SWIFTStandards XML

for Implementors

Table of contents

1 Introduction

1.1 Purpose of this document

This document is the implementation manual for SWIFTStandards XML messages.

It is currently aligned with the SWIFTNet V4.0 product line, and it will be updated as such when SWIFTNet will be enhanced.

It explains

- what the various XML components are
- how they relate to the SWIFTStandards Financial Dictionary
- how this SWIFStandards Financial Dictionary is organised to promote reusability

1.2 Intended audience

- Software engineers working on the implementation of the SWIFTStandards XML messages in the financial user community
- Software engineers designing the user interface for SWIFTStandards XML messages
- SWIFT Application vendors
- SWIFT Customers developing their own SWIFT applications

1.3 Prerequisites

Prior knowledge the reader must have, before reading this document:

1. basic knowledge of UML (http://www.omg.org/uml)
2. basic understanding of SWIFTStandards modeling methodology (http://www.swift.com/index.cfm?item_id=7291)
3. in-depth knowledge of XML (http://www.w3c.org/TR/2000/REC-xml-20001006)
4. in-depth knowledge of XML Schema (http://www.w3c.org/TR/xmlschema-0/), (http://www.w3c.org/TR/xmlschema-1/) and (http://www.w3c.org/TR/xmlschema-2/)

1.4 Terms and definitions

Business Actor
A physical business user (i.e. person, organisation or infrastructure), playing one or more Business Roles in a Business Process. A Business Actor is uniquely identified in the Financial Dictionary.

Example: Bank, Corporate

Business Area
A set of strongly related business activities, that provide a self-standing business value to a set of Business Actors. A Business Area may be refined in other Business Areas (i.e. hierarchical structure). At the lowest level it is defined by a set of Business Processes. A Business Area is uniquely identified in the Financial Dictionary.

Example: Pre-Trade, Post-Trade/Pre-settlement

Business Association
A semantic relation between two Business Components. A Business Association is uniquely identified in the Financial Dictionary. There can be several Business Associations between two Business Components if the semantics of the relations are different.

Example: a Party services an Account

Business Component
A representation of a (part of a) key business notion and characterised by specific Business Elements. Each Business Component may have one or more Associations with other Business Components. A Business Component is uniquely identified in the Financial Dictionary.

Business Concept
Dictionary Item with a business semantic meaning, i.e. Business Actor, Business Component, Business Element, Business Rule or Association.

Business Element
A characteristic of a Business Component. A Business Element is uniquely identified in its Business Component.

Business Information
A generic name covering Business Components, Business Elements and Associations between these.

Business Model
An abstract definition of a (part of a) business area showing the main Business Processes and Business Concepts relevant to this (part of a) Business Area.

Business Process
A main business activity within a Business Area that allows the industry to achieve its business objectives. A Business Process may be refined in other Business Processes (i.e. hierarchical structure).

Business Role
A functional role played by a Business Actor in a particular Business Process.

Business Rule
A business constraint attached to a Business Concept and defining specific conditions applicable to that Business Concept or to its associated Business Concepts if any. A Business Rule is uniquely identified in the scope of a Business Concept.

Data Type
A Data Type unambiguously specifies the set of valid values of a Business Element or of a Message Element. The set of valid values may be defined via a format specification or via an enumeration. A Data Type is uniquely identified in the Financial Dictionary.

Data Type Representation
A Data Type Representation complements the definition of a Data Type with the technical information required for implementation and processing; “boolean”, “integer”, “string” are some examples of Data Type representation.

Derived Message Element
A derived Message Element is a Message Element that has a different name than its corresponding Business Element.

Dictionary Item
An item that is uniquely identified in the Data Dictionary.

Enumerated Code Value
One possible code value in a list of possible code values assigned to a Business or Message Element. It must be defined as a code of 1 up to 4 uppercase alphanumeric characters.

Enumerated Code Value List
A list of all possible code values assigned to a Business or Message Element. An Enumerated Code Value List is uniquely identified in the Data Dictionary.

Message
A set of structured information exchanged between Business Actors, in the scope of a Business Process. A Message is uniquely identified.

Message Component
A reusable Dictionary Item that is a building block for assembling messages. It is normally linked to a Business Component and characterised by specific Message Elements. A Message Component is uniquely identified in the Financial Dictionary.

Message Concept
Dictionary Item used for Message Definition, i.e. Message Component, Message Element or Message Rule.

Message Definition
The formal description of a Message. The Message Definition is built as a tree structure of reused Message Components.

Message Element
A characteristic of a Message Component. A Message Element is uniquely identified in its Message Component.

Message Flow Diagram
A Message Flow Diagram addresses the dynamic view of a system. It depicts the ordered sequence of messages that may be exchanged between Business Actors. A Message Flow Diagram is uniquely identified.

Message Rule
A specific constraint that is specified at the level of a Message or of a Message Component. A Message Rule is uniquely identified in the Message or in the Message Component.

Technical Message Element
A Technical Message Element is a Message Element that only makes sense in a specific message context and hence only exists in that Message.

2 Overview of the methodology for SWIFTStandards Message Development

The methodology comprises 5 activities[1]:

- The **Business Analysis** focuses on getting a good understanding of the business objectives of the considered Business Area.
- The **Requirements Analysis** focuses on discovering the communication and interaction requirements related to the Business Processes that are part of the considered Business Area.
- The **Logical Analysis** and **Logical Design** specify the Message Standard that meets the identified communication and interaction requirements. The Message Standard is defined independently of any physical implementation and includes Message Flow Diagrams and Message Definitions.
- The **Technical Design** delivers the physical implementation of Message Definitions and Message Rules in their corresponding SWIFTStandards XML Schema[2].

These five activities are applied in an iterative and incremental way for the SWIFTStandards Message Development.

Bearing the above in mind, this document aims at explaining **bottom-up** how in XML, all the various artefacts fit in a more global picture of the Financial Dictionary.

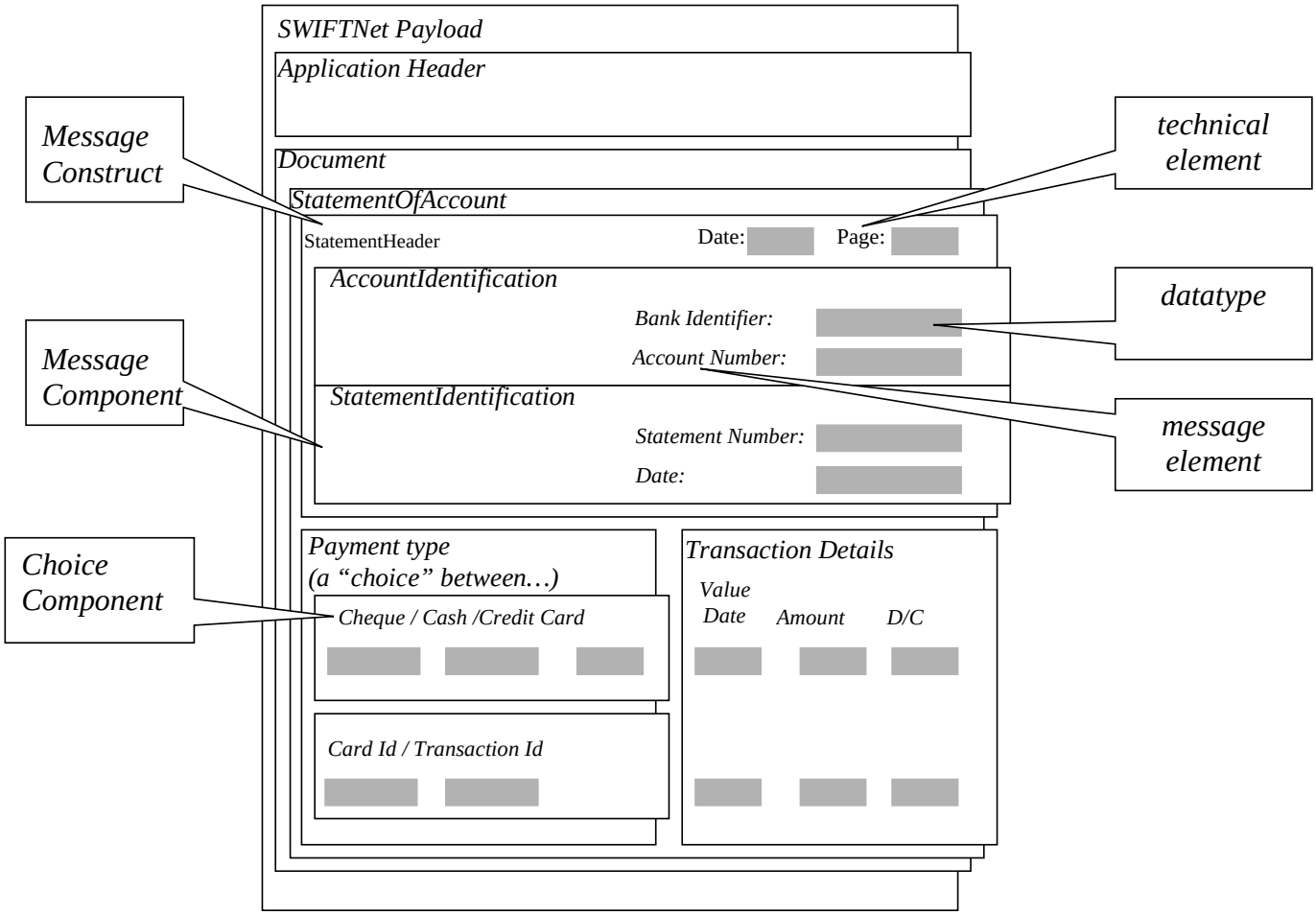
[1] For a more detailed explanation, see the SWIFTStandards Modeling Methodology document (http://www.swift.com/index.cfm?item_id=7291)

[2] Only XML Schema's are generated, not XML DTD's.

3 Message Definition

3.1 Example

A Message Definition Diagram contains the formal representation of a Message. A Message is composed of different kinds of components, which we call artefacts. It is important to understand what these various artefacts can be. Below example tries to visualise this.



3.2 Message Definition artefacts

3.2.1 Message

A Message is composed of Message Constructs, Message Components, Choice Constructs, Choice Components, Message Elements. A Message is uniquely identified within the SWIFTStandards Business Domain.

In the above example, StatementOfAccount is the message name.

3.2.2 Message Construct

A **Message Construct** represents a structural element of a "document" (e.g. Statement Header). A Message Construct has no business meaning. Its sole purpose is to better structure a Message to simplify it or to have a better validation. A Message Construct cannot be reused.

3.2.3 Message Component

A **Message Component** is composed of **Message Elements** and/or other Message Components. All Message Components are derived from Business Components[3]. (See also chapter Traceability from Message Definition Diagram to business artefacts)

Message Components form the reusable items with which the Messages must be built. They can be different because of their specific subset of Message Elements or because of specific constraints such as specific Message Rules or cardinality constraints. A Message Component has a unique name within the SWIFTStandards Business Domain and a set of Message Elements.

3.2.4 Choice

There are also cases where a choice has to be made between several Message Components. This can be done by either using a **Choice Component**, whereby the Choice Component itself does have a real business meaning and is as such reusable, or by using a **Choice**

[3] Business Components cannot be used directly in Message Definition Diagrams.

Construct, whereby the Choice Construct itself does not have a real business meaning and is as such not reusable.

3.2.5 Message Element

Message Elements are derived from the Business Elements of the Business Component corresponding to the Message Component.

Message Elements may contain some additional semantic than the Business Element from which they are derived.

Since a Message Element is local to its Message Component in which it is decalred, the name of a Message Element is unique within its Message Component. (See also chapter Traceability from Message Definition artefacts to business artefacts).

A Message Element is typed with either a Message Component, or with a SWIFTStandards-defined Data Type.

Beside the 'Common' Message Elements, there are two special cases of Message Elements:

1. A Technical Message Element.
   A Technical Message Element is a Message Element that only makes sense in a message context. A specific Datatype can be created, but it has no corresponding Business Element.
2. A Derived Message Element.
   A Derived Message Element is a Message Element that has a different name than its corresponding Business Element. This can occur when its name has more meaning or because it was derived from a Business Element of another Business Component

3.2.6 Data Type

To recapitulate, in a Message Definition Diagram all Message Elements have a type of which some refer to Message Components and the rest refer to **SWIFTStandards-defined Data Types**.

3.2.6.1 SWIFTStandards-defined Data Types

- are grouped together into so-called Representation classes. A Representation Class has a number of characteristics that are passed on ('inherited by') all Data Types that are using that representation class (e.g. an XML attribute, the XML primitive[4] type it is using, etc). In this way, characteristics common to a number of Data Types are grouped together.
- can be further constraint by XML facets (see also XML facets) and by Business Rules.
- are globally unique, across the SWIFTStandards Business Domain and across all Messages.

Following table summarizes what these different Representations are:

Representation Class name	Definition
Identifier	A character string to identify and distinguish uniquely, one instance of an object in an identification scheme together with relevant supplementary information. The Identifier allows to define an identification scheme and the organization managing this scheme
Code	A character string (letters, figures or symbols) that for brevity and/or language independence may be used to represent or replace a definitive value or text of an attribute together with relevant supplementary information
Indicator	A list of two values that indicate a condition such as credit/debit, true/false, ...
Text	A character string that may be used to describe a concept
Quantity	A number of non-monetary units where the unit of quantity is explicit or implied.
Amount	A number of monetary units specified in a currency where the unit of currency is explicit or implied.

[4] Primitive" as defined by W3C's data type specification document http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

DateTime	A measurement of time.
Rate	A quantity or amount measured with respect to another measured quantity or amount, or a fixed or appropriate charge, cost or value.

For a more detailed explanation on the different representations of Data Types, refer to Appendix A Data Type Representations.

3.2.6.2 Primitive[5] Data Types

SWIFTStandards Data Types are encoded as defined by W3C, defined at http://www.w3.org/TR/xmlschema-2/#dt-encoding.

Following XML built-in primitive types are supported:

XML Name	Description
`string`	Set of finite sequences of UTF-8 characters
`boolean`	Has the value space of boolean constants “True” or “False”
`integer`	Corresponds to 32 bits integer type
`decimal`	Arbitrary precision decimal numbers
`date`	Corresponds to a date. See ISO 8601. Format CCYY-MM-DD
`time`	Corresponds to a time. See ISO8601. Format HH:MM:SS +- offset to UTC
`dateTime`	Corresponds to a date and time. See ISO8601. Format CCYY-MM-DDTHH:MM:SS +- offset to UTC
`duration`	Corresponds to a period in time. See ISO8601. Format PnYnMnDTnHnMnS
`gDay`	It is a set of one-day long, annually periodic instances. The time zone must be UTC. Format: --MM-DD.
`gMonth`	Represents a time period that starts at midnight on the first day of the month and lasts until the midnight that ends the last day of the month. Format: --MM--.
`gYear`	Represents a time period that starts at the midnight that starts the first day of the year and ends at the midnight that ends the last day of the

[5] “Primitive” as defined by W3C’s data type specification document http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

	year. It is a set of one-year long, non-periodic instances. Format: CCYY
`gMonthday`	Represents a set of one-day long, monthly periodic instances. The time zone must be UTC. Format: ---DD.
`base64Binary`	Represents Base64-encoded arbitrary binary data

3.3 Traceability from Message Definition artefacts to Business artefacts

One of the goals of this SWIFTStandards Financial Dictionary is to promote reusability and to be able to do a better impact analysis in case of changes and updates to Messages. Hence the need to have a link between the various Message artefacts and their underlying Business Elements / Business Components.

Below describes how these Message artefacts are linked to their corresponding Business Elements / Components, using what we call 'traceability links'.

3.3.1 Business Component

Business Components are defined in the Business Information Model. They cannot be used directly in Message Definition Diagrams.

3.3.1.1 Message Component derived from Business Component

Message Components are directly derived from – i.e. they are tracing to - the Business Components and can be considered as "views" on Business Components used in Messages. Several Message Components can be derived from – i.e. traced to – one single Business Component.

In some cases, Message Components contain Message Elements coming from different Business Components (these are called Derived Message Elements).

[7] Or complexTypes with simpleContent

3.3.2 Business Element

A Business Component is composed of Business Elements. The name of a Business Element is unique within a given Business Component. A Business Element can never be used directly in a Message Definition Diagram.

3.3.2.1 Message Element derived from a Business Element

All Message Elements trace to Business Elements (except for Technical Message Elements, which have no trace).

There may also be situations where Message Elements in one Message Component come from multiple related Business Components. These will all trace to their respective Business Components.

3.3.3 Example

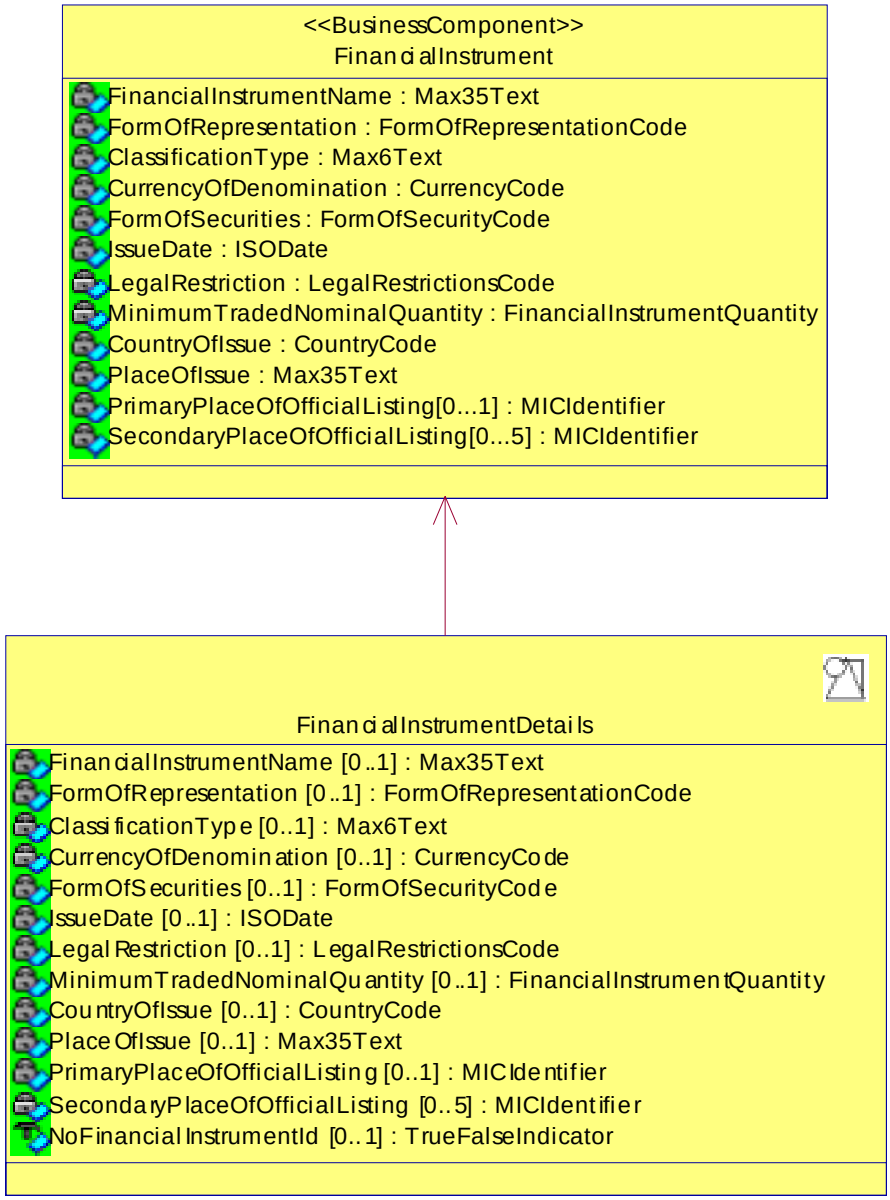


Figure 3-1: Deriving a MessageComponent from a BusinessComponent

Note

"NoFinancialInstrumentID" was created only for FinancialInstrumentDetails. It has no traceability link. Its purpose is to indicate whether a FinancialInstrumentIdentification Message Component is in the message, or not. But this information can also be derived from the presence/absense of that Message Component.

4 SWIFTStandards XML Schema and XML Instance

4.1 Introduction

This chapter explains how a generated SWIFTStandards XML Schema (and its corresponding XML instance) is mapped to its corresponding Message Definition Diagram.

4.2 Mapping rules

In an XML Schema, any SWIFTStandards XML element

- has a name
- is typed either by a simpleType or a complexType
- can have a number of XML attributes

4.2.1 XML element name

Where can the XML element name be found in the Message Definition Diagram?

For a SWIFTStandards XML element mapped from a Message:

- The name of the Message.

For a SWIFTStandards XML element mapped from a Message Element:

- The XML name of the Message Element or by default the name of the Message Element.

4.2.2 XML simpleType

XML simpleTypes are mapped from SWIFTStandards-defined Data Types that are using following Representation Classes:

- *Identifier*
- *Code*
- *Indicator*
- *Text*

- *DateTime*

4.2.3 XML complexType

If an XML element has a complexType, then the complexType is mapped from Message Component or Message Construct.

4.2.3.1 XML complexTypes with simpleContent

In this case the complexType is mapped from SWIFTStandards-defined Data Type that is using one of following Representation Classes:

- *Quantity*
- *Amount*
- *Rate*

4.2.4 XML Attributes

4.2.4.1 xsi:type

When a Message Component inherits characteristics from another Message Component, the xsi:type is present to indicate in the instance the Message Component that was inherited from.

By using xsi:type in the instance, the schema does not need to define any additional attribute on types since xsi:type implicitely refers to a type defined in the schema. The attribute "xsi:type" is required to indicate the chosen type in the SWIFTStandards XML instance.

4.2.4.2 Data Type Representation Attribute

SWIFTStandards-defined Data Types that have an XML attribute must use one of following Representation Classes:

Representation Class	XML Attribute
Quantity	Unit
Amount	Currency
Rate	Unit

4.2.5 Summary

Representation	XML derived type	XML primitive type	XML attribute	Supported XML Schema Facets
Identifier	simpleType	string	None	None
Code	simpleType	string	None	Enumeration
Indicator	simpleType	boolean	None	None
Text	simpleType	string	None	Pattern, Length, MinLength, MaxLength
Quantity	complexType with simpleContent	decimal	Unit	minInclusive, minExclusive, maxInclusive, maxExclusive, totalDigits and FractionDigits
Amount	complexType with simpleContent	decimal	Currency	minInclusive, minExclusive, maxInclusive, maxExclusive, totalDigits and FractionDigits
DateTime	simpleType	* DateTime * Date * gYear * gMonth * gDay * gMonthDay * gYearMonth * Time * Duration	None	minInclusive, minExclusive, maxInclusive, maxExclusive
Rate	complexType with simpleContent	decimal	Unit	minInclusive, minExclusive, maxInclusive, maxExclusive totalDigits and FractionDigits

4.3 Traceability from XML Schema to its Message Definition Diagram

Any SWIFTStandards XML artefact (e.g. XML element, simpleType, complexType, attribute) has a corresponding Message Definition artefact (e.g. Message Element, Message Component, Construct).
All Message Definition artefacts have a name (independent of the syntax) and an XML name. Typically the (syntax-independent) name and the XML name are the same.

SWIFT Standards Message artefact	XML representation
Message	An XML document
MessageComponent	complexType
MessageConstruct	complexType
MessageElement	Element in a complexType
ChoiceComponent	An XML Choice model group
ChoiceConstruct	An XML Choice model group
SWIFTStandards-defined Data Type	See Appendix A for a detailed mapping.

5 UML representation

This chapter explains how the SWIFTStandards concepts are expressed in UML.

SWIFT Standards Business Concepts	UML Modeling representation
BusinessComponent	A class with stereotype BusinessComponent
BusinessElement	➢ An attribute of a class, which type an be either a Datatype or a BusinessComponent ➢ A relation and a role name linked to a BusinessComponent.
SWIFTStandards-defined Data Type	The stereotype of the class indicates the type of representation.

SWIFT Standards Message Concepts	UML Modeling representation
Message	Class with stereotype <<Message>>
MessageComponent	Class with stereotype <<MessageComponent>>
MessageElement	➢ An attribute in the class with stereotype <<MessageComponent>> or <<ChoiceComponent>>. The type of the attribute is necessarily a MessageComponent, a ChoiceComponent or a Data Type. ➢ An aggregation link from the class <<MessageComponent>> or <<ChoiceComponent>> to another MessageComponent or ChoiceComponent.
ChoiceComponent	Class with stereotype <<ChoiceComponent>>
MessageConstruct	Class with stereotype <<MessageConstruct>>
A MessageElement references a MessageComponent or a Datatype	The referenced MessageComponent or Data Type is the type of the Attribute.

6 SWIFTStandards Financial Dictionary artefacts

The Dictionary contains only reusable artefacts. Following artefacts will be accessable through the Dictionary:

- Business Components
- Business Association within its Business Components
- Business Elements + Business Rule within their Business Component
- Message Components
- Message Elements within their Message Component
- SWIFTStandards-defined Data Types

Non-reusable artefacts can only be found in the SWIFTStandards Reference Guide. For the sake of completeness, following are the non-reusable artefacts that have been discussed previously:

- Business Processes, Information Flows
- Messages
- Message Constructs

7 XML schema features used in SWIFTStandards XML

7.1 Namespaces in XML schema and XML instances

SWIFTStandards XML schema and XML instances use four namespaces:

- the default (non qualified) namespace. All schema have their own default namespace generated according to the following regular expression: “urn:swift:xsd:\$+”. Where the “+” must be replaced by the message name optionally prefixed by the collaboration name separated by a ‘.’.
- xs: W3C XML schema namespace (not used in instances)
- xsi: W3C XML schema-instance namespace
- a target namespace (for schema only) which is the same as the default namespace with a user selected prefix.

Schema snippet:

```
<schema
   xmlns=”urn:swift:xsd:$NoticeOfExecution”
   xmlns:xs=” http://www.w3.org/2001/XMLSchema”
   targetNamespace=”urn:swift:xsd:$NoticeOfExecution”>
```

Instance snippet:

```
<Document
   xmlns=”urn:swift:xsd:$NoticeOfExecution”
   xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”>

   <NoticeOfExecution>…
```

For further information on a complete SWIFTNet Message, refer to the SWIFTNet Headers document.

7.2 XML facets on simpleTypes[7]

The following sections describe the facets that may be used in the XML schema.

7.2.1 pattern

Pattern matching allows lexical validation on strings, which syntax can be described using regular expressions, (commonly referred to as "Perl expressions").

This facet only applies to Representation of type Text.

7.2.2 length, minLength, maxLength

XML schema allows restriction of the value space of any string value (i.e.: integer, date etc are not affected) by using the following constraining facets:

- length
- minLength
- maxLength

Those facets only apply on strings, and their values must be positive integer values.

7.2.3 minInclusive, maxInclusive, minExclusive, maxExclusive

XML schema allows restriction of the value space of any numerical value by using the following constraining facets:

- minInclusive
- minExclusive
- maxInclusive
- maxExclusive

Those facets only apply to numerical values (Integer, BigDecimal) and to time measurement related values (Date, Time, DateTime, gDay, gMonth, gYear, gMonthday, duration) and their value must be constants of the same type than the numeric value they apply to.

7.2.4 enumeration

XML schema allows restriction of the value space of an enumeration by using the enumeration constraining facet.

This facet only applies to enumerations, and their value must be part of the original enumeration from which they restrict.

Enumerations can only apply to Data Types using representation <<Code>>.

7.2.5 totalDigits, fractionDigits

Fixed-point decimal values need a totalDigits specification (i.e. the maximum number of decimal digits in values of Data Types derived from decimal: totalDigits), as well as a fractionDigits specification (i.e. the maximum number of decimal digits in the fractional part of values of Data Types derived from decimal: fractionDigits).

The value of the totalDigits facet must be a positive integer.

The value of the fractionDigits facet must be a non-negative integer.

8 Other characteristics

8.1 Run-time Schema versus documentation information

Run-time schemas only contain information required to validate XML instances. Documentation or implementation information –information that is not used by the XML parser - is not part of SWIFTStandards XML Schema's.

8.2 Granularity of Schemas

There is one Schema per message.

8.3 Naming conventions

8.3.1 Message Components and Business Components

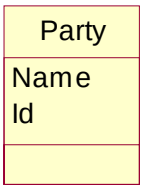
All Message Components, Business Components and Data Types must have a unique name within the SWIFTStandards Business Domain.

8.3.2 Name scoping

One of the characteristics of Message Elements is that they are scoped within the Message Component they are used in. This means that any Message Element 'inherits' the properties (i.e. in this case the name) of the Message Component it is used in.

Example:

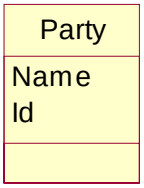
Suppose an Account with two properties: Id and Name:



This could translate to following XML Instance snippet:

```
<Account>
   <Name>data</Name>
   <Id>data</Id>
</Account>
```

Suppose within the same message, a Party with two properties: Id and Name. Id and Name have a different meaning and properties than the ID and Name from Account, but have the same name. Their meaning depends on the Message Component they're in. In other words, their name is scoped within the component they are used in.



8.4 Character set

SWIFTStandards XML uses UTF-8 as the (default) character encoding mechanism, for the following reasons:

- It has the most efficient method of character representation:
  - It is the shortest method to represent the characters which are currently the most commonly used in a financial environment (ASCII and EBCDIC characters)
  - It can still represent almost any known character
- It is interoperable with many other encoding schemes through (automatable) conversion algorithms.

Example:

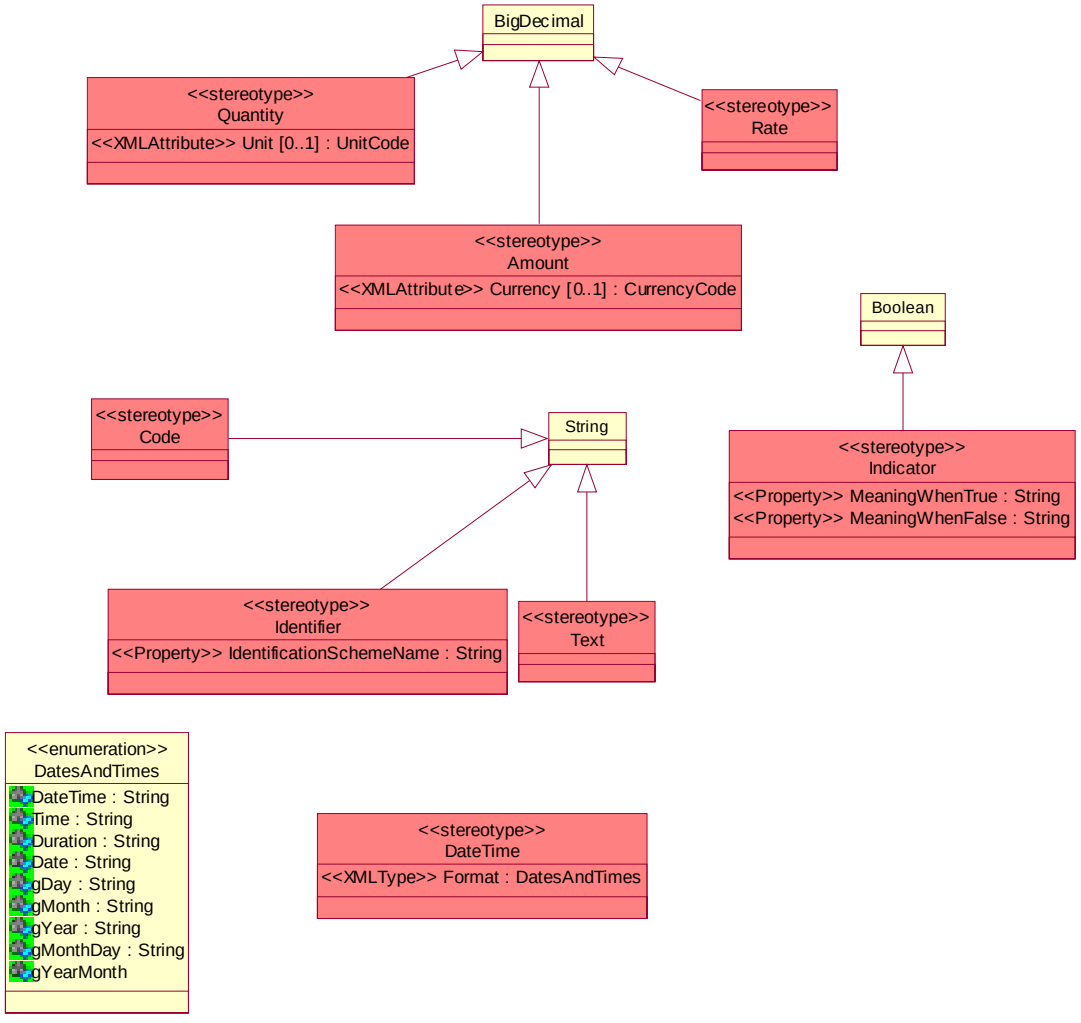
<?xml version="1.0" encoding="UTF-8"?>

8.5 Schema Versioning

The version number is an integral part of the message name.

A APPENDIX Data Type Representations

A.1 Data Type Metamodel



Notes:

Each SWIFTStandards-defined Data Type is identified by a class and stereotyped by a representation class name. A representation class has a number of characteristics that are

passed on ('inherited by') all Data Types that are using that representation class. In this way, characteristics common to a number of Data Types are grouped together.

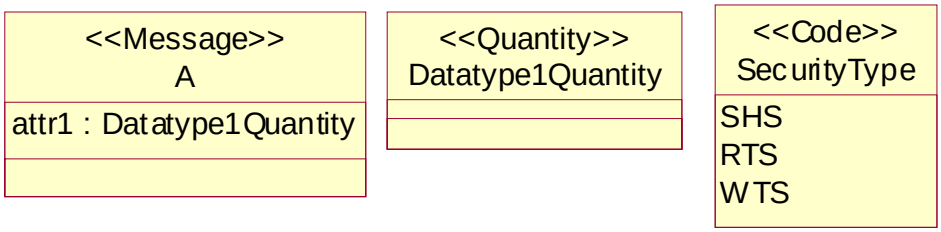
Stereotype <<XMLAttribute>> indicates that the values the XML attribute of this Representation are usable by the associated SWIFTStandards-defined Data Type. The Data Type definition could supersede this definition, by not using the XML Attribute. One of these values will appear in the XML instance in case of ambiguity[8].
Conversely, if there is no ambiguity (i.e. the datatype has an XML attribute with only one or no values), it must NOT appear in the instance and must not be declared in its XML Schema.

Stereotype <<Property>> indicates that the values this characteristic has will NOT be declared in the XML Schema. Instead this is a property inherent to this Data Type that will only appear in Financial Dictionary, the Standards Reference Guide and other implementation information, since their content/value is static across instances, so it must not be validated.

Stereotype <<XMLType>> (only used in representation class DateTime) indicates actually which XML built-in type (Date, Time, DateTime, gDay, gMonth, gYear, gMonthday or duration) will be used by a specific Data Type.

Data Types are globally unique, across all business domains and across all messages.

A.2 Data Type using representation class <<Quantity>>



Properties:

- Since the representation class Quantity (see metamodel) has an attribute that is stereotyped as being a <<XMLAttribute>>, any Data Type that is stereotyped by <<Quantity>> must also specify whether there is a list of possible values attached. In this case there is (called "SecurityType") hence the corresponding Schema defines for

[8] Ambiguity occurs when this XML attribute can have more than one value. Then the XML attribute must be present in the XML instance and it must be declared in its XML Schema.

Message Element <attr1> an XML attribute named 'UnitCode' with a enumerated list of values a specified in the Class 'SecurityType'.

- An enumerated value is constrained within a list of possible values.
- The values for the enumerated items are taken from the UML initial value given to each of the UML enumerated attributes.

Suppose this Data Type has an additional constraint (=XML facet) that the maximum quantity may not exceed 20000 units.

Instance snippet:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <attr1 UnitCode="SHS">1000</attr1>
</A>
```

Schema snippet:

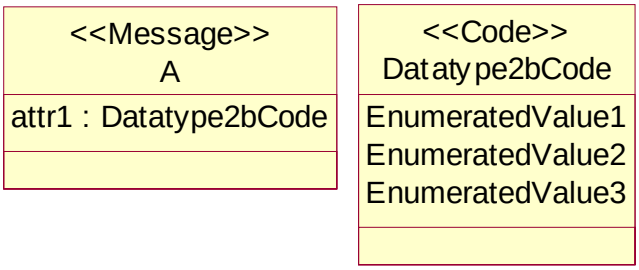
```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype1Quantity"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Datatype1Quantity">
  <xs:simpleContent>
   <xs:restriction base="xs:decimal">
      <xs:maxInclusive value="20000">
      <xs:attribute name="UnitCode" type="SecurityType"/>
   </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="SecurityType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="SHS"/>
      <xs:enumeration value="RTS"/>
      <xs:enumeration value="WTS"/>
   </xs:restriction>
</xs:simpleType>
```

A.3 Data Type using representation class <<Code>>



Properties:

- This Data Type is used when the values of the list have a meaningful (i.e.semantic) value within the context of the message (e.g. the trade types). SWIFTStandards-defined Data Types using <<Code>> reference an internal list (i.e. a list specified in the schema). Datatype2bCode is an enumeration of which one of the Enumerated Values has to be chosen in the instance.
- An enumerated value is constrained within a list of possible values.
- The values for the enumerated items are taken from the UML initial value given to each of the UML enumerated attributes.

UML	**SWIFTStandards XML instance**
Class contains an enumeration of possible values	SWIFTStandards XML element contains the chosen value

Instance snippet:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <attr2>EnumeratedValue2</attr2>
</A>
```

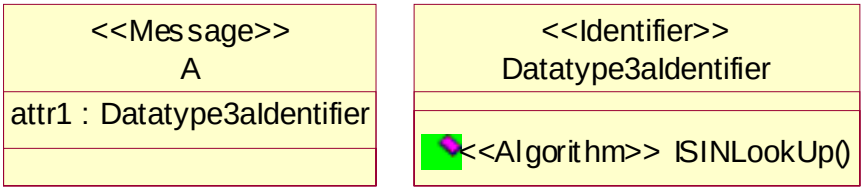
Schema snippet:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr2" type="xs:Datatype2bCode"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype2bCode">
   <xs:restriction base="xs:string">
      <xs:enumeration value="EnumeratedValue1"/>
      <xs:enumeration value="EnumeratedValue2"/>
      <xs:enumeration value="EnumeratedValue3"/>
   </xs:restriction>
</xs:simpleType>
```

A.4 Data Type using representation class <<Identifier>>



Properties:

- This Data Type is used when the values of the list have no meaningful (i.e.semantic) value within the context of the message (e.g. BIC addresses). SWIFTStandards-defined Data Types using <<Identifier>> have an external list (i.e. not specified in the schema). The class has no attributes and an operation is added with stereotype <<algorithm>> that refers to the 'external' list (in this case an ISIN database).

Instance snippet:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <attr1>AnythingGoesHere</attr1>
</A>
```

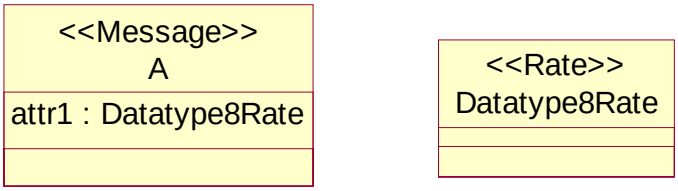
Schema snippet:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype3aIdentifier"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype3aIdentifier">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>
```

A.5 Data Type using representation class <<Rate>>



Instance snippet:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <attr1>95.6</attr1>
</A>
```

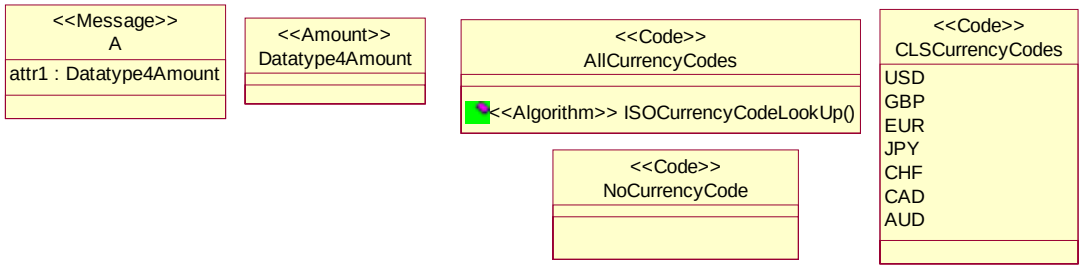
Schema snippet:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype8Rate"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype8Rate">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>
```

A.6 Data Type using representation class <<Amount>>



Properties:

- Since the representation class Amount (see metamodel) has an attribute with a type named CurrencyCode which is stereotyped as being a <<XMLAttribute>>, any Data Type that is stereotyped by <<Amount>> may also specify the values this attribute can have.
- There are two possible cases:
  1. The attribute has more than one possible value. Then the corresponding Schema defines for Message Element <attr1> an XML attribute named 'CurrencyCode' with a list of possible values (here specified in the Classes 'CLSCurrencyCodes' or 'AllCurrencyCodes' (which is an external list)).
  2. The attribute has one or no possible values. Then there is no ambiguity and the Data Type has no XML attribute.
- In the below case the datatype4Amount is referring to "AllCurrencyCodes". However, since this one is validated externally hence the corresponding Schema doesn't contain the list of valid values.

Instance snippet:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <attr1 CurrencyCode="USD">95.6</attr1>
</A>
```

Schema snippet:

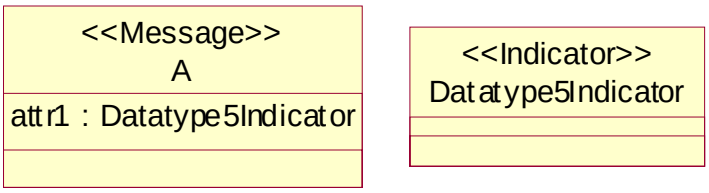
```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype4Amount"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Datatype4Amount">
  <xs:simpleContent>
   <xs:restriction base="xs:decimal">
      <xs:attribute name="CurrencyCode" type="AllCurrencyCodes"/>
   </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="AllCurrencyCodes">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>
```

A.7 Data Type using representation class <<Indicator>>



Properties:

- A Data Type stereotyped by representation class <<Indicator>> indicates that the Message Element must have a Boolean value (true or false).
- None of the attributes in the metamodel for <<Indicator>> appear in an XML instance.

Instance snippet:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <attr1>true</attr1>
</A>
```

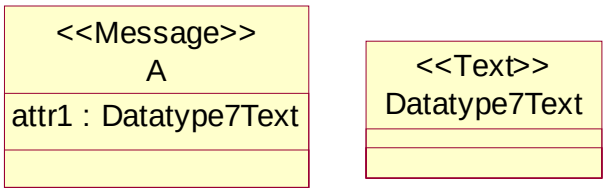
Schema snippet:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype5Indicator"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype5Indicator">
   <xs:restriction base="xs:boolean">
   </xs:restriction>
</xs:simpleType>
```

A.8 Data Type using representation class <<Text>>



Properties:

- A Data Type stereotyped by representation class <<Text>> indicates that the Message Element contains textual information.

Instance snippet:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <attr1>any narrative text</attr1>
</A>
```

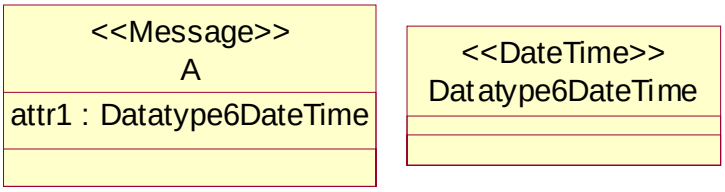
Schema snippet:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype7Text"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype7Text">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>
```

A.9 Data Type using representation class <<dateTime>>



Properties:

- Representation class 'DateTime' has a meta attribute Format which is stereotyped <<XMLType>>. This means that any Data Type that is using representation class <<DateTime>> has to indicate from which XML primitive Data Type it is restricting.
- Suppose the primitive used for Datatype6DateTime is dateTime and an additional constraint is added namely that the date should be equal or later than January first, 2002.

Instance snippet:

```
<A xmlns="urn:swift:xsd:$A" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <attr1>2002-11-23</attr1>
</A>
```

Schema snippet:

```
<!-- <<message>> A -->
<xs:element name="A" type="A"/>

<!-- class: A -->
<xs:complexType name="A">
   <xs:sequence>
      <xs:element name="attr1" type="xs:Datatype6DateTime"/>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="Datatype6DateTime">>
   <xs:restriction base="xs:dateTime">
      <xs:minInclusive value="2002-01-01T00:00:00"/>
   </xs:restriction>
</xs:simpleType>
```

B XML DTD limitations

XML DTD requires all element declarations (i.e. element names) to be unique within their namespace. This means that two elements with the same name but a different content model (i.e. a different structure) are not allowed within DTD.

In XML Schema, all XML elements are local, that is they are unique within their parent element. All complexTypes are global, that is they are unique within the Financial Dictionary.

XML Schema does not have that restriction. This means that following Message Definition patterns are **issues in DTD generation**, but aren't in Schema generation:

- Two Message Component aggregation names are the same and have the same content model:

  This is not an issue for schemas, as those aggregation names will be defined in two different complexTypes.

- Two Message Component aggregation names or 2 Message Element names are the same, and they have a different content model:

  This is not an issue for schemas as long as the aggregations or Message Elements belong to different Message Components.

- A Message Component aggregation name and a Message Element name are the same:

  This is not an issue for schemas as long as the aggregation and Message Element belong to different Message Components.

- Two Message Components have the same name:

  As the name of the Message Component will be used for naming the associated complexType in the schema, this is NOT allowed.

End of document