of information** or any breach by you of this user agreement.

2.9 Governing law and arbitration

This user agreement will be governed by and construed in accordance with the laws of the Kingdom of Belgium.

In the event of any dispute, controversy or claim arising out of or in connection with this user agreement, or the breach, termination or invalidity thereof, the parties agree to submit the matter to settlement proceedings under the ICC (International Chamber of Commerce) ADR rules. If the dispute has not been settled pursuant to the said rules within 45 days following the filing of a request for ADR or within such other period as the parties may agree in writing, such dispute shall be finally settled under the rules of arbitration of the International Chamber of Commerce by three arbitrators appointed in accordance with the said rules of arbitration. All related proceedings should be at the place of the ICC in Paris, France.

The language to be used in the arbitral proceedings shall be English.

Highlights

This is the first issue of SX005G and, therefore are no changes to highlight.

End of data module

Table of contents

The listed documents are included in Issue 1.0 dated 2017-12-31, of this publication.

End of data module

Chapter 1

Introduction to the specification

Table of contents

Chapter 1.1

Purpose

Table of contents

Page

List of tables

References

Table 1 References

Chap No./Document No.	Title
SX001G	Glossary for the S-Series ILS specifications
SX002D	Common data model for the S-Series ILS specifications
SX004G	Unified Modeling Language (UML) model readers' guidance

1 General

The S-Series ILS specification XML schema implementation guidance is a document defining rules for managing the data exchanged via update messages between two systems that are compliant with the respective S-Series ILS specifications.

XML schemas defined for the respective S-Series ILS specifications define data that can be sent from one data system to another in order to support the ILS specifications business process under consideration. This specification defines rules for managing the data exchanged via update messages between two systems. The data contained in an exchange file (ie, XML document) can be new to the receiving data system or can be updates of data already received.

2 Purpose

The purpose of the S-Series ILS specification XML schema implementation guidance is to provide clear instruction on how to manage and exchange data between two data systems compliant to respective S-Series ILS specifications.

3 Background

The international aerospace and defense community has, over the past 20 years, invested considerable effort developing specifications in the field of ILS. The work was accomplished by integrated working groups composed of industry and customer organizations in a collaborative

environment. Customer organizations included representatives from national ministries and departments of defense from Europe and the United States. Aerospace and defense associations provided guidance and supported the work as required. The structure and functional coverage of these specifications was largely determined by North Atlantic Treaty Organization (NATO) requirements specified during an international workshop in Paris in 1993.

Beginning in 2003, the relationships between supporting industry organizations were formalized through a series of Memorandums of Understanding (MOU). Initially, AeroSpace and Defense Industries Association of Europe (ASD) and Aerospace Industries Association (AIA) signed an MOU to jointly develop and maintain S1000D.

In 2010, ASD and AIA signed an MOU to promote a common, interoperable, international suite of integrated logistics support specifications and jointly develop the S-Series ILS specifications. This MOU authorized the formation of the AIA/ASD ILS Specifications Council, whose responsibilities include liaising between ASD and AIA, developing and maintaining the S-Series ILS specifications, administering joint meetings and identifying additional areas of harmonization.

The need for a set of common and harmonized technologies and data models to be used by all S-Series ILS specifications was recognized in 2011 by the ILS Specifications Council. Its creation and maintenance was assigned to the Data Modeling and Exchange Working Group (DMEWG) and includes eg, a harmonized glossary (SX001G), a common data model (SX002D) and a UML readers' guide (SX004G). The latest addition to this set of harmonized technologies is the SX005G, S-Series ILS specification XML schema implementation guidance.

Chapter 1.2

Scope

Table of contents

Page

List of tables

References

Table 1 References

Chap No./Document No.	Title
S1000D	International specification for technical publications using a common source database
S2000M	International specification for material management - Integrated data processing
S3000L	International procedure specification for Logistics Support Analysis LSA
S4000P	International specification for developing and continuously improving preventive maintenance
S5000F	International specification for in-service data feedback
S6000T	International specification for training analysis and design - Human performance analysis, training analysis and training design
SX000i	International guide for the use of the S-Series Integrated Logistic Support (ILS) specifications
SX001G	Glossary for the S-Series ILS specifications
SX002D	Common data model for the S-Series ILS specifications
SX003X	Compatibility matrix for the S-Series ILS Specifications
SX004G	Unified Modeling Language (UML) model readers' guidance

1 General

SX005G, S-Series ILS specifications XML schema implementation guidance is a document addressing the requirements for the data exchange messages for the S-Series ILS specifications. This specification defines the rules for managing the data exchanged via update messages between two systems. The data contained in an exchange file (ie, XML document), can be new to the receiving data system or can be updates of data already received.

2 Scope

Within the scope of the document are:

- XML schemas for update messages between two systems
- Rules for managing update messages between two systems
- Action code definition for update messages between two systems
- Tailoring of valid values that must be exchanged between two systems that exchange data

3 S-Series ILS specifications

The AeroSpace and Defense Industries Association of Europe (ASD) and Aerospace Industries Association of America (AIA) S-Series ILS specifications are currently available or in the process of development, including:

- S1000D - International specification for technical publications using a common source database
- S2000M - International specification for material management - Integrated data processing
- S3000L - International procedure specification for Logistics Support Analysis (LSA)
- S4000P - International specification for developing and continuously improving preventive maintenance
- S5000F - International specification for in-service data feedback
- S6000T - International specification for training analysis and design - Human performance analysis, training analysis and training design
- SX000i - International guide for the use of the S-Series Integrated Logistic Support (ILS) specifications
- SX001G - Glossary for the S-Series ILS specifications
- SX002D - Common data model for the S-Series ILS specifications
- SX003X - Compatibility matrix for the S-Series ILS specifications
- SX004G - Unified Modeling Language (UML) model readers' guidance
- SX005G - S-Series ILS specification XML schema implementation guidance

Chapter 1.3

How to use the specification

Table of contents

Page

List of tables

References

Table 1 References

Chap No./Document No.	Title
Chap 1	Introduction to the specification
Chap 2	S-Series ILS specification XML schema implementation guidance

1 General

This chapter gives an overview of the organization of the specification and the fundamental reading rules.

2 Organization of the specification

2.1 Chapter 1 - Introduction to the specification

Chap 1 provides a summarized view on purpose, background and scope of SX005G.

2.2 Chapter 2 - XML schema implementation guidance

Chap 2 is step by step definition of constructs and rules that supports the exchange of updated information. It does this by starting with a very simplistic example, which can be managed through a set of basic rules. It then adds complexity to the examples and gives the reader an understanding on how each added complexity is to be managed.

Chap 2 is organized by the following topics:

- Basic use of action (crud) using a simple UML class as an example
- Managing multi-valued attributes in an update message for a business object
- Managing multi-valued associations in an update message
- Managing composition aggregations in an update message
- Managing attribute groups in an update message
- Other considerations with respect to exchange of classification values

End of data module

Chapter 1.4

Maintenance of the specification

Table of contents

Page

List of tables

References

Table 1 References

Chap No./Document No.	Title
SX000i	International guide for the use of the S-Series Integrated Logistic Support (ILS) specifications

1 Maintenance of the specification

SX005G is maintained by the DMEWG operating under the supervision of the ILS specifications Council.

Both the DMEWG and the ILS specifications Council include representatives from ASD and AIA member companies and nations. SX005G is maintained in accordance with SX000i.

Chapter 2

XML schema implementation guidance

Table of contents

Chapter 2.1

XML schema implementation guidance - Overview

Table of contents

Page

List of tables

References

Table 1 References

Chap No./Document No.	Title
None	

1 Overview

XML schemas defined for the respective S-Series ILS specifications define data that can be sent from one data system to another in order to support the ILS specifications business process under consideration. This specification defines rules for managing the data exchanged via update messages between two systems. The exchange file containing the update messages (defined as an XML document) will be referred to as a message in this document. The data in the message may be new to the receiving data system or may be updates of data already received. The action class and the identified action/crud code define actions to be identified for each instance of a "business object" (UML class) in a message file. The crud codes are discussed in Para 2.

Note

CRUD stands for Create, Read, Update and Delete, and is a term often used to summarize different types of commands or queries that can be performed on relational databases.

Note

It is the receiving system's responsibility to process the update messages created by the sending system. If a receiving data system's data is updated independent of the sending system data update process, the message (data values or element action codes) created by the sending system can become not current or valid.

2 Basic use of the action/crud code

The use of the action/crud code attribute is explained here in a step by step manner starting with its most simplistic use.

2.1 Action/crud codes

The acronym crud refers to all of the major functions that need to be implemented in a relational database application to consider it complete. Each letter in the acronym can be mapped to a standard SQL statement:

Operation	SQL	Code
Create	INSERT	I
Delete (Destroy)	DELETE	D
Update	UPDATE	U
Read (Retrieve)	SELECT	N
Replace	REPLACE	R

Although a relational database is a common persistence layer in software applications, there are numerous others (eg, an object database, an XML database, flat text files, custom file formats, tape, or card).

For the S-Series schemas, the crud codes define specific actions that the receiving data system should implement on the receiving system data, assuming no other updates have been made to the data in the receiving system applicable to the data being sent from the sending system.

Note

Action/crud codes are defined on a business object (UML class) level and not for individual attributes and associations.

The following chapters demonstrate the use of action/crud codes through a set of example UML models and their related XML schemas. It demonstrates how data may be exchanged / updated using the respective action/crud code. The complexity of the examples will increase as the different update transactions are presented.

Chapter 2.2

XML schema implementation guidance - A simple UML class

Table of contents

Page

List of tables

List of figures

References

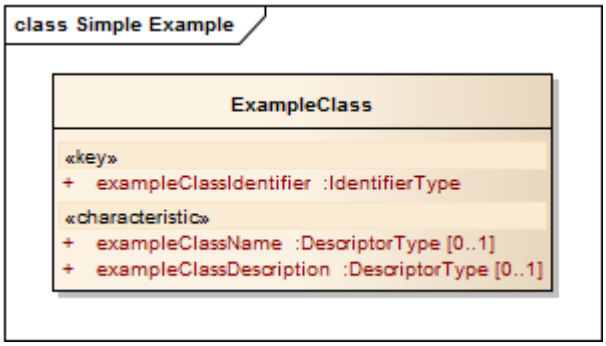
Table 1 References

Chap No./Document No.	Title
None	

1 A simple UML class

A simple use of the action code of the crud element is when a UML class and its corresponding XML construct do not allow for any multiple attributes and/or multiple associations.

Fig 1 illustrates a "simple" UML class, "ExampleClass", which has three attributes, exampleClassIdentifier, exampleClassName and exampleClassDescription. The exampleClassIdentifier is defined to be of the S-Series <<primitive>> IdentifierType, and exampleClassName and exampleClassDescrption are defined as being of the S-Series <<primitive>> DescriptorType.



ICN-B6865-SX005G0001-001-01

Fig 1 Simple class example

The flattened XML schema representation for the ExampleClass, in accordance with the S-Series ILS specifications XML Schema Authoring Rules, will look like:

```
    <!—START OF THE exampleClass XML SCHEMA -->

<xsd:complexType
   name="exampleClass">
   <xsd:annotation>
      <xsd:appinfo>SX001G:ExampleClass</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>

   <!--UML-Attributes-->

   <xsd:element
      name="classId"
      type="exampleClassIdentifier"/>
   <xsd:element
      name="name"
      type="exampleClassName" minOccurs="0"
      nillable="true"/>
   <xsd:element
      name="classDescr"
      type="exampleClassDescription" minOccurs="0"
      nillable="true"/>
   </xsd:sequence>
   <xsd:attribute
      name="uid"
      use="optional">
      <xsd:simpleType>
         <xsd:restriction
            base="xsd:ID">
            <xsd:pattern
                value="ec[1-9][0-9]*"/>
         </xsd:restriction>
       </xsd:simpleType>
   </xsd:attribute>
   <xsd:attribute
```

```
        name="crud"
        type="crudCodeValues"
        default="I"/>
</xsd:complexType>


<xsd:complexType
   name="exampleClassIdentifier">
   <xsd:annotation>
      <xsd:appinfo>SX001G:exampleClassIdentifier</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="id"
         type="nonEmptyString"
         nillable="true"/>
      <xsd:element
         name="class"
         type="value:classExampleIdentifierClassValues"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="setBy"
         type="organizationReference"
         minOccurs="0"
         nillable="true"/>
    </xsd:sequence>
</xsd:complexType>


<xsd:complexType
   name="exampleClassName">
   <xsd:annotation>
      <xsd:appinfo>SX001G:exampleClassName</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="descr"
         type="nonEmptyString"
         nillable="true"/>
      <xsd:element
         name="lang"
         type="value:languageCodeValues"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="date"
         type="xsd:date"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="providedBy"
```

```
            type="organizationReference"
            minOccurs="0"
            nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>


   <xsd:complexType
      name="exampleClassDescription">
      <xsd:annotation>
         <xsd:appinfo>SX001G:exampleClassDescription</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="descr"
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
            name="lang" type="value:languageCodeValues"
            minOccurs="0"
            nillable="true"/>
         <xsd:element
            name="date"
            type="xsd:date"
            minOccurs="0"
            nillable="true"/>
         <xsd:element
            name="providedBy"
            type="organizationReference" minOccurs="0"
            nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>


      <!—END OF THE exampleClass XML SCHEMA -->
```

Note

SimpleTypes defining valid values and allowed crud values are not included in the XML schema above.

2 First message (baseline message)

When data for the ExampleClass is sent for the first time, as part of a message, all instances must have the associated action/crud code set to "I", insertedElement.

The example below illustrates a message which includes two instances of "ExampleClass":

```
<exampleClass
   crud="I">
   <classId>
       <id>1</id>
   </classId>
   <name>
```

```
            <descr>Name1</descr>
        </name>
        <classDescr>
            <descr>Just a text description.</descr>
        </classDescr>
    </exampleClass>
    <exampleClass
        crud="I">
        <classId>
            <id>2</id>
        </classId>
        <name>
            <descr>Name2</descr>
        </name>
    </exampleClass>
```

Note

Use of crud="I" can be omitted since the crud attribute is defaulted to "I". However, it's defined explicitly in these examples for clarity.

Note

For the initial examples, the optional characterizations of attribute values are not included. These optional characterizations will be discussed in Chap 2.3.

2.1 Update message - deletedElement

Use of the action/crud code "D" (deletedElement) means that an entire record (business object) is to be deleted. This will result in the deletion of the complete business object including all its attributes and embedded business objects, nested through relationships, composite aggregation associations and implemented interfaces.

Note

In deleting a record, the complete key information must be provided ie, if the identifier attribute is using the identifierClass and/or identifierSetBy attributes then these also must be provided as part of the key information for the record to be deleted. However, these first examples are kept as easy as possible,

The example below illustrates an update message which identifies the ExampleClass where the exampleClassIdentifier <id> value equals "2" must be deleted:

```
<exampleClass
    crud="D">
    <classId>
        <id>2</id>
    </classId>
</exampleClass>
```

The resulting dataset at the receiving end must then look like (based on the example in Para 2):

```
<exampleClass>
    <classId>
        <id>1</id>
    </classId>
    <name>
       <descr>Name1</descr>
    </name>
```

```
    <classDescr>
        <descr>Just a text description.</descr>
    </classDescr>
</exampleClass>
```

Note

The resulting dataset must only include one instance of exampleClass after that the update procedure has been performed.

2.2 Update message - insertedElement

Use of the action/crud code "I" (insertedElement) as part of an update message means that there is a complete new record that is to be added into the existing dataset. This will result in the addition of a new business object including all its attributes and embedded elements.

The example below illustrates an update message which identifies the ExampleClass with the exampleClassIdentifier="3" must be added to the dataset:

```
<exampleClass
    crud="I">
    <classId>
        <id>3</id>
    </classId>
    <name>
        <descr>Name3</descr>
    </name>
    <classDescr>
        <descr>Just a textual description.</descr>
    </classDescr>
</exampleClass>
```

The resulting dataset of the receiving system must then look like (based on the example in Para 2.1):

```
<exampleClass>
    <classId>
        <id>1</id>
    </classId>
    <name>
        <descr>Name1</descr>
    </name>
    <classDescr>
        <descr>Just a text description.</descr>
    </classDescr>
</exampleClass>
<exampleClass>
    <classId>
        <id>3</id>
    </classId>
    <name>
        <descr>Name3</descr>
    </name>
    <classDescr>
        <descr>Just a textual description.</descr>
```

```
    </classDescr>
</exampleClass>
```

2.3 Update message – updatedElement

Use of the action/crud code "U" (updatedElement), as part of an update message, means that an existing record (attribute and embedded elements) must be changed. This will result in an updated business object.

Important characteristics for use of the action/crud code updatedElement are that:

- Key information must be included
- Characteristics (attributes and relationships) that have not changed must not be re-sent
- Characteristics (attributes and relationships) that are to be deleted must be defined as `xsi:nil="true"`.

Note

Updating identifier values, including key field change, is discussed in Chap 2.3.

The example below illustrates an update message which identifies the ExampleClass where the exampleClassIdentifier <id> value equals "1" has changed name, and the ExampleClass where the exampleClassIdentifier <id> value equals "3" has a description that is to be deleted:

```
<exampleClass
    crud="U">
    <classId>
        <id>1</id>
    </classId>
    <name>
        <descr>New name1</descr>
    </name>
</exampleClass>
<exampleClass
    crud="U">
    <classId>
        <id>3</id>
    </classId>
    <classDescr
    xsi:nil="true"/>
</exampleClass>
```

The resulting dataset at the receiving end must then look like (based on the example in Para 2.2):

```
<exampleClass>
    <classId>
        <id>1</id>
    </classId>
    <name>
        <descr>New name1</descr>
    </name>
    <classDescr>
        <descr>Just a text description.</descr>
    </classDescr>
</exampleClass>
```

```
<exampleClass>
   <classId>
       <id>3</id>
   </classId>
   <name>
       <descr>Name3</descr>
   </name>
</exampleClass>
```

2.4 Update message – replacedElement

Use of the action/crud code "R" (replacededElement) as part of an update message means that all existing characteristics (attributes and embedded elements) for an existing record (business object) must be deleted and be replaced with the data defined in the update message. This will result in a re-instantiated “business object”.

Important characteristics for use of the action/crud code replacedElement are that:

- Key information of the re-instantiated business object must be included as the first attribute.
- All other valid information (including additional identifiers) must be part of the message ie, if a characteristic (attribute and embedded elements) is left empty then is it assumed to be empty. The old characteristics for the business object are no longer valid.

The example below illustrates an update message which identifies the ExampleClass where the exampleClassIdentifier <id> value equals "3" is to be replaced ie, all current information in the receiving system must be replaced by the information in the update message:

```
<exampleClass
   crud="R">
   <classId>
       <id>3</id>
   </classId>
   <name>
       <descr>Replaced name3</descr>
   </name>
   <classDescr>
       <descr>Replaced textual description.</descr>
   </classDescr>
</exampleClass>
```

The resulting dataset at the receiving end must then look like (based on the example in Para 2.3):

```
<exampleClass>
   <classId>
       <id>1</id>
   </classId>
   <name>
       <descr>New name1</descr>
   </name>
   <classDescr>
       <descr>Just a text description.</descr>
   </classDescr>
</exampleClass>
```

```
<exampleClass
   <classId>
       <id>3</id>
   </classId>
   <name>
       <descr>Replaced name3</descr>
   </name>
   <classDescr>
       <descr>Replaced textual description.</descr>
   </classDescr>
</exampleClass>
```

2.5 Update message – nonChangedElement

Use of the action/crud code "N", (nonChangedElement), is required for navigating in nested structures and does not really apply to the simple example given in this section.

Important characteristics for use of the action/crud code nonChangedElement are that:

- Key (identifier) information of the nonChanged business object must be provided
- All other valid information must be left empty with the exception of nested elements which are needed for further navigation to these elements that are to be updated.

However, it can be used (but it is not really meaningful) as described in the example below:

```
<exampleClass
   crud="N">
   <classId>
       <id>3</id>
   </classId>
</exampleClass>
```

The resulting dataset at the receiving end must then look like (based on the example in Para 2.4):

```
<exampleClass>
   <classId>
       <id>1</id>
   </classId>
   <name>
       <descr>New name1</descr>
   </name>
   <classDescr>
       <descr>Just a text description.</descr>
   </classDescr>
</exampleClass>
<exampleClass>
   crud="R">
   <classId>
       <id>3</id>
   </classId>
   <name>
       <descr>Replaced name3</descr>
   </name>
```

```
    <classDescr>
        <descr>Replaced textual description.</descr>
    </classDescr>
</exampleClass>
```

Note

The resulting dataset is identical to the dataset that was the starting point for this update message (Para 2.4).

End of data module

Chapter 2.3

XML schema implementation guidance - Managing multi-valued attributes in an update message for a business object

Table of contents

Page

List of tables

List of figures

References

Table 1 References

Chap No./Document No.	Title
Chap 2.4	XML schema implementation guidance - Managing multi-valued associations in an update message
SX002D	Common data model for the S-Series ILS specifications
SX004G	Unified Modelling Language (UML) model readers' guidance

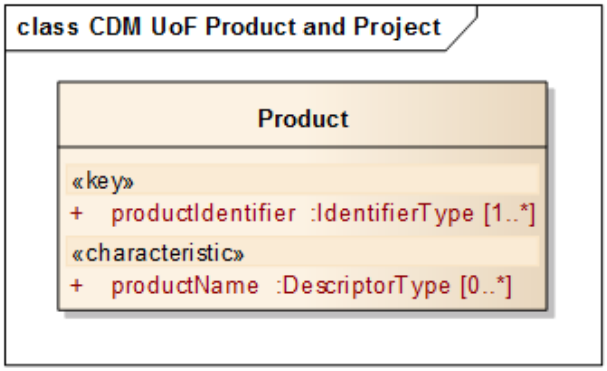
1 Introduction

This chapter focuses on describing how to manage multi-valued attributes, defined in the S-Series ILS specification as UML class attributes with cardinalities greater than one. An attributes in the S-Series ILS specifications UML models can be defined as either an S-Series <<primitive>>, an S-Series <<compoundAttribute>> or a UML built-in primitive eg, boolean or integer.

Note

Associations can also be modelled as attributes in a UML model. However, according to the S-Series UML modelling style guide (SX004G), associations between business objects are always modelled explicitly using explicit associations between UML classes. Multi-valued associations are described in Chap 2.4.

An example where a UML class has multi-valued attributes is the business object Product in SX002D, Common Data Model. Its attributes, productIdentifer and productName can have multiple values.



ICN-B6865-SX005G0002-001-01

Fig 1 SX002D Product and Project UoF UML diagram, Product

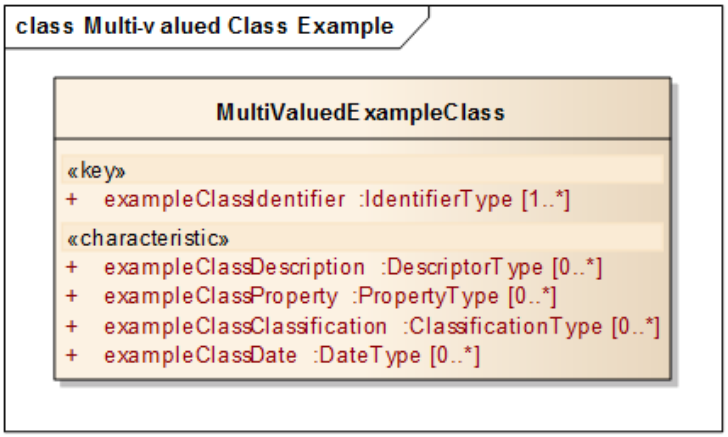
The productIdentifier is a <<key>> attribute ie, it provides the identifier(s) of the Product business object. SX002Dhas defined that the productIdentifier, implemented via the IdentifierType <<primitive>>, may have 1 or many product identifier values.

The productName is a <<characteristic>> attribute of the Product. SX002D has defined that the productName, implemented via the DescriptorType <<primitive>>, can have 0 or many product name values.

2 An example UML class containing multi-values attributes

The following chapters use a more complex example UML class (and XML schema) in order to demonstrate how multi-valued attributes must be managed in update messages.

The example UML class to be used is illustrated has five attributes, each representing a different S-Series <<primitive>>. Refer to Fig 2.



ICN-B6865-SX005G0003-001-01

Fig 2 Example class with multi-valued attributes

The flattened XML schema representation for the MultiValuedExampleClass, in accordance with the S-Series ILS specifications XML Schema Authoring Rules, will look like:

```
    <!—START OF THE multiValuedExampleClass XML SCHEMA -->

<xsd:complexType
   name="multiValuedExampleClass">
```

```
        <xsd:annotation>
           <xsd:appinfo>SX001G:multiValuedExampleClass</xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>

        <!--UML-Attributes-->

           <xsd:element
              name="classId"
              type="exampleClassIdentifier"
              maxOccurs="unbounded"/>
           <xsd:element
              name="classDescr"
              type="exampleClassDescription"
              minOccurs="0"
              maxOccurs="unbounded"
              nillable="true"/>
           <xsd:element
              name="classProp"
              type="exampleClassProperty"
              minOccurs="0"
              maxOccurs="unbounded"
              nillable="true"/>
           <xsd:element
              name="classClassif"
              type="exampleClassClassification"
              minOccurs="0"
              maxOccurs="unbounded"
              nillable="true"/>
           <xsd:element
              name="classDate"
              type="exampleClassDate"
              minOccurs="0"
              maxOccurs="unbounded"
              nillable="true"/>
        </xsd:sequence>
        <xsd:attribute
           name="uid"
           use="optional">
           <xsd:simpleType>
              <xsd:restriction
                 base="xsd:ID">
                 <xsd:pattern
                    value="mvec[1-9][0-9]*"/>
              </xsd:restriction>
           </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute
           name="crud"
           type="crudCodeValues"
           default="I"/>
     </xsd:complexType>
```

```
<xsd:complexType
   name="exampleClassIdentifier">
   <xsd:annotation>
      <xsd:appinfo>SX001G:exampleClassIdentifier</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="id"
         type="nonEmptyString"
         nillable="true"/>
      <xsd:element
         name="class"
         type="value:classExampleIdentifierClassValues"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="setBy"
         type="organizationReference"
         minOccurs="0"
         nillable="true"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType
   name="exampleClassDescription">
   <xsd:annotation>
      <xsd:appinfo>SX001G:exampleClassDescription</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="descr"
         type="nonEmptyString"
         nillable="true"/>
      <xsd:element
         name="lang"
         type="value:languageCodeValues"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="date"
         type="xsd:date"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="providedBy"
         type="organizationReference"
         minOccurs="0"
         nillable="true"/>
   </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType
   name="exampleClassProperty">
   <xsd:annotation>
      <xsd:appinfo>SX001G:exampleClassProperty</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="date"
         type="xsd:date"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="vdtm"
         type="value:valueDeterminationCodeValues"
         minOccurs="0"
         nillable="true"/>
      <xsd:choice>
         <xsd:sequence>
            <xsd:element
               name="unit"
               type="unit:unit"/>
               <xsd:choice>
                  <xsd:sequence>
                     <xsd:element
                        name="value"
                        type="xsd:decimal"
                        nillable="true"/>
                  </xsd:sequence>
                  <xsd:sequence>
                     <xsd:element
                        name="nomVal"
                        type="xsd:decimal"/>
                     <xsd:element
                        name="lowOff"
                        type="xsd:decimal"/>
                     <xsd:element
                        name="uppOff"
                        type="xsd:decimal"/>
                  </xsd:sequence>
                  <xsd:sequence>
                     <xsd:element
                        name="lowVal"
                        type="xsd:decimal"/>
                     <xsd:element
                        name="uppVal"
                        type="xsd:decimal"/>
                  </xsd:sequence>
               </xsd:choice>
            </xsd:sequence>
            <xsd:sequence>
```

```
                <xsd:element
                   name="txt" type="nonEmptyString"
                   nillable="true"/>
             </xsd:sequence>
         </xsd:choice>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType
      name="exampleClassClassification">
      <xsd:annotation>
        <xsd:appinfo>SX001G:exampleClassClassification</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="code"
            type="value:exampleClassClassificationCodeValues"
            nillable=”true”/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType
      name="exampleClassDate">
      <xsd:annotation>
         <xsd:appinfo>SX001G:exampleClassDate</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="date"
            type="xsd:date"
            nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>

       <!-- Supporting XML schema elements -->

   <xsd:complexType
      name="organizationReference">
      <xsd:sequence>
         <xsd:element
            name="orgId"
            type="organizationIdentifier"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType
      name="organizationIdentifier">
      <xsd:annotation>
         <xsd:appinfo>SX001G:organizationIdentifier</xsd:appinfo>
       </xsd:annotation>
       <xsd:sequence>
```

```
            <xsd:element
              name="id"
              type="nonEmptyString"
              nillable="true"/>
            <xsd:element
              name="class"
              type="value:organizationIdentifierClassValues"
              minOccurs="0"
              nillable="true"
              default="CAGE"/>
            <xsd:element
              name="setBy"
              type="organizationReference"
              minOccurs="0"
              maxOccurs="0"/>
         </xsd:sequence>
      </xsd:complexType>

      <xsd:simpleType
         name="nonEmptyString">
         <xsd:restriction
            base="xsd:token">
            <xsd:minLength
               value="1"/>
          </xsd:restriction>
      </xsd:simpleType>

         <!—END OF THE multiValuedExampleClass XML SCHEMA -->
```

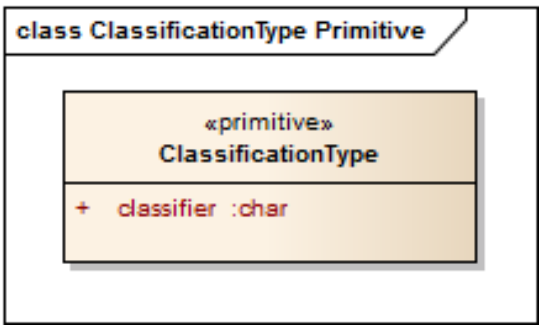
Note

SimpleTypes defining valid values and allowed crud values are not included in the XML schema above.

3 Multi-valued ClassificationType attributes

3.1 ClassificationType basics

The ClassificationType <<primitive>> does not have any associated characterization attributes.



ICN-B6865-SX002D0013-001-01

Fig 3 ClassificationType primitive

Typically, attributes of data type ClassificationType would not have multiple values if there's no associated applicability statement. Refer to Para 7. However, there might be cases where

multiple classifications might be applied without any associated applicability statement eg, multiple classifications for hazardous material.

3.2 ClassificationType update rules

If a multi-valued classification is defined without an associated applicability statement, then the following rules must be applied for update messages:

- Any classification(s) in the receiving system must be overwritten by the classification(s) in the update message (if provided)
- If no element for a ClassificationType attribute is provided, then the existing values must be unchanged
- To delete ALL existing classifications, the update of the attribute must be defined as `xsi:nil="true"`.

3.3 ClassificationType update examples

In order to illustrate the update rules, the exampleClassClassification ClassificationType attribute as defined for the MultiValuedExampleClass in Para 2 is used.

Note

The following examples are also using the exampleClassIdentifier attribute (<classId>) in order to be able to identify instances of MultiValuedExampleClass.

3.3.1 Update multi-valued ClassificationType attribute

The <`multiValuedExampleClass`> instance that is to be updated in this example has two instances of the <`classClassif`> ClassificationType attribute:

```
<multiValuedExampleClass>
   <classId>
       <id>1</id>
   </classId>
   <classClassif>
       <code>CODE1</code>
   </classClassif>
   <classClassif>
       <code>CODE2</code>
   </classClassif>
</multiValuedExampleClass>
```

If the desired update is to change one of the classifications eg, change CODE2 to CODE3 then both valid classifications must be provided in the update message, even if the first already exists in the receiving dataset. Ie, the update message must look like:

```
<multiValuedExampleClass
   crud="U">
   <classId>
       <id>1</id>
   </classId>
   <classClassif>
       <code>CODE1</code>
   </classClassif>
   <classClassif>
       <code>CODE3</code>
   </classClassif>
</multiValuedExampleClass>
```

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
       <id>1</id>
   </classId>
   <classClassif>
       <code>CODE1</code>
   </classClassif>
   <classClassif>
       <code>CODE3</code>
   </classClassif>
</multiValuedExampleClass>
```

Note

The update message is basically a combined statement of firstly, delete all existing classifications, and secondly add the new ones (ie, there is no explicit information on what has really changed in the update message).

If only the changed classification (ie, CODE3) was sent, then the resulting dataset at the receiving end must look like:

```
<multiValuedExampleClass>
   <classId>
       <id>1</id>
   </classId>
   <classClassif>
       <code>CODE3</code>
   </classClassif>
</multiValuedExampleClass>
```

In this case, one classification would be missing.

3.3.2 ClassificationType attribute is not present in the update message

If the <classClassif> element is not present in the update message (as in the example below) then the <classClassif> values in the receiving dataset must be unchanged.

```
<multiValuedExampleClass
   crud="U">
   <classId>
       <id>1</id>
   </classId>
</multiValuedExampleClass>
```

If update message above was to be applied to the original dataset defined in Para 3.3.1, then the resulting dataset must look the same as it was before the update message was sent:

```
<multiValuedExampleClass>
   <classId>
       <id>1</id>
   </classId>
   <classClassif>
      <code>CODE1</code>
   </classClassif>
```

```
    <classClassif>
        <code>CODE2</code>
    </classClassif>
</multiValuedExampleClass>
```

3.3.3 ClassificationType attribute values to be deleted in an update message

If all classification values for the <classClassif> element is to be deleted, then the update message must instantiate the <classClassif> element with xsi:nil="true" (see example below).

```
<multiValuedExampleClass
    crud="U">
    <classId>
        <id>1</id>
    </classId>
    <classClassif
        xsi:nil="true"/>
</multiValuedExampleClass>
```

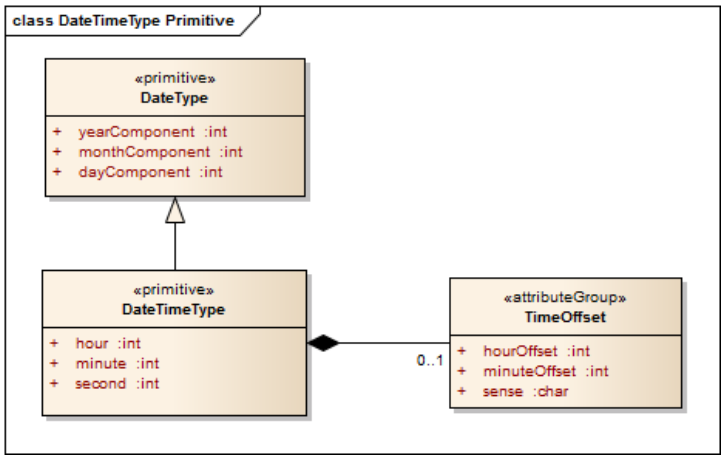
If update message above was to be applied to the original dataset defined in Para 3.3.1, then all instances of <classClassif> must be deleted ie, the resulting dataset must look like:

```
<multiValuedExampleClass>
    <classId>
        <id>1</id>
    </classId>
</multiValuedExampleClass>
```

4 Multi-valued Date and DateTime attributes

4.1 DateType and DateTimeType basics

Similar to the ClassificationType <<primitive>> the dateType and dateTimeType <<primitive>> does not include any associated characterization attributes.



ICN-B6865-SX002D0014-001-01

Fig 4 DateType and DateTimeType primitives

Typically, attributes of data type DateType and DateTimeType would not have multiple values if there's no associated applicability statement. Refer to Chap 2.4.

4.2 DateType and DateTimeType update rules

If a multi-valued date (or dateTime) is defined, without an associated applicability statement, then the following rules must be applied for update messages:

- Any date(s) in the receiving system must be overwritten by the date(s) in the update message (if provided)
- If no element for a DateType attribute is provided, then the existing values must be unchanged
- To delete ALL existing date values for the specific date attribute, the update of the attribute must be defined as `xsi:nil="true"`.

4.3 DateType update examples

In order to illustrate the update rules, the exampleClassDate DateType attribute as defined for the MultiValuedExampleClass in Para 2 is used.

Note

The following examples are also using the exampleClassIdentifier attribute (<`classId`>) in order to be able to identify instances of MultiValuedExampleClass.

4.3.1 Update multi-valued DateType attribute

The <`multiValuedExampleClass`> instance that is to be updated in this example has two instances of the <`classDate`> DateType attribute:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDate>
      <date>2016-10-10</date>
   </classDate>
   <classDate>
      <date>2016-10-11</date>
   </classDate>
</multiValuedExampleClass>
```

If the desired update is to change one of the dates eg, change 2016-10-11 to 2016-10-12 then both valid dates must be provided in the update message, even if the first already exists in the receiving dataset. Ie, the update message must look like:

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classDate>
      <date>2016-10-10</date>
   </classDate>
   <classDate>
      <date>2016-10-12</date>
   </classDate>
</multiValuedExampleClass>
```

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDate>
      <date>2016-10-10</date>
   </classDate>
   <classDate>
      <date>2016-10-12</date>
   </classDate>
</multiValuedExampleClass>
```

Note

The update message is basically a combined statement of firstly, delete all existing dates, and secondly add the new ones (ie, there is no explicit information on what has really changed in the update message).

If only the changed date (ie, 2016-10-12) was sent, then the resulting dataset at the receiving end must look like:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDate>
      <date>2016-10-12</date>
   </classDate>
</multiValuedExampleClass>
```

showing, one date is missing.

4.3.2 DateType attribute is not present in the update message

If the <classDate> element is not present in the update message (as in the example below) then the <classDate> values in the receiving dataset must be unchanged.

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
</multiValuedExampleClass>
```

If update message above was to be applied to the original dataset defined in Para 4.3, then the resulting dataset must look the same as it was before the update message was sent:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDate>
      <date>2016-10-10</date>
   </classDate>
   <classDate>
      <date>2016-10-12</date>
```

```
   </classDate>
</multiValuedExampleClass>
```

4.3.3 DateType attributes to be deleted in an update message

If all date values for the <classDate> element is to be deleted, then the update message must instantiate the <classDate> element with xsi:nil="true". For example:

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classDate
      xsi:nil="true"/>
</multiValuedExampleClass>
```

If update message above was to be applied to the original dataset defined in Para 4.3.2 then all instances of <classdate> must be deleted ie, the resulting dataset must look like:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
</multiValuedExampleClass>
```

5 Multi-valued DescriptorType attributes

5.1 DescriptorType basics

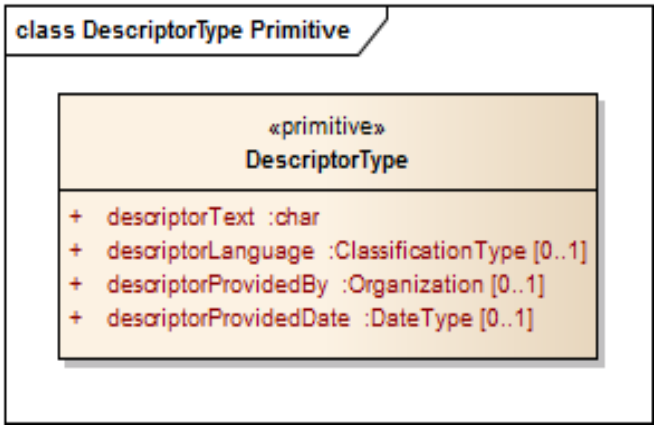
The DescriptorType <<primitive>> has four attributes:

- Firstly, the "primary" attribute descriptorText
- Secondly, three optional characterization attributes, descriptorLanguage (ClassificationType <<primitive>>), descriptorProvidedBy (Organization) and descriptorProvideByDate (DateType <<primitive>>).

These optional characterization attributes define uniqueness for the descriptorText attribute. Stated differently, multiple values for an attribute of data type DescriptorType should have different values for the optional characterization attributes.

Note

The examples provided in this document assume multiple values of primitive data type attributes will have different values of optional characterization attributes. These characterization attributes define uniqueness for a single property value instance.



ICN-B6865-SX002D0015-001-01

Fig 5 DescriptorType primitive

Note

Individual S-Series ILS specifications can choose to deprecate (restrict) the use of one or more of the DescriptorType characterization attributes (descriptorLanguage, descriptorProvidedBy and descriptorProvidedDate) as part of their tailoring process. However, these kinds of tailoring do not affect the principles defined here, but just limit the described scenarios.

5.2 DescriptorType update rules

If a multi-valued descriptor is defined, without an associated applicability statement, then the following rules must be applied for update messages:

- Any descriptors in the receiving system must be overwritten by the descriptors in the update message (if provided), if the optional characterization attributes in the update message match the optional characterization attributes of an existing descriptor.
- If no element for a descriptor attribute is provided, then the existing values must be unchanged
- To delete ALL existing descriptor values for a specific descriptorType attribute, the update of the attribute must be defined as `xsi:nil="true"`.

5.3 DescriptorType update examples

In order to illustrate the update rules, the exampleClassDescription DescriptorType attribute as defined for the MultiValuedExampleClass in Para 2 is used.

The MultiValuedExampleClass may have 0, 1 or many instances of its exampleClassDescription attribute, where each instance can have 0 or 1 associated language code (descriptorLanguage), 0 or 1 associated organization (descriptorProvidedBy implying that organization defined the descriptor value), and/or 0 or 1 date (descriptorProvidedDate), defining the date that the descriptor was provided.

Therefore, an update message for the MultiValuedExampleClass may have 0, 1 or many exampleClassDescription(s), each with 0 or 1 values for its optional characterization attributes in terms of descriptorLanguage, the descriptorProvidedBy and descriptorProvideByDate, respectively.

If the exchange message is defined with a specific combination of non-blank optional characterization attributes for a descriptorText value, it is the intent of the exchange message to update or insert the descriptorText for the matching non-blank characterization attributes.

Note

The following examples are also using the exampleClassIdentifier element (<classId>) in order to be able to identify instances of MultiValuedExampleClass.

5.3.1.1 Adding an instance to a multi-valued DescriptorType attribute

The <multiValuedExampleClass> instance that is to be updated in this example has two instances of <classDecr> (exampleClassDescription) DescriptorType attribute.

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDescr>
       <descr>Just a text description.</descr>
      <lang>en</lang>
      <date>2016-03-01</date>
   </classDescr>
   <classDescr>
      <descr>An additional textual description.</descr>
      <lang>en</lang>
      <date>2016-03-04</date>
   </classDescr>
</multiValuedExampleClass>
```

Note

In the example, above, the two instances of the exampleClassDescription attribute (<classDescr>) have different <date> characteristic values.

If an instance of the exampleClassDescription attribute (<classDescr>) in an update message do not match with an existing set of characterizations (descriptorLanguage, descriptorProvidedBy and descriptorProvideByDate) in the receiving system, then the exampleClassDescription in the update must be inserted, with its related additional characterizations.

Assume that an update message for the instance of <multiValuedExampleClass> defined above contains two additional descriptions ie, instances of <classDescr>:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDescr>
      <descr>Even more descriptive information.</descr>
      <lang>en</lang>
      <date>2016-03-08</date>
   </classDescr>
   <classDescr>
      <descr>Svensk beskrivning.</descr>
      <lang>se</lang>
      <date>2016-03-08</date>
   </classDescr>
</multiValuedExampleClass>
```

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDescr>
      <descr>Just a text description.</descr>
      <lang>en</lang>
      <date>2016-03-01</date>
   </classDescr>
   <classDescr>
      <descr>An additional textual description.</descr>
      <lang>en</lang>
      <date>2016-03-04</date>
   </classDescr>
   <classDescr>
      <descr>Even more descriptive information.</descr>
      <lang>en</lang>
      <date>2016-03-08</date>
   </classDescr>
   <classDescr>
      <descr>Svensk beskrivning.</descr>
      <lang>se</lang>
      <date>2016-03-08</date>
   </classDescr>
</multiValuedExampleClass>
```

5.3.2 Updating an existing instance of a multi-valued DescriptorType attribute

In order to update the descriptorText for an existing instance of exampleClassDescription the characterizations of an instance of exampleClassDescription in an update message must match the characterizations for an already existing instance of exampleClassDescription.

The following update message will update an existing exampleClassDescription in the dataset described in Para 5.3.1.1, since the optional characterization attributes match an already existing instance of `<classDescr>`.

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classDescr>
      <descr>A corrected text description.</descr>
      <lang>en</lang>
      <date>2016-03-01</date>
   </classDescr>
</multiValuedExampleClass>
```

Note

In the example above, the descriptor optional characterization attributes (descriptorLanguage and descriptorProvidedDate) match the optional characterization attributes of an existing (previously exchanged) instance of exampleClassDescription.

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDescr>
      <descr> A corrected text description.</descr>
      <lang>en</lang>
      <date>2016-03-01</date>
   </classDescr>
   <classDescr>
      <descr>An additional textual description.</descr>
      <lang>en</lang>
      <date>2016-03-04</date>
   </classDescr>
   <classDescr>
      <descr>Even more descriptive information.</descr>
      <lang>en</lang>
      <date>2016-03-08</date>
   </classDescr>
   <classDescr>
      <descr>Svensk beskrivning.</descr>
      <lang>se</lang>
      <date>2016-03-08</date>
   </classDescr>
</multiValuedExampleClass>
```

5.3.3 Deletion of all instances of a DescriptorType attribute

To delete ALL intances of exampleClassDescription for the `<multiValuedExampleClass>` the update message for the `<multiValuedExampleClass>` exampleClassDescription attribute must be identified as `xsi:nil="true"`. The receiving system should delete ALL the existing exampleClassDescriptions.

The following update message is to be applied to the resulting dataset in Para 5.3.2:

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classDescr
      xsi:nil="true"/>
</multiValuedExampleClass>
```

The resulting dataset must then look like:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
</multiValuedExampleClass>
```

5.3.4 Deleting a single instance of a DescriptorType attribute

In order to delete a single instance of exampleClassDescription the characterizations of an instance of exampleClassDescription in an update message must match the characterizations for an already existing instance of exampleClassDescription, and the value for its descriptorText (<descr>) element must be set to `xsi:nil="true"`.

The following update message will delete a single instance of exampleClassDescription in the resulting dataset described in Para 5.3.2.

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classDescr>
      <descry
         xsi:nil="true">
      <lang>en</lang>
      <date>2016-03-01</date>
   </classDescr>
</multiValuedExampleClass>
```

Note

In the example above, the descriptor type optional characterization attributes (descriptorLanguage and descriptorProvidedDate) match the optional characterization attributes of an existing (previously exchanged) instance of exampleClassDescription.

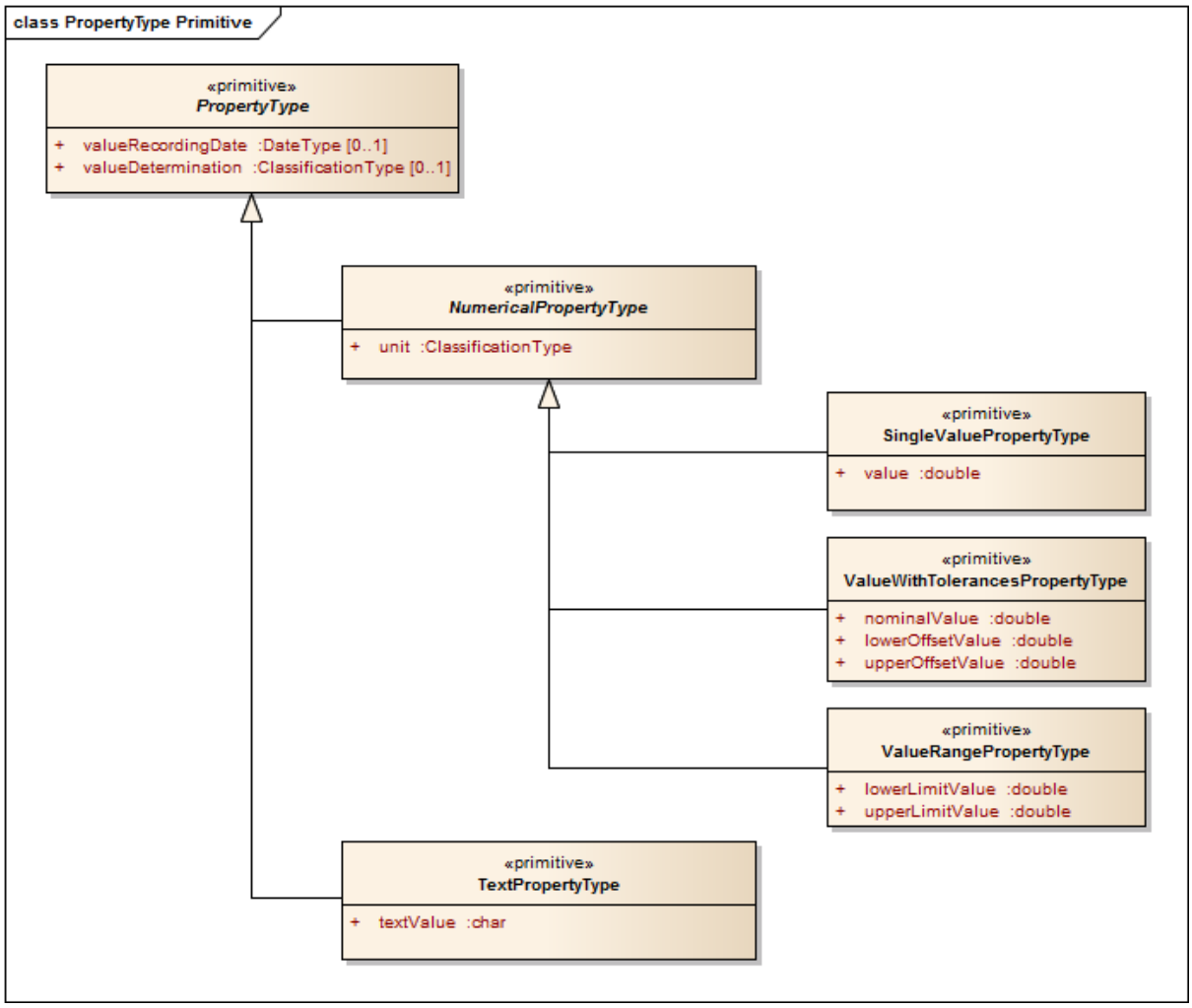
The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classDescr>
      <descr>An additional textual description.</descr>
      <lang>en</lang>
      <date>2016-03-04</date>
   </classDescr>
   <classDescr>
      <descr>Even more descriptive information.</descr>
      <lang>en</lang>
      <date>2016-03-08</date>
   </classDescr>
   <classDescr>
      <descr>Svensk beskrivning.</descr>
      <lang>se</lang>
      <date>2016-03-08</date>
   </classDescr>
</multiValuedExampleClass>
```

6 Multi-valued PropertyType attributes

6.1 PropertyType basics

The PropertyType primitive has four different representations of property values; single values, value with tolerances, value ranges and text values.



ICN-B6865-SX002D0017-001-01

Fig 6 PropertyType primitive

All representations of property values have two optional characterization attributes in common; valueDetermination (ClassificationType <<primitive>>) and valueRecordingDate (DateType <<primitive>>).

Numeric property values also have a third mandatory characterization attribute, unit (ClassificationType primitive>).

These characterization attributes define uniqueness for a single property value instance. Stated differently, multiple values for a class attribute of data type PropertyType should have different values for the characterization attributes.

Note

Individual S-Series ILS specifications can choose to deprecate (restrict) the use of one or more of the PropertyType optional characterization attributes (valueDetermination and valueRecordingDate) as part of their tailoring process. However, these kinds of tailoring do not affect the principles defined here, but just limit the described scenarios.

6.2 PropertyType update rules

If a multi-valued property is defined, without an associated applicability statement, then the following rules must be applied for update messages:

- Any property values in the receiving system must be overwritten by the property values in the update message (if provided), if the optional characterization attributes in the update message match the optional characterization attributes of an already existing property value.
- If no element for a property value attribute is provided, then the existing values must be unchanged
- To delete ALL existing property values for a specific property attribute, the update of the attribute must be defined as `xsi:nil="true"`.

6.3 PropertyType update examples

In order to illustrate the update rules, the exampleClassProperty PropertyType attribute as defined for the MultiValuedExampleClass in Para 2 is used.

The MultiValuedExampleClass may have 0, 1 or many instances of its exampleClassProperty attribute, where each instance may have 0 or 1 values for the valueDetermination and/or valueRecordingDate. There must always be a related unit value for numeric property values.

Note

The following examples are also using the exampleClassIdentifier attribute (<classId>) in order to be able to identify instances of MultiValuedExampleClass.

6.3.1 Adding an instance to a multi-valued PropertyType attribute

The <`exampleClass2`> instance that is to be updated in this example has two instances of <`classProp`> (exampleClassProperty) PropertyType attribute, both being instances of single numeric values.

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <vdtm>REQ</vdtm>
      <unit>FH</unit>
      <value>15</value>
   </classProp>
   <classProp>
      <vdtm>SPE</vdtm>
      <unit>FH</unit>
      <value>17</value>
   </classProp>
</multiValuedExampleClass>
```

Note

In the example, above, the two instances of the exampleClassProperty attribute (<`classProp`>) have different <`vdtm`> (value determination) characteristic values, but the same units. The set of characterizations must be unique in the context of a given multi-valued attribute. It is assumed that there can NOT be multiple identical instances of a primitive value with the same optional characterization attribute values.

The example illustrates two property values, the first being a required value (valueDetermination = REQ) defined as 15 Flight Hours (FH), the second being a specified (valueDetermination = SPE) value defined as 17 Flight Hours (FH).

The receiving system should assume that if the characterizations of an instance of exampleClassProperty in an update message matches an already existing exampleClassProperty then the property value must be updated. If the characterizations of an instance of exampleClassProperty in an update message do not match with an existing set of characterizations (valueDetermination, valueRecordingDate, and unit) in the receiving system, then the exampleClassProperty in the update must be inserted, with its related additional characterizations.

Assuming that an update message for the instance of <multiValuedExampleClass> defined above contains one additional property value ie, an instance of <exampleClassProperty>:

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <date>2016-10-15</date>
      <vdtm>MEA</vdtm>
      <unit>FH</unit>
      <value>19</value>
   </classProp>
</multiValuedExampleClass>
```

This value states that there is now also a measured (MEA) value, recorded on OCT 15 2016.

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <vdtm>REQ</vdtm>
      <unit>FH</unit>
      <value>15</value>
   </classProp>
   <classProp>
      <vdtm>SPE</vdtm>
      <unit>FH</unit>
      <value>17</value>
   </classProp>
   <classProp>
      <date>2016-10-15</date>
      <vdtm>MEA</vdtm>
      <unit>FH</unit>
      <value>19</value>
   </classProp>
</multiValuedExampleClass>
```

6.3.2 Updating an existing instance of a multi-valued PropertyType attribute

In order to update the value for an existing instance of exampleClassProperty the characterizations of an instance of exampleClassProperty in an update message must match the characterizations for an already existing instance of exampleClassProperty.

The following update message will update an existing exampleClassProperty in the dataset described in Para 6.3.1.

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <date>2016-10-15</date>
      <vdtm>MEA</vdtm>
      <unit>FH</unit>
      <value>20</value>
   </classProp>
 </multiValuedExampleClass>
```

Note

In the example above, the characterization attributes (valueRecordingDate, valueDetermination and unit) match the charaterization attributes of an existing (previously exchanged) instance of exampleClassProperty.

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <vdtm>REQ</vdtm>
      <unit>FH</unit>
      <value>15</value>
   </classProp>
   <classProp>
      <vdtm>SPE</vdtm>
      <unit>FH</unit>
      <value>17</value>
   </classProp>
   <classProp>
      <date>2016-10-15</date>
      <vdtm>MEA</vdtm>
      <unit>FH</unit>
      <value>20</value>
   </classProp>
</multiValuedExampleClass>
```

6.3.3 Deletion of all instances of a PropertyType attribute

To delete ALL intances of exampleClassProperty for the multiValuedExampleClass, the update message for the multiValuedExampleClass exampleClassProperty attribute must be identified as

xsi:nil="true". The receiving system should delete ALL the existing instances of exampleClassProperty for that instance of multiValuedExampleClass.

The following update message is to be applied to the resulting dataset in Para 6.3.2:

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classProp
      xsi:nil="true"/>
</multiValuedExampleClass>
```

The resulting dataset must then look like:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
</multiValuedExampleClass>
```

6.3.4 Deleting a single instance of a PropertyType attribute

In order to delete a single instance of exampleClassProperty the characterizations of an instance of exampleClassProperty in an update message must match the characterizations for an already existing instance of exampleClassProperty, and the value for its <value> element must be set to xsi:nil="true".

Note

Regardless of whether the existing value is a single value, a value with tolerances or a value range, the instance to be deleted must use the single value representation and define it as xsi:nil="true".

The following update message will delete the specified value instance of exampleClassProperty in the dataset described in Para 6.3.2.

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <vdtm>SPE</vdtm>
      <unit>FH</unit>
      <value
         xsi:nil="true"/>
   </classProp>
</multiValuedExampleClass>
```

Note

In the example above, the characterization attributes valueDetermination (vdtm) and unit match the attributes of an existing instance of exampleClassProperty.

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <vdtm>REQ</vdtm>
      <unit>FH</unit>
      <value>15</value>
   </classProp>
    <classProp>
       <date>2016-10-15</date>
       <vdtm>MEA</vdtm>
       <unit>FH</unit>
       <value>20</value>
   </classProp>
</multiValuedExampleClass>
```

6.3.5 Changing the value representation

An important aspect of updating property values is that the updated property value must not necessarily have the same value representation as the existing property value ie, the updated value might be eg, a value range as where the existing is a single value.

The following update message will change the required value instance of exampleClassProperty in the dataset described in Para 6.3.4, and at the same time change its value representation from a single numerical value to a value range.

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <vdtm>REQ</vdtm>
      <unit>FH</unit>
      <lowVal>14</lowVal>
      <uppVal>16</uppVal>
   </classProp>
</multiValuedExampleClass>
```

Note

In the example above, the characterization attributes valueDetermination (vdtm) and unit match the attributes of an existing instance of exampleClassProperty.

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classProp>
      <vdtm>REQ</vdtm>
      <unit>FH</unit>
      <lowVal>14</lowVal>
```

```
        <uppVal>16</uppVal>
    </classProp>
    <classProp>
        <date>2016-10-15</date>
        <vdtm>MEA</vdtm>
        <unit>FH</unit>
        <value>20</value>
    </classProp>
</multiValuedExampleClass>
```

Multi-valued identifiers (IdentifierType)

6.4 Identifier basics

This section describes how to manage multi-values identifiers. Identifiers (keys) are normally defined as being of data type IdentifierType. However, there can be key attributes that are defined as being of other data types. If so they must apply the same rules as defined here.

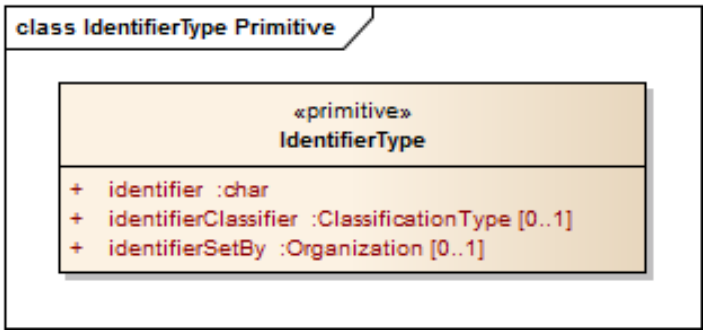
Note

There can also be attributes in the data model that are defined as being of data type IdentifierType. Such attributes must be treated in the same way as defined for multi-valued DescriptorType attributes. Refer to Para 5.

The IdentifierType primitive has three attributes:

- Firstly the "primary" identifier attribute
- Secondly, two optional characteristization attributes, identifierClassifier (ClassificationType <<primitive>>) and identifierSetBy (Organization).

These optional characterization attributes define uniqueness for the identifier attribute. Stated differently, multiple identifiers must have different values for the optional characterization attributes.



ICN-B6865-SX002D0016-001-01

Fig 7 IdentifierType primitive

Note

Individual S-Series ILS specifications can choose to deprecate (restrict) the use of one or more of the IdentifierType characterization attributes (identifierClassifier and identifierSetBy) as part of their tailoring process. However, these kinds of tailoring do not affect the principles defined here, but just limit the described scenarios.

6.5 IdentifierType update rules

If a multi-valued identifier is defined, without an associated applicability statement, then the following rules must be applied for update messages:

- Any identifier values in the receiving system must be overwritten by the identifier values in the update message (if provided), if the optional characterization attributes in the update message match the optional characterization attributes of an already existing identifier.
- If no element for an identifier value attribute is provided, then the existing values must be unchanged
- ALL existing identifier values for a specific identifier attribute may NOT be deleted, there must be at least one identifier defined.

6.6 Identifier update examples

In order to illustrate the update rules, the exampleClassIdentifier IdentifierType attribute as defined for the MultiValuedExampleClass in Para 2 is used.

The MultiValuedExampleClass can have 1 or many identifiers (ie, instances of its exampleClassIdentifier attribute). Each identifier can have 0 or 1 associated identifierClassifier determining the type of identifier being provided, and/or 0 or 1 organization (identifierSetBy) defining the organization that "owns" the identifier. If multiple identifiers are defined, then the values for the optional characterization attributes identifierClassifier and identifierSetBy, must be different.

An update message for the MultiValuedExampleClass must have one identifier (ie, instance of exampleClassIdentifier) which locates the instance of MultiValuedExampleClass that is to be updated. This means that there must be an exact match between the attributes for the exampleClassIdentifier (identifier, identifierClassifier and identifierSetBy) in the update message and an existing exampleClassIdentifier in the receiving system. This identifier must be defined as the first identifier for the MultiValuedExampleClass in the update message.

6.6.1 Adding an identifier for multi-valued identifiers

The <`multiValuedExampleClass`> instance that is to be updated in this example has one identifier ie, one instance of the <`classId`> (exampleClassIdentifier) IdentifierType attribute.

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
      <class>Type1</class>
   </classId>
</multiValuedExampleClass>
```

Note

In the example, above, the instance of exampleClassIdentifier (<`classId`>) is determined to be a "Type1" identifier.

If an instance of the exampleClassIdentifier attribute (<`classId`>) in an update message do not match with an existing set of IdentifierType values (identifier, identifierClassifier and identifierSetBy) in the receiving system, then the exampleClassIdentifier in the update must be inserted, with its related additional characterizations.

The following update message will then add two additional identifiers (instances of <`classId`>) to the instance of <`multiValuedExampleClass`> defined above:

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
      <class>Type1</class>
   </classId>
```

```
    <classId>
        <id>X11</id>
        <class>Type2</class>
    </classId>
    <classId>
        <id>Q1-A</id>
        <class>Type3</class>
    </classId>
</multiValuedExampleClass>
```

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
    <classId>
        <id>1</id>
        <class>Type1</class>
    </classId>
    <classId>
        <id>X11</id>
        <class>Type2</class>
    </classId>
    <classId>
        <id>Q1-A</id>
        <class>Type3</class>
    </classId>
</multiValuedExampleClass>
```

6.6.2 Deleting an identifier for multi-valued identifiers

Identifier values of a class may be deleted via an update action message. Deleting an identifier as part of an update action (crud code) requires that the class instance in the receiving system must have two or more identifiers, since a business object must have at least one identifier defined.

The class instance, for which the identifier is to be deleted, must have at least two instances of the identifier attribute in the update message. Firstly, the identifier that locates the class instance that is to be updated and secondly, the identifier(s) that is to be deleted. The identifier that is to be deleted must have an exact match for its optional characterization attributes (identifierClassifier and identifierSetBy) and its identifier attribute set to `xsi:nil="true"`.

The `<multiValuedExampleClass>` instance that is to be updated in this example has three identifiers ie, three instances of the `<classId>` (exampleClassIdentifier) IdentifierType attribute.

```
<multiValuedExampleClass>
    <classId>
        <id>1</id>
        <class>Type1</class>
    </classId>
    <classId>
        <id>X11</id>
        <class>Type2</class>
    </classId>
    <classId>
        <id>Q1-A</id>
```

```
      <class>Type3</class>
   </classId>
</multiValuedExampleClass>
```

Note

In the multiValueExampleClass the business object above, may be identified by either or the three identifier values: "1", "X11", or "Q1-A" with each of the identifier values having unique characterizations of class type: "Type1", "Type2", "Type3", respectively.

The following update message will delete two of the identifiers (instances of <classId>) from the instance of <multiValuedExampleClass> defined above:

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>X11</id>
      <class>Type2</class>
   </classId>
   <classId>
      <id
         xsi:nil="true"/>
      <class>Type1</class>
   </classId>
   <classId>
      <id
         xsi:nil="true"/>
      <class>Type3</class>
   </classId>
</multiValuedExampleClass>
```

Note

The previous update message locates the business object to be updated via the identifier, "X11", of Type2 classification.

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>X11</id>
      <class>Type2</class>
   </classId>
</multiValuedExampleClass>
```

6.6.3 Key field (business object identifier) change

To change an identifier of a business object, the following pattern should be applied:

1. Locate the business object via its existing identifier value (including all its optional characterization attributes)
2. Update the identifier value by entering second identifier instance with the new value, including all its optional characterization attributes.

Note

After the business object is located with an identifier value (and its applicable optional characterization attribute values), the specific identifier of the business object that is to be

changed must be presented as an attribute of the class instance in the update message. If this identifier has related optional characterization attributes, they must be provided also.

Note

The key field change can change one or many of the identifier attributes in terms of its identifier string, identifierClassifier and/or identifierSetBy, ie, a key field change does not always have to change the identifier string value, it can just be a change for its optional characterization attributes.

Note

In different data standards depicting logistics support analysis and provisioning data (eg, Mil-Std-1388-2B and GEIA-STD-0007), business objects were identified with one and only one set of identification attributes. In addition, these identifier values were replicated as part of the identifier in related business objects. For these data standards, there is required special logic to be implemented to change a business object identifier.

In GEIA-STD-0007, Revision C, the parent and related business object identifiers are referenced, not repeated in the identification of a business object.

6.6.3.1 Change the identifier value

In the first example there is a key field change that changes the identifier, but not its associated optional characterization attributes. The `<multiValuedExampleClass>` in the following example changes the "Type1" identifier (`<id>` element) from "`XYZ.11`" to "`XYZ-11`". The instance of `<multiValuedExampleClass>` to be changed looks like:

```
<multiValuedExampleClass>
   <classId>
      <id>XYZ.11</id>
      <class>Type1</class>
   </classId>
</multiValuedExampleClass>
```

The following update message will change the identifier (`<id>`) value to "`XYZ-11`":

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>XYZ.11</id>
      <class>Type1</class>
   </classId>
      <classId>
         <id>XYZ-11</id>
         <class>Type1</class>
   </classId>
</multiValuedExampleClass>
```

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>XYZ-11</id>
      <class>Type1</class>
   </classId>
</multiValuedExampleClass>
```

Note

Two instances of an identifierType attribute cannot exist with the exact same set of optional characterization attributes.

6.6.3.2 Change the optional characterization attributes but not the identifier value

The second example just updates the associated optional characterization attibute (<class>) value. The <multiValuedExampleClass> in the following example changes the classification (determination) for the "XYZ-11" identifier (<id> element) from "Type1" to "Type2". The instance of <multiValuedExampleClass> to be changed looks like:

```
<multiValuedExampleClass>
   <classId>
      <id>XYZ-11</id>
      <class>Type1</class>
   </classId>
</multiValuedExampleClass>
```

The following update message will change the optional characterization attributes for the identifier with (<id>) value to "XYZ-11":

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>XYZ-11</id>
      <class>Type1</class>
   </classId>
   <classId>
      <id
         xsi:nil="true"/>
      <class>Type1</class>
   </classId>
   <classId>
      <id>XYZ-11</id>
      <class>Type2</class>
   </classId>
</multiValuedExampleClass>
```

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>XYZ-11</id>
      <class>Type2</class>
   </classId>
</multiValuedExampleClass>
```

Note

If the deletion of an identifier of a business object results in 0 identifiers for the business object in the receiving system, then this would cause an error, because all the business objects at that are identified as key must have at least one identifier value.

Note

The identifier with the optional characterization attributes that is to be changed must be deleted, otherwise would the new identifier be regarded to be an additional identifier.

Note

The business object that is to be updated must first be located using one of its identifier values. The identifier used to locate the business object can be any of its existing identifiers, and must not be the identifier to be changed, if the business object has multiple identifiers.

7 Multi-valued attributes using Applicability

Dependent on the specification, an additional applicability expression may be assigned to primitive type elements eg, DescriptorType, PropertyType and ClassificationType. There can only be one ApplicabilityStatement assigned to each instance of a multi-valued attribute. If applicability is assigned to the multi-valued attribute, that applicability must be included in the update messages with the multi-valued attribute values and be regarded as any other optional characterization attributes for the attribute under consideration.

A simple applicability statement in accordance with the SX002D Common Data Model XML schema can be represented in XML as:

```
<applic>
   <applicDef>
      <applicDescr>
         <descr>Arctic conditions</descr>
      </applicDescr>
   </applicDef>
</applic>
```

Note

The XML schema definition for applicability statement is not included in this document. See the respective S-Series ILS Specifications XML schema for details.

If the receiving system contains an instance of <`multiValuedExampleClass`>, where an instance of its exampleClassIdentifier attribute (<`classId`>) has a defined applicability as an optional characterization attribute, then its applicability must be included in the information for that attribute in order to update its classification.

The <`multiValuedExampleClass`> instance that is to be updated in this example has two instances of the <`classClassif`> ClassificationType attribute, each with an associated applicability statement:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classClassif>
      <code>CODE1</code>
      <applic>
         <applicDef>
            <applicDescr>
               <descr>Arctic conditions</descr>
            </applicDescr>
         </applicDef>
      </applic>
   </classClassif>
   <classClassif>
      <code>CODE2</code>
      <applic>
```

```
            <applicDef>
               <applicDescr>
                  <descr>Desert conditions</descr>
               </applicDescr>
            </applicDef>
         </applic>
      </classClassif>
   </multiValuedExampleClass>
```

If the desired update is to change one the classifications for desert conditions from "CODE2" to "CODE3" then the applicability statement must be included as the characterization for the classification in order to determine the classification to be updated, ie, the update message must look like:

```
<multiValuedExampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <classClassif>
      <code>CODE3</code>
      <applic>
         <applicDef>
            <applicDescr>
               <descr>Desert conditions</descr>
            </applicDescr>
         </applicDef>
      </applic>
   </classClassif>
</multiValuedExampleClass>
```

The dataset at the receiving end must then result in:

```
<multiValuedExampleClass>
   <classId>
      <id>1</id>
   </classId>
   <classClassif>
      <code>CODE1</code>
         <applic>
            <applicDef>
               <applicDescr>
                  <descr>Arctic conditions</descr>
               </applicDescr>
            </applicDef>
         </applic>
   </classClassif>
   <classClassif>
      <code>CODE3</code>
      <applic>
         <applicDef>
            <applicDescr>
```

```
                <descr>Desert conditions</descr>
            </applicDescr>
        </applicDef>
    </applic>
  </classClassif>
</multiValuedExampleClass>
```

Note

There are too many cases where applicability statements can be used for them to be explained here. However, if an applicability statement is assigned to a multi-valued attribute then that applicability must still be regarded as any other optional characterization attribute for the attribute under consideration.

Chapter 2.4

XML schema implementation guidance - Managing multi-valued associations in an update message

Table of contents

Page

List of tables

List of figures

References

Table 1 References

Chap No./Document No.	Title
SX002D	Common data model for the S-Series ILS specifications

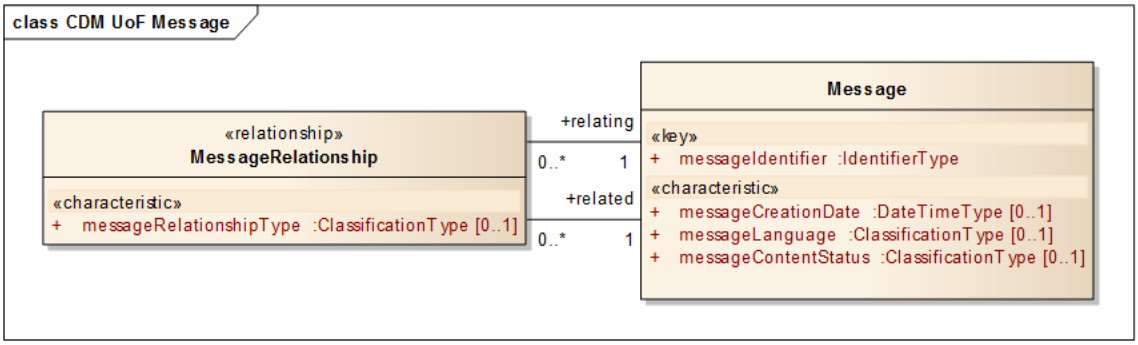
1 Introduction

Regular associations are represented using <<relationship>> classes which establishes associations between pairs of relating/related business objects.

Each <<relationship>> class always include the identification of the related class, either by identifier values for the related object, or by a uidRef value which uniquely identifies a class instance within the message. It can also have additional related characteristic attributes and an associated applicability statement, which adds characteristics to the association.

Each <<relationship>> class also has an associated crud attribute, which enables change information to be associated with individual relationship instances as part of an update message.

An example of an association in in SX002D, Common Data Model, is MessageRelationship which establishes associations between instances of Message (). MessageRelationship also includes a characterization attribute, messageRelationshipType, to further characterize the association with respect to the type of relationship. This also enables multiple relationships between two Message instances. Refer to Fig 1.



ICN-B6865-SX005G0004-001-00

Fig 1 SX002D Message and MessageRelationship

Associations are mostly multi-valued ie, the class that owns the association can have one or many associated <<relationship>> classes. The example in Fig 2 defines that an instance of Message can have zero, one or many associated Messages via the relating association with the MessageRelationship <<relationship>> class.

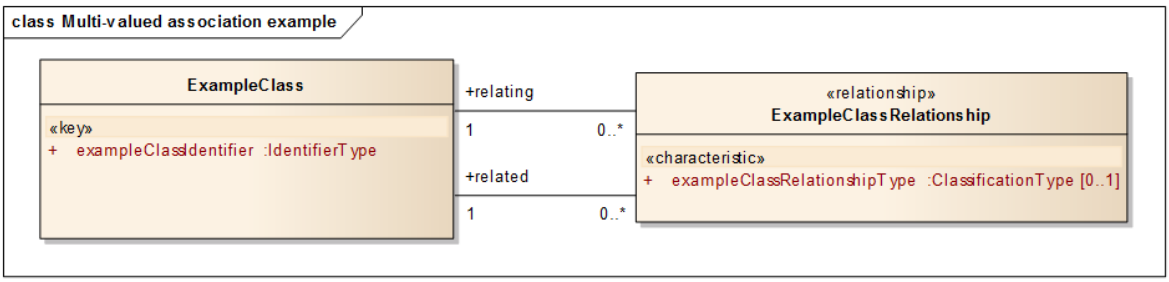
Note

The combination of the relating and the related business object identifiers defines the uniqueness of the association.

2 An example of a multi-valued association using a <<relationship>> class

The following sections use the ExampleClass and its associated ExampleClassRelationship <<relationship>> class, to demonstrate how multi-valued associations must be managed in update messages.

The example UML classes to be used are illustrated in Fig 2.



ICN-B6865-SX005G0005-001-00

Fig 2 ExampleClass <<relationship>>

The flattened XML schema representation for the ExampleClass and ExampleClassRelationship classes, in accordance with the S-Series ILS specifications XML Schema Authoring Rules, will look like:

```
<!—START OF THE exampleClass XML SCHEMA -->


<xsd:complexType
   name="exampleClass">
   <xsd:annotation>
      <xsd:appinfo>SX001G:ExampleClass</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>

<!--UML-Attributes-->

      <xsd:element
         name="classId"
         type="exampleClassIdentifier"/>

      <!--UML-EmbAssocs-->

      <xsd:element
         name="relClass"
         type="exampleClassRelationship"
         minOccurs="0"
         maxOccurs="unbounded"
         nillable="true"/>
   </xsd:sequence>
   <xsd:attribute
      name="uid"
      use="optional">
      <xsd:simpleType>
         <xsd:restriction
            base="xsd:ID">
            <xsd:pattern
               value="ec[1-9][0-9]*"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:attribute>
   <xsd:attribute
      name="crud"
      type="crudCodeValues"
      default="I"/>
</xsd:complexType>
<xsd:complexType
   name="exampleClassRelationship">
   <xsd:annotation>
      <xsd:appinfo>SX001G:ExampleClassRelationship</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>

   <!--UML-Attributes-->

      <xsd:element
```

```
            name="relType"
            type="exampleClassRelationshipType"
            minOccurs="0"
            nillable="true"/>

         <!--UML-EmbAssocs-->

         <xsd:element
            name="ecRef"
            type="exampleClassReference"/>
      </xsd:sequence>
      <xsd:attribute
         name="uid"
         use="optional">
         <xsd:simpleType>
            <xsd:restriction
               base="xsd:ID">
               <xsd:pattern
                  value="ecr[1-9][0-9]*"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute
         name="crud"
         type="crudCodeValues"
         default="I"/>
   </xsd:complexType>
   <xsd:complexType
      name="exampleClassIdentifier">
      <xsd:annotation>
         <xsd:appinfo>SX001G:exampleClassIdentifier</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="id"
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
            name="class"
            type="value:classExampleIdentifierClassValues"
            minOccurs="0"
            nillable="true"/>
         <xsd:element
            name="setBy"
            type="organizationReference"
            minOccurs="0"
            nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType
      name="exampleClassRelationshipType">
      <xsd:annotation>
```

```
      <xsd:appinfo>SX001G:exampleClassRelationshipType</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
      <xsd:element
         name="code"
         type="value:exampleClassRelationshipTypeCodeValues"
         nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>

   <!-- Supporting XML schema elements -->

   <xsd:complexType name="exampleClassReference">
      <xsd:sequence
         minOccurs="0">
         <xsd:element
            name="classId"
            type="exampleClassIdentifier"/>
      </xsd:sequence>
      <xsd:attribute
         name="uidRef"
         use="optional">
         <xsd:simpleType>
            <xsd:restriction
               base="uidRef">
               <xsd:pattern value="ec[1-9][0-9]*"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
      <xsd:assert
         test="(@uidRef and not(classId)) or (not(@uidRef) and
   classId)"/>
   </xsd:complexType>
   <xsd:complexType
      name="organizationReference">
      <xsd:sequence>
         <xsd:element
            name="orgId"
            type="organizationIdentifier"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType
      name="organizationIdentifier">
      <xsd:annotation>
         <xsd:appinfo>SX001G:organizationIdentifier</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="id"
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
```

```
            name="class"
            type="value:organizationIdentifierClassValues"
            minOccurs="0" nillable="true" default="CAGE"/>
         <xsd:element
            name="setBy"
            type="organizationReference"
            minOccurs="0"
            maxOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:simpleType
      name="nonEmptyString">
      <xsd:restriction
         base="xsd:token">
         <xsd:minLength
            value="1"/>
      </xsd:restriction>
   </xsd:simpleType>


      <!—END OF THE exampleClass XML SCHEMA -->
```

Note

SimpleTypes defining allowed valid values and crud values are not included in the XML schema above.

Note

References in S-Series ILS Specifications XML schemas can be made using either <<key>> information or the uidRef (IDREF) attribute.

3 Association update rules

3.1 Introduction

In order to illustrate the update rules, SX00the XML schema for ExampleClass and ExampleClassRelationship defined in Para 2 is used.

The <exampleClass> instance that is to be updated in this example contains one identifier (classId):

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
</exampleClass>
```

Note

All additional exampleClass attributes defined in previous Paras are omitted due to keeping the examples as simple as possible.

3.2 Insert an <<relationship>> association

If the desired update is to add a relationship from exampleClass instance with the identifier value set to "1", to another exampleClass instance with identifier value set to "5" then the update message must look like:

```
<exampleClass
   crud="I">
```

```
      <classId>
         <id>5</id>
      </classId>
   </exampleClass>
   <exampleClass
      crud="U">
      <classId>
         <id>1</id>
      </classId>
      <relClass
         crud="I">
         <relType>
            <code>Type1Relationship</code>
         </relType>
         <ecRef>
            <classId>
               <id>5</id>
            </classId>
         </ecRef>
      </relClass>
   </exampleClass>
```

Note

The class to which the relationship is added has the crud value set to "U" (Update) and the relationship itself has the crud value set to "I" (Insert).

The dataset at the receiving end must then result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
   <relClass>
      <relType>
         <code>Type1Relationship</code>
      </relType>
      <ecRef>
         <classId>
            <id>5</id>
         </classId>
      </ecRef>
   </relClass>
</exampleClass>
<exampleClass>
   <classId>
      <id>5</id>
   </classId>
</exampleClass>
```

Note

Information about the relationship is embedded within the class instance that owns the relationship.

An update message for adding an additional relationship from exampleClass instance with the identifier value set to “1", to another exampleClass instance with identifier value set to "4" must look like:

```
<exampleClass
   crud="I">
   <classId>
      <id>4</id>
   </classId>
</exampleClass>
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <relClass
   crud="I">
      <relType>
         <code>Type1Relationship</code>
      </relType>
      <ecRef>
         <classId>
            <id>4</id>
         </classId>
      </ecRef>
   </relClass>
</exampleClass>
```

The dataset at the receiving end must then result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
   <relClass>
      <relType>
         <code>Type1Relationship</code>
      </relType>
      <ecRef>
         <classId>
            <id>4</id>
         </classId>
      </ecRef>
   </relClass>
   <relClass>
      <relType>
         <code>Type1Relationship</code>
      </relType>
      <ecRef>
         <classId>
            <id>5</id>
         </classId>
      </ecRef>
```

```
   </relClass>
</exampleClass>
<exampleClass>
   <classId>
      <id>4</id>
</classId>
</exampleClass>
<exampleClass>
   <classId>
      <id>5</id>
   </classId>
</exampleClass>
```

3.3 Delete a single <<relationship>> association

Deleting a single <<relationship>> association is done by providing the relationship instance and set the crud value for the relationship to "D" (Delete).

The generic rule is that more than one relationship instances for the same two relating and related class instances cannot exist at the same time. Therefore, it is enough to provide the identifier for the class that is related by the relationship.

Note

If multiple relationships exist at the same time between the same pair of relating and related class instances, then these relationships must be distinguished by applicability statements. Therefore, the applicability statement must be part of the information that must be provided for a <<relationship>> delete action.

If the desired update is to delete a single relationship instance, then the update message must look like:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <relClass
      crud="D">
      <ecRef>
         <classId>
            <id>4</id>
         </classId>
      </ecRef>
   </relClass>
</exampleClass>
```

If this update is applied to the resulting dataset in Para 3.2, then the updated dataset at the receiving end must result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
   <relClass>
      <relType>
         <code>Type1Relation</code>
```

```
            </relType>
            <ecRef>
               <classId>
                  <id>5</id>
               </classId>
            </ecRef>
         </relClass>
      </exampleClass>
      <exampleClass>
         <classId>
            <id>4</id>
         </classId>
      </exampleClass>
      <exampleClass>
         <classId>
            <id>5</id>
         </classId>
      </exampleClass>
```

Note

Deleting the relationship between one business object and another does not affect the existence of the related business object.

3.4 Delete all existing <<relationship>> associations

Deletion of existing associations can be performed by setting the association element to `xsi:nil="true"`.

The following update message will delete all instances of exampleClassRelationship defined for exampleClass with id 1 in the dataset described in Para 3.2.

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <relClass
      xsi:nil="true"/>
</exampleClass>
```

The dataset at the receiving end must then result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
</exampleClass>
<exampleClass>
   <classId>
      <id>4</id>
   </classId>
</exampleClass>
<exampleClass>
   <classId>
      <id>5</id>
```

```
   </classId>
</exampleClass>
```

Note

Deleting the relationship does not mean that the related instance is to be deleted.

3.5 Update the characterizations for an association

Updating the characterizations for a <<relationship>> association is done by providing the relationship instance and set the crud value for the relationship to "U" (Update).

As mentioned in Para 3.3, the general rule is that more than one relationship instance for the same two relating and related class instances cannot exist at the same time. Therefore, it is enough to provide the identifier for the class that is related by the relationship.

Note

If multiple relationships exist at the same time between the same pair of relating and related class instances, then the multiple relationships must be distinguished by an applicability statement. Therefore, the applicability statement must also be part of the information that must be provided for a <<relationship>> update action.

If the desired update is to change the characterization attribute relationship type (from “Type1Relation” to a “Type2Relation”) for the relationship between ExampleClass with identifier value "1" and the ExampleClass with identifier value "4", then the update message must look like:

```
<exampleClass
   crud="N">
   <classId>
      <id>1</id>
   </classId>
   <relClass
      crud="U">
      <relType>
         <code>Type2Relationship</code>
      </relType>
      <ecRef>
         <classId>
            <id>4</id>
         </classId>
      </ecRef>
   </relClass>
</exampleClass>
```

Note

The crud action code for the exampleClass is defined as "N" (non-changed) due to the fact there is no change to the exampleClass instance as such. The relationship with the related exampleClass instance still exists.

If this update is applied to the resulting dataset in Para 3.2, then the updated dataset at the receiving end must result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
   <relClass>
```

```
            <relType>
               <code>Type2Relationship</code>
            </relType>
            <ecRef>
               <classId>
                  <id>4</id>
               </classId>
            </ecRef>
         </relClass>
         <relClass>
            <relType>
               <code>Type1Relationship</code>
            </relType>
            <ecRef>
               <classId>
                  <id>5</id>
               </classId>
            </ecRef>
         </relClass>
      </exampleClass>
      <exampleClass>
         <classId>
            <id>4</id>
         </classId>
      </exampleClass>
      <exampleClass>
         <classId>
            <id>5</id>
         </classId>
      </exampleClass>
```

4 Relationship associations with Applicability

If multiple relationships exist at the same time between the same pair of relating and related class instances, then these multiple instances must be distinguished by applicability statements.

Therefore, the applicability statement must also be part of the information that must be provided for <<relationship>> delete and update actions, respectively.

Note

The same logic as being defined for Multi-valued attributes using Applicability (refer to Chap 2.3) applies to <<relationship>> associations.

Chapter 2.5

XML schema implementation guidance - Managing composition aggregations in an update message

Table of contents

Page

List of tables

List of figures

References

Table 1 References

Chap No./Document No.	Title
SX002D	Common data model for the S-Series ILS specifications

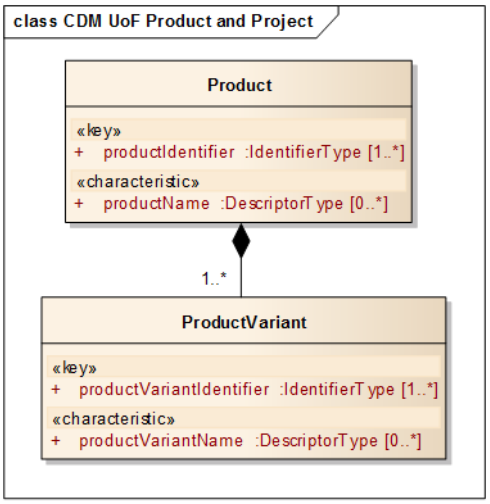
1 Introduction

A composition aggregation is a form of an aggregation relationship, where the children are dependent upon the lifecycle for the parent class ie, the child class instances cannot outlive its parent class instance.

Things worth emphasizing for composition aggregation associations are:

- The class instance that represents the part (child) in a composition association is always embedded as part of the whole (parent) class instance
- Composition associations in themselves do not have any characterizations or applicability statements.

An example of a composition aggregation association in SX002D, Common Data Model, is the association between Product and ProductVariant. This model defines that a ProductVariant must be defined in the context of a Product.



ICN-B6865-SX005G0006-001-00

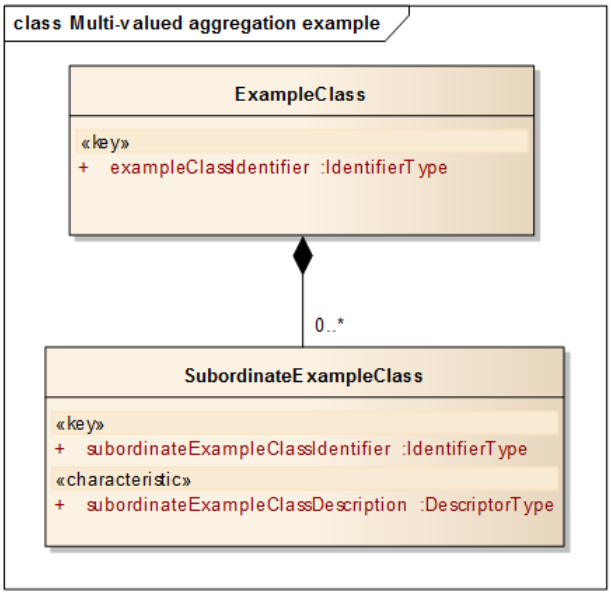
Fig 1 SX002D Product and ProductVariant

Composition aggregation associations are mostly multi-valued ie, the class that owns the association can have one or many associated part class instances. The example in Fig 1 defines that an instance of Product can have one or many associated ProductVariants via the composition aggregation association.

2 An example of a multi-valued composition aggregation

The following sections use the ExampleClass and SubordinateExampleClass classes, to demonstrate how multi-valued composition aggregation associations must be managed in update messages.

The example UML classes to be used are illustrated in Fig 2.



ICN-B6865-SX005G0007-001-00

Fig 2 ExampleClass compostion aggregation

The flattened XML schema representation for the ExampleClass and SubordinateExampleClass classes, in accordance with the S-Series ILS specifications XML Schema Authoring Rules, will look like:

```
   <!—START OF THE exampleClass XML SCHEMA -->

<xsd:complexType name="exampleClass">
   <xsd:annotation>
      <xsd:appinfo>SX001G:ExampleClass</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>

   <!--UML-Attributes-->

      <xsd:element
         name="classId"
         type="exampleClassIdentifier"/>

      <!--UML-CompAssocs-->

      <xsd:element
         name="compClass"
         type="subordinateExampleClass"
         minOccurs="0"
         maxOccurs="unbounded"
         nillable="true"/>
   </xsd:sequence>
   <xsd:attribute
      name="uid"
      use="optional">
      <xsd:simpleType>
         <xsd:restriction
```

```
                base="xsd:ID">
                <xsd:pattern
                value="ec[1-9][0-9]*"/>
             </xsd:restriction>
          </xsd:simpleType>
       </xsd:attribute>
       <xsd:attribute
          name="crud"
          type="crudCodeValues"
          default="I"/>
    </xsd:complexType>
    <xsd:complexType
       name="subordinateExampleClass">
       <xsd:annotation>
          <xsd:appinfo>SX001G:SubordinateExampleClass</xsd:appinfo>
       </xsd:annotation>
       <xsd:sequence>

          <!--UML-Attributes-->

          <xsd:element
             name="classId"
             type="subordinateExampleClassIdentifier"/>
          <xsd:element
             name="classDescr"
             type="subordinateExampleClassDescription"
             minOccurs="0"
             nillable="true"/>
       </xsd:sequence>
       <xsd:attribute
          name="uid"
          use="optional">
          <xsd:simpleType>
             <xsd:restriction
                base="xsd:ID">
                <xsd:pattern
                   value="subec[1-9][0-9]*"/>
             </xsd:restriction>
          </xsd:simpleType>
       </xsd:attribute>
       <xsd:attribute
          name="crud"
          type="crudCodeValues" default="I"/>
    </xsd:complexType>
    <xsd:complexType
       name="exampleClassIdentifier">
       <xsd:annotation>
          <xsd:appinfo>SX001G:exampleClassIdentifier</xsd:appinfo>
       </xsd:annotation>
       <xsd:sequence>
          <xsd:element
             name="id"
```

```
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
            name="class"
            type="value:classExampleIdentifierClassValues"
            minOccurs="0"
            nillable="true"/>
         <xsd:element
            name="setBy"
            type="organizationReference"
            minOccurs="0"
            nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType
      name="subordinateExampleClassIdentifier">
      <xsd:annotation>
         <xsd:appinfo>SX001G:subordinateExampleClassIdentifier
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="id"
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
            name="class"
            type="value:classExampleIdentifierClassValues"
            minOccurs="0"
            nillable="true"/>
         <xsd:element
            name="setBy"
            type="organizationReference"
            minOccurs="0"
            nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType
      name="subordinateExampleClassDescription">
      <xsd:annotation>
         <xsd:appinfo>SX001G:subordinateExampleClassDescription
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="descr"
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
            name="lang"
            type="value:languageCodeValues"
            minOccurs="0"
```

```
            nillable="true"/>
         <xsd:element
            name="date"
            type="xsd:date"
            minOccurs="0"
            nillable="true"/>
         <xsd:element
            name="providedBy"
            type="organizationReference"
            minOccurs="0"
            nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>

   <!-- Supporting XML schema elements -->

   <xsd:complexType
      name="organizationReference">
      <xsd:sequence>
         <xsd:element
            name="orgId"
            type="organizationIdentifier"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType
      name="organizationIdentifier">
      <xsd:annotation>
         <xsd:appinfo>SX001G:organizationIdentifier</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="id"
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
            name="class"
            type="value:organizationIdentifierClassValues"
            minOccurs="0"
            nillable="true"
            default="CAGE"/>
         <xsd:element
            name="setBy"
            type="organizationReference"
            minOccurs="0"
            maxOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
      <xsd:simpleType
            name="nonEmptyString">
         <xsd:restriction
            base="xsd:token">
            <xsd:minLength
```

```
            value="1"/>
         </xsd:restriction>
      </xsd:simpleType>

      <!—END OF THE exampleClass XML SCHEMA -->
```

Note

SimpleTypes defining allowed valid values and crud values are not included in the XML schema above.

3 Composition aggregation update rules

3.1 Introduction

In order to illustrate the update rules, the XML schema for ExampleClass and SubordinateExampleClass defined in Para 2 is used.

The <exampleClass> instance that is to be updated in this example contains one identifier (classId):

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
</exampleClass>
```

Note

All additional exampleClass attributes and relationships defined in previous paragraphs are omitted due to keeping the examples as simple as possible.

3.1.1 Insert a composition aggregation class instance

If the desired update is to add a subordinateExampleClass instance with the identifier value set to "A", to the exampleClass instance with identifier value set to "1" then the update message must look like:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <compClass
      crud="I">
      <classId>
         <id>A</id>
      </classId>
      <classDescr>
         <descr>A text description</descr>
      </classDescr>
   </compClass>
</exampleClass>
```

Note

The class to which the composition instance is added has the crud value set to "U" (Update) and the added composition instance itself has the crud value set to "I" (Insert).

The dataset at the receiving end must then result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
   <compClass>
      <classId>
         <id>A</id>
      </classId>
      <classDescr>
         <descr>A text description</descr>
      </classDescr>
   </compClass>
</exampleClass>
```

An update message for adding an additional composition instance with identifier value "B" to the exampleClass instance with the identifier value set to "1" must look like:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <compClass
      crud="I">
      <classId>
         <id>B</id>
      </classId>
      <classDescr>
         <descr>B description</descr>
      </classDescr>
   </compClass>
</exampleClass>
```

The dataset at the receiving end must then result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
   <compClass>
      <classId>
         <id>A</id>
      </classId>
      <classDescr>
         <descr>A text description</descr>
      </classDescr>
   </compClass>
   <compClass>
      <classId>
         <id>B</id>
      </classId>
      <classDescr>
         <descr>B description</descr>
```

```
      </classDescr>
   </compClass>
</exampleClass>
```

3.2 Delete a single composition aggregation class instance

Deleting a single composition class instance is done by providing the identification for the part class instance and set the crud value for the relationship to "D" (Delete).

If the desired update is to delete the composition instance with identifier value "A" then must the update message look like:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <compClass
      crud="D">
      <classId>
         <id>A</id>
      </classId>
   </compClass>
</exampleClass>
```

If this update is applied to the resulting dataset in Para 3.1.1 then must the updated dataset at the receiving end result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
   <compClass>
      <classId>
         <id>B</id>
      </classId>
      <classDescr>
         <descr>B description</descr>
      </classDescr>
   </compClass>
</exampleClass>
```

3.3 Delete all existing composition aggregation class instances

Deletion of existing composition instances can be performed by setting the composition element to `xsi:nil="true"`.

The following update message will delete all instances of subordinateExampleClass defined for exampleClass with identifier value "1" in the dataset described in Para 3.1.1.

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <compClass
```

```
      xsi:nil="true"/>
</exampleClass>
```

The dataset at the receiving end must then result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
</exampleClass>
```

3.4 Update the characterizations for a composition instance

Updating the characterizations for a composition class instance is done by providing the relationship instance and set the crud value for the updated composition class instance to "U" (Update).

If the desired update is to change the characterization for the SubordinateClassInstance with identifier value "A" then the update message must look like:

```
<exampleClass
   crud="N">
   <classId>
      <id>1</id>
   </classId>
   <compClass
      crud="U">
      <classId>
         <id>A</id>
      </classId>
      <classDescr>
         <descr>An updated A description</descr>
      </classDescr>
   </compClass>
</exampleClass>
```

Note

The crud action code for the exampleClass is defined as "N" (non-changed) due to the fact there is no change to the exampleClass instance as such. Its part class SubordinateExampleClass with identifier value "A" still exists.

If this update is applied to the resulting dataset in Para 3.1.1 then the updated dataset at the receiving end must result in:

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
   <compClass>
      <classId>
         <id>A</id>
      </classId>
      <classDescr>
         <descr>An updated A description</descr>
      </classDescr>
```

```
      </compClass>
      <compClass>
         <classId>
            <id>B</id>
         </classId>
         <classDescr>
            <descr>B description</descr>
         </classDescr>
      </compClass>
   </exampleClass>
```

3.5 Composition aggregation classes with Applicability

A composition aggregation class in many cases may have an associated applicability statement. If so, the applicability statement must be part of the optional characterization attributes used to locate a specific business object. That is, the applicability statement must be provided together with one of the business object identifiers defined for the composition aggregation class (Refer to Chap 2.3)

Note

The same basic logic as being defined for Multi-valued attributes using Applicability (Refer to Chap 2.3) also applies to composition aggregation classes.

Chapter 2.6

XML schema implementation guidance - Managing attribute groups in an update message

Table of contents

Page

List of tables

List of figures

References

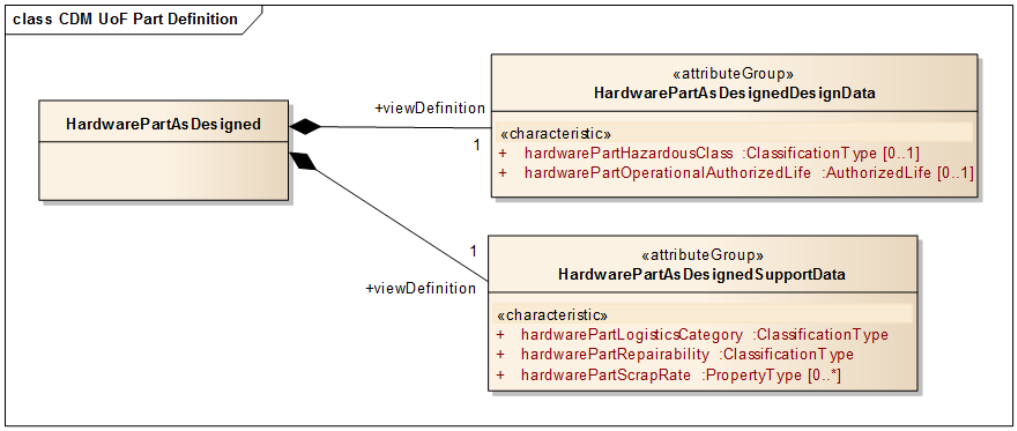
Table 1 References

Chap No./Document No.	Title
SX002D	Common data model for the S-Series ILS specifications

1 Introduction

An attribute group is typically used when the list of attributes defined for a class gets too extensive, and/or when there is some logic that is important to make explicit (eg, stakeholders such as "Design" vs "Support"). These attribute groups typically have the cardinality of one (1) ie, there is no other dependency between the attributes listed in the attribute group. An example where a UML class has single instance attribute groups is the business object HardwarePartAsDesigned in SX002D, the Common Data Model. The HardwarePartAsDesigned

class has two associated attribute groups, HardwarePartAsDesignedDesignData and HardwarePartAsDesignedSupportData, respectively

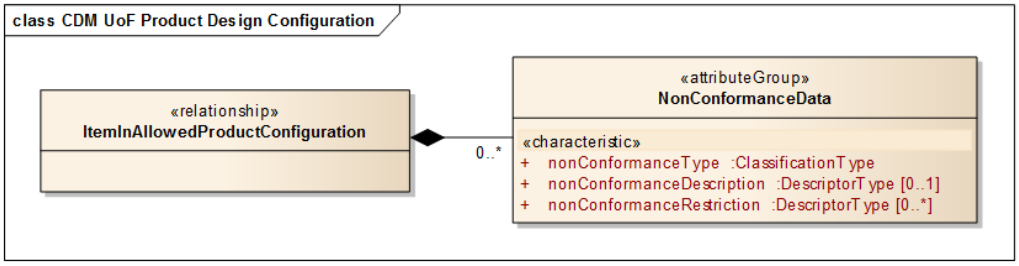


ICN-B6865-SX005G0008-001-00

Fig 1 SX002D HardwarePartAsDesigned and its HardwarePartAsDesignedDesignData and HardwarePartAsDesignedSupportData attribute groups

Attribute groups can also be used to group attributes where the individual attribute lose its meaning without being put in context of the others. This type of attribute group typically has the cardinality of many and may have a defined applicability for the group of attributes being defined rather than for individual attributes.

An example where a UML class has this type of attribute group is the business object ItemInAllowedProductConfiguration in SX002D, Common Data Model. An instance of the ItemInAllowedProductConfiguration class can have one or more sets of defined non confomance statements, each being defined by an instance of the NonConformanceData attribute group.



ICN-B6865-SX005G0009-001-00

Fig 2 SX002D ItemInAllowedProductConfiguration and its NonConformanceData attribute group

2 Single instance attribute group

Single instance attribute groups are enumerations of individual attributes, where each individual attribute must be updated the same way as defined for the respective primitive type in Chap 2.1 and Chap 2.3.

Note

Single instance attribute groups are represented using the XSD <group> element ie, there is no explicit element in an XML document that can be used to refer to the attribute group as such.

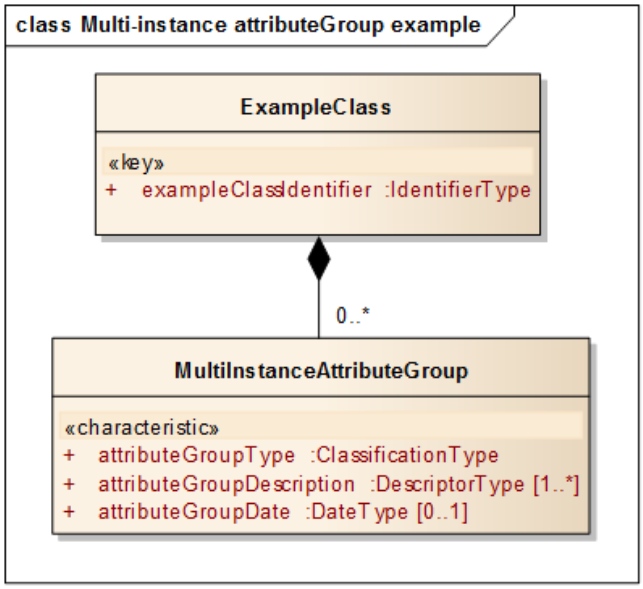
3 Multiple instance attribute group

Multiple instance attribute groups are sets of related attributes that should be updated together. If one attribute value is to be updated or deleted, all the non-blank attribute values in the multiple instance attribute group must be updated.

3.1 An example of a multi-instance attribute group

The following sections use the ExampleClass and MultiInstanceAttributeGroup classes, to demonstrate how multi-instance attribute groups must be managed in update messages.

The example UML classes to be used are illustrated in Fig 3.



ICN-B6865-SX005G0010-001-00

Fig 3 ExampleClass mult-instance attribute group

The flattened XML schema representation for the ExampleClass and its MultiValuedAttributeGroup, in accordance with the S-Series ILS specifications XML Schema Authoring Rules, will look like:

```
<!—START OF THE exampleClass XML SCHEMA -->


<xsd:complexType
   name="exampleClass">
   <xsd:annotation>
      <xsd:appinfo>SX001G:ExampleClass</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>

      <!--UML-Attributes-->

      <xsd:element
         name="classId"
         type="exampleClassIdentifier"/>

<!--UML-AttGroups-->
```

```
            <xsd:group
               ref="multiInstanceAttributeGroup"/>
         </xsd:sequence>
         <xsd:attribute
            name="uid"
            use="optional">
            <xsd:simpleType>
               <xsd:restriction
                  base="xsd:ID">
                  <xsd:pattern
                     value="ec[1-9][0-9]*"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute
            name="crud"
            type="crudCodeValues"
            default="I"/>
      </xsd:complexType>
      <xsd:group
         name="multiInstanceAttributeGroup">
         <xsd:annotation>
            <xsd:appinfo>SX001G:MultiInstanceAttributeGroup
            </xsd:appinfo>
         </xsd:annotation>
         <xsd:sequence>
            <xsd:element
               name="exampleClassAttrGroup"
               minOccurs="0"
               maxOccurs="unbounded"
               nillable="true">
      <xsd:complexType>
         <xsd:sequence>

            <!--UML-Attributes-->

            <xsd:element
               name="attrGroupType"
               type="attributeGroupType"/>
            <xsd:element
               name="attrGroupDescr"
               type="attributeTypeDescriptionDescription"
               maxOccurs="unbounded"
               nillable="true"/>
            <xsd:element
               name="attrGroupDate"
               type="attributeGroupTypeDate"
               minOccurs="0"
               nillable="true"/>
         </xsd:sequence>
      </xsd:group>
```

```
<xsd:complexType
   name="attributeGroupDate">
   <xsd:annotation>
      <xsd:appinfo>SX001G:attributeGroupDate</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="date"
         type="xsd:date"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType
   name="attributeGroupDescription">
   <xsd:annotation>
      <xsd:appinfo>SX001G:attributeGroupDescription</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="descr"
         type="nonEmptyString"
         nillable="true"/>
      <xsd:element
         name="lang"
         type="value:languageCodeValues"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="date"
         type="xsd:date"
         minOccurs="0"
         nillable="true"/>
      <xsd:element
         name="providedBy"
         type="organizationReference"
         minOccurs="0"
         nillable="true"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType
   name="attributeGroupType">
   <xsd:annotation>
      <xsd:appinfo>SX001G:attributeGroupType</xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="code"
         type="value:attributeGroupTypeCodeValues"
         nillable="true"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType
   name="exampleClassIdentifier">
```

```
      <xsd:annotation>
         <xsd:appinfo>SX001G:exampleClassIdentifier</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="id"
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
            name="class"
            type="value:classExampleIdentifierClassValues"
            minOccurs="0"
            nillable="true"/>
         <xsd:element
            name="setBy"
            type="organizationReference"
            minOccurs="0"
            nillable="true"/>
      </xsd:sequence>
   </xsd:complexType>

   <!-- Supporting XML schema elements -->

   <xsd:complexType
      name="organizationReference">
      <xsd:sequence>
         <xsd:element
            name="orgId"
            type="organizationIdentifier"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType
      name="organizationIdentifier">
      <xsd:annotation>
         <xsd:appinfo>SX001G:organizationIdentifier</xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element
            name="id"
            type="nonEmptyString"
            nillable="true"/>
         <xsd:element
            name="class"
            type="value:organizationIdentifierClassValues"
            minOccurs="0"
            nillable="true"
            default="CAGE"/>
         <xsd:element
            name="setBy
            type="organizationReference"
            minOccurs="0"
            maxOccurs="0"/>
```

```
   </xsd:sequence>
</xsd:complexType>
<xsd:simpleType
   name="nonEmptyString">
   <xsd:restriction
      base="xsd:token">
      <xsd:minLength
         value="1"/>
   </xsd:restriction>
</xsd:simpleType>

   <!—END OF THE exampleClass XML SCHEMA -->
```

Note

SimpleTypes defining allowed valid values and crud values are not included in the XML schema above.

3.2 Multi-instance attribute group update rules

3.2.1 Introduction

In order to illustrate the update rules, the XML schema for ExampleClass and MultiInstanceAttributeGroup defined in Para 6.3.1 is used.

The <exampleClass> instance that is to be updated in this example contains one identifier (<classId>):

```
<exampleClass>
   <classId>
      <id>1</id>
   </classId>
</exampleClass>
```

Note

All additional exampleClass attributes and relationships defined in previous Paras are omitted due to keeping the examples as simple as possible.

3.2.2 Insert a multi-instance attribute group instance

If the desired update is to add a first instance of the attributeGroup to the exampleClass instance with identifier value set to "1" then the update message must look like:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <exampleClassAttrGroup>
      <attrGroupType>
         <code>A</code>
      </attrGroupType>
      <attrGroupDescr>
         <descr>Attribute group A instance description.</descr>
      </attrGroupDescr>
   </exampleClassAttrGroup>
</exampleClass>
```

Adding another instance of the attributeGroup to the exampleClass instance with identifier value set to "1" must be implemented by providing ALL attributes for ALL instances of the multiple instance attribute group defined for the business object, along with all the implemented interfaces of the attributes in the attribute group. The update message must look like:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <exampleClassAttrGroup>
      <attrGroupType>
         <code>A</code>
      </attrGroupType>
      <attrGroupDescr>
         <descr>Attribute group A instance description.</descr>
      </attrGroupDescr>
   </exampleClassAttrGroup>
   <exampleClassAttrGroup>
      <attrGroupType>
         <code>B</code>
      </attrGroupType>
      <attrGroupDescr>
         <descr>Attribute group B instance description.</descr>
      </attrGroupDescr>
   </exampleClassAttrGroup>
</exampleClass>
```

The dataset at the receiving end must then result in:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <exampleClassAttrGroup>
      <attrGroupType>
         <code>A</code>
      </attrGroupType>
      <attrGroupDescr>
         <descr>Attribute group A instance description.</descr>
      </attrGroupDescr>
   </exampleClassAttrGroup>
   <exampleClassAttrGroup>
      <attrGroupType>
         <code>B</code>
      </attrGroupType>
      <attrGroupDescr>
         <descr>Attribute group B instance description.</descr>
      </attrGroupDescr>
   </exampleClassAttrGroup>
</exampleClass>
```

3.2.3 Update of multiple instance attribute group

To update an attribute in a multiple instance attribute group, ALL attributes for ALL instances of the multiple instance attribute group implemented for the business object must be provided, along with all the implemented interfaces of the attributes in the attribute group.

If the desired update is to change the description for the first instance of the multi instance attribute group defined in the example in Para 6.3.2.2, then the update message must look like:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <exampleClassAttrGroup>
      <attrGroupType>
         <code>A</code>
      </attrGroupType>
      <attrGroupDescr>
         <descr>Updated attribute group A instance description.
         </descr>
      </attrGroupDescr>
   </exampleClassAttrGroup>
   <exampleClassAttrGroup>
      <attrGroupType>
         <code>B</code>
      </attrGroupType>
      <attrGroupDescr>
         <descr>Attribute group B instance description.</descr>
      </attrGroupDescr>
   </exampleClassAttrGroup>
</exampleClass>
```

This shows that, ALL instances of the attribute group, including its attribute values and additional optional characterization attributes like associated applicability statements, must be re-sent.

An update of an attribute group assumes the business object exists and the additional characteristic attributes related to the attribute group are intended to be updated, via the update message.

Note

The additional characteristic attributes of the attributes in an attribute group are defined and managed in the update message as discussed in Chap 2.3.

3.2.4 Delete a single instance of a multiple instance attribute group

Deleting a single instance of a multiple instance attribute group requires that ALL attribute group instances that are still valid must be re-sent.

If the desired update is to delete the second instance of the multi instance attribute group defined in the example in Para 3.2.3, then the update message must look like:

```
<exampleClass
   crud="U">
   <classId>
      <id>1</id>
   </classId>
   <exampleClassAttrGroup>
```

```
            <attrGroupType>
               <code>A</code>
            </attrGroupType>
            <attrGroupDescr>
               <descr>Updated attribute group A instance description.
               </descr>
            </attrGroupDescr>
         </exampleClassAttrGroup>
      </exampleClass>
```

If this update is applied to the resulting dataset in Para 3.2.3 then must the updated dataset at the receiving end result in:

```
      <exampleClass
         crud="U">
         <classId>
            <id>1</id>
         </classId>
         <exampleClassAttrGroup>
            <attrGroupType>
               <code>A</code>
            </attrGroupType>
            <attrGroupDescr>
               <descr>Updated attribute group B instance description.
               </descr>
            </attrGroupDescr>
         </exampleClassAttrGroup>
      </exampleClass>
```

3.2.5 Delete all existing multiple attribute group instances

To delete all instances of a multiple instanced attribute group, including all its implemented interfaces, the update of the multiple instance attribute group must be defined as `xsi:nil="true"`.

The following update message will delete all instances of multiInstanceAttributeGroup defined for the exampleClass with identifier value "1" in the dataset described in Para 3.2.3.

```
      <exampleClass
         crud="U">
         <classId>
            <id>1</id>
         </classId>
         <exampleClassAttrGroup
            xsi:nil="true"/>
      </exampleClass>
```

The dataset at the receiving end must then result in:

```
      <exampleClass>
         <classId>
            <id>1</id>
         </classId>
      </exampleClass>
```

Chapter 2.7

XML schema implementation guidance - Other Considerations

Table of contents

Page

List of tables

References

Table 1 References

Chap No./Document No.	Title
None	

1 Introduction

Data exchanged in the exchange file may require special processing dependent on exchange partner agreements.

2 Exchange of classification values

Classification values are defined as "code" values in the exchange file eg, breakdownElementRelationshipType,

```
<xsd:complexType
   name="breakdownElementRelationshipType">
   <xsd:annotation>
      <xsd:appinfo>SX001G:breakdownElementRelationshipType
      </xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         name="code"
         type= "value:breakdownElementRelationshipTypeCodeValues"
         nillable="true"/>
   </xsd:sequence>
</xsd:complexType>
```

For some classification type elements, it may make sense to define a commonly used short character string to represent the classification. The common values may be defined in related valid values schemas. The valid values must be defined between data exchange partners, prior to the exchange of any date.

For example, if the "replacebilityStrategy" element is exchanged, the "replaceabilityStrategyCodeValues" exchange value, LRU, should be interpreted by the receiving system as a "lineReplaceable" unit. Using the enumerated values in the exchange file will facilitate an accurate exchange of these classification values.

```
<xsd:simpleType
   name="replaceabilityStrategyCodeValues">
   <xsd:restriction
      base="xsd:string">

      <!--Data Element Examples -->

      <xsd:enumeration
         value="LRU">
         <xsd:annotation>
            <xsd:appinfo>SX001G:lineReplaceable</xsd:appinfo>
         </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration
         value="SRU">
         <xsd:annotation>
            <xsd:appinfo>SX001G:shopReplaceable</xsd:appinfo>
         </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration
         value="NA">
         <xsd:annotation>
            <xsd:appinfo>SX001G:notApplicable</xsd:appinfo>
         </xsd:annotation>
      </xsd:enumeration>
   </xsd:restriction>
</xsd:simpleType>
```

2.1 Program (Exchange Partner) specific classifications

If Program (Exchange Partner) specific classifications are needed, it is recommended that these classification values be included in an applicable valid values file, so the exchange partners have one set of files for classification values.

There should be some obvious identification of the added / modified classification data in the valid values file. Typically, the classification values have a "source" reference in the <xsd:appinfo> attribute of the <xsd:annotation> element of the classification . A Program (Exchange Partner) specific value could be used to identify the added or modified classification information.

```
<xsd:simpleType
   name="replaceabilityStrategyCodeValues">
   <xsd:restriction
      base="xsd:string">
```

```
        <!--  Data Element Examples -->

        <xsd:enumeration
           value="LRU">
           <xsd:annotation>
              <xsd:appinfo>SX001G:lineReplaceable</xsd:appinfo>
           </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration
           value="SRU">
           <xsd:annotation>
              <xsd:appinfo>SX001G:shopReplaceable</xsd:appinfo>
           </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration
           value="NA">
           <xsd:annotation>
              <xsd:appinfo>SX001G:notApplicable</xsd:appinfo>
           </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration
           value="ADDRS">
           <xsd:annotation>
              <xsd:appinfo>
               PROGRAM1:AdditionalReplaceabilityStrategy
              </xsd:appinfo>
           </xsd:annotation>
        </xsd:enumeration>
     </xsd:restriction>
  </xsd:simpleType>
```

Note

It is recommended that all classifications defined in a specification valid values schema should have at least one valid value defined.

2.2 Optional characterization attributes for primitive data type attributes

It is strongly recommended that multiple values of attributes of primitive type descriptor, property or identifier should have unique sets of optional characterization attribute values. Without using different optional characterization attributes to differentiate the primitive data type values, the specific primitive values would not able to be identified for update or deletion. If the multiple values for a primitive data type attribute differ by one or all of the optional characterization attributes, then the update or deletion of the specific value may be supported in the update message. If the multiple primitive data type values do not differ by the optional characterization attributes, then the update and delete of the primitive data type attribute values must be implemented like the update and delete of the classification primitive data type. To update or delete a classification value, all classification values must be deleted and the current values added back.

In addition, having multiple values for primitive data type attributes, without unique sets of optional characterization attributes does not allow the database system to accurately select a specific value of the primitive type data element. As defined, the primitive data type attribute may have one or many values. For example, a descriptor may have many values. Without defining optional characterization attributes with the values, which descriptor value should be used for a specific condition or data selection? The primitive data types are defined to allow a flexible

schema structure for the business object definition. The flexibility should be supported with the applicable data to ensure accurate use of the business object data.

Note

It is strongly recommended to not assign an ApplicabilityStatement to the usage of a classification value in the context of a primitive value characterization, example the identifierClassifier for an identifier. The problem is the interpretation of the "applicability of the value" since the characterization attributes are part of the unique identification of the value.

End of data module