XML Naming and Design Rules For CCTS 2.01

Version 2.1.1

31 August 2021

1 Status of this Document

This Technical Specification is being developed in accordance with the ECE/TRADE/C/CEFACT/2010/24/Rev.2.3 Open Development Process.

This document is for publication.

2 UN/CEFACT XML Naming and Design Rules Project Team Participants

We would like to recognise the following for their significant participation to the development of this Technical Specification.

Project Team Leader:

Chris Hassler

Lead Editor:

Mary Kay Blantz

Contributors:

Sylvia Webb
Karina Duvinger
Kevin Smith
Ewa Iwicka
Sue Probert
Michael Dill
Gerhard Heemskerk
Hidekazu Enjo
Hisanao Sugamata

Contributors to Previous Versions

Mark Crawford
Paula Heilig
Gunter Stuhec
Garret Minikawa
Hisanao Sugamata
Frank Lin
K.K. Suen
Luc Mouchot
Thomas Bikeev
Jostein Frømyr
Sue Probert
Alain Dechamps
Michael Dill

3 Table of Contents

4 Introduction

This UN/CEFACT – *XML Naming and Design Rules* Technical Specification describes and specifies the rules and guidelines that will be applied by UN/CEFACT when developing XML schema.

This technical specification provides a way to identify, capture and maximize the reuse of business information expressed as XML schema components to support and enhance information interoperability across multiple business situations.

4.1 Scope and Focus

This UN/CEFACT – *XML Naming and Design Rules* Technical Specification can be employed wherever business information is being shared or exchanged amongst and between enterprises, governmental agencies, and/or other organizations in an open and worldwide environment using XML schema for defining the content of the business information payload.

This technical specification will form the basis for standards development work of technical experts developing XML schema based on information models developed in accordance with the UN/CEFACT *Core Components Technical Specification – Part 8 of the ebXML Framework (CCTS)*, version 2.01 plus corrigenda.

This version was amended from the original to correct certain errors in the original text, as well as to fulfill the following goals: (1) decoupling the unqualified data type schema from this specification, to allow for easier maintenance of the data type catalogue, (2) to enable relative path names, making implementation of the schemas easier, (3) incorporating the Core Component Business Document Assembly specification, which did not exist at the time the first version was created, and (4) selectively decoupling any or all code list and identifier value enumerations from the qualified data type, to allow for two-phase code validation processing and flexibility.

[Note] - The term “decoupling” refers to decoupling of unqualified data type schema from the data type catalogue (CCT) and/or decoupling a particular qualified data type from a set of value enumerations.

4.2 Audience

The primary audience for this UN/CEFACT – *XML Naming and Design Rules* Technical Specification are members of the UN/CEFACT Bureau Programme Support who are responsible for development and maintenance of UN/CEFACT XML schema. The intended audience also includes the wider membership of the other UN/CEFACT groups who will participate in the process of creating and maintaining UN/CEFACT XML schema.

Additional audiences are designers of tools who need to specify the conversion of user input into XML schema representation adhering to the rules defined in this document. Additionally, designers of XML schema outside of the UN/CEFACT Forum community may find the rules contained herein suitable as design rules for their own organization. Since the constructs defined in CCTS are consistent with UML classes, attributes, and associations, these design rules can easily be applied to non CCTS constructs as well.

4.3 Structure of this Specification

The UN/CEFACT *XML Naming and Design Rules* Technical Specification has been divided into 5 main sections:

- Section 4 provides general information about the document itself as well as normative statements in respect to conformance, and guiding principles applied in developing this specification.
- Section 5 provides information on this specification’s dependency and relationship to CCTS. Furthermore, this section describes the approach taken to modularity in order to maximize the reuse of business information expressed as XML schema components and the general naming conventions applied. (Normative, except for Section 5.4, which is Informative)

- Section 6 provides the general conventions applied with respect to the use of the XML schema language. (Normative)
- Section 7 provides detailed rules applicable to each of the schema modules defined by the modularity approach. (Normative)
- Section 8 provides guidelines and rules related to XML instance documents. (Normative)

The document also contains the following Appendices:

- Appendix A Related Documents (Informative)
- Appendix B Overall Structure (Normative)
- Appendix C BPS Approved Acronyms and Abbreviations (Normative)
- Appendix D Use cases for code lists and identifier lists. (Informative)
- Appendix E Annotation Templates (Informative)
- Appendix F Naming and Design Rules List (Normative)
- Appendix G Glossary (Informative)

4.4 Terminology and Notation

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in Internet Engineering Task Force (IETF) Request For Comments (RFC) 2119.[1] Wherever xsd: appears this refers to a construct taken from the W3C XML schema specification. Wherever ccts: appears this refers to a construct taken from the CCTS.

Example – A representation of a definition or a rule. Examples are

informative. [Note] – Explanatory information. Notes are informative.

[Rn] – Identification of a rule that requires conformance. Rules are normative. In order to ensure continuity across versions of the specification, rule numbers that are deleted will not be re-issued, and any new rules will be assigned the next higher number - regardless of location in the text.

`Courier` – All words appearing in **`bolded courier font`** are values, objects or

keywords. When defining rules the following annotations are used:

- [ ] = optional
- < > = Variable
- | = choice

4.5 Related Documents

Related documents referenced in this specification are listed in Appendix A.

4.6 Conformance

Applications will be considered to be in full conformance with this technical specification if they comply with the content of normative sections, rules and definitions.

[R1]	Conformance shall be determined through adherence to the content of normative sections, rules and definitions.

4.7 Guiding Principles

The following guiding principles were used as the basis for all design rules contained in this document:

[1] Key words for use in RFCs to Indicate Requirement Levels - Internet Engineering Task Force, Request For Comments 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

- Relationship to UMM – UN/CEFACT XML Schema Definition Language (XSD) Schema will be based on UMM metamodel adherent Business Process Models.
- Relationship to Information Models – UN/CEFACT XSD Schema will be based on information models developed in accordance with the UN/CEFACT – *Core Components Technical Specification*.
- Schema Creation– UN/CEFACT XML design rules will support schema creation through handcrafting as well as automatic generation.
- Interchange and Application Use – UN/CEFACT XSD Schema and instance documents are intended for business-to-business and application-to-application use.
- Tool Use and Support - The design of UN/CEFACT XSD Schema will not make any assumptions about sophisticated tools for creation, management, storage, or presentation being available.
- Legibility - UN/CEFACT XML instance documents should be intuitive and reasonably clear in the context for which they are designed.
- Schema Features - The design of UN/CEFACT XSD Schema should use the most commonly supported features of W3C XSD Schema.
- Technical Specifications – UN/CEFACT XML Naming and Design Rules will be based on Technical Specifications holding the equivalent of W3C recommended status.
- Schema Specification – UN/CEFACT XML Naming and Design rules will be fully conformant with W3C XML Schema Definition Language.
- Interoperability - The number of ways to express the same information in a UN/CEFACT XSD Schema and UN/CEFACT XML instance document is to be kept as close to one as possible.
- Maintenance – The design of UN/CEFACT XSD Schema must facilitate maintenance.
- Context Sensitivity - The design of UN/CEFACT XSD Schema must ensure that context-sensitive document types are not precluded.
- Relationship to Other Namespaces - UN/CEFACT XML design rules will be cautious about making dependencies on other namespaces.
- Legacy formats - UN/CEFACT XML Naming and Design Rules are not responsible for sustaining legacy formats.

5 General XML Construct

This section defines rules related to general XML constructs to include:

- Overall Schema Structure
- Relationship to CCTS
- Naming and Modelling Constraints
- Reusability Scheme
- Modularity Model
- Namespace Scheme
- Schema Location
- Versioning Scheme

5.1 Overall Schema Structure

UN/CEFACT has determined that the World Wide Web Consortium (W3C) XML schema definition (XSD) language is the generally accepted schema language experiencing the broadest adoption. Accordingly, all UN/CEFACT normative schema will be expressed in XSD. All references to XML schema will be as XSD schema or UN/CEFACT XSD Schema.

[R2]	All UN/CEFACT XSD Schema design rules MUST be based on the *W3C XML Schema Recommendations: XML Schema Part 1: Structures and XML Schema Part 2: Data Types*

The W3C is the recognized source for XML specifications. W3C specifications can hold various statuses. Only those W3C specifications holding recommendation status are guaranteed by the W3C to be stable specifications.

[R3]	All UN/CEFACT XSD Schema and UN/CEFACT conformant XML instance documents MUST be based on the W3C suite of technical specifications holding recommendation status.

To maintain consistency in lexical form, all UN/CEFACT XSD Schema need to use a standard structure for all content. This standard structure is contained in Appendix B.

[R4]	UN/CEFACT XSD Schema MUST follow the standard structure defined in Appendix B.

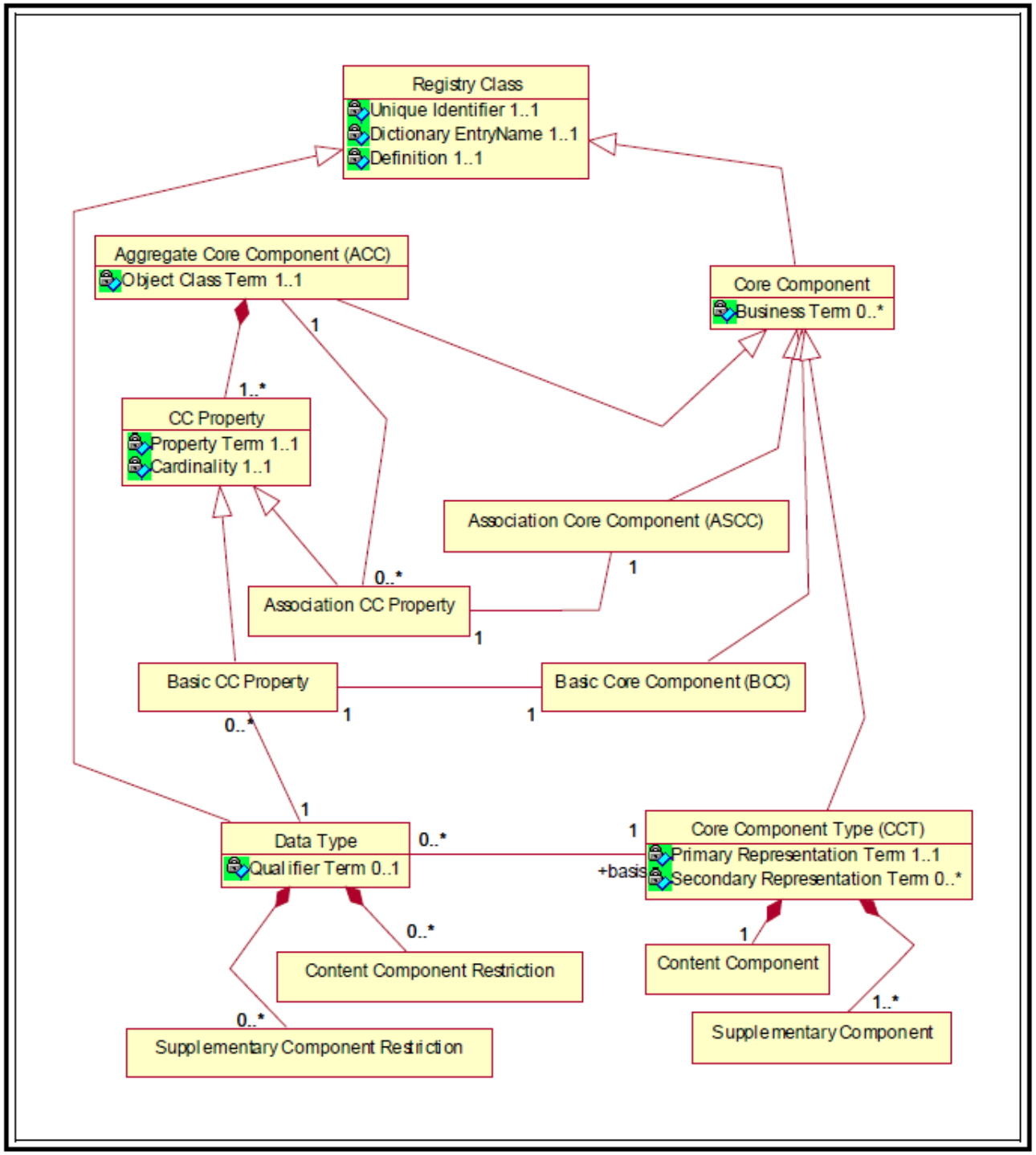
5.2 Relationship to CCTS

All UN/CEFACT business information modelling and business process modelling employ the methodology and model described in CCTS.

5.2.1 CCTS

CCTS defines context neutral and context specific information building blocks. Context neutral information components are defined as Core Components (**`ccts:CoreComponents`**). Context neutral **`ccts:CoreComponents`** are defined in CCTS as "A building block for the creation of a semantically correct and meaningful information exchange package. It contains only the information pieces necessary to describe a specific concept."[2] Figure 5-1 illustrates the various pieces of the overall **`ccts:CoreComponents`** metamodel.

[2] *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.01 (Second Edition), UN/CEFACT, 15 November 2003, plus corrigenda*



345 **Figure 5-1 Core Component Metamodel**

346 5.2.2 Business Information Entities

347 In the CCTS model, context neutral core components are instantiated as context specific
348 components for business information payload and model harmonization. The context specific
349 components are defined as Business Information Entities. (See CCTS Section 6.2 for a detailed
350 discussion of the UN/CEFACT context mechanism.) [3] Context specific CCTS Business

[3] *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.0 (Second Edition),* UN/CEFACT, 15 November 2003

Information Entities are defined in CCTS as "A piece of business data or a group of pieces of business data with a unique business semantic definition."[4]

Figure 5-2 illustrates the various pieces of the overall **`ccts:BusinessInformationEntity`** metamodel and their relationship with the **`ccts:CoreComponents`** metamodel.

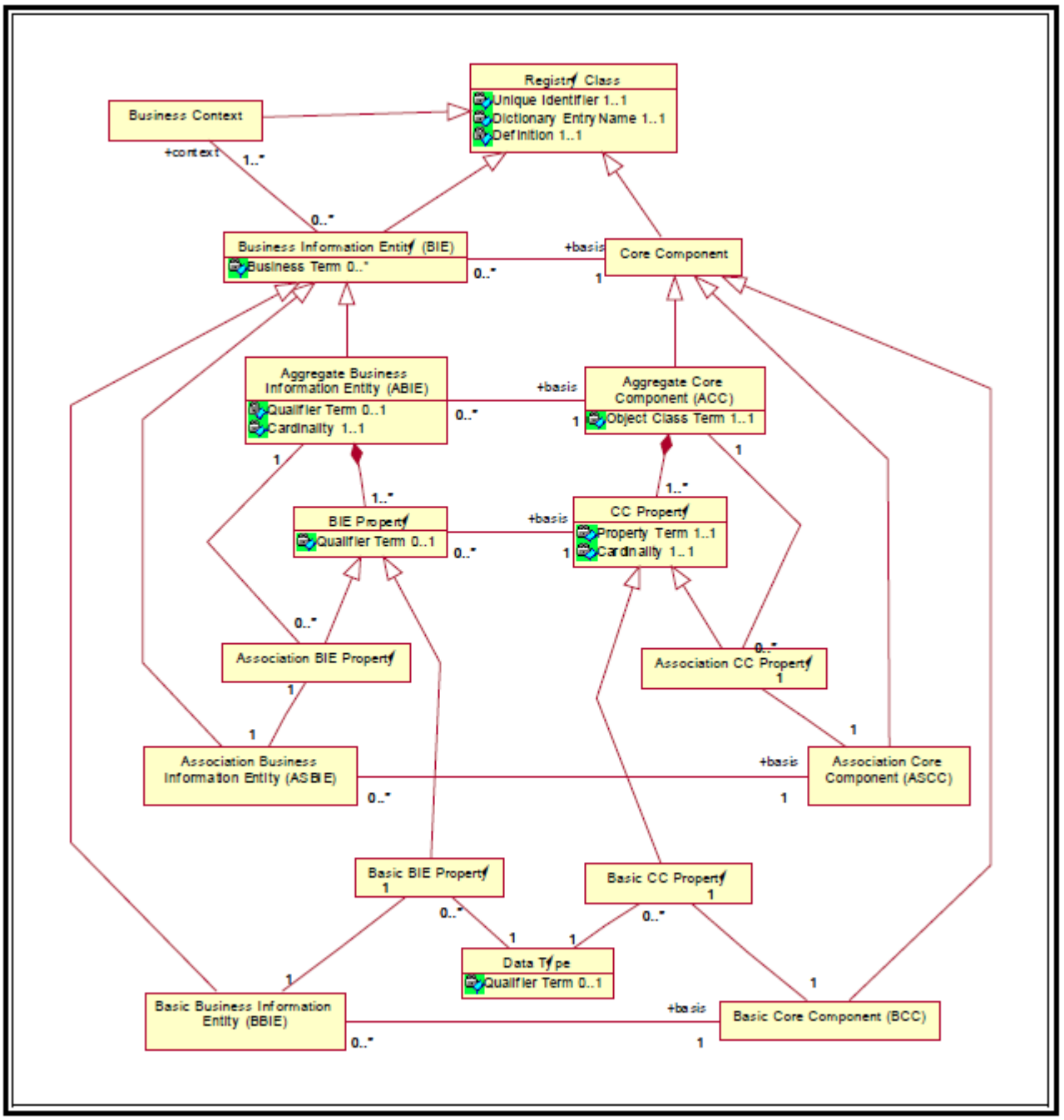


Figure 5-2 Context Specific Business Information Entity Metamodel

5.2.3 The XML Constructs

UN/CEFACT XML design rules are closely coupled with CCTS. UN/CEFACT XSD Schema will be developed from fully conformant Business Information Entities that are based on fully conformant Core Components. Figure 5-3 shows the relationship between CC's, BIE's and XSD artefacts. The grey boxes reflect CCTS constructs (Core Component Types, Data Types, Core Components, and

[4] Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.01, UN/CEFACT, 15 November 2003

Business Information Entities), and the other boxes reflect XSD constructs (**`xsd:type`**, **`xsd:element`**, **`xsd:attribute`**). The relationships follow the following basic principles:

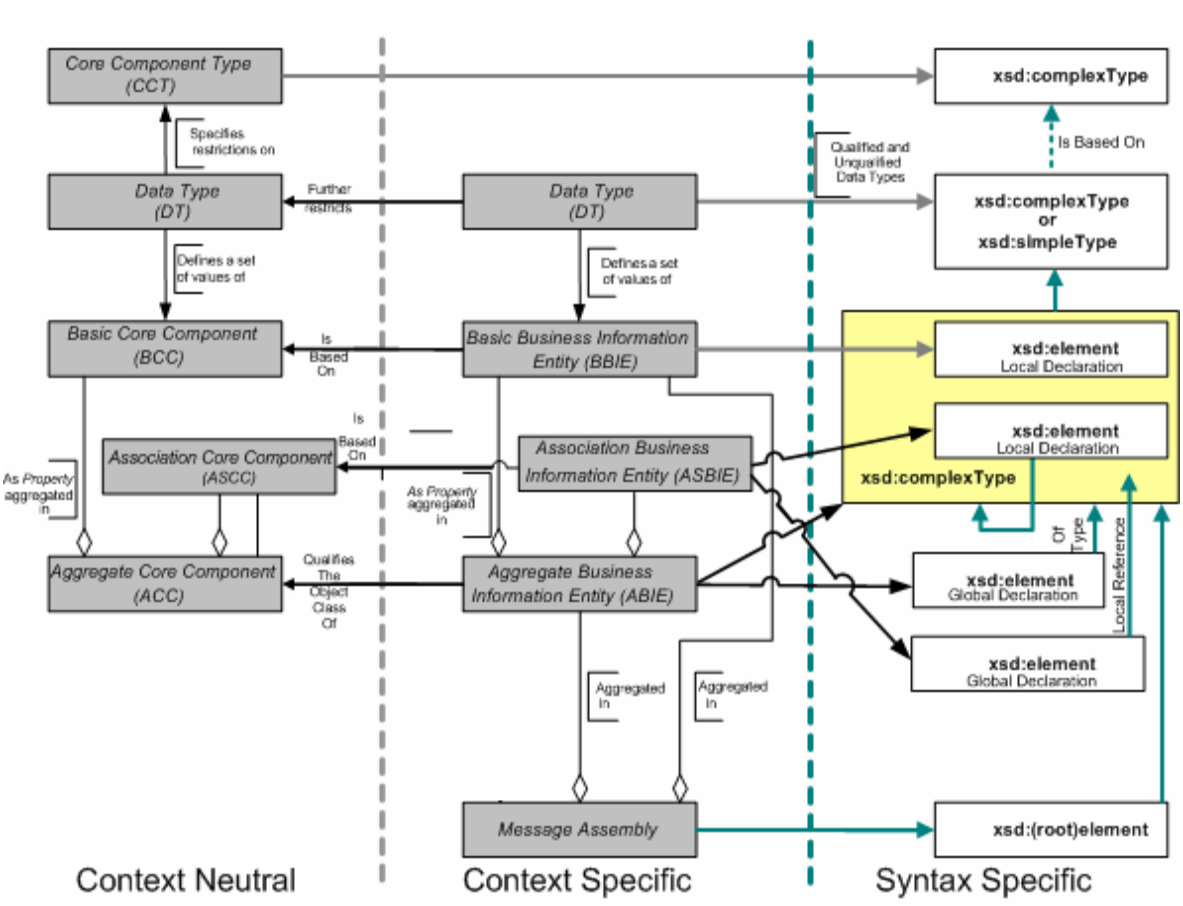


Figure 5-3 Relationship between CCTS and XSD Artefacts in UN/CEFACT XSD Schema

- The business information payload (Message Assembly) is represented as a **`xsd:complexType`** definition and global element declaration in an UN/CEFACT XSD Schema. The global element declaration is based on (is of type) **`xsd:complexType`** that represents the document level ABIE. The global element appears in, and is designated as the root element of, UN/CEFACT conformant XML instances.
- An ABIE is defined as a **`xsd:complexType`** and a corresponding global **`xsd:element`** is declared.
- Depending on the type of association, an ASBIE will be declared as either a local element or as a global element. If the ASBIE is a composition it will be declared as a local element within the **`xsd:complexType`** representing the associating ABIE. If it is not a composition (i.e., aggregation) the ASBIE is included in the content model by referencing the global element that was declared for the associated ABIE. The ASBIE element is in itself based on (is of type) **`xsd:complexType`** of the associated ABIE. In this way the content model of the associated ABIE is included in the content model of the associating ABIE.

> [Note]
>
> Per CCTS, an ABIE can contain other ABIEs in ever higher levels of aggregation. When an ABIE contains another ABIE, this is accomplished through the use of ASBIEs. The ASBIE is the linking mechanism that shows the hierarchical relationship between ABIE constructs. When an ASBIE is used, we refer to the ABIE that contains it as the associating ABIE, and the ABIE that it represents as the associated ABIE.

- A BBIE is declared as a local element within the **`xsd:complexType`** representing the parent ABIE. The BBIE is based on a (is of type) qualified or unqualified data type (DT).

- A DT is defined as either a **xsd:complexType** or **xsd:simpleType**. DT's are based on Core Component Type **xsd:complexType** from the Core Component Type (CCT) schema module. These data types can be unqualified (no additional restrictions above those imposed by the CCT type) or qualified (additional restrictions above those imposed by the CCT type). XSD built-in data types will be used whenever the facets of the built-in data type are equivalent to the CCT supplementary components for that data type.

> [Note]
>
> Data Types are not derived from the CCT complex types using **xsd:restriction** because whereas all CCTs are defined as complex types with attributes representing their supplementary components, in some cases built-in XSD data types whose facets correspond to the supplementary components are leveraged . See Section 7.5 for more information.

- A CCT is defined as a **xsd:complexType**. Supplementary components are declared as attributes for the CCT **xsd:complexType**. CCTs are contained in the Core Component Type Schema Module which is considered the normative XSD expression of CCTS Core Component Type.

5.3 Naming and Modelling Constraints

UN/CEFACT XSD Schema are derived from components created through the application of CCTS, UN/CEFACT Modelling Methodology (UMM) process modelling and data analysis, and Core Component Business Document Assembly (CCBDA). UN/CEFACT XSD Schema contain XML syntax specific constructs that follow the naming and design rules in this specification. Those naming and design rules have taken advantage of the features of XSD to incorporate naming constraint rules that in many cases result in truncation of the CCTS dictionary entry names. However, the fully conformant CCTS dictionary entry names of the underlying CCTS registry artefact are preserved as part of the **xsd:<annotation>** element accompanying each element declaration in UN/CEFACT schema, and can be reconstructed through use of XPath expressions. The XML fully qualified XPath ties the information to its standardized semantics as described in the underlying CCTS construct and CCTS dictionary entry name, while the XML element or attribute name is a truncation that reflects the hierarchy inherent in the XML construct. There are differences in the rules for naming of elements, attributes, and types.

[R5]	Each element or attribute XML name MUST have one and only one fully qualified XPath(FQXP)

This rule and the other rules on element naming imply that a part of the fully qualified XPath will always represent the CCTS dictionary entry name of the corresponding ABIE, BBIE, ASBIE or DT.

Example 5-1: Fully Qualified XPath

```
/CrossIndustryInvoice/CIExchangedDocumentContext/SpecifiedTransaction/Identifier
/Tender/ProcuringOrganization/Name/Text
```

The official language for UN/CEFACT is English. All official XML constructs as published by UN/CEFACT will be in English. XML development work may very well occur in other languages, however official submissions for inclusion in the UN/CEFACT XML library must be in English. Other language translations of UN/CEFACT published XML components are at the discretion of users.

[R6]	Element, attribute and type names MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary.

Following the *ebXML Architecture Specification* and commonly used best practice, Lower Camel Case (LCC) is used for naming attributes and Upper Camel Case (UCC) is used for naming elements and types. Lower Camel Case capitalizes the first character of each word except the first word and compounds the name. Upper Camel Case capitalizes the first character of each word and compounds the name.

[R7]	Lower camel case (LCC) MUST be used for naming attributes

Example 5-2: Attribute

```
<xsd:attribute name="unitCode" .../>
```

[R8] Upper camel case (UCC) MUST be used for naming elements and types.

Example 5-3: Element

```
<xsd:element name="LastReportedSubmissionDateTime" ...>
```

Example 5-4: Type

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

[R9] Element, attribute and type names MUST be in singular form unless the concept itself is plural.

Example 5-5: Singular and Plural Concept Form

Allowed - Singular:

```
<xsd:element name="GoodsCharacteristic" ...>
```

Not Allowed - Plural:

```
<xsd:element name="ItemsQuantity" ...>
```

[R10] Element, attribute and type names MUST be drawn from the following character set: **`a-z`** and **`A-Z.`** Any special characters such as spaces, underscores, and periods that exist in the underlying Dictionary Entry Names MUST be removed.

Example 5-6: Non-Letter Characters

Not Allowed

```
<xsd:element name="LanguageCode8" ...>
```

The CCTS allows for the use of periods, spaces and other separators in the dictionary entry name. XML best practice is to not include these in an XML tag name. Additionally, XML 1.0 specifically prohibits the use of certain reserved characters in XML tag names.

[R11] This rule has been combined with [R10].

Example 5-7: Spaces in Name

Not Allowed

```
<xsd:element name="Customized_ Language. Code:8" ...>
```

[R12] XML element, attribute and type names MUST NOT use acronyms, abbreviations, or other word truncations, except those included in the UN/CEFACT controlled vocabulary or listed in Appendix C.

[R13] The acronyms and abbreviations listed in Appendix C MUST always be used.

[R14] Acronyms and abbreviations at the beginning of an attribute declaration MUST appear in all lower case. All other acronym and abbreviation usage in an attribute declaration must appear in upper case.

[R15] Acronyms MUST appear in all upper case for all element declarations and type definitions.

Example 5-8: Acronyms and Abbreviations

Allowed – ID is an approved abbreviation

```
<xsd:attribute name="currencyID"
```

Not Allowed – Cd is not an approved abbreviation, if it was an approved abbreviation it must appear in all upper case

```
<xsd:simpleType name="temperatureMeasureUnitCdType>
```

5.3.1 Element Naming Conventions

The fully qualified XPath anchors the use of a construct to a particular location in a business information payload. The dictionary definition identifies any semantic dependencies that the FQXP has on other elements and attributes within the UN/CEFACT library that are not otherwise enforced or made explicit in its structural definition. The dictionary serves as a traditional data dictionary, and also serves some of the functions of traditional implementation guides.

5.4 Reusability Scheme (Informative)

UN/CEFACT is committed to transitioning to an object based approach for its process models and core component implementation efforts as supported in both UMM and CCTS. UN/CEFACT deliberated adopting a type based approach (named types), a type and element based approach, or an element based approach.

A type based approach for XML management provides the closest alignment with the process modelling methodology described in UMM. Type information is beginning to be accessible when processing XML instance documents. Post Schema-Validation Infoset (PSVI) capabilities are beginning to emerge that support this approach, such as “data-binding” software that compiles schema into ready-to-use object classes and is capable of manipulating XML data based on their types. The most significant drawback to a type based approach is the risk of developing an inconsistent element vocabulary where elements are declared locally and allowed to be reused without regard to semantic clarity and consistency across types. UN/CEFACT manages this risk by carefully controlling the creation of BBIEs and ASBIEs with fully defined semantic clarity that are only usable within the ABIE in which they appear. This is accomplished through the relationship between BBIEs, ASBIEs and their parent ABIE and the strict controls put in place for harmonization and approval of the semantic constructs prior to their XSD instantiation.

A purely type based approach does, however, limit the ability to reuse elements, especially in technologies such as Web Services Description Language (WSDL). UN/CEFACT has thus decided to implement what is known as a “hybrid approach” as this provides benefits over a purely type based approach. Most significantly it increases reusability of library content both at the modelling and xsd level.

The key principles of the “hybrid approach” are:

- All classes (PurchaseOrderRequest, Seller_Party, Buyer_Party, Ordered_LineItem and ProductOrService_Item in figure 5-4) are declared as a **`xsd:complexType`**.
- All attributes of a class are declared as a local **`xsd:element`** within a **`xsd:complexType`**.
- Composition associations (e.g. PurchaseOrderRequest. Ordered. Ordered_LineItem in figure 5-4) are locally declared as a **`xsd:element`** within a **`xsd:complexType`**. A composition ASBIE is defined as a specialized type of ASBIE that represents a composition relationship between the associating ABIE and the associated ABIE.
- Associations that are not defined as composites (e.g. PurchaseOrderRequest.Buyer. Buyer_Party, PurchaseOrderRequest. Seller. SellerParty in figure 5-4) are globally declared as a **`xsd:element`**.

The rules pertaining to the ‘hybrid approach’ are contained in sections 7.3.4 and 7.3.5 for type and element declaration.

Figure 5-4 shows an example UML model and example 5-9 shows the resulting XSD declarations.

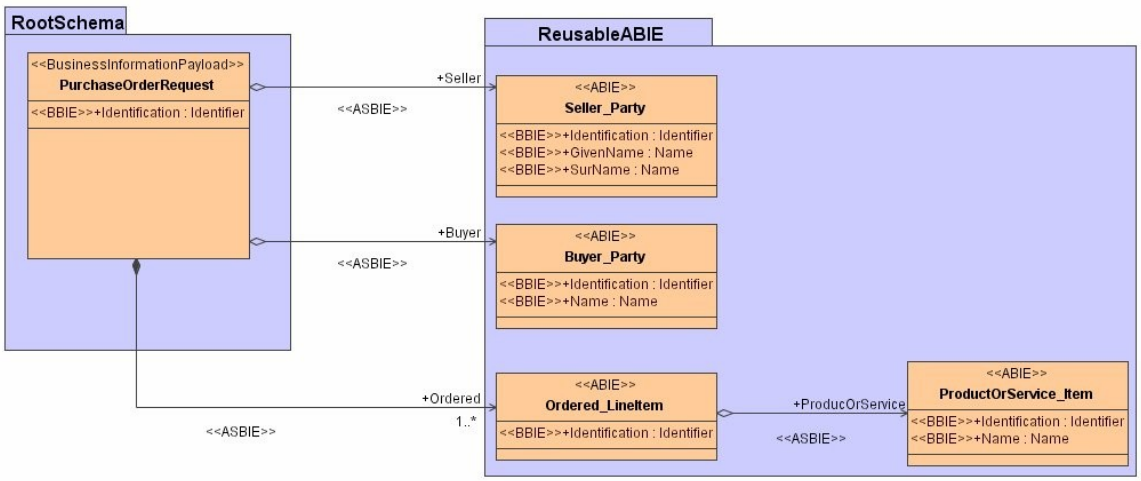


Figure 5-4 UML model example

Example 5-9: xsd declarations representing Figure 5-4

```
    <xsd:element name="PurchaseOrderRequest" type="rsm:PurchaseOrderRequestType"/>
    <xsd:element name="BuyerParty" type="ram:BuyerPartyType"/>
    <xsd:element name="OrderedLineItem" type="ram:OrderedLineItemType"/>
    <xsd:element name="ProductOrServiceItem" type="ram:ProductOrServiceItemType"/>
    <xsd:element name="SellerParty" type="ram:SellerPartyType"/>
    <xsd:complexType name="PurchaseOrderRequestType">
       <xsd:sequen
           ce>
          <xsd:element name="ID" type="udt:IDType"/>
          <xsd:element ref="ram:SellerParty"/>
          <xsd:element ref="ram:BuyerParty"/>
          <xsd:element name="OrderedLineItem"
   type="ram:OrderedLineItemType" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="BuyerPartyType">
       <xsd:sequen
           ce>
          <xsd:element name="ID" type="udt:IDType"/>
          <xsd:element name="Name" type="udt:NameType"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="OrderedLineItemType">
       <xsd:sequen
           ce>
          <xsd:element name="ID" type="udt:IDType"/>
          <xsd:element name="ProductOrServiceItem" type="ram:ProductOrServiceItemType"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ProductOrServiceItemType">
       <xsd:sequen
           ce>
          <xsd:element name="ID" type="udt:IDType"/>
          <xsd:element name="Name" type="udt:NameType"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SellerPartyType">
       <xsd:sequen
           ce>
          <xsd:element name="ID" type="udt:IDType"/>
          <xsd:element name="GivenName" type="udt:NameType"/>
          <xsd:element name="Surname" type="udt:NameType"/>
       </xsd:sequence>
    </xsd:complexType>
```

5.5 Modularity Model

Modularity in schema design promotes reuse and provides significant management capabilities. Modules can be either unique in their functionality, or represent splitting of larger schema files for

performance or manageability enhancement. A modularity model provides an efficient and effective mechanism for importing and including components as needed rather than dealing with complex, multi-focused schema.

Accordingly UN/CEFACT has defined a number of schema modules to support this approach. Figure 5-5 portrays the UN/CEFACT modularity model. UN/CEFACT categorizes modules into business information payload(s) and external schema modules. The business information payload consists of root schema and internal schema modules that reside in the same namespace as the root schema. The external schema modules consist of a set of reusable schema for ABIEs, unqualified data types, qualified data types, code lists and identifier lists. Each of these schema modules resides in its own namespace. Dependencies exist amongst the various modules as shown in figure 5-5.

The root schema module always includes any internal schema residing in its namespace. It also always imports the ABIE reusable, unqualified and qualified data type schema modules. It may import root schemas from other namespaces as well as reusable schema from other standards bodies. The internal schema module may include other internal schema modules from its own namespace, and may reference – through the root schema module– other root schema modules and their internal schema modules. It may also import the unqualified data type, qualified data type, and reusable ABIE schema modules.

The reusable ABIE schema module always imports the unqualified data type and qualified data type schema modules. The unqualified data type schema imports necessary code list schema modules and may import identifier list schema modules. The qualified data type schema modules always import the unqualified data type schema module as well as necessary code list and identifier list schema modules.

The core component type schema module is provided as reference documentation and is used as the basis for the unqualified data type schema module. The modularity approach has been designed so that there are no circular imports.

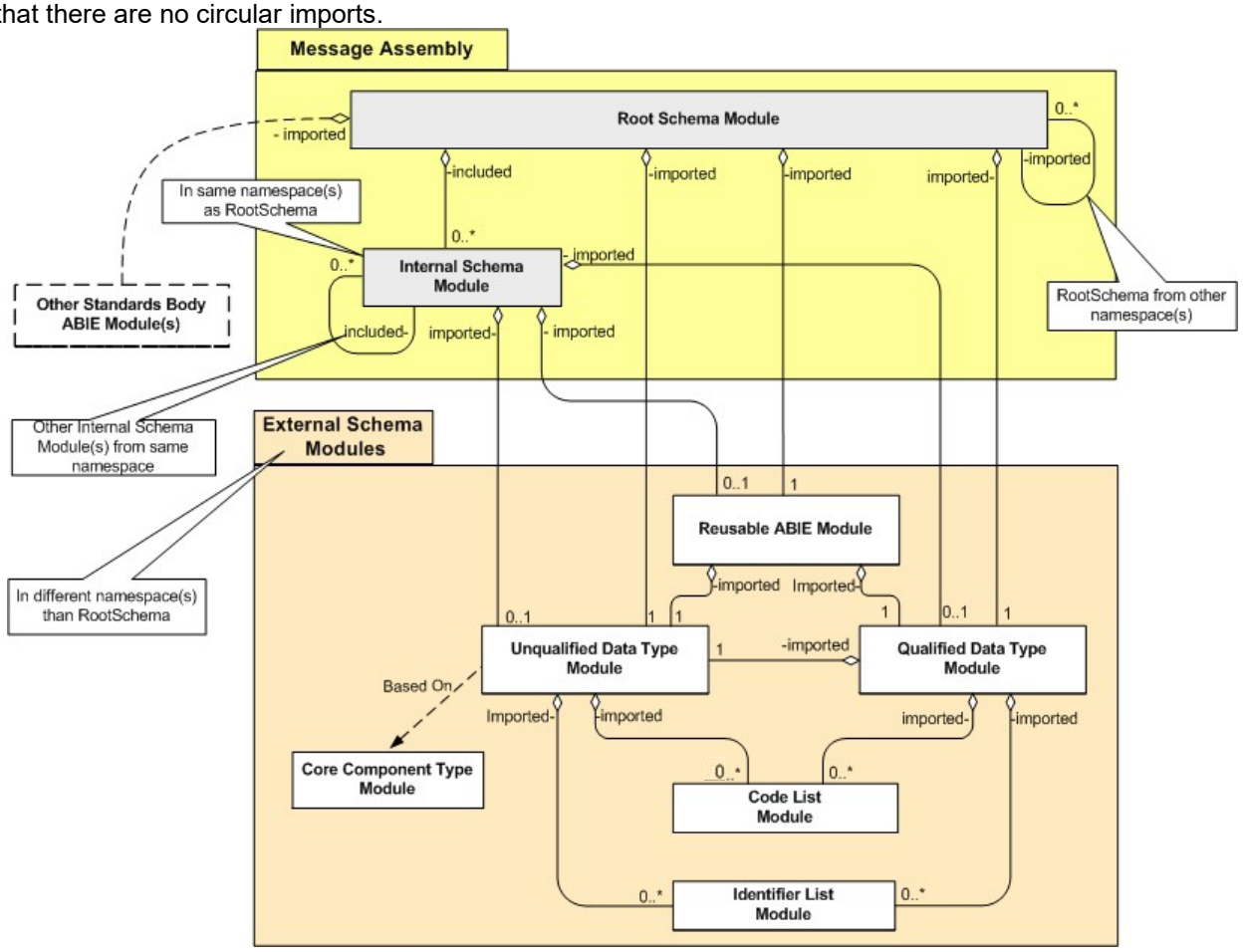


Figure 5-5 UN/CEFACT XSD Schema Modularity Scheme

To ensure consistency, and for standardization of namespace tokens as addressed elsewhere in this specification, all schema modules identified above are referred to by their formal name or token value in the table below:

Schema Module Name	Token
Root Schema Schema Module	rsm
Core Component Type Schema Module	cct
Reusable Aggregate Business Information Entity Schema Module	ram
Unqualified Data Type Schema Module	udt
Qualified Data Type Schema Module	qdt
Code List Schema Module	clm
Identifier List Schema Module	ids

[R16] The schema module file name for modules other than code lists or identifier lists MUST of the form **`<SchemaModuleName>_<Version>.xsd`**, with periods, spaces, or other separators and the words Schema Module removed.

[R17] The schema module file name for code lists and identifier lists, MUST be of the form **`<AgencyName>_<ListName>_<Version>.xsd`**, with periods, spaces, or other separators removed.

[R18] In representing versioning schemes in file names, only the major version should be included.

5.5.1 Root Schema

UN/CEFACT incorporates a modularity concept that leverages the benefits previously described. In the UN/CEFACT XML repository, there are a number of UN/CEFACT root schema, each of which expresses a separate business function.

[R19] A root schema MUST be created for each unique business information payload.

To ensure uniqueness, root schema modules will be given unique names that reflect the business function being addressed by the schema. This business function is described in the UN/CEFACT Requirements Specification Mapping (RSM) document as the target business information payload. Accordingly, the business information payload name representing the business function will form the basis for the root schema name.

[R20] Each UN/CEFACT root schema module MUST be named <BusinessInformationPayload> Schema Module.

The UN/CEFACT modularity approach enables the reuse of individual root schema without having to import the entire UN/CEFACT root schema library. Additionally, a root schema can import individual modules without having to import all UN/CEFACT XSD schema modules. Each root schema will define its own dependencies. A root schema should not duplicate reusable XML constructs contained in other schema, rather it should reuse existing constructs available elsewhere. Specifically, root schema will import or include other schema modules to maximize reuse through **`xsd:include`** or **`xsd:import`** as appropriate.

[R21] A root schema MUST NOT replicate reusable constructs available in schema modules capable of being referenced through **`xsd:include`** or **`xsd:import`**.

Schema modules used by the root schema need to be treated as either internal or external schema modules so correct namespace decisions can be made.

[R22] UN/CEFACT XSD schema modules MUST either be treated as external schema modules, or as internal schema modules of the root schema.

5.5.2 Internal Schema

The Core Component Business Document Assembly (CCBDA) specification provides a mechanism for restricting ABIEs in order to assemble a single message. Messages in an XML context correspond to a root schema, and as such, the restricted ABIEs would be declared in an internal schema. These ABIEs will be defined as **`xsd:complexType`** in an internal schema module rather than in the reusable ABIE schema module, (See Section 5.5.3.4 below). UN/CEFACT XSD Schema may have zero or more internal schema modules.

Internal schema modules will reside in the same namespace as their parent root schema. Since the internal schema reside in the same namespace as the root, the root schema uses **`xsd:include`** to incorporate these internal modules. The UN/CEFACT XSD schema modularity approach ensures that logical associations exist between root and internal schema modules and that individual schema modules can be reused to the maximum extent possible.

[R23]	All UN/CEFACT internal schema modules MUST be in the same namespace as their corresponding **`rsm:RootSchema.`**

UN/CEFACT internal schema modules will have a semantically meaningful name. Internal schema module names will identify the parent root schema module, the internal schema module function, and the schema module itself.

[R24]	Each UN/CEFACT internal schema module MUST be named **`<ParentRootSchemaModuleName><InternalSchemaModuleFunction> Schema Module`**

Example 5-10: UN/CEFACT internal schema module name

```
TravelReservationRequestFlightInformation
Where:
TravelReservationRequest represents the parent root schema module name
FlightInformation represents the internal schema module function
```

5.5.3 External Schema

To adhere to the principles and rules contained in Section 7, schema modules will be created for reusable components. These schema modules are referred to as external schema modules because they reside in a different namespace from the root schema. Root schema may import one or more of these external schema modules. UN/CEFACT has identified the need for the following external schema modules:

- Unqualified Data Type
- Qualified Data Type
- Reusable ABIE
- Code List
- Identifier List
- Other Standards Body ABIE module

> [Note]
>
> The terms “unqualified data type” and “qualified data type” refer to the ISO 11179 concept of qualifiers for name constructs, not to the xml namespace concept of qualified and unqualified

These external schema modules are reflected in Figure 5-6.

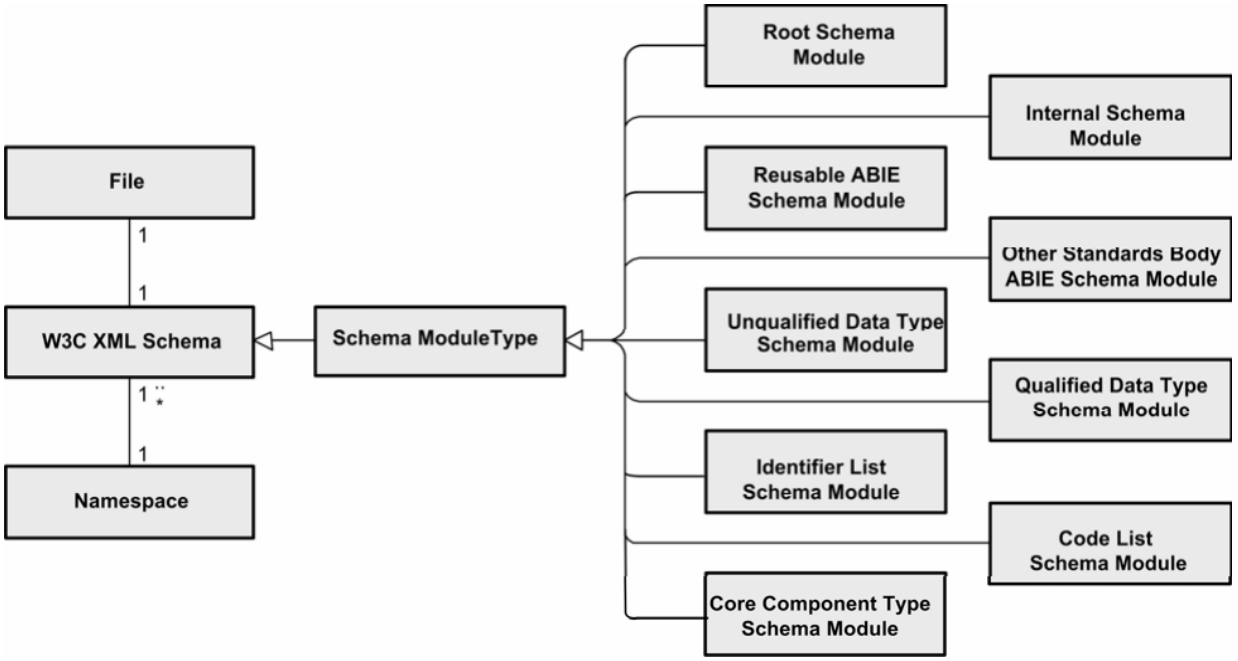


Figure 5-6 UN/CEFACT XSD Schema Modules

5.5.3.1 Core Component Type Schema Module

A schema module is required to represent the normative form for CCTs from CCTS. This schema module will be used as the normative reference for all CCTS based XML instantiations. This schema will form the basis of the UDT schema module, however it will never be imported directly into any UN/CEFACT schema module.

[R25]	A Core Component Type schema module MUST be created.

The Core Component Type schema module will have a standardized name that uniquely differentiates it from other UN/CEFACT XSD schema modules.

[R26]	The **`cct:CoreComponentType`** schema module MUST be named 'Core Component Type Schema Module'.

5.5.3.2 Unqualified Data Type Schema Module

A schema module is required to represent the normative form data types for each CCT as expressed in the CCTS meta model. These data types are based on the XSD constructs from the CCT schema module but where possible reflect the use of XSD built-in data types defined as **`xsd:simpleType`** rather than their parent CCT **`xsd:complexType`**. As such, the unqualified data type schema module does not import the CCT schema module.

An unqualified data type is defined for all approved CCTS primary and secondary representation terms.

[R203]	An Unqualified Data Type MUST NOT contain any restriction on their source CCTs other than those defined in CCTS and agreed upon best practices.
[R27]	An Unqualified Data Type schema module MUST be created

The unqualified data type schema module will have a standardized name that uniquely differentiates it from other UN/CEFACT XSD schema modules.

[R28]	The **`udt:UnqualifiedDataType`** schema module MUST be named 'Unqualified Data TypeSchema Module'

5.5.3.3 Qualified Data Type Schema Module

As data types are reused for different BIEs, restrictions on the data type may be applied. These restricted data types are referred to as qualified data types. These qualified data types will be defined in a separate qualified data type schema module. The qualified data type schema module

will import the Unqualified Data Type Schema Module. In the future, this single qualified data type schema module may be segmented into additional modules if deemed necessary.

[R29] A Qualified Data Type schema module MUST be created.

The qualified data type schema module will have a standardized name that uniquely differentiates it from other UN/CEFACT XSD schema modules.

[R30] The **`qdt:QualifiedDataType`** schema module MUST be named 'Qualified Data Type Schema Module'.

5.5.3.4 Reusable Aggregate Business Information Entity Schema Module

A single reusable aggregate business information entity schema module is required. This schema module will contain a type definition and element declaration for every reusable ABIE in the UN/CEFACT Core Component Library. In the future this single reusable schema module may be segmented into additional modules if deemed necessary. This single reusable schema module may be compressed for runtime performance considerations if necessary. Compression means that a runtime version of the reusable ABIE schema module would be created that would consist of a subset of the ABIE constructs. This subset would consist only of those ABIEs necessary to support the specific root schema being validated.

[R31] A Reusable Aggregate Business Information Entity schema module MUST be created.

The reusable aggregate business information entity schema module will have a standardized name that uniquely differentiates it from other UN/CEFACT XSD schema modules.

[R32] The **`ram:ReusableAggregateBusinessInformationEntity`** schema module MUST be named 'Reusable Aggregate Business Information Entity Schema Module'.

5.5.3.5 Code List Schema Modules

In cases where a code list is required or used, reusable code list schema modules will be created to minimize the impact of code list changes on root and other reusable schema. Each reusable code list schema module will contain enumeration values for codes and code values.

[R33] Reusable Code List schema modules MUST be created to convey code list enumerations.

Code list schema modules will have a standardized name that uniquely differentiates it from other UN/CEFACT XSD schema modules and external organization generated code list modules.

[R34] The name of each **`clm:CodeList`** schema module MUST be of the form: **`<Code List Agency Identifier|Code List Agency Name><Code List Identification Identifier|Code List Name> - Code List Schema Module`**
Where:
Code List Agency Identifier = Identifies the agency that maintains the code list
Code List Agency Name = Agency that maintains the code list
Code List Identification Identifier = Identifies a list of the respective corresponding codes
Code List Name = The name of the code list as assigned by the agency that maintains the code list

Example 5-11: Name of UN/CEFACT Account Type Code Schema Module

```
63139 - Code List Schema Module
where:
6 = Code list agency identifier for UN/CEFACT as defined in UN/CEFACT code
    list 3055
3139 = Code list identification identifier for Contact Type Code in UN/CEFACT
    directory
```

Example 5-12: Name for a code using agency name and code list name

```
    Planning Level Code - Code List Schema Module
```

5.5.3.6 Identifier List Schema Modules

Whereas codes are normally part of a finite list that are suitable for runtime validation, identifiers may or may not be suitable for creation as a discrete list of identification schemes and subsequently validated during runtime. In those cases where runtime validation is required against a used identifier scheme, a separate identifier list schema module will be created to minimize the impact of identifier list changes on root and other reusable schema. Each reusable identifier list schema module will contain enumerated values for the identifiers.

[R35]	An identifier list schema module MUST be created to convey enumerated values for each identifier list that requires runtime validation.

Identifier list schema modules will have a standardized name that uniquely differentiates it from other UN/CEFACT XSD schema modules or external organization generated schema modules.

[R36]	The name of each **`ids:IdentifierList`** schema module MUST be of the form: **`<Identifier Scheme Agency Identifier|Identifier Scheme Agency Name><Identifier Scheme Identifier|Identifier Scheme Name> - Identifier List Schema Module`** Where: Identifier Scheme Agency Identifier = identification of the agency that maintains the identifier list Identifier Scheme Agency Name = Agency that maintains the identifier list Identifier Scheme Identifier = identification of the identifier list Identification Scheme Name = Name as assigned by the agency that maintains the identifier list

Example 5-13: Name of ISO Country Identifier schema module

```
53166-1 - Identifier List Schema Module
where:
5 = Code list agency identifier for ISO as defined in UN/CEFACT code list 3055
3166-1 =  Identifier scheme identifier for Two Alpha Country Identifier in ISO
```

5.5.3.7 Other Standards Body Aggregate Business Information Entity Schema Modules

Other Standards Body ABIE schema modules are those reusable XML constructs created by standards bodies other than UN/CEFACT and made publicly available. UN/CEFACT will only import other Standards Body ABIE schema modules when their contents are in strict conformance to the requirements of the CCTS and this specification.

[R37]	Imported schema modules MUST be fully conformant with the UN/CEFACT *XML Naming and Design Rules* Technical Specification and the UN/CEFACT *Core Components Technical Specification.*

5.6 Namespace Scheme

A namespace is a collection of names for elements, attributes and types that serve to uniquely distinguish the collection from the collection of names in another namespace. As defined in the W3C XML specification, “XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.”[5] This enables interoperability and consistency in the XML artefacts for the library of reusable types and schema modules. The UN/CEFACT reusability methodology maximizes the reuse of defined named types, a combination of locally and globally declared elements, and attributes (See Section 5.4).

In addition, the modularity approach of multiple reusable schema modules (See Section 5.5) prescribe just such a method. There exist specific relationships between the various internal and external schema modules identified in Section 5.5 with respect to their namespaces. These relationships are defined in Figure 5-5. Accordingly, a sufficiently robust namespace scheme is essential.

[5] World Wide Web Consortium, *Namespaces in XML*, 14 January 1999

5.6.1 Namespace Scheme

In establishing a UN/CEFACT approach to namespaces, it is important to recognize that in addition to XML requirements, many other requirements exist for a standardized namespace approach. Accordingly, a master UN/CEFACT namespace scheme must be sufficiently flexible and robust to accommodate both XML and other syntax requirements. Figure 5-7 reflects such an approach and will be used as the basis for determining the namespace structure and rules that follow.

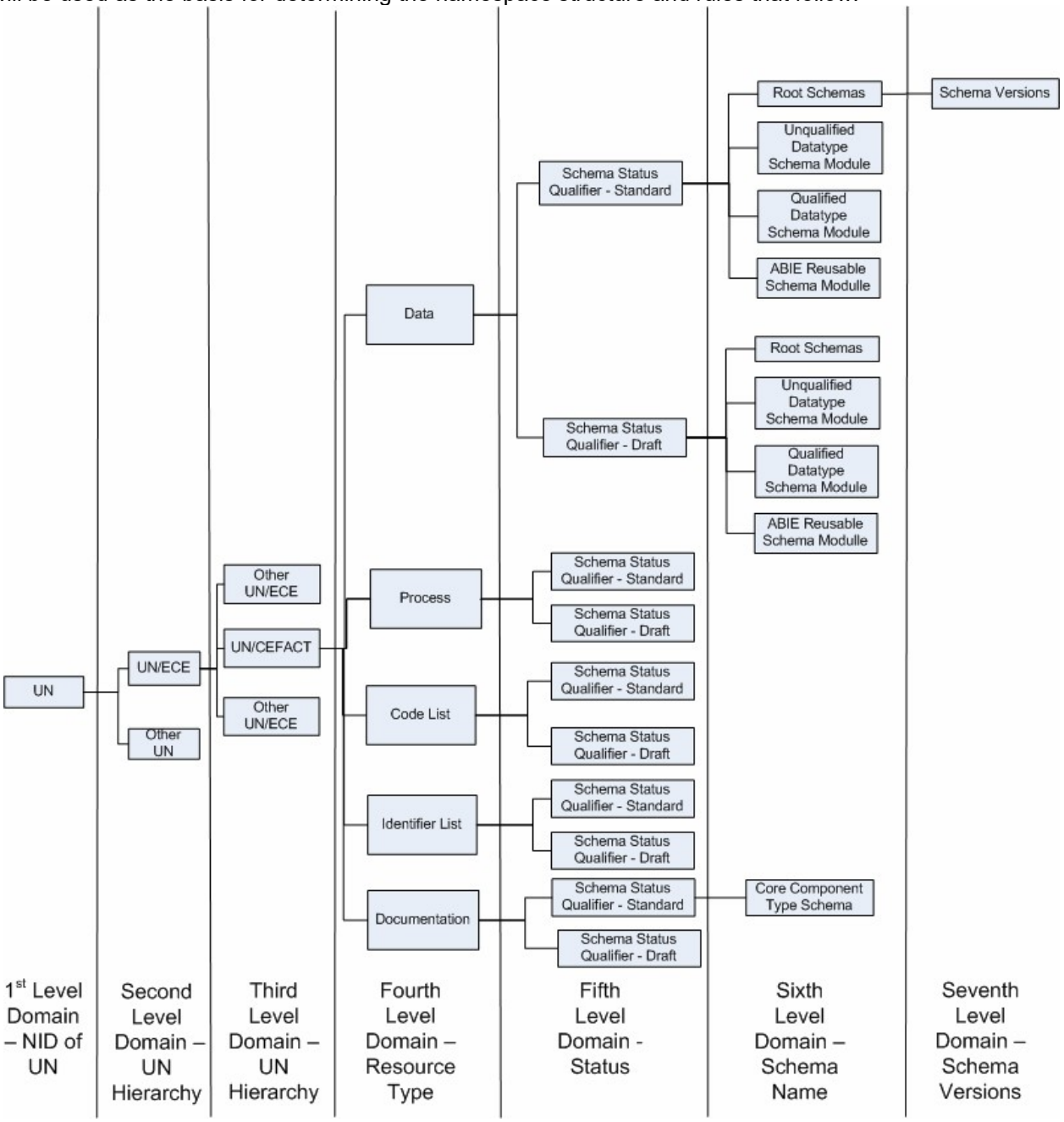


Figure 5-7: UN/CEFACT Namespace Scheme

5.6.2 Declaring Namespace

Best practice dictates that every schema module have its own namespace with the exception that internal schema modules will be in the same namespace as the root schema.

[R38]	Every UN/CEFACT defined or imported schema module MUST have a namespace declared, using the **`xsd:targetNamespace`** attribute.

5.6.3 Namespace Persistence

Namespaces also provide a means for achieving consistency and harmonization between schema versions. UN/CEFACT has chosen to align namespace versioning with schema versioning and modularity. The UN/CEFACT modularity approach provides for grouping of reusable schemas by a root schema. Many of these schema are intended to be reused across multiple schema. Others are unique to a particular root schema. The root schema and those schema modules that are unique to it are considered a schema set. The contents of a schema set are so interrelated that proper management dictates that both versioning and namespace of all members of the set be synchronized. Schema sets are therefore assigned to a single, versioned namespace. Other schema modules are also best managed by being assigned to their own unique versioned namespaces. Accordingly, with the exception of internal schema modules, each UN/CEFACT XSD schema module will have its own namespace and each namespace will be versioned.

[R39]	Every version of a defined or imported schema module other than internal schema modules MUST have its own unique namespace.

Once a namespace declaration is published, any change would result in an inability to validate instance documents citing the namespace. Accordingly, a change in the construct or contents of the namespace should not be allowed.

[R40]	UN/CEFACT published namespace declarations MUST NOT be changed, and its contents MUST NOT be changed unless such change does not break backward compatibility.

5.6.4 Namespace Uniform Resource Identifiers

Namespaces must be persistent. Namespaces should be resolvable. Uniform Resource Indicators (URIs) are used for identifying a namespace. Within the URI space, options include Uniform Resource Locators (URLs) and Uniform Resource Names (URNs). URNs have an advantage in that they are persistent. URLs have an advantage in that they are resolvable. After careful consideration, UN/CEFACT has determined that URNs are most appropriate as persistence is of a higher priority, and efforts are underway to make URNs resolvable.

[R41]	UN/CEFACT namespaces MUST be defined as Uniform Resource Names.

To ensure consistency, each UN/CEFACT namespace will have the same general structure. This namespace structure will follow the provisions of Internet Engineering Task Force (IETF) Request For Comments (RFC) 2141 – URN Syntax. That specification calls for a standardized URN syntax structure as follows: (phrases enclosed in quotes are REQUIRED):

<URN> ::= "urn:" <NID> ":" <NSS>

where :

<NID> = the Namespace Identifier
<NSS> = the Namespace Specific String.
The leading "urn:" sequence is case-insensitive.

The Namespace identifier determines the syntactic interpretation of the Namespace Specific String. Following this pattern, the UN/CEFACT namespace general structure for a namespace name should be: **`urn:un:unece:uncefact:<schematype>:<status>:<name>:<version>`**

Where:

- Namespace Identifier (NID) = un
- Namespace Specific String = **`unece:uncefact:<schematype>:<status>:<name>:<version>`** with unece and uncefact as fixed value second and third level domains within the NID of un
- schematype = a token identifying the type of schema module: **`data|process|codelist|identifierlist|documentation`**
- status = the status of the schema as: **`draft|standard`**
- name = the name of the schema module (using upper camel case) with periods, spaces, or other separators and the words 'schema module' removed.
- version = The major version number. Sequentially assigned, first release starting with the number 1.

[R42] The names for namespaces MUST have the following structure while the schema is at draft status:
urn:un:unece:uncefact:<schematype>:<status>:<name>:<major>

Where:
schematype = a token identifying the type of schema module:
data|process|codelist|identifierlist|documentation
status = a token identifying the standards status of the schema module:
draft|standard
name = the name of the schema module (using upper camel case) with periods, spaces, or other separators and the words 'schema module' removed.
major = the major version number. Sequentially assigned, first release starting with the number 1.

[R43] This rule was combined with [R42].

Example 5-14: Namespace Name at Draft Status

```
"urn:un:unece:uncefact:data:draft:UnqualifiedDataType:1"
```

Example 5-15: Namespace Name at Specification Status

```
"urn:un:unece:uncefact:data:standard:UnqualifiedDataType:1"
```

5.6.5 Namespace Constraint

To ensure consistency in declaring namespaces, a namespace should only be declared for an XML construct by the owner of that namespace – unless specifically designed as a generic namespace such as xsi. Accordingly, UN/CEFACT namespaces will only contain XML constructs created and assigned by UN/CEFACT.

[R44] UN/CEFACT namespace values will only be assigned to UN/CEFACT developed objects.

5.6.6 UN/CEFACT XSD Namespace Schema Tokens

Namespace URIs are typically represented by tokens rather than citing the entire URI as the qualifier in qualified XML constructs. UN/CEFACT has developed a token pattern for each type of UN/CEFACT schema module. These token patterns are identified in the applicable schema module subsection in Section 7.

5.7 Schema Location

Schema locations are required to be in the form of a URI scheme. Schema locations are typically based on their namespaces. Schema locations are typically defined as URL based URI schemes because of resolvability limitations of URN based URI scheme. However, UN/CEFACT XSD Schema use a URN based URI scheme for namespace declarations because persistence is considered more important than resolvability. In recognition of the need for resolvability of schema location, until such time as URNs become fully resolvable, UN/CEFACT will store schema in locations identified using a URL based URI scheme.

[R45] The general structure for schema location MUST be:
../<schematype>/<status>/<name>_<major>.<minor>[p <revision>].xsd

Where:
schematype = a token identifying the type of schema module:
data|process|codelist|identifierlist|documentation
status = the status of the schema as: draft|standard
name = the name of the schema module (using upper camel case) with periods, spaces, or other separators and the words 'schema module' removed.
major = the major version number, sequentially assigned, first release starting with the number 1.
minor = the minor version number within a major release, sequentially assigned, first release starting with the number 0.

revision = sequentially assigned alphanumeric character for each revision of a minor release. Only applicable where status = draft.

[R46] Each **xsd:schemaLocation** attribute declaration MUST contain a resolvable URL, and in the case of an absolute path, a persistent URL.

[R47] This rule has been removed.

5.8 Versioning

The versioning scheme for UN/CEFACT XSD schema modules is composed of a major version number and where appropriate, a minor version number. Major version numbers are reflected in the namespace declaration while minor version numbers are only reflected in the schema location. Major and minor version numbers are also declared in the version attribute in the **xsd:schema** element.

[R48] The **xsd:schema** version attribute MUST always be declared.

[R49] The **xsd:schema** version attribute MUST use the following template:
<xsd:schema ... version="<major>.<minor>">

[R50] Every schema version namespace declaration MUST have the URI of:
urn:un:unece:uncefact:<schematype>:<status>:<name>:<major>

5.8.1 Major Versions

A major version of a UN/CEFACT XSD schema module constitutes significant and/or non-backwards compatible changes. If any XML instance based on such older major version UN/CEFACT XSD Schema attempts validation against the newer version, it may experience validation errors. A new major version will be produced when significant and/or non-backward compatible changes occur, i.e.

- Removing or changing values in enumerations
- Changing of element names, type names and attribute names
- Changing the structures so as to break polymorphic processing capabilities
- Deleting or adding mandatory elements or attributes
- Changing cardinality from mandatory to optional

Major version numbers are reflected in the namespace declaration as follows:

urn:un:unece:uncefact:<schematype>:<status>:<name>:<major> Where:

- major = the first version starts with the number 1.

Major version numbers should be based on logical progressions to ensure semantic understanding of the approach and guarantee consistency in representation. Non-negative, sequentially assigned incremental integers satisfy this requirement.

[R51] Every UN/CEFACT XSD Schema and schema module major version number MUST be a sequentially assigned incremental integer greater than zero.

5.8.2 Minor Versions

Within a major version of an UN/CEFACT XSD schema module there can be a series of minor, or backward compatible, changes. The minor versioning of an UN/CEFACT XSD schema module determines its compatibility with UN/CEFACT XSD schema modules with preceding and subsequent minor versions within the same major version. The minor versioning scheme thus helps to establish backward and forward compatibility. Minor versions will only be increased when compatible changes occur, i.e

- Adding values to enumerations
- Optional extensions
- Add optional elements

[R52] Minor versioning MUST be limited to declaring new optional XSD constructs, extending existing XSD constructs, or refinements of an optional nature.

967 Minor versions are reflected in the schema location as identified in section 5.7, but are not reflected
968 in the namespace declaration. Minor versions will be declared using the **`xsd:version`** attribute in
969 the **`xsd:schema`** element. It is only necessary to declare the minor version in the internal schema
970 version attribute since instance documents with different minor versions are compatible with the
971 major version held in the same namespace. By using the version attribute in each document
972 instance, the application can provide the appropriate logic switch for different compatible versions
973 without having knowledge of the schema version at which the document instance was delivered.

974 Just like major version numbers, minor version numbers should be based on logical progressions to
975 ensure semantic understanding of the approach and guarantee consistency in representation. Non-
976 negative, sequentially assigned incremental integers satisfy this requirement.

977 Minor version changes are not allowed to break compatibility with previous minor versions.
978 Compatibility includes consistency in naming of the schema constructs to include elements,
979 attributes, and types. UN/CEFACT minor version changes will not include renaming the schema
980 construct.

981 [R53] For UN/CEFACT minor version changes, the name of the schema construct MUST NOT
982 change.

983 Semantic compatibility across minor versions is essential.

984 [R54] Changes in minor versions MUST NOT break semantic compatibility with prior versions
985 having the same major version number.

986 For a particular namespace, the parent major version and subsequent minor versions of a major
987 version establish a linearly linked relationship. Since each major version is assigned its own
988 namespace, for conformance purposes, the first minor version must incorporate all XML constructs
989 present in the parent major version, and each new minor version needs to incorporate all XML
990 constructs present in the immediately preceding minor version.

991 [R55] UN/CEFACT minor version schema MUST incorporate all XML constructs from the
992 immediately preceding major or minor version schema.

6 General XML Schema Language Conventions

6.1 Schema Construct

[R56] The **xsd:elementFormDefault** attribute MUST be declared and its value set to **qualified**.

[R57] The **xsd:attributeFormDefault** attribute MUST be declared and its value set to **unqualified**.

[R58] The **xsd** prefix MUST be used in all cases when referring to http://www.w3.org/2001/XMLSchema as follows: **xmlns:xsd=http://www.w3.org/2001/XMLSchema**.

Example 6-1: Element and Attribute Form Default

```
<xsd:schema targetNamespace=" ... see namespace ...
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified"
 attributeFormDefault="unqualified">
```

6.1.1 Constraints on Schema Construction

[R59] **xsd:appInfo** MUST NOT be used.

[R60] **xsd:notation** MUST NOT be used.

[R61] **xsd:wildcard** MUST NOT be used.

[R62] The **xsd:any** element MUST NOT be used.

[R63] The **xsd:any** attribute MUST NOT be used.

[R64] Mixed content MUST NOT be used (excluding documentation).

[R65] **xsd:substitutionGroup** MUST NOT be used.

[R66] **xsd:ID/xsd:IDREF** MUST NOT be used.

[R67] **xsd:key/xsd:keyref** MUST be used for information association.

[R68] The absence of a construct or data MUST NOT carry meaning.

6.2 Attribute and Element Declarations

6.2.1 Attributes

6.2.1.1 Usage of Attributes

User declared attributes are only used to convey the supplementary components of core component types. However, predefined **xsd:attributes** will be used as described elsewhere in this document.

[R69] User declared attributes MUST only be used to convey core component type (CCT) supplementary component information.

The user declared attributes can represent different types of values. Some of the values can be variable information or can be based on code lists or identifier schemes.

[R70] A **xsd:attribute** that represents a supplementary component with variable information MUST be based on the appropriate XSD built-in data type.

[R71] A **xsd:attribute** that represents a supplementary component which represents codes MUST be based on the **xsd:simpleType** of the appropriate code list.

[R72] A **xsd:attribute** that represents a supplementary component which represents identifiers MUST be based on the **xsd:simpleType** of the appropriate identifier scheme.

6.2.1.2 Constraints on Attribute Declarations

In general, the absence of an element in an XML schema does not have any particular meaning - it may indicate that the information is unknown, or not applicable, or the element may be absent for some other reason. The XML schema specification does however provide a feature, the **xsd:nillable** attribute, whereby an element may be transferred with no content, but still use its attributes and thus carry semantic meaning. In order to respect the principles of the CCTS and to retain semantic clarity the nillability feature of XSD will not be used.

[R73] The **xsd:nillable** attribute MUST NOT be used.

6.2.2 Elements

6.2.2.1 Usage of Elements

Elements are declared for the document level business information payload, ABIEs, BBIEs, and ASBIEs.

6.2.2.2 Element Declaration

[R74] Empty elements MUST NOT be used.

[R75] Every BBIE leaf element declaration MUST be of the **udt:UnqualifiedDataType** or **qdt:QualifiedDataType** that represents the source basic business information entity (BBIE) data type.

Example 6-2: Element Declaration

```
<xsd:complexType name="AcknowledgementType">
   <xsd:annotation>
     … see annotation …
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="AcknowledgementDocument"
         type="ram:AckowledgementDocumentType”  minOccurs="0">
         <xsd:annot
           ation>
               … see annotation …
         </xsd:annotation>
      </xsd:element>
            <xsd:element name="ProjectParty" type="ram:ProjectPartyType"
         <xsd:annota
            tion>
           … see annotation …
         </xsd:annot
         ation>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

6.2.2.3 Constraints on Element Declarations

[R76] The **xsd:all** element MUST NOT be used.

6.3 Type Declarations

6.3.1 Usage of Types

[R77] All type definitions MUST be named.

Example 6-3: Type Definition Name

```
<xsd:complexType name="IDType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:sequence>
      ... see element declaration ...
   </xsd:sequence>
</xsd:complexType>
```

Data types are intended to be reused to the maximum extent possible. If an existing data type has the same semantic meaning and structure (facet restrictions) as the intended data type, then the existing data type should be used rather than creating a semantically equivalent duplicate data type.

[R78]	Data type definitions with the same semantic meaning MUST NOT have an identical set of facet restrictions.

6.3.2 Simple Type Definitions

`xsd:simpleTypes` must always be used where they satisfy the user's business requirements. Where these business requirements cannot be satisfied, user defined complex type definitions will be used.

Example 6-4: Simple Types in Unqualified Data Type Schema Module

```
<xsd:simpleType name="TextType">
   <xsd:annotation>
            … see
         annotation
            …
   </xsd:annotation>
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
```

Example 6-5: Simple Types in Code Lists Module

```
<xsd:simpleType name="CurrencyCodeContentType">
   <xsd:restriction base="xsd:token">
      <xsd:enumeration value="ADP">
         …see enumeration of code lists …
      </xsd:enumeration>
      <xsd:annotation>
         … see annotation …
      </xsd:annotation>
   </xsd:restriction>
</xsd:simpleType>
```

6.3.3 Complex Type Definitions

User defined complex types may be used when XSD built-in data types do not satisfy the business requirements or when an aggregate business information entity (ABIE) must be defined.

Example 6-6: Complex Type of Object Class "ProjectContactType"

```
<xsd:complexType name="ProjectContactType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:sequence>
      ... see element declaration ...
   </xsd:sequence>
</xsd:complexType>
```

6.4 User of XSD Extension and Restriction

The general philosophy is that all UN/CEFACT XSD schema constructs will follow the model defined in Figure 5.1. These schema constructs are based on the concept that the underlying semantic structures of the core components and business information entities are normative forms of standards that developers are not allowed to alter without coordination of appropriate UN/CEFACT Domains. Accordingly, as business requirements dictate, new schema constructs will be created and new types defined and elements declared as appropriate. The concept of derivation through the

use of **xsd:extension** and **xsd:restriction** will only be used in limited circumstances as described below.

6.4.1 Extension

[R79] **xsd:extension** MUST only be used in the **cct:CoreComponentType** schema module and the **udt:UnqualifiedDataType** schema module. When used it MUST only be used for declaring **xsd:attributes** to accommodate relevant supplementary components.

6.4.2 Restriction

The CCTS specification employs the concept of semantic restriction in creating specific instantiations of core components. Accordingly, **xsd:restriction** will be used as appropriate to define types that are derived from the existing types. Where used, the derived types must always be renamed. Simple and complex type restrictions may be used. **xsd:restriction** can be used for facet restriction and/or attribute restriction.

[R80] When **xsd:restriction** is applied to a **xsd:simpleType** or **xsd:complexType** that represents a data type the derived construct MUST use a different name.

Example 6-7: Restriction of Simple Type

```
<xsd:simpleType name="TaxAmountType">
   <xsd:annotation>
            ... see
         annotation ...
   </xsd:annotation>
   <xsd:restriction base="udt:AmountType">
      <xsd:totalDigits value="10"/>
      <xsd:fractionDigits value="3"/>
   </xsd:restriction>
</xsd:simpleType>
```

6.5 Annotation

All UN/CEFACT XSD schema constructs will use **xsd:annotation** to provide the documentation specified in Section 7 of CCTS.

[R81] Each UN/CEFACT defined or declared construct MUST use the **xsd:annotation** element for required CCTS documentation.

> [Note]
>
> In order to conform to this specification, this rule also applies to any construct imported from other standards bodies.

6.5.1 Documentation

The annotation documentation will be used to convey all metadata as specified in the CCTS, i.e., to convey the semantic content carried in the XML construct. Therefore, all elements specified for the documentation are defined in the Core Component Technical Specification namespace. The current version of this namespace is:
urn:un:unece:uncefact:documentation:standard:CoreComponentsTechnicalSpecification:2.

Thus, all schema modules must contain the following namespace declaration:
ccts="urn:un:unece:uncefact:documentation:standard:CoreComponentsTechnicalSpe cification:2."

and all documentation elements must be prefixed with 'ccts'.

The following annotations are required as defined in section 7 in type definitions and element declarations (the representation of each item in XML code is shown in parenthesis):

- **Unique Identifier**: The unique identifier assigned to the artefact in the library. (UniqueID)

- **Acronym:** The abbreviation of the type of component. (Acronym)
  - **BBIE –** Basic Business Information Entity
  - **ABIE –** Aggregate Business Information Entity
  - **ASBIE –** Associated Business Information Entity
  - **CCT –** Core Component Type
  - **QDT –** Qualified Data Type
  - **UDT –** Unqualified Data Type
- **Dictionary Entry Name**: The complete name (not the tag name) of the artefact in the library. (DictionaryEntryName**)**
- **Name:** The name of the supplementary component or business information payload. (Name)
- **Version**: The version of the artefact as assigned by the registry. (Version)
- **Definition**: The semantic meaning of the artefact. (Definition)
- **Cardinality**: An indication of whether the property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the object. (Cardinality)
- **Object Class Term**: The Object Class represented by the artefact. (ObjectClassTerm)
- **Object Class Qualifier Term:** A term(s) that qualifies the Object Class. (ObjectClassQualifierTerm)
- **Property Term**: The Property Term represented by the artefact. (PropertyTerm)
- **Property Qualifier Term:** A term(s) that qualifies the Property Term. (PropertyQualifierTerm)
- **Associated Object Class Term**: The Associated Object Class Term represented by the artefact. (AssociatedObjectClassTerm)
- **Associated Object Class Qualifier Term:** A term(s) that qualifies the Associated Object ClassTerm. (AssociatedObjectClassQualifierTerm)
- **Association Type**: The association type of the Association Business Information Entity. (AssociationType)
- **Primary Representation Term**: The Primary Representation Term represented by the artefact. (PrimaryRepresentationTerm)
- **Data Type Qualifier Term:** A term(s) that qualifies the Data Type Term. (DataTypeQualifierTerm)
- **Primitive Type**: The primitive data type as assigned to the artefact by CCTS. (PrimitiveType)
- **Business Process Context Value**: A valid value describing the Business Process contexts for which this construct has been designed. Default is 'In All Contexts'. (BusinessProcessContextValue)
- **Geopolitical/Region Context Value**: A valid value describing the Geopolitical/Region contexts for which this construct has been designed. Default is 'In All Contexts'. (GeopoliticalOrRegionContextValue)
- **Official Constraints Context Value**: A valid value describing the Official Constraints contexts for which this construct has been designed. Default is 'None'. (OfficialConstraintContextValue)
- **Product Context Value**: A valid value describing the Product contexts for which this construct has been designed. Default is 'In All Contexts'. (ProductContextValue)
- **Industry Context Value**: A valid value describing the Industry contexts for which this construct has been designed. Default is 'In All Contexts'. (IndustryContextValue)
- **Business Process Role Context Value**: A valid value describing the Role contexts for which this construct has been designed. Default is 'In All Contexts'. (BusinessProcessRoleContextValue)
- **Supporting Role Context Value**: A valid value describing the Supporting Role contexts for which this construct has been designed. Default is 'In All Contexts'. (SupportingRoleContextValue)
- **System Capabilities Context Value**: A valid value describing the Systems Capabilities contexts for which this construct has been designed. Default is 'In All Contexts'. (SystemCapabilitiesContextValue)

- **Usage Rule:** A constraint that describes specific conditions which are applicable to the artefact. (UsageRule)
- **Business Term:** A synonym term under which the artefact is commonly known and used in business. (BusinessTerm)
- **Example:** A possible value for the artefact. (Example)

Appendix F specifies normative information on the specific annotation required for each of the artefacts.

> Note: The list above defines the minimum annotation documentation requirements. However, additional annotation documentation may be included when necessary.

Example 6-8: Example of annotation

```
<xsd:annotation>
  <xsd:documentation xml:lang="en">
    <ccts:UniqueID>UN01005559</ccts:UniqueID>
    <ccts:Acronym>BBIE</ccts:Acronym>
    <ccts:DictionaryEntryName>CI_ Note. Content. Code</ccts:DictionaryEntryName>
    <ccts:Version>1.0</ccts:Version>
    <ccts:Definition>The code specifying the content of this CI
    note.</ccts:Definition>
    <ccts:Cardinality>0..1</ccts:Cardinality>
    <ccts:ObjectClassTerm>Note</ccts:ObjectClassTerm>
    <ccts:ObjectClassQualifierTerm>CI</ccts:ObjectClassQualifierTerm>
    <ccts:PropertyTerm>Content</ccts:PropertyTerm>
    <ccts:PrimaryRepresentationTerm>Code</ccts:PrimaryRepresentationTerm>
    <ccts:BusinessProcessContextValue>Cross Industry
          Trade</ccts:BusinessProcessContextValue>
    <ccts:GeopoliticalOrRegionContextValue>In All
       Contexts</ccts:GeopoliticalOrRegionContextValue>
    <ccts:OfficialConstraintContextValue>None</ccts:OfficialConstraintContextValue>
    <ccts:ProductContextValue>In All Contexts</ccts:ProductContextValue>
    <ccts:IndustryContextValue>In All Contexts</ccts:IndustryContextValue>
    <ccts:BusinessProcessRoleContextValue>In All
       Contexts</ccts:BusinessProcessRoleContextValue>
    <ccts:SupportingRoleContextValue>In All
       Contexts</ccts:SupportingRoleContextValue>
    <ccts:SystemCapabilitiesContextValue>In All
       Contexts</ccts:SystemCapabilitiesContextValue>
  </xsd:documentation>
</xsd:annotation>
```

Each UN/CEFACT construct containing a code should include documentation that will identify the code list(s) that must be minimally supported when the construct is used.

The following table provides a summary view of the annotation data as defined in section 6.

	rem:RootSchema	ABIE: xsd:complexType and xsd:element	BBIE: xsd:element	ASBIE xsd:element	cct:CoreComponentType	Supplementary component	udt:UnqualifiedDataType	qdt:QualfiedDataType
Unique Identifier	M	M	M	M	M	O	M	M
Acronym	M	M	M	M	M	M	M	M
Dictionary Entry Name		M	M	M	M	M	M	M
Name	M							
Version	M	M	M	M	M		M	M
Definition	M	M	M	M	M	M	M	M
Cardinality			M	M		M		
Object Class Term		M	M	M		M		
Object Class Qualifier Term		O	O	O				
Property Term			M	M		M		
Property Qualifier Term			O	O				
Associated Object Class Term				M				
Associated Object Class Qualifier Term				O				
Association Type				M				
Primary Representation Term			M		M	M	M	M
Data Type Qualifier Term								M
Primitive Type					M	M	M	M
Business Process Context Value	M, R	O, R	O, R	O, R				O, R
Geopolitical/Region Context Value	O, R	O, R	O, R	O, R				O, R
Official Constraints Context Value	O, R	O, R	O, R	O, R				O, R

	rem:RootSchema	ABIE: xsd:complexType and xsd:element	BBIE: xsd:element	ASBIE xsd:element	cct:CoreComponentType	Supplementary component	udt:UnqualifiedDataType	qdt:QualifiedDataType
Product Context Value	O, R	O, R	O, R	O, R				O, R
Industry Context Value	O, R	O, R	O, R	O, R				O, R
Business Process Role Context Value	O, R	O, R	O, R	O, R				O, R
Supporting Role Context Value	O, R	O, R	O, R	O, R				O, R
System Capabilities Context Value	O, R	O, R	O, R	O, R				O, R
Usage Rule		O, R	O, R	O, R	O, R	O, R	O, R	O, R
Business Term		O, R	O, R	O, R				
Example		O, R	O, R	O, R				O, R

Key:
M - mandatory
O - optional
R - repeating

> Note
>
> When a particular optional annotation element contains no value, it may be omitted from the schema.

7 XML Schema Modules

This section describes the requirements of the various XML schema modules that will be incorporated within the UN/CEFACT library.

7.1 Root Schema

The root schema serves as the container for all other schema content that is required to fulfil a business information exchange. The root schema resides in its own namespace and imports external schema modules as needed. It may also include internal schema modules that reside in its namespace.

7.1.1 Schema Construct

Each root schema will be constructed in a standardized format in order to ensure consistency and ease of use. The specific format is shown in the example below and must adhere to the format of the relevant sections as detailed in Appendix B.

Example 7-1: Structure of RootSchema Module

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ================================================================== -->
<!-- =====  [MODULENAME] Schema Module                            ===== -->
<!-- ================================================================== -->
<!--
   Schema agency:  UN/CEFACT
   Schema version: 2.0
   Schema date:     [SCHEMADATE]



... see intellectual property disclaimer ...
-->
<xsd:schema
 targetNamespace="urn:un:unece:uncefact:data:draft:[MODULENAME]:1"
 ... see namespaces ...
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
 <!-- ================================================================== -->
 <!-- =====  Imports                                               ===== -->
 <!-- ================================================================== -->
 <!-- =====  Import of [MODULENAME]                                ===== -->
 <!-- ================================================================== -->
 <   see imports
 <!-- ================================================================== -->
 <!-- =====  Include                                               ===== -->
 <!-- ================================================================== -->
 <!-- ===== Include of [MODULENAME]                                ===== -->
 <!-- ================================================================== -->
 ... see includes ...
 <!-- ================================================================== -->
 <!-- =====  Element Declarations                                  ===== -->
 <!-- ================================================================== -->
 <!-- =====  Root Element Declarations                             ===== -->
 <!-- ================================================================== -->
    See element declarations…
 <!-- ================================================================== -->
 <!-- =====  Type Definitions                                      ===== -->
 <!-- ================================================================== -->
 <!-- =====  Type Definitions: [TYPE]                              ===== -->
 <!-- ================================================================== -->
 <xsd:complexType name="[TYPENAME]">
    <xsd:restriction base="xsd:token">
        ... see type definition ....
    </xsd:restriction>
 </xsd:complexType>
</xsd:schema>
```

7.1.2 Namespace Scheme

All root schemas published by UN/CEFACT will be assigned a unique token by BPS to represent the namespace prefix. This token will be prefixed by 'rsm'.

[R82] The root schema module MUST be represented by a unique token.

Example 7-2: Namespace of Root Schema Module

```
"xmlns:rsm="urn:un:unece:uncefact:data:draft:CrossIndustryInvoice:1"
```

[Note]

Throughout this specification, the token 'rsm' is used for the unique root schema token.

7.1.3 Imports and Includes

[R83] The **`rsm:RootSchema`** MUST import the following schema modules:
- **`ram:ReusableABIE`** Schema Module
- **`udt:UnqualifiedDataType`** Schema Module
- **`qdt:QualifiedDataType`** Schema Module

The root schema will include all internal schema modules that reside in its namespace. The root schema may import other external schema modules as necessary provided they conform to UN/CEFACT naming and design rules. One root schema (root schema A) may also make use of ABIEs defined as part of another root schema (root schema B) or that root schema's internal schema module. In other words, reuse type definitions and element declarations defined in another namespace. An example may be that the root schema for a Purchase Order Response message (root schema A) makes use of ABIEs defined as part of the schema definition for a Purchase Order Request message (root schema B). If that is the case then such type definitions and element declarations should be imported in to the root schema (root schema A). To achieve this only the root schema (root schema B) in the namespace containing the type definitions and element declarations needed should be imported as this in itself included the subordinate internal schema modules.

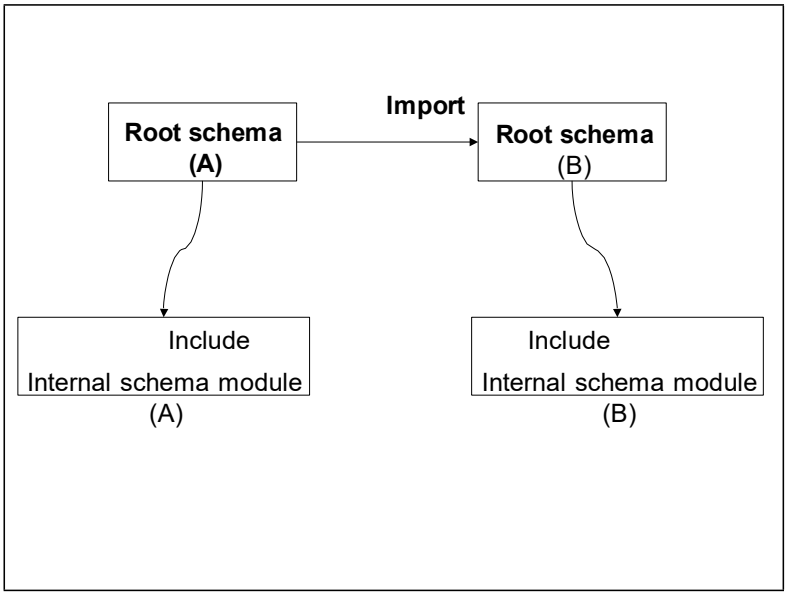


Figure 7-1: Imports and Includes of Schema Modules in Root Schema

[R84] A **rsm:RootSchema** in one UN/CEFACT namespace that is dependent upon type definitions or element declaration defined in another namespace MUST import the **rsm:RootSchema** from that namespace.

[R85] A **rsm:RootSchema** in one UN/CEFACT namespace that is dependent upon type definitions or element declarations defined in another namespace MUST NOT import Schema Modules from that namespace other than the **rsm:RootSchema.**

[R86] The **rsm:RootSchema** MUST include any internal schema modules that reside in the root schema namespace.

7.1.4 Root Element Declaration

Each UN/CEFACT business information payload message has a single root element that is globally declared in the root schema. The global element is named according to the business information payload that it represents and references the target information payload that contains the actual business information.[6]

[R87] A single global element known as the root element, representing the business information payload, MUST be declared in a **rsm:RootSchema**.

[R88] The name of the root element MUST be the name of the business information payload with separators and spaces removed.

[R89] The root element declaration must be of **xsd:complexType** that represents the business information payload.

Example 7-3: Name of Root Element

```
<!-- ================================================================== -->
<!-- ===== Root Element                                          ===== -->
<!-- ================================================================== -->
   <xsd:element name="CrossIndustryInvoice" type="rsm:CrossIndustryInvoiceType">
   <xsd:annotation>
        ... see annotation ...
   </xsd:annotation>
</xsd:element>
```

7.1.5 Type Definitions

Root schemas are limited to defining a single **xsd:complexType** and a declaring a single global element that fully describe the business information payload.

[R90] Root schema MUST define a single **xsd:complexType** that fully describes the business information payload.

[R91] The name of the root schema**xsd:complexType** MUST be the name of the root element with the word 'Type' appended.

Example 7-4: Name of Complex Type Definition

```
<!-- ================================================================== -->
<!-- ===== Root Element ===== -->
<!-- ============================ -->
<xsd:element name="PurchaseOrderRequest" type="rsm:PurchaseOrderRequestType">
    <xsd:annotation>
        ... see annotation ...
    </xsd:annotation>
</xsd:element>
<xsd:complexType name="PurchaseOrderRequestType">
    <xsd:sequence>
        ...
    </xsd:sequence>
```

[6] All references to root element represent the globally declared element in a UN/CEFACT schema module that is designated as the root element for instances that use that schema.

```
</xsd:complexType>
```

7.1.6 Annotations

[R92] The **rsm:RootSchema** root element declaration MAY have a structured set of annotations present in the following pattern:

- UniqueID (required): The identifier that references the business information payload instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be RSM.
- Name (required): The name of the business information payload.
- Version (required): An indication of the evolution over time of a business information payload.
- Definition (required): A brief description of the business information payload.
- BusinessProcessContextValue (required, repetitive): The business process with which this business information is associated.
- GeopoliticalorRegionContextValue (optional, repetitive): The geopolitical/region contexts for this business information payload.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this business information payload.
- ProductContextValue (optional, repetitive): The product context for this business information payload.
- IndustryContextValue (optional, repetitive): The industry context for this business information payload.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this business information payload.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this business information payload.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this business information payload.

7.2 Internal Schema

A UN/CEFACT internal schema module will contain schema constructs representing ABIEs that are specific to a given root schema, such as restricted ABIEs created through CCBDA. Internal schema modules reside in the same namespace as their root schema. These constructs are subject to the same rules as those for reusable ABIEs as provided in sections 7.3.4, 7.3.5, and 7.3.6.

7.2.1 Schema Construct

Each internal schema will be constructed in a standardized format in order to ensure consistency and ease of use. Each internal schema format must adhere to the format of the relevant sections as detailed in Appendix B.

7.2.2 Namespace Scheme

[R93] All UN/CEFACT internal schema modules MUST be in the same namespace as their corresponding **rsm:RootSchema**.

The UN/CEFACT internal schema modules do not declare a target namespace, but instead reside in the namespace of their parent root schema. All internal schema modules are accessed from the root schema using **xsd:include**.

[R94] The internal schema module MUST be represented by the same token as its **rsm:RootSchema**.

7.2.3 Imports and Includes

The internal schema module may import or include other schema module as necessary to support validation.

7.3 Reusable Aggregate Business Information Entity Schema

The UN/CEFACT ABIE schema module is a schema instance that contains all of the reusable ABIEs. This schema module may thus be used (imported into) in conjunction with any of the UN/CEFACT root schema.

7.3.1 Schema Construct

The reusable ABIE schema will be constructed in a standardized format in order to ensure consistency and ease of use. The specific format is shown below and must adhere to the format of the relevant sections as detailed in Appendix B.

Example 7-5: Structure of Reusable ABIEs Schema Module

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ========================================================================= -->
<!-- =====  Reusable ABIEs Schema Module                                  ===== -->
<!-- ========================================================================= -->
<!--
    Schema agency:    UN/CEFACT
    Schema version:   2.0
    Schema date:      [SCHEMADATE]

         ... see intellectual property disclaimer ...

-->
<xsd:schema targetNamespace=
 ... see namespace declaration ... xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <!-- ========================================================================= -->
 <!-- ===== Imports                                                       ===== -->
 <!-- ========================================================================= -->
 ... see imports ...
 <!-- ========================================================================= -->
 <!-- ===== Type Definitions                                              ===== -->
 <!-- ========================================================================= -->
 ... see type definitions ...
</xsd:schema>
```

7.3.2 Namespace Scheme

[R95] The Reusable Aggregate Business Information Entity schema module MUST be represented by the token **ram**.

Example 7-6: Namespace of Reusable Aggregate Business Information Entity Schema Module

```
"urn:un:unece:uncefact:data:draft:ReusableAggregateBusinessInformationEntity:1"
```

Example 7-7: Schema-Element of Reusable ABIEs Schema Module

```
<xsd:schema targetNamespace=
 "urn:un:unece:uncefact:data:draft:ReusableAggregateBusinessInformationEntity:1"
 xmlns:ram=
 "urn:un:unece:uncefact:data:draft:ReusableAggregateBusinessInformation:1"
```

7.3.3 Imports and Includes

[R96] The **ram:ReusableAggregateBusinessInformationEntity** schema MUST import the following schema modules:
– **udt:UnqualifiedDataType** Schema Module
– **qdt:QualifiedDataType** Schema Module

Example 7-8: Import of required modules

```
 <!-- ========================================================================= -->
 <!-- ===== Imports                                                       ===== -->
 <!-- ========================================================================= -->
 <!-- ===== Import of Qualified Data Type Schema Module                   ===== -->
 <!-- ========================================================================= -->
 <xsd:import namespace=
    "urn:un:unece:uncefact:data:draft:QualifiedDataType:1"
 schemaLocation="http://www.unece.org/uncefact/data/draft/QualifiedDataType_1.xsd"/
```

```
    >
    <!-- ================================================================== -->
    <!-- ===== Import of Unqualified Data Type Schema Module          ===== -->
    <!-- ================================================================== -->
     <xsd:import
        namespace="urn:un:unece:uncefact:data:draft:UnqualifiedDataType:1"
     schemaLocation="http://www.unece.org/uncefact/data/draft/UnqualifiedDataTypes_1.xsd"/>
```

7.3.4 Type Declarations

[R97] For every object class (ABIE) identified in the UN/CEFACT syntax-neutral model, a named **xsd:complexType** MUST be defined.

[R98] The name of the ABIE **xsd:complexType** MUST be the **ccts:DictionaryEntryName** with the spaces and separators removed, approved abbreviations and acronyms applied, and with the 'Details' suffix replaced with 'Type'.

For every complex type definition based on an ABIE object class, its XSD content model will be defined such that it reflects each property of the object class as an element declaration, with its cardinality and sequencing within the schema XSD content model determined by the details of the source aggregate business information entity (ABIE).

[R99] Every aggregate business information entity (ABIE) **xsd:complexType** definition content model MUST use the **xsd:sequence** and/or **xsd:choice** elements to reflect each property (BBIE or ASBIE) of its class.

[R100] Recursion of **xsd:sequence** and/or **xsd:choice** MUST NOT occur.

No complex type may contain a sequence followed by another sequence or a choice followed by another choice. However, it is permissible to alternate sequence and choice as in example 7.9. Note that the choice construction will not be used in the base reusable ABIE UN/CEFACT schemas, as it cannot be directly modeled in CCTS. However, third party schemas that implement those restrictions would still be conformant.

Example 7-9: Sequence within an object class

```
    <xsd:complexType name="AcknowledgementDocumentType" >
       <xsd:annotation>
          ...see annotation...
       </xsd:annotation>
       <xsd:sequence>
          <xsd:element name="MultipleReferencesIndicator" type="udt:IndicatorType"
             minOccurs="0 ">
              <xsd:annotation>
                ...see annotation...
              </xsd:annotation>
          </xsd:element>
          <xsd:element name="ID" type="udt:IDType" minOccurs="0" >
              <xsd:annotation>
                ...see annotation...
              </xsd:annotation>
          </xsd:element>
          <xsd:element name="TypeCode" type="qdt:DocumentTypeCode" minOccurs="0"
             maxOccurs="unbounded">
              <xsd:annotation>
                ...see annotation...
              </xsd:annotation>
          </xsd:element>
          ...
       </xsd:sequence>
    </xsd:complexType>
```

Example 7-10: Choice

```
    <xsd:complexType name="LocationType">
       <xsd:annotation>
             ... see annotation ...
       </xsd:annotation>
       <xsd:choice>
```

```
            <xsd:element name="GeoCoordinate" type="ram:GeoCoordinateType"
                minOccurs="0">
                  <xsd:annotation>
                  ... see annotation ...
                  </xsd:annotation>
            </xsd:element>
            <xsd:element name="Address" type="ram:AddressType"
                minOccurs="0">
                  <xsd:annotation>
                  ... see annotation ...
                  </xsd:annotation>
            </xsd:element>
            <xsd:element name="Location" type="ram:LocationType"
                minOccurs="0">
                  <xsd:annotation>
                  ... see annotation ...
                  </xsd:annotation>
            </xsd:element>
         </xsd:choice>
         </xsd:complexType>
```

Example 7-11: Sequence + Choice within Object Class "PeriodType"

```
         <xsd:complexType name="PeriodType">
         ...
            <xsd:sequence>
            <xsd:element name="DurationDateTime"
               type="qdt:DurationDateTimeType" minOccurs="0"
               maxOccurs="unbounded">
               ...
            </xsd:element>
            ...
            <xsd:choice>
                  <xsd:sequence>
                  <xsd:element name="StartTime" type="udt:TimeType"
                     minOccurs="0">
                        ...
                  </xsd:element>
                  <xsd:element name="EndTime" type="udt:TimeType"
                     minOccurs="0">
                        ...
                  </xsd:element>
                  </xsd:sequence>
                  <xsd:sequence>
                  <xsd:element name="StartDate" type="udt:DateType"
                     minOccurs="0">
                        ...
                  </xsd:element>
                  <xsd:element name="EndDate" type="udt:DateType"
                     minOccurs="0">
                        ...
                  </xsd:element>
                  </xsd:sequence>
                  <xsd:sequence>
                  <xsd:element name="StartDateTime" type="udt:DateTimeType"
                     minOccurs="0">
                        ...
                  </xsd:element>
                  <xsd:element name="EndDateTime" type="udt:DateTimeType"
                     minOccurs="0">
                        ...
                  </xsd:element>
                  </xsd:sequence>
            </xsd:choice>
         </xsd:sequence>
      </xsd:complexType>
```

[R101] The order and cardinality of the elements within an ABIE **`xsd:complexType`** MUST be according to the structure of the ABIE as defined in the model.

Example 7-12: Type definition of an ABIE

```
<!-- ================================================================== -->
<!-- ===== Type Definitions                                       ===== -->
<!-- ================================================================== -->
<xsd:complexType name="AgriculturalPlotType" >
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="ID" type="udt:IDType" >
          <xsd:annotation>
             ... see annotation ...
          </xsd:annotation>
      </xsd:element>
      ... see element declaration ....
   </xsd:sequence>
</xsd:complexType>
```

7.3.5 Element Declarations and References

Every ABIE will have a globally declared element.

[R102] For each ABIE, a named **xsd:element** MUST be globally declared.

[R103] The name of the ABIE **xsd:element** MUST be the **ccts:DictionaryEntryName** with the separators and 'Details' suffix removed and approved abbreviations and acronyms applied.

[R104] Every ABIE global element declaration MUST be of the **xsd:complexType** that represents the ABIE.

The content model of the complex type definitions will include both element declarations for BBIEs and ASBIEs whose **ccts:AssociationType** is Composition, and element references to the globally declared elements for ASBIEs whose **ccts:AssociationType** is not Composition. The BBIEs will always be declared locally.

[R105] For every BBIE identified in an ABIE, a named **xsd:element** MUST be locally declared within the `xsd:complexType` representing that ABIE.

[R106] Each BBIE element name declaration MUST be the property term and qualifiers and the representation term of the basic business information entity (BBIE). Where the word 'identification' is the final word of the property term and the representation term is 'identifier', the term 'identification' MUST be removed. Where the word 'indication' is the final word of the property term and the representation term is 'indicator', the term 'indication' MUST be removed from the property term.

[R107] If the representation term of a BBIE is 'text', 'text' MUST be removed.

[R108] The BBIE element MUST be based on an appropriate data type that is defined in the UN/CEFACT **qdt:QualifiedDataType** or **udt:UnqualifiedDataType** schema modules.

The ASBIEs whose **ccts:AssociationType** is Composition will always be declared locally.

[R109] For every ASBIE whose **ccts:AssociationType** is a composition, a named **xsd:element** MUST be locally declared.

[R110] For each locally declared ASBIE, the element name MUST be the ASBIE property term and qualifier term(s) and the object class term and qualifier term(s) of the associated ABIE.

[R111] For each locally declared ASBIE, the element declaration MUST be of the **sd:complexType** that represents its associated ABIE.

For each ASBIE who's **ccts:AssociationType** is not a composition, the globally declared element for the associated ABIE will be included in the content model of the associating ASBIE.

[R112] For every ASBIE whose **ccts:AssociationType** is not a composition, the globally declared element for the associated ABIE must be referenced using **xsd:ref**.

Example 7-13: Element declaration and reference within an ABIE type definition

```
<xsd:complexType name="PurchaseOrderRequestType">
     <xsd:sequence>
       <xsd:element name="ID" type="udt:IDType"/>
       <xsd:element name="SellerParty" type="ram:SellerPartyType"/>
       <xsd:element name="BuyerParty" type="ram:BuyerPartyType"/>
       <xsd:element ref="ram:OrderedLineItem" maxOccurs="unbounded"/>
    </xsd:sequence>
 </xsd:complexType>
```

7.3.6 Annotation

[R113] For every ABIE **`xsd:complexType`** and **`xsd:element`** definition a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references an ABIE instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be ABIE.
- DictionaryEntryName (required): The official name of an ABIE.
- Version (required): An indication of the evolution over time of an ABIE instance.
- Definition (required): The semantic meaning of an ABIE.
- ObjectClassTerm (required): The Object Class Term of the ABIE.
- ObjectClassQualifierTerm (optional): Qualifies the Object Class Term of the ABIE.
- BusinessProcessContextValue (optional, repetitive): The business process with which this ABIE is associated.
- GeopoliticalorRegionContexValue (optional, repetitive): The geopolitical/region contexts for this ABIE.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this ABIE.
- ProductContextValue (optional, repetitive): The product context for this ABIE.
- IndustryContextValue (optional, repetitive): The industry context for this ABIE.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this ABIE.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this ABIE.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this ABIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the ABIE.
- BusinessTerm (optional, repetitive): A synonym term under which the ABIE is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of an ABIE.

[R114] This rule was combined with [R113].

Example 7-14: Annotation of an ABIE

```
<xsd:complexType name="AgriculturalPlotType" >
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         <ccts:UniqueID>UN01002651</ccts:UniqueID>
         <ccts:Acronym>ABIE</ccts:Acronym>
         <ccts:DictionaryEntryName>Agricultural_ Plot. Details</ccts:DictionaryEntryName>
         <ccts:Version>1.0</ccts:Version>
         <ccts:Definition>A small piece of land used in agriculture.</ccts:Definition>
         <ccts:ObjectClassTerm>Plot</ccts:ObjectClassTerm>
         <ccts:ObjectClassQualifierTerm>Agricultural</ccts:ObjectClassQualifierTerm>
         <ccts:BusinessProcessContextValue>Crop Data Sheet</ccts:BusinessProcessContextValue>
         <ccts:GeopoliticalorRegionalContextValue>Global</ccts:
         GeopoliticalorRegionalContextValue >
         <ccts:OfficialConstraintContextValue>European, National, Local
         Regulations</ccts:OfficialConstraintContextValue>
         <ccts:ProductContextValue>Arable crop</ccts:ProductContextValue>
         <ccts:IndustryContextValue>Agricultural</ccts:IndustryContextValue>
         <ccts:BusinessProcessRoleContextValue>In All
         Contexts</ccts:BusinessProcessRoleContextValue>
```

```
            <ccts:SupportingRoleContextValue>In All Contexts</ccts:SupportingRoleContextValue>
            <ccts:SystemCapabilitiesContextValue>In All
            Contexts</ccts:SystemCapabilitiesContextValue>
          </xsd:documentation>
        </xsd:annotation>
        ...
    </xsd:complexType>
```

[R115] For every BBIE **`xsd:element`** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references a BBIE instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be BBIE.
- DictionaryEntryName (required): The official name of the BBIE.
- VersionID (required): An indication of the evolution over time of a BBIE instance.
- Definition (required): The semantic meaning of the BBIE.
- Cardinality (required): Indication whether the BBIE Property represents a not-applicable, optional, required and/or repetitive characteristic of the ABIE.
- ObjectClassTerm (required): The Object Class Term of the parent ABIE.
- ObjectClassQualifierTerm (optional): Qualifies the Object Class Term of the parent ABIE.
- PropertyTerm (required): The Property Term of the BBIE.
- PropertyQualifierTerm (optional): Qualifies the Property Term of the BBIE.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the BBIE.
- BusinessProcessContextValue (optional, repetitive): The business process with which this BBIE is associated.
- GeopoliticalorRegionContextValue (optional, repetitive): The geopolitical/region contexts for this BBIE.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this BBIE.
- ProductContextValue (optional, repetitive): The product context for this BBIE.
- IndustryContextValue (optional, repetitive): The industry context for this BBIE.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this BBIE.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this BBIE.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this BBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to this BBIE.
- BusinessTerm (optional, repetitive): A synonym term under which the BBIE is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of a BBIE.

Example 7-15: Annotation of a BBIE

```
<xsd:element name="ID" type="udt:IDType" >
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         <ccts:UniqueID>UN01002652</ccts:UniqueID>
         <ccts:Acronym>BBIE</ccts:Acronym>
         <ccts:DictionaryEntryName>Agricultural_ Plot. Identification.
         Identifier</ccts:DictionaryEntryName>
         <ccts:Version>1.0</ccts:Version>
         <ccts:Definition>The unique identifier for this agricultural plot.
         </ccts:Definition>
         </ccts:Cardinality>1</ccts:Cardinality>
         <ccts:ObjectClassTerm>Plot</ccts:ObjectClassTerm>
```

```
                <ccts:ObjectClassQualifierTerm>Agricultural</ccts:ObjectClassQualifierTerm>
                <ccts:PropertyTerm>Identification</ccts:PropertyTerm>
                <ccts:PrimaryRepresentationTerm>Identifier</ccts:PrimaryRepresentationTerm>
                <ccts:BusinessProcessContextValue>Crop Data
                Sheet</ccts:BusinessProcessContextValue>
                <ccts:GeopoliticalOrRegionContextValue>Global</ccts:GeopoliticalOrRegionCon
                textValue>
                <ccts:OfficialConstraintContextValue>European, National, Local
                Regulations</ccts:OfficialConstraintContextValue>
                <ccts:ProductContextValue>Arable crop</ccts:ProductContextValue>
                <ccts:IndustryContextValue>Agricultural</ccts:IndustryContextValue>
                <ccts:BusinessProcessRoleContextValue>In All
                Contexts</ccts:BusinessProcessRoleContextValue>
                <ccts:SupportingRoleContextValue>In All
                Contexts</ccts:SupportingRoleContextValue>
                <ccts:SystemCapabilitiesContextValue>In All
                Contexts</ccts:SystemCapabilitiesContextValue>
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>
```

[R116] For every ASBIE **`xsd:element`** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references an ASBIE instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be ASBIE.
- DictionaryEntryName (required): The official name of the ASBIE.
- Version (required): An indication of the evolution over time of the ASBIE instance.
- Definition (required): The semantic meaning of the ASBIE.
- Cardinality (required): Indication whether the ASBIE Property represents a not-applicable, optional, required and/or repetitive characteristic of the ABIE.
- ObjectClassTerm (required): The Object Class Term of the associating ABIE.
- ObjectClassQualifierTerm (optional): A term that qualifies the Object Class Term of the associating ABIE.
- PropertyTerm (required): The Property Term of the ASBIE.
- PropertyQualifierTerm (Optional): A term that qualifies the Property Term of the ASBIE.
- AssociatedObjectClassTerm (required): The Object Class Term of the associated ABIE.
- AssociatedObjectClassQualifierTerm (optional): Qualifies the Object Class Term of the associated ABIE.
- AssociationType (required): The Association Type of the ASBIE.
- BusinessProcessContextValue (optional, repetitive): The business process with which this ASBIE is associated.
- GeopoliticalorRegionContextValue (optional, repetitive): The geopolitical/region contexts for this ASBIE.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this ASBIE.
- ProductContextValue (optional, repetitive): The product context for this ASBIE.
- IndustryContextValue (optional, repetitive): The industry context for this ASBIE.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this ASBIE.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this ASBIE.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this ASBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the ASBIE.

- BusinessTerm (optional, repetitive): A synonym term under which the ASBIE is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of an ASBIE.

Example 7-16: Annotation of an ASBIE

```
<xsd:element name="IncludedInAgriculturalCountrySubDivision" type="ram:
   AgriculturalCountrySubDivisionType" minOccurs="0" maxOccurs="unbounded">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         <ccts:UniqueID>UN01002659</ccts:UniqueID>
         <ccts:Acronym>ASBIE</ccts:Acronym>
         <ccts:DictionaryEntryName>Agricultural_ Plot. Included In. Agricultural_
         Country Sub-Division</ccts:DictionaryEntryName>
         <ccts:Version>1.0</ccts:Version>
         <ccts:Definition>An agricultural country sub-division in which this
         agricultural plot is included.</ccts:Definition>
         <ccts:Cardinality>0..n</ccts:Cardinality>
         <ccts:ObjectClassTerm>Plot</ccts:ObjectClassTerm>
         <ccts:ObjectClassQualifierTerm>Agricultural</ccts:ObjectClassQualifierTe
         rm>
         <ccts:AssociationType>composition</ccts:AssociationType>
         <ccts:PropertyTerm>Included In</ccts:PropertyTerm>
         <ccts:AssociatedObjectClassTerm>Country Sub-
         Division</ccts:AssociatedObjectClassTerm>
         <ccts:AssociatedObjectClassQualifierTerm>Agricultural</ccts:AssociatedOb
         jectClassQualifierTerm>
         <ccts:BusinessProcessContextValue>Crop Data
         Sheet</ccts:BusinessProcessContextValue>
         <ccts:GeopoliticalOrRegionContextValue>Global</ccts:GeopoliticalOrRegion
         ContextValue>
         <ccts:OfficialConstraintContextValue>European, National, Local
         Regulations</ccts:OfficialConstraintContextValue>
         <ccts:ProductContextValue>Arable crop</ccts:ProductContextValue>
         <ccts:IndustryContextValue>Agricultural</ccts:IndustryContextValue>
         <ccts:BusinessProcessRoleContextValue>In All
         Contexts</ccts:BusinessProcessRoleContextValue>
         <ccts:SupportingRoleContextValue>In All
         Contexts</ccts:SupportingRoleContextValue>
         <ccts:SystemCapabilitiesContextValue>In All
      Contexts</ccts:SystemCapabilitiesContextValue></xsd:documentation>
   </xsd:annotation>
</xsd:element>
```

7.4 Core Component Type

7.4.1 Use of Core Component Type Module

The purpose of the core component type module is to define the core component types on which the unqualified data types are based. This module is only for reference and will not be included/imported in any schema.

7.4.2 Schema Construct

The core component type schema module will be constructed in a standardized format in order to ensure consistency and ease of use. The specific format is shown below and must adhere to the format of the relevant sections as detailed in Appendix B.

Example 7-17: Structure of Core Component Type Schema Module

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ================================================================== -->
<!-- ===== Core Component Type Schema Module                      ===== -->
<!-- ================================================================== -->
<!--
    Scheme agency:    UN/CEFACT
    Scheme version:   2.0
    Schema date:   [SCHEMADATE]

    ... see intellectual property disclaimer ...
```

```
    -->
    <xsd:schema targetNamespace=
        ... see namespace ... xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     elementFormDefault="qualified" attributeFormDefault="unqualified">
     <!-- ================================================================== -->
     <!-- ===== Type Definitions                                       ===== -->
     <!-- ================================================================== -->
     <!-- ===== CCT: AmountType                                        ===== -->
     <!-- ================================================================== -->
     ... see type definitions ...
    </xsd:schema>
```

7.4.3 Namespace Scheme

[R117] The core component type (CCT) schema module MUST be represented by the token cct.

Example 7-18: Namespace of Core Component Type Schema Module

```
"urn:un:unece:uncefact:documentation:draft:CoreComponentType:2"
```

Example 7-19: Schema-element of Core Component Type Schema Module

```
<xsd:schema
 targetNamespace="urn:un:unece:uncefact:documentation:draft:CoreComponentType:2"
 xmlns:cct="urn:un:unece:uncefact:documentation:draft:CoreComponentType:2"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
```

7.4.4 Imports and Includes

The core component type schema module does not import or include any other schema modules.

[R118] The **`cct:CoreCoreComponentType`** schema module MUST NOT include or import any other schema modules.

7.4.5 Type Definitions

[R119] Every core component type MUST be defined as a named **`xsd:complexType`** in the **`cct:CoreComponentType`** schema module.

[R120] The name of each **`xsd:complexType`** based on a core component type MUST be the dictionary entry name of the core component type (CCT), with the separators and spaces removed and approved abbreviations applied.

[R121] Each core component type **`xsd:complexType`** definition MUST contain one **`xsd:simpleContent`** element.

[R122] The core component type **`xsd:complexType`** definition **`xsd:simpleContent`** element MUST contain one **`xsd:extension`** element. This **`xsd:extension`** element must include an XSD based attribute that defines the specific XSD built-in data type required for the CCT content component.

[R123] Within the core component type **`xsd:extension`** element a **`xsd:attribute`** MUST be declared for each supplementary component pertaining to that core component type.

Example 7-20: Type definition of a CCT

```
    <!-- ================================================================== -->
    <!-- ===== Type Definitions                                       ===== -->
    <!-- ================================================================== -->
    <!-- =====  AmountType                                            ===== -->
    <!-- ================================================================== -->
     <xsd:complexType name="AmountType">
        <xsd:annotation>
            ... see annotation ...
        </xsd:annotation>
        <xsd:simpleContent>
           <xsd:extension base="xsd:decimal">
              <xsd:attribute name="currencyID" type="xsd:token" use="optional"
              <xsd:annotation>
```

```
                    ... see annotation ...
                </xsd:annotation>
            </xsd:attribute>
            ... see attribute declaration ...
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

7.4.6 Attribute Declarations

The current CCTS does not specify the components of the CCT supplementary component dictionary entry name. However, in order to ensure a standard approach to declaring the supplementary components as attributes, BPS has applied the naming concepts from ISO 11179, part 5. Specifically, BPS has defined the dictionary entry name as it is stated in CCTS in terms of object class, property term, and representation term. These components are identified in the annotation documentation for each supplementary component in the CCT schema module.

[R124] Each core component type supplementary component **`xsd:attribute`** name MUST be the CCTS supplementary component dictionary entry name with the separators and spaces removed.

[R125] If the object class of the supplementary component dictionary entry name contains the name of the representation term of the parent CCT, the duplicated object class word or words MUST be removed from the supplementary component **`xsd:attribute`** name.

[R126] If the object class of the supplementary component dictionary entry name contains the term 'identification', the term 'identification' MUST be removed from the supplementary component **`xsd:attribute`** name.

[R127] If the representation term of the supplementary component dictionary entry name is 'text', the representation term MUST be removed from the supplementary component **`xsd:attribute`** name.

[R128] The attribute representing the supplementary component MUST be based on the appropriate XSD built-in data type.

Example 7-21: Supplementary component other than code or identifier

```
<xsd:complexType name="BinaryObjectType">
   …
   <xsd:simpleContent>
      <xsd:extension base="xsd:base64Binary">
         <xsd:attribute name="format" type="xsd:string" use="optional">
            …
         </xsd:attribute>
         …
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

7.4.7 Extension and Restriction

The core component type schema module is a generic module based on the underlying core component types. No restriction or extension is appropriate.

7.4.8 Annotation

[R129] For every core component type **`xsd:complexType`** definition a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references the Core Component Type instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. . In this case the value will always be CCT.
- DictionaryEntryName (required): The official name of a Core Component Type.
- Version (required): An indication of the evolution over time of a Core Component Type instance.

- Definition (required): The semantic meaning of a Core Component Type.
- PrimaryRepresentationTerm (required): The primary representation term of the Core Component Type.
- PrimitiveType (required): The primitive data type of the Core Component Type.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Core Component Type.

Example 7-22: Annotation of a CCT

```
... see type definition ...
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         <ccts:UniqueID>UNDT000001</ccts:UniqueID>
         <ccts:Acronym>CCT</ccts:Acronym>
         <ccts:DictionaryEntryName>Amount. Type</ccts:DictionaryEntryName>
         <ccts:Version>1.0</ccts:Version>
         <ccts:Definition>A number of monetary units specified in a currency
               where the unit of the currency is explicit or
               implied.</ccts:Definition>
        <ccts:PrimaryRepresentationTerm>Amount</ccts:PrimaryRepresentationTerm>
        <ccts:PrimitiveType>decimal</ccts:PrimitiveType>
      </xsd:documentation>
   </xsd:annotation>
 ... see type definition ...
```

[R130] For every supplementary component **`xsd:attribute`** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (optional): The identifier that references the Supplementary Component instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of Supplementary Component. In this case the value will always be SC.
- DictionaryEntryName (required): The official name of the Supplementary Component.
- Definition (required): The semantic meaning of the Supplementary Component.
- Cardinality (required): The cardinality of the Supplementary Component.
- ObjectClassTerm (required): The Object Class of the Supplementary Component.
- PropertyTerm (required): The Property Term of the Supplementary Component.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the Supplementary Component.
- PrimitiveType (required): The primitive data type of the Supplementary Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Core Component.

Example 7-23: Annotation of a supplementary component

```
... see attribute declaration ...
<xsd:annotation>
   <xsd:documentation xml:lang="en">
      <ccts:Acronym>SC</ccts:Acronym>
      <ccts:DictionaryEntryName>Amount. Currency. Identifier</ccts:DictionaryEntryName>
      <ccts:Definition>The currency of the amount.</ccts:Definition>
      <ccts:ObjectClassTerm>Amount</ccts:ObjectClassTerm>
      <ccts:PropertyTerm>Currency</ccts:PropertyTerm>
      <ccts:PrimaryRepresentationTerm>Identifier</ccts:PrimaryRepresentationTerm>
      <ccts:PrimitiveType>string</ccts:PrimitiveType>
   </xsd:documentation>
</xsd:annotation>
... see attribute declaration ...
```

7.5 Unqualified Data Type

7.5.1 Use of Unqualified Data Type Module

The unqualified data type schema module will define data types for all primary and secondary representation terms as specified in the CCTS. All data types will be defined as **`xsd:complexType`** or **`xsd:simpleType`** and will only reflect restrictions as specified in CCTS and agreed upon industry best practices.

7.5.2 Schema Construct

The unqualified data types schema will be constructed in a standardized format in order to ensure consistency and ease of use. The specific format is shown below and must adhere to the format of the relevant sections as detailed in Appendix B.

Example 7-24: Structure of unqualified data type schema module

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ========================================================================= -->
<!-- ===== Unqualified Data Type Schema Module                          ===== -->
<!-- ========================================================================= -->
<!--
    Schema agency:    UN/CEFACT
    Schema version:   2.0
    Schema date:      [SCHEMADATE]

    ... see intellectual property disclaimer ...

-->
<xsd:schema targetNamespace=
    ... see namespace ...
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
 attributeFormDefault="unqualified">
 <!-- ========================================================================= -->
 <!-- ===== Imports                                                       ===== -->
 <!-- ========================================================================= -->
    ... see imports ...
 <!-- ========================================================================= -->
 <!-- ===== Type Definitions                                              ===== -->
 <!-- ========================================================================= -->
 <!-- ===== Amount. Type                                                  ===== -->
 <!-- ========================================================================= -->
 <xsd:complexType name="AmountType">
    ... see type definition ...
 </xsd:complexType>
 ...
</xsd:schema>
```

7.5.3 Namespace Scheme

[R131] The Unqualified Data Type schema module namespace MUST be represented by the token **udt**.

Example 7-25: Namespace of unqualified data type schema module

```
"urn:un:unece:uncefact:data:draft:UnqualifiedDataType:1"
```

Example 7-26: Schema-element of unqualified data type schema module

```
<xsd:schema targetNamespace=
 "urn:un:unece:uncefact:data:draft:UnqualifiedDataType:1"
 xmlns:udt=
 "urn:un:unece:uncefact:data:draft:UnqualifiedDataType:1"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
```

7.5.4 Imports and Includes

To maximize reusablility and minimize maintenance costs, it is strongly discouraged that the Unqualified Data Type schema will import any code list or identifier list schema modules. The schema may indicate that certain code or identifier lists are recommended for use by certain supplementary components, but those code lists or schema modules should be bound only if absolutely necessary. The Unqualified Data Type schema will not import any other schema modules.

[R132] The **udt:UnqualifiedDataType** schema MUST only import the following schema modules:
- **ids:IdentifierList** schema modules
- **clm:CodeList** schema modules

Example 7-27: Imports

```
<!-- ===== Imports                                                    ===== -->
<!-- ========================================================================= -->
<!-- ===== Imports of Code Lists                                      ===== -->
<!-- ========================================================================= -->
<xsd:import namespace=
   "urn:un:unece:uncefact:codelist:standard:6:3055:D12A"
schemaLocation="www.unece.org/uncefact/codelist/standard/UNECE_AgencyIdentificationCode_
D12A.xsd"/>
```

7.5.5 Type Definitions

Each unqualified data type is represented in the unqualified data type schema module as either a **`xsd:complexType`** or a **`xsd:simpleType`**. Unqualified data types are defined based on the core component types as specified in the CCTS.

[R133] An unqualified data type MUST be defined for each approved primary and secondary representation terms identified in the CCTS Permissible Representation Terms table.

[R134] The name of each unqualified data type MUST be the dictionary entry name of the primary or secondary representation term, with the word 'Type' appended, the separators and spaces removed and approved abbreviations applied.

In accordance with rules and principles in this document, the unqualified data type will be based on XSD built-in data types *whenever the XSD built-in data type meets the functionality of all the supplementary components for that data type.*

[R135] For every unqualified data type whose supplementary components map directly to the properties of a XSD built-in data type, the unqualified data type MUST be defined as a named **`xsd:simpleType`** in the **`udt:UnqualifiedDataType`** schema module.

[R136] Every unqualified data type **`xsd:simpleType`** MUST contain one **`xsd:restriction`** element. This **`xsd:restriction`** element MUST include an **`xsd:base`** attribute that defines the specific XSD built-in data type required for the content component.

When the unqualified data type does not directly map to an **`xsd:simpleType`** due to the supplementary components needing to be expressed, the unqualified data type will be defined as an **`xsd:complexType`**. If, however, some implementers want to use the simple type but others want to use the complex type, the unqualified data type should allow a choice between the two, using `xsd:choice`.

[R137] For every unqualified data type whose supplementary components are not equivalent to the properties of a XSD built-in data type, the unqualified data type MUST be defined as an **`xsd:complexType`** in the **`udt:UnqualifiedDataType`** schema module.

[R138] Every unqualified data type **`xsd:complexType`** definition MUST contain one **`xsd:simpleContent`** element.

[R139] Every unqualified data type **`xsd:complexType xsd:simpleContent`** element MUST contain one **`xsd:extension`** element. This **`xsd:extension`** element must include an **`xsd:base`** attribute that defines the specific XSD built-in data type required for the content component.

[R204] When a combination of the complex and simple types are necessary to support business requirements, the element MUST be declared as an `xsd:complexType` with an `xsd:choice` between elements declared as the two different alternatives.

7.5.6 Attribute Declarations

Each core component supplementary component is declared as an attribute of the complex type. In certain circumstances, continually providing the attributes necessary to convey code and identifier list metadata for multiple repetitions of the same element may prove burdensome. The namespace scheme for code lists and identification scheme lists has been specifically designed to include some of the supplementary components for the CCTs **`Code. Type`** and **`Identifier. Type`**. If an implementation desires this metadata to be

conveyed as attributes rather than part of the namespace declaration, a qualified data type with the additional attributes representing the missing supplementary components can be specified.

[R140] Within the unqualified data type **`xsd:complextype xsd:extension`** element an **`xsd:attribute`** MUST be declared for each supplementary component pertaining to the underlying CCT.

The attributes representing supplementary components will be named based on their underlying CCT supplementary component. The user declared attributes can be based on:

- XSD built-in types, if a specific supplementary component represents a variable value,
- Simple types of a code list, if the specific supplementary component represents a code value, or
- Simple types of an identifier scheme, if the specific supplementary component represents an identifier value.

For some CCTs, the CCTS identifies restrictions in the form of pointing to certain restrictive code or identifier lists. These restrictive lists will be declared in the code list or identifier schema module and the unqualified data type may reference these lists.

[R141] Each supplementary component **`xsd:attribute`** name MUST be the supplementary component name with the separators and spaces removed, and approved abbreviations and acronyms applied.

[R142] If the object class of the supplementary component dictionary entry name contains the name of the representation term, the duplicated object class word or words MUST be removed from the supplementary component **`xsd:attribute`** name.

[R143] If the object class of the supplementary component dictionary entry name contains the term 'identification', the term 'identification' MUST be removed from the supplementary component **`xsd:attribute`** name.

[R144] If the representation term of the supplementary component dictionary entry name is 'text', the representation term MUST be removed from the supplementary component **`xsd:attribute`** name.

Example 7-28: Type definitions of unqualified data types

```
<!-- ===== Type Definitions                                        ===== -->
<!-- ================================================================== -->
<!-- ===== Amount. Type                                            ===== -->
<!-- ================================================================== -->
<xsd:complexType name="AmountType">
   <xsd:annotation>
       ... see annotation ...
   </xsd:annotation>
   <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
         <xsd:attribute name="currencyID"
            type="clm5ISO42173A:ISO3AlphaCurrencyCodeContentType" use="optional">
            <xsd:annotation>
                ... see annotation ...
            </xsd:annotation>
         </xsd:attribute>
         <xsd:attribute name="currencyCodeListVersionID" type="xsd:token" use="optional">
            <xsd:annotation>
                ... see annotation ...
            </xsd:annotation>
         </xsd:attribute>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
<!-- ================================================================== -->
<!-- ===== Binary Object. Type                                     ===== -->
<!-- ================================================================== -->
<xsd:complexType name="BinaryObjectType">
   <xsd:annotation>
       ... see annotation ...
   </xsd:annotation>
   <xsd:simpleContent>
```

```
            <xsd:extension base="xsd:base64Binary">
               <xsd:attribute name="mimeCode" type="clmIANAMIMEMediaType:MIMEMediaTypeContentType">
                  <xsd:annotation>
                      ... see annotation ...
                  </xsd:annotation>
               </xsd:attribute>
               <xsd:attribute name="encodingCode"
               type="clm60133:CharacterSetEncodingCodeContentType" use="optional">
                  <xsd:annotation>
                      ... see annotation ...
                  </xsd:annotation>
               </xsd:attribute>
               <xsd:attribute name="characterSetCode"
               type="clmIANACharacterSetCode:CharacterSetCodeContentType"
               use="optional">
                    <xsd:annotation>
                     ... see annotation ...
                    </xsd:annotation>
               </xsd:attribute>
               <xsd:attribute name="uri" type="xsd:anyURI" use="optional">
                    <xsd:annotation>
                     ... see annotation ...
                    </xsd:annotation>
               </xsd:attribute>
               <xsd:attribute name="filename" type="xsd:string" use="optional">
                    <xsd:annotation>
                     ... see annotation ...
                    </xsd:annotation>
               </xsd:attribute>
            </xsd:extension>
         </xsd:simpleContent>
      </xsd:complexType>
```

The user declared attributes are dependent on the type of representation term of the specific supplementary component.

[R145] If the representation term of the supplementary component is 'Code' and validation is required, then the attribute representing this supplementary component MUST be based on the defined **`xsd:simpleType`** of the appropriate external imported code list.

Example 7-29: Supplementary Component is a Code

```
      <xsd:complexType name="MeasureType">
         <xsd:simpleContent>
            <xsd:extension base="xsd:decimal">
               <xsd:attribute name="unitCode"
                  type="clm6Recommendation20:MeasurementUnitCommonCodeContentType"use="optional">
                  …
               </xsd:attribute>
               …
            </xsd:extension>
         </xsd:simpleContent>
      </xsd:complexType>
```

[R146] If the representation term of the supplementary component is 'Identifier' and validation is required, then the attribute representing this supplementary component MUST be based on the defined **`xsd:simpleType`** of the appropriate external imported identifier list.

Example 7-30: Supplementary component is an identifier

```
      <xsd:complexType name="AmountType">
         <xsd:annotation>
            ...
         </xsd:annotation>
         <xsd:simpleContent>
            <xsd:extension base="xsd:decimal">
                  <xsd:attribute name="currencyID"
                        type="clm5ISO42173A:ISO3AlphaCurrencyCodeContentType" use="optional">
                        ...
                  </xsd:attribute>
            </xsd:extension>
         </xsd:simpleContent>
      </xsd:complexType>
```

[R147] If the representation term of the supplementary component is other than 'Code' or 'Identifier', then the attribute representing this supplementary component MUST be based on the appropriate XSD built-in data type.

Example 7-31: Supplementary component other than code or identifier

```
<xsd:complexType name="BinaryObjectType">
   …
   <xsd:simpleContent>
      <xsd:extension base="xsd:base64Binary">
         <xsd:attribute name="format" type="xsd:string" use="optional">
            …
         </xsd:attribute>
         …
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

7.5.7 Extension and Restriction

The unqualified data types can be further restricted through the creation of qualified data types. These qualified data types are defined in the **qdt:QualifiedDataType** schema module.

7.5.8 Annotation

[R148] For every unqualified data type **xsd:complexType** or **xsd:simpleType** definition a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references an Unqualified Data Type instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be UDT.
- DictionaryEntryName (required): The official name of the Unqualified Data Type.
- Version (required): An indication of the evolution over time of the Unqualified Data Type instance.
- Definition (required): The semantic meaning of the Unqualified Data Type.
- PrimaryRepresentationTerm (required): The primary representation term of the Unqualified Data Type.
- PrimitiveType (required): The primitive data type of the Unqualified Data Type.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Unqualified Data Type.

Example 7-32: Annotation of unqualified type definition

```
   .. see complex type definition ...
<xsd:annotation>
   <xsd:documentation xml:lang="en">
      <ccts:UniqueID>UNDT000001</ccts:UniqueID>
      <ccts:Acronym>UDT</ccts:Acronym>
      <ccts:DictionaryEntryName>Amount. Type</ccts:DictionaryEntryName>
      <ccts:Version>2.01</ccts:Version>
      <ccts:Definition>A number of monetary units specified in a currency where the
unit of the currency is explicit or implied.</ccts:Definition>
      <ccts:PrimitiveType>decimal</ccts:PrimitiveType>
   </xsd:documentation>
</xsd:annotation>
... see complex type definition ...
```

[R149] For every supplementary component **xsd:attribute** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (optional): The identifier that references a Supplementary Component instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be SC.
- Dictionary Entry Name (required): The official name of the Supplementary Component.
- Definition (required): The semantic meaning of the Supplementary Component.

- Cardinality (mandatory): The cardinality of the Supplementary Component.
- ObjectClassTerm (mandatory): The Object Class of the Supplementary Component.
- PropertyTerm (mandatory): The Property Term of the Supplementary Component.
- PrimaryRepresentationTerm (mandatory): The Primary Representation Term of the Supplementary Component.
- PrimitiveType (mandatory): The primitive data type of the Unqualified Data Type.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Component.

Example 7-33: Annotation of a supplementary component

```
... see complex type definition ...
<xsd:attribute name="currencyID" type="iso4217:CurrencyCodeContentType"
   use="required">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         <ccts:Acronym>SC</ccts:Acronym>
         <ccts:DictionaryEntryName>Amount. Currency. Identifier
         </ccts:DictionaryEntryName>
         <ccts:Definition>The currency of the amount.</ccts:Definition>
         <ccts:Cardinality>0..1</ccts:Cardinality>
         <ccts:ObjectClassTerm>Amount</ccts:ObjectClassTerm>
         <ccts:PropertyTerm>Currency</ccts:PropertyTerm>
         <ccts:PrimaryRepresentationTerm>Identifier<ccts:PrimaryRepresentationTerm>
         <ccts:PrimitiveType>decimal</ccts:PrimitiveType>
         <ccts:usageRule>By default, use latest version of ISO 4217.</ccts:usageRule>
      </xsd:documentation>
   </xsd:annotation>
</xsd:attribute>
... see complex type definition ...
```

7.6 Qualified Data Type

Ensuring consistency of qualified data types with the UN/CEFACT modularity and reuse goals requires creating a single schema module that defines all qualified data types. The qualified data type schema module name must follow the UN/CEFACT schema module naming approach. The qualified data type schema module will be used by the reusable ABIE schema module and all root schema modules.

7.6.1 Use of Qualified Data Type Schema Module

The data types defined in the unqualified data type schema module are of type **`xsd:complexType`** or **`xsd:simpleType`**. These types are intended to be suitable as the **`xsd:base`** type for some, but not all, BBIEs represented as **`xsd:elements`**. As business process modelling reveals the need for specialized data types, new qualified types will need to be defined. The qualified data types will also be necessary to define code lists and identifier lists. These new qualified data types must be based on an unqualified data type and must represent a semantic or technical restriction of the unqualified data type. Technical restrictions must be implemented as a **`xsd:restriction`** or a new **`xsd:simpleType`** if the supplementary components of the qualified data type map directly to the properties of a XSD built-in data type.

7.6.2 Schema Construct

The qualified data type schema will be constructed in a standardized format in order to ensure consistency and ease of use. The specific format is shown below and must adhere to the format of the relevant sections as detailed in Appendix B.

Example 7-34: Structure of qualified data type schema module

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ================================================================== -->
<!-- =====  Qualified Data Type Schema Module                       ===== -->
<!-- ================================================================== -->
<!--
    Schema agency:  UN/CEFACT
    Schema version:    2.0
    Schema date:    [SCHEMADATE]
... see intellectual property disclaimer ...
```

```
-->
<xsd:schema targetNamespace=
 ... see namespace ... xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
 <!-- ================================================================== -->
 <!-- ===== Imports                                                ===== -->
 <!-- ================================================================== -->
 ... see imports ...
 <!-- ================================================================== -->
 <!-- ===== Type Definitions                                       ===== -->
 <!-- ================================================================== -->
 ... see type definitions ...
</xsd:schema>
```

7.6.3 Namespace Scheme

[R150] The Qualified Data Type schema module namespace MUST be represented by the token **qdt**.

Example 7-35: Namespace name

```
"urn:un:unece:uncefact:data:draft:QualifiedDataType:1"
```

Example 7-36: Schema element

```
<xsd:schema targetNamespace="urn:un:unece:uncefact:data:draft:QualifiedDataType:1"
 xmlns:udt="urn:un:unece:uncefact:data:draft:UnqualifiedDataTypeSchema:1"
 xmlns:qdt="urn:un:unece:uncefact:data:draft:QualifiedDataTypeSchemaModule:1"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

7.6.4 Imports and Includes

Qualified data types will be derived from data types defined in the unqualified data types, code list, and identifier list schema modules. Code or identifier lists should be bound to a qualified data type only when absolutely necessary to avoid introducing complications in the maintenance of implementations.

[R151] The **qdt:QualifiedDataType** schema module MUST import the **udt:UnqualifiedDataType** schema module.

[R205] If a coupled design approach is used, then the qdt:QualifiedDataType schema module MUST import all code list and identifier scheme schemas used in the module.

> [Note]
>
> If needed, relevant UN/CEFACT and external code list and identifier scheme schema modules not imported by the **udt:UnqualifiedDataType** schema module may be imported.

7.6.5 Type Definitions

[R152] Where required to change facets of an existing unqualified data type, a new data type MUST be defined in the **qdt:QualifiedDataType** schema module.

[R153] A qualified data type MUST be based on an unqualified or qualified data type and add some semantic and/or technical restriction to the base data type.

[R154] The name of a qualified data type MUST be the name of its base unqualified or qualified data type with separators and spaces removed and with its qualifier term added.

The qualified data types can be derived from an unqualified or qualified data type **xsd:complexType** or **xsd:simpleType** or the code or identifier list schema module content type.

[R155] When a qualified data type is based on an unqualified data type that contains an `xsd:choice` element, then the qualified data type MUST be based on one or the other of the elements, but not both.

Example 7-37: Type Definitions

```
<!-- ========================================================================= -->
<!-- ===== Type Definitions                                            ===== -->
<!-- ========================================================================= -->
<!-- ===== Qualified Data Type based on DateTime Type                  ===== -->
<!-- ========================================================================= -->
<!-- =====  Formatted_ DateTime. Type                                  ===== -->
<!-- ========================================================================= -->
<xsd:complexType name="FormattedDateTimeType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:sequence>
        <xsd:simpleContent>
               <xsd:extension base="xsd:string">
                      <xsd:attribute name=”format” type=”xsd:string”>
                             <xsd:annotation>
                                    ... see annotation ...
                             </xsd:annotation>
                      </xsd:attribute>
               </xsd:extension>
        </xsd:simpleContent>
   </xsd:sequence>
</xsd:simpleType>
...
<!-- ========================================================================= -->
<!-- ===== Qualified Data Type based on Identifier. Type               ===== -->
<!-- ========================================================================= -->
<!-- =====  Country_ Identifier. Type                                  ===== -->
<!-- ========================================================================= -->
<xsd:complexType name="CountryIDType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:simpleContent>
        <xsd:extension base="ids5ISO316612A:ISOTwoletterCountryCodeContentType">
               <xsd:attribute name=”schemeID” type=”xsd:token” use=”optional”>
                      <xsd:annotation>
                      ... see annotation ...
                      </xsd:annotation>
               </xsd:attribute>
               ...
        </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
...
```

[R156] Every qualified data type based on an unqualified or qualified data type **`xsd:complexType`** whose supplementary components do not map directly to the properties of a XSD built-in data type

- MUST be defined as a **`xsd:complexType`**
- MUST contain one **`xsd:simpleContent`** element
- MUST contain one **`xsd:restriction`** element
- MUST include the unqualified data type as its **`xsd:base`** attribute.

[R157] Every qualified data type based on an unqualified or qualified data type **`xsd:simpleType`**

- MUST contain one **`xsd:restriction`** element
- MUST include the unqualified or qualified data type as its **`xsd:base`** attribute or if the facet restrictions can be achieved by use of a XSD built-in data type, then that XSD built-in data type may be used as the **`xsd:base`** attribute.

[R158] Every qualified data type based on a single code list or identifier list **`xsd:simpleType`** MUST contain one **`xsd:restriction`** element or **`xsd:union`** element. When using the **`xsd:restriction`** element, the **`xsd:base`** attribute MUST be set to the code list or identifier list schema module defined simple type with appropriate namespace qualification. When using the **`xsd:union`** element, the **`xsd:member`** type attribute MUST be set to the code list or identifier list schema module defined simple types with appropriate namespace qualification.

XML declarations for using code lists in qualified data types are shown in the following examples.

Example 7-38: Usage of only one Code List

```
<xsd:simpleType name="TemperatureMeasureUnitCodeType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:restriction base="clm6Recommendation20:MeasurementUnitCommonCodeContentType">
      <xsd:length value="3"/>
      <xsd:enumeration value="BTU">
          <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:Name>British thermal unit</ccts:Name>
            </xsd:documentation>
          </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="CEL">
          <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:Name>degree Celsius</ccts:Name>
            </xsd:documentation>
          </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="FAH">
          <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:Name>degree Fahrenheit</ccts:Name>
            </xsd:documentation>
          </xsd:annotation>
      </xsd:enumeration>
   </xsd:restriction>
</xsd:simpleType>
```

Example 7-39: Combination of Code Lists

```
<xsd:simpleType name="AccountDutyCodeType">
   <xsd:annotation>
       ... see annotation ...
   </xsd:annotation>
   <xsd:union memberType="clm64437:AccountTypeCodeContentType
      clm65153:DutyTaxFeeTypeCodeContentType"/>
</xsd:simpleType>
```

[R159] Every qualified data type that has a choice of two or more code lists or identifier lists
MUST be defined as an **`xsd:complexType`**
MUST contain the **`xsd:choice`** element whose content model must consist of element references for the alternative code lists or identifier lists to be included with appropriate namespace qualification.

Example 7-40: Usage of alternative Code Lists

```
<xsd:complexType name="PersonPropertyCodeType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:choice>
      <xsd:element ref="clm63479:MaritalCode"/>
      <xsd:element ref="clm63499:GenderCode"/>
   </xsd:choice>
</xsd:complexType>
```

7.6.6 Attribute and Element Declarations

There will be no element declarations in the qualified data type schema module. Attribute declarations in the qualified data type schema will either be those present in the base data type with further restrictions applied as required, or represented as XSD built-in data type facets such as those conveyed in the namespace declaration for code and identifier lists or representing further restrictions to **`xsd:dateTime`**.

[R160] The qualified data type **`xsd:complexType`** definition **`xsd:simpleContent`** element MUST only restrict attributes declared in its base type, or MUST only restrict facets equivalent to inherited supplementary components.

Example 7-41: Qualified Data Type Restricting an Identification Scheme

```
<xsd:complexType name="PartyIDType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:simpleContent>
      <xsd:restriction base="udt:IDType">
         <xsd:attribute name="schemeName" use="prohibited"/>
         <xsd:attribute name="schemeAgencyName" use="prohibited"/>
         <xsd:attribute name="schemeVersionID" use="prohibited"/>
         <xsd:attribute name="schemeDataURI" use="prohibited"/>
      </xsd:restriction>
   </xsd:simpleContent>
</xsd:complexType>
```

[Note] - In the case of decoupling a particular qualified data type from a set of value enumerations, the supplementary component attribute declarations MAY have a default= value for guidance. However, doing so obligates users to explicitly override the value if it does not apply when used.

7.6.7 Annotation

[R161] Every qualified data type definition MAY contain a structured set of annotations in the following sequence and pattern:

- UniqueID (required): The identifier that references a Qualified Data Type instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be QDT.
- DictionaryEntryName (required): The official name of the Qualified Data Type.
- Version (required): An indication of the evolution over time of the Qualified Data Type instance.
- Definition (required): The semantic meaning of the Qualified Data Type.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the Qualified Data Type.
- DataTypeQualifierTerm (required): A term that qualifies the Representation Term in order to differentiate it from its underlying Unqualified Data Type and other Qualified Data Type.
- PrimitiveType (required): The primitive data type of the Qualified Data Type.
- BusinessProcessContextValue (optional, repetitive): The business process context for this Qualified Data Type is associated.
- GeopoliticalorRegionContextValue (optional, repetitive): The geopolitical/region contexts for this Qualified Data Type.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this Qualified Data Type.
- ProductContextValue (optional, repetitive): The product context for this Qualified Data Type.
- IndustryContextValue (optional, repetitive): The industry context for this Qualified Data Type.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this Qualified Data Type.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this Qualified Data Type.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this Qualified Data Type.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Qualified Data Type.
- Example (optional, repetitive): Example of a possible value of a Qualified Data Type.

Example 7-42: Annotation of qualified data types

```
... see type definition ...
<xsd:annotation>
```

```
        <xsd:documentation xml:lang="en">
           <ccts:UniqueID>UN02000133</ccts:UniqueID>
           <ccts:Acronym>QDT</ccts:Acronym>
           <ccts:DictionaryEntryName>Subject_ Code. Type</ccts:DictionaryEntryName>
           <ccts:Version>1.0</ccts:Version>
           <ccts:Definition>A character string used to represent a subject code.
           </ccts:Definition>
          <ccts:PrimaryRepresentationTerm>Code</ccts:PrimaryRepresentationTerm>
           <ccts:PrimitiveType>string</ccts:PrimitiveType>
           <ccts:DataTypeQualifierTerm>Subject</ccts:DataTypeQualifierTerm>
         </xsd:documentation>
    </xsd:annotation>
    ... see type definition ...
```

[R162] For every supplementary component **xsd:attribute** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (optional): The identifier that references a Supplementary Component of a Core Component Type instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be SC.
- DictionaryEntryName (required): The official name of a Supplementary Component.
- Definition (required): The semantic meaning of a Supplementary Component.
- Cardinality (required): Indication whether the Supplementary Component Property represents a not-applicable, optional, required and/or repetitive characteristic of the Core Component Type.
- ObjectClassTerm (required): The Object Class Term of the associated Supplementary Component.
- PropertyTerm (required): The Property Term of the associated Supplementary Component.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the associated Supplementary Component.
- PrimitiveType (required): The Primitive Type of the associated Supplementary Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Component.

7.7 Code Lists

Codes are an integral component of any business to business information flow. Codes have been developed over time to facilitate the flow of compressed, standardized values that can be easily validated for correctness to ensure consistent data. In order for the XML instance documents to be fully validated by the parsers, any codes used within the XML document need to be available as part of the schema validation process. Many international, national and sectoral agencies create and maintain code lists relevant to their area. If required to be used within an information flow, these code lists will be stored in their own schema, and are referred to as external code lists. For example, many of the existing code lists that exist in the United Nations Code List (UNCL) will be stored as external code list schema for use within other UN/CEFACT XSD Schema.

It should be noted that the use of enumerated code lists in messages is entirely optional. Great care should be taken when using them due to the issues in maintenance that can be created by their use.

[R163] Each UN/CEFACT maintained code list MUST be defined in its own schema module.

External code lists must be used when they exist in schema module form and when they can be directly imported into a schema module.

UN/CEFACT may design and use an internal code list schema where an existing external code list schema needs to be extended, or where no suitable external code list schema exists. If a code list schema is created, it should be globally scoped and designed for reuse and sharing.

[R164] Internal code list schema MUST NOT duplicate existing external code list schema when the existing ones are available to be imported.

7.7.1 Schema Construct

The code list schema module will follow the general pattern for all UN/CEFACT XSD schema modules. Following the generic module information, the body of the schema will consist of code list definitions of the following general form:

Example 7-43: Structure of code lists

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ============================================================================ -->
<!-- =====  6Recommendation20 - Code List Schema Module                  ===== -->
<!-- ============================================================================ -->
<!--
   Schema agency:     UN/CEFACT
   Schema version:    2.0
   Schema date:       [SCHEMADATE]

   Code list name:    Measurement Unit Common Code
   Code list agency:  UNECE
   Code list version: 3

 ... see intellectual property disclaimer ...

-->
<xsd:schema targetNamespace=" ... see namespace ...
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
 <!-- ============================================================== -->
 <!-- ===== Root Element                                      ===== -->
 <!-- ============================================================== -->
    ... see root element declaration ...
 <!-- ============================================================== -->
 <!-- ===== Type Definitions                                  ===== -->
 <!-- ============================================================== -->
 <!-- ===== Type Definition: Measurement Unit Common Code Content Type == -->
 <!-- ============================================================== -->
    ... see type definition ...
</xsd:schema>
```

7.7.2 Namespace Name for Code Lists

The namespace name for code list is unique in order to convey some of the supplementary component information rather than including them as attributes. Specifically, the UN/CEFACT namespace structure for a namespace name of a code list should be:

urn:un:unece:uncefact:codelist:<status>:<Code List Agency Identifier|Code List Agency Name Text>:<Code List Identification Identifier|Code List Name Text>:<Code List Version Identifier>

Where:

- Namespace Identifier (NID) = **un**
- Namespace Specific String = unece:uncefact:codelist:
- <status> with unece and uncefact as fixed value second and third level domains within the NID of un and the code list as a fixed schema type.
- Supplementary Component String for unique identifiying of code lists = <Code List. Agency Identifier|Code List. Agency Name. Text>:<Code List. Identification. Identifier|Code List. Name. Text>:<Code List. Version. Identifier>

[R165] The namespace names for code list schemas MUST have the following structure:

urn:un:unece:uncefact:codelist:<status>:<Code List Agency Identifier|Code List Agency Name Text>:<Code List Identification. Identifier|Code List Name Text>:<Code List Version. Identifier>

Where:
codelist = this token identifying the schema as a code list
status = a token identifying the standards status of this code list: `draft|standard`
Code List Agency Identifier = identifies the agency that manages a code list. The default agencies used are those from DE 3055 but roles defined in DE 3055 cannot be used.
Code List Agency Name Text = the name of the agency that maintains the code list.
Code List Identification Identifier = identifies a list of the respective corresponding codes. listID is only unique within the agency that manages this code list. Code List Name Text = the name of a list of codes.

Code List Version Identifier = identifies the version of a code list.

[R166] This rule was combined with [R165].

Example 7-44: Namespace name of a code list with an agency and a code list identifier at draft status

```
"urn:un:unece:uncefact:codelist:draft:6:3403:D.04A"
where
6 = the value for UN/ECE in UN/CEFACT data element 3055 representing
    the Code List. Agency. Identifier
3403 = UN/CEFACT data element tag for Name type code representing the
    Code List. Identification. Identifier
D.04A = the version of the UN/CEFACT directory
```

Example 7-45: Namespace name of proprietary code list at draft status

```
"urn:un:unece:uncefact:codelist:draft:Security_Initiative:Document_Security:1.2"
where
SecurityInitiative = the code list agency name of a responsible agency, which
                     is not defined in UN/CEFACT data element 3055
                      representing the Code List. Agency. Identifier
DocumentSecurity = the value for Code List. Name. Text
1.2 = the value for Code List. Version. Identifier
```

Example 7-46: Namespace name of a code list with an agency and a code list identifier at standard status

```
"urn:un:unece:uncefact:codelist:standard:6:3403:D.04A"
where
6 = the value for UN/ECE in UN/CEFACT data element 3055 representing
    the Code List. Agency. Identifier
3403 = UN/CEFACT data element tag for Name status code representing the
    Code List. Identification. Identifier
D.04A = the version of the UN/CEFACT directory
```

Example 7-47: Namespace name of proprietary code list at standard status

```
"urn:un:unece:uncefact:codelist:standard:Security_Initiative:Document_Security:1.2"
where
SecurityInitiative = the code list agency name of a responsible agency, which
                     is not defined in UN/CEFACT data element 3055
                      representing the Code List. Agency. Identifier
DocumentSecurity = the value for Code List. Name. Text
1.2 = the value for Code List. Version. Identifier
```

Versioning for code lists published by external organisations is outside of the control of UN/CEFACT. As with UN/CEFACT published code lists and identifier list schema the value of the Code List Version Identifier will follow the same rules as for versioning of other schema modules.

7.7.3 UN/CEFACT XSD Schema Namespace Token for Code Lists

A unique token will be defined for each namespace of code lists. The token representing the namespace for code lists should be constructed based on the identifier of the agency maintaining the code list and the identifier of the specific code list as issued by the maintenance agency except where there is no identifier. When there is no identifier, the name for the agency and/or code list should be used instead. This will typically be true when proprietary code lists are used. This method of token construction will provide uniqueness with a reasonably short token. When the code list is used for a qualified data type with a restricted set of valid code values, the qualified data type name is required to be used to distinguish one set of restricted values from another.

The agency maintaining the code list will generally be either identified by the agency code as specified in data element 3055 in the UN/CEFACT Code List directory or the agency name if the agency does not have a code value in 3055. The identifier of the specific code list will generally be the data element tag of the corresponding list in the UN/CEFACT directory. If there is no corresponding data element, then the name of the code list will be used.

In cases where the code list schema is a restricted set of values of a published code list schema, the code list schema will be associated with a qualified data type, and the name of the qualified data type will be included as part of the namespace token to ensure uniqueness from the unrestricted code list schema.

[R167] Each UN/CEFACT maintained code list schema module MUST be represented by a unique token constructed as follows:

clm[Qualified data type name]<Code List Agency Identifier|Code List Agency Name Text><Code List Identification Identifier|Code List Name Text>

with any repeated words eliminated.

Example 7-48: Code list token with an agency and a code list identifier

```
The code list token for Name Type. Code is clm63403
where
6 = the value for UN/ECE in UN/CEFACT data element 3055 representing
    the Code List. Agency. Identifier
3403 = UN/CEFACT data element tag for Name status code representing
    the Code List. Identification. Identifier
```

Example 7-49: Code list token for a qualified data type with an agency and code list identifiers

```
Code list token for Person_Name Type. Code is clmPersonNameType63403
where
PersonNameType = name of the qualified data type
6 = the value for UN/ECE in UN/CEFACT data element 3055 representing
    the Code List. Agency. Identifier
3403 = UN/CEFACT data element tag for Name status code representing
    the Code List. Identification. Identifier
```

Example 7-50: Code list token for a proprietary code list

```
Code list token for a proprietary code list for Document Security is
clmSecurityInitiativeDocumentSecurity
where
SecurityInitiative = the code list agency name of a repsonsible agency, which
is not defined in UN/CEFACT data element 3055
      representing the Code List. Agency. Identifier
DocumentSecurity = the value for Code List. Name. Text
```

Based on the constructs identified in the above examples, a namespace declaration for a code list would appear as shown in Example 7-51.

Example 7-51: Target namespace declaration for a code list

```
<xsd:schema
 targetNamespace="urn:un:unece:uncefact:codelist:draft:6:4437:D.04A"
 xmlns:clm64437="urn:un:unece:uncefact:codelist:draft:6:4437:D.04A"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
```

> [Note]
>
> External developers are encouraged to follow the above construct rule when customizing schema for code lists to ensure that there is no namespace conflict.

7.7.4 Schema Location

Schema locations of code lists are typically defined as URL based URI schemes because of resolvability limitations of URN based URI schemes. However, UN/CEFACT XSD Schema of code lists use a URN based URI scheme for namespace declarations because persistence is considered more important than resolvability. In recognition of the need for resolvability of schema location, until such time as URNs become fully resolvable, UN/CEFACT will store schema of code lists in locations identified using a URL based URI scheme aligned with the URN based URI scheme used for the namespace declaration as follows:

urn:un:unece:uncefact:codelist:<status>:<Code List. Agency Identifier|Code List. Agency Name. Text>:<Code List. Identification. Identifier|Code List. Name. Text>:<Code List. Version. Identifier>

[R168] The structure for schema location of code lists MUST be:

../codelist/<status>/<Code List. Agency Identifier|Code List Agency Name Text>/<Code List Identification Identifier|Code List Name Text>_<Code List Version Identifier>.xsd

Where:
schematype = a token identifying the type of schema module: `codelist`
status = the status of the schema as: `draft|standard`
Code List Agency Identifier = identifies the agency that manages a code list. The default agencies used are those from DE 3055. Code List Agency Name Text = the name of the agency that maintains the code list.
Code List Identification Identifier = identifies a list of the respective corresponding codes. listID is only unique within the agency that manages this code list.
Code List Name Text = the name of a list of codes.
Code List Version Identifier = identifies the version of a code list.

[R169] Each **xsd:schemaLocation** attribute declaration of a code list MUST contain a resolvable URL, and if an absolute path is used, it MUST also be persistent.

[R170] This rule has been removed.

7.7.5 Imports and Includes

UN/CEFACT Code List Schema Modules are standalone schema modules and will not import or include any other schema modules.

[R171] Code List schema modules MUST not import or include any other schema modules.

7.7.6 Type Definitions

[R172] Within each code list module one, and only one, named **xsd:simpleType** MUST be defined for the content component.

[R173] The name of the **xsd:simpleType** MUST be the name of code list root element with the word 'ContentType' appended.

Example 7-52: Simple type definition of code lists

```
<!-- ================================================================== -->
<!-- ===== Type Definitions                                       ===== -->
<!-- ================================================================== -->
<!-- =====  Type Definition: Party Role Type Code          ===== -->
<!-- ================================================================== -->
<xsd:simpleType name="PartyRoleCodeContentType">
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="AA">
          ... see enumeration ...
      </xsd:enumeration>
      ...
   </xsd:restriction>
</xsd:simpleType>
```

[R174] The **xsd:restriction** element base attribute value MUST be set to **xsd:token**.

[R175] Each code in the code list MUST be expressed as an **xsd:enumeration**, where the **xsd:value** for the enumeration is the actual code value.

Example 7-53: Enumeration facet of code lists

```
... see type definition ...
<xsd:enumeration value="AA">
   <xsd:annotation>
       ... see annotation
   </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="AB">
   <xsd:annotation>
```

```
            ... see annotation
         </xsd:annotation>
      </xsd:enumeration>
      ...
```

The purpose of the code list schema module is to define the list of allowable values (enumerations) that can appear within a particular element. Facet restrictions may be included in code lists if desired for additional edit check validation.

7.7.7 Element and Attribute Declarations

Each code list schema module will have a single **xsd:simpleType** defined. This single **xsd:simpleType** definition will have a **xsd:restriction** expression whose base is a XSD built-in data type. The **xsd:restriction** will be used to convey the content component enumeration value(s).

[R176] For each code list a single root element MUST be globally declared.

[R177] The name of the code list root element MUST be the name of the code list following the naming rules as defined in section 5.3.

[R178] The code list root element MUST be of a type representing the actual list of code values.

Example 7-54: Root element declaration of code lists

```
<!-- ================================================================== -->
<!-- ===== Root Element                                           ===== -->
<!-- ================================================================== -->
<xsd:element name="PartyRoleCode" type="clm63035:PartyRoleCodeContentType"/>
```

The global declaration of a root element for each code list allows the use of code lists from different namespaces in a schema module when using **xsd:choice**.

Example 7-55: Usage of a choice of code lists

```
<xsd:complexType name="CalculationCurrencyCode">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:choice>
      <xsd:element ref="clm54217-N:CurrencyCode"/>
      <xsd:element ref="clm54217-A:CurrencyCode"/>
   </xsd:choice>
 </xsd:complexType>
```

7.7.8 Extension and Restriction

Users of the UN/CEFACT library may identify any subset or superset they wish from a specific code list for their own trading community requirements by defining a qualified data type.

Representation of a qualified data type of code lists could be

- a combination of several individual code lists using **xsd:union**
- a choice between several code lists, using **xsd:choice**
- subsetting an existing code list using **xsd:restriction**

Each of these can easily be accommodated in this syntax solution as required by the user's business requirements. Appendix D provides detailed examples of the various code list options.

7.7.9 Annotation

In order to facilitate a clear and unambiguous understanding of the list of allowable codes within an element, annotations will be provided for each enumeration to provide the code name and description.

[R179] Each code list **xsd:enumeration** MAY contain a structured set of annotations in the following sequence and pattern:

- Name (required): The name of the code.
- Description (optional): Descriptive information concerning the code.

Example 7-56: Annotation of codes

```
<xsd:enumeration value="AI">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         <ccts:Name>Successful job applicant</ccts:Name>
         <ccts:Description>Person who has been chosen for a job. </ccts:Description>
      </xsd:documentation>
   </xsd:annotation>
</xsd:enumeration>...
```

7.8 Identifier List Schema

When required, separate schema modules will be defined for identification schemes that have a token, and optionally a description, and that have the same functionality as a code list. In this way, XML instance documents containing these identifiers can be fully validated by the parsers. Other identifier schemes should be defined as a qualified or unqualified data type as appropriate.

External identifier lists must be used when they exist in schema module form and when they can be directly imported into a schema module.

UN/CEFACT may design and use an internal identifier list where an existing external identifier list needs to be extended, or where no suitable external identifier list exists. If an identifier list is created, the lists should be globally scoped and designed for reuse and sharing.

It should be noted that the use of enumerated identifier lists in messages is entirely optional. Great care should be taken when using them due to the issues in maintenance that can be created by their use.

[R180] Internal identifier lists schema MUST NOT duplicate existing external identifier list schema when the existing ones are available to be imported.

[R181] Each UN/CEFACT maintained identifier list MUST be defined in its own schema module.

7.8.1 Schema Construct

The identifier list schema module will follow the general pattern for all UN/CEFACT XSD schema modules. Following the generic module information, the body of the schema will consist of identifier list definitions of the following general form:

Example 7-57: Structure of identifier lists

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ========================================================================= -->
<!-- =====  Agency Identifier - Identifier List Schema Module        ===== -->
<!-- ========================================================================= -->
<!--
   Schema agency:     UN/CEFACT
   Schema version:    2.0
   Schema date:       [SCHEMADATE]

   Identifier list name:        Agency Identifier
   Identifier list agency:      UNECE
    Code list version:   3

 ... see intellectual property disclaimer ...

-->
<xsd:schema targetNamespace=" ... see namespace ...
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
 <!-- ===== Root Element                                              ===== -->
 <!-- ========================================================================= -->
    ... see root element declaration ...
 <!-- ===== Type Definitions                                          ===== -->
 <!-- ========================================================================= -->
 <!-- ===== Type Definition: Agency Identifier            ===== -->
 <!-- ========================================================================= -->
    ... see type definition ...
</xsd:schema>
```

7.8.2 Namespace Name For Identifier List Schema

The namespace name for identifier list is unique in order to convey some of the supplementary component information rather than including them as attributes. Specifically, the UN/CEFACT namespace structure for a namespace name of an identifier list schema should be:

urn:un:unece:uncefact:identifierlist:<status>:<Identifier Scheme Agency Identifier|Identifier Scheme Agency Name Text>:< Identifier Scheme Identifier|Identifier Scheme Name Text>:< Identifier Scheme Version. Identifier>

Where:

- Namespace Identifier (NID) = **un**
- Namespace Specific String =
- unece:uncefact:codelist:<status> with unece and uncefact as fixed value second and third level domains within the NID of un and the code list as a fixed schema type.
- Supplementary Component String for unique identifiying of identifier schemes = <Identifier Scheme Agency Identifier|Identifier Scheme Agency Name Text>:< Identifier Scheme Identifier|Identifier Scheme Name Text>:< Identifier Scheme Version Identifier>

[R182] The names for namespaces MUST have the following structure:

urn:un:unece:uncefact:identifierlist:<status>:<Identifier Scheme. Agency Identifier|Identifier Scheme Agency Name Text>:<Identifier Scheme Identifier|Identifier Scheme Name Text>:<Identifier Scheme Version Identifier>

Where:
status = the token identifying the publication status of this identifier scheme schema = `draft|standard`
identifierlist = this token identifying the schema as an identifier scheme
Identifier Scheme Agency Identifier = the identification of the agency that maintains the identification scheme.
Identifier Scheme Agency Name. Text = the name of the agency that maintains the identification list.
Identifier Scheme Identifier = the identification of the identification scheme.
Identifier Scheme Name. Text = the name of the identification scheme.
Identifier Scheme Version. Identifier = the version of the identification scheme.

[R183] This rule was combined with [R182].

Example 7-58: Namespace name of an identifier list schema with an agency and an identifier list schema identifier at draft status

```
"urn:un:unece:uncefact:identifierlist:draft:5:3166:2001"

where
5 = the value for ISO in UN/CEFACT data element 3055 representing the Code List.
    Agency. Identifier
4217 = ISO identifier scheme identifier for country code representing the Code List.
    Identification. Identifier
2001 = the version of the ISO country identifier list.
```

Example 7-59: Namespace of an identifier list schema with an agency and an identifier list schema identifier at standard status

```
"urn:un:unece:uncefact:identifierlist:standard:5:3166:2001"
where
5 = the value for ISO in UN/CEFACT data element 3055 representing the Code List.
    Agency. Identifier
4217 = ISO identifier scheme identifier for country code representing the Code List.
    Identification. Identifier
2001 = the version of the ISO country identifier list.
```

Versioning for identifier list schemas published by external organisations is outside of the control of UN/CEFACT. As with UN/CEFACT published identifier list schema the value of the Identifier Scheme Version Identifier will follow the same rules as for versioning of other schema modules.

7.8.3 UN/CEFACT XSD Namespace Token for Identifier List Schema

A unique token will be defined for each namespace of an identifier list schema. The token representing the namespace for identifier lists should be constructed based on the identifier of the agency maintaining the identification list and the identifier of the specific identification list as issued by the maintenance agency. This method of token construction will provide uniqueness with a reasonably short token. When the identifier list is used for a qualified data type with a restricted set of valid identifier values, the qualified data type name is required to be used to distinguish one set of restricted values from another.

The agency maintaining the identification list will be either identified by the agency code as specified in data element 3055 in the UN/CEFACT EDIFACT directory. The identifier of the identification list will be the identifier as allocated by the identification scheme agency.

In cases where the identifier scheme is a restricted set of values of a published identifier list, the identifier list schema will be associated with a qualified data type, and the name of the qualified data type will be included as part of the namespace token to ensure uniqueness from the unrestricted identifier list schema.

[R184] Each UN/CEFACT maintained identifier list schema module MUST be represented by a unique token constructed as follows:

ids[Qualified data type name]<Identification Scheme Agency Identifier><Identification Scheme Identifier>

with any repeated words eliminated.

Example 7-60: Identifier list token

```
Token for the ISO Country Codes would be:  ids53166-1
where:
5 = the Identification Scheme Agency Identifier for ISO in codelist 3055 3166-1 = the
Identification Scheme Identifier as allocated by ISO.
```

Based on the constructs identified in Example 7-60, a namespace declaration for an identifier list would appear as shown in Example 7-61.

Example 7-61: Target Namespace declaration for an Identifier list

```
<xsd:schema
 targetNamespace="urn:un:unece:uncefact:identifierlist:draft:5:3166-1:1997"
 xmlns:ids53166-1="urn:un:unece:uncefact:identifierlist:draft:5:3166-1:1977"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
```

> [Note]
>
> External developers are encouraged to follow the above construct rule when customizing schema for identifier lists to ensure that there is no namespace conflict.

7.8.4 Schema Location

Schema locations of identifier list schema are typically defined as URL based URI schemes because of resolvability limitations of URN based URI schemes. However, UN/CEFACT XSD Schema of identifier lists use a URN based URI scheme for namespace declarations because persistence is considered more important than resolvability. In recognition of the need for resolvability of schema location, until such time as URNs become fully resolvable, UN/CEFACT will store schema of identifier list in locations identified using a URL based URI scheme aligned with the URN based URI scheme used for the namespace declaration as follows:

urn:un:unece:uncefact:identifierlist:<status>:<Identifier Scheme Agency Identifier|Identifier Scheme Agency Name Text>:< Identifier Scheme Identifier|Identifier Scheme Name Text>:< Identifier Scheme Version. Identifier>

[R185] The structure for schema location of identifier lists MUST be:

[../identifierlist/<status>/<Identifier **Scheme Agency Identifier|Identifier Scheme Agency Name Text>/< Identifier Scheme Identifier|Identifier Scheme Name Text>_< Identifier Scheme Version Identifier>.xsd**

Where:
schematype = a token identifying the type of schema module: identifierlist
status = the status of the schema as: **`draft|standard`**
Identifier Scheme. Agency Identifier = the identification of the agency that maintains the identification scheme.
Identifier Scheme. Agency Name. Text = the name of the agency that maintains the identification scheme.
Identifier Scheme. Identifier = the identification of the identification scheme.
Identifier Scheme. Name. Text = the name of the identification scheme.
Identifier Scheme. Version. Identifier = the version of the identification scheme.

[R186] Each **`xsd:schemaLocation`** attribute declaration of an identifier list schema MUST contain a resolvable URL, and if an absolute path is used, it MUST also be persistent.

[R187] This rule has been removed.

7.8.5 Imports and Includes

UN/CEFACT Identifier List Schema Modules are standalone schema modules and will not import or include any other schema modules.

[R188] Identifier list schema modules MUST NOT import or include any other schema modules.

7.8.6 Type Definitions

A restriction has to be declared in order to define the content component (the simple type) as a restriction of the unqualified data type in order to comply with parser requirements. The restriction itself is the list of enumerations.

[R189] Within each identifier list schema module one, and only one, named **`xsd:simpleType`** MUST be defined for the content component.

[R190] The name of the **`xsd:simpleType`** MUST be the name of the identifier list root element with the word 'ContentType' appended.

Example 7-62: Simple type definition of an identifier list

```
<!-- ===== Type Definitions                                        ===== -->
<!-- ========================================================================= -->
<xsd:simpleType name="PaymentTermsDescriptionIdentifierContentType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="1">
            ... see enumeration ...
        </xsd:enumeration>
    </xsd:restriction>
</xsd:simpleType>
```

[R191] The **`xsd:restriction`** element base attribute value MUST be set to **`xsd:token`**.

[R192] Each identifier in the identifier list MUST be expressed as an **`xsd:enumeration`**, where the **`xsd:value`** for the enumeration is the actual identifier value.

Example 7-63: Enumeration facet of an identifier list

```
... see type defintion ...
<xsd:enumeration value="1">
    <xsd:annotation>
        ... see annotation
    </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="2">
    <xsd:annotation>
        ... see annotation
    </xsd:annotation>
</xsd:enumeration>
...
```

The purpose of the identifier list schema module is to define the list of allowable values (enumerations) that can appear within a particular element. Therefore, no other facet restrictions are allowed.

[R193] Facets other than **xsd:enumeration** MUST NOT be used in the identifier list schema module.

7.8.7 Attribute and Element Declarations

Each identifier list schema module will have a single **xsd:simpleType** defined. This single **xsd:simpleType** definition will have a **xsd:restriction** expression whose base is a XSD built-in data type. The **xsd:restriction** will be used to convey the content component enumeration value(s).

[R194] For each identifier list a single root element MUST be globally declared.

[R195] The name of the identifier list root element MUST be the name of the identifier list following the naming rules as defined in section 5.3.

[R196] The identifier list root element MUST be of a type representing the actual list of identifier values.

Example 7-64: Root element declaration of identifier lists

```
<!-- ================================================================== -->
<!-- ===== Root Element                                           ===== -->
<!-- ================================================================== -->
<xsd:element name="PaymentTermsDescriptionIdentifier" type="ids64277:
PaymentTermsDescriptionIdentifierContentType"/>
```

The global declaration of a root element for each identifier list allows the use of identifier lists from different namespaces in a schema module when using **xsd:choice**.

Example 7-65: Usage of a choice of identifier lists

```
<xsd:complexType name="CalculationCurrencyCode">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:choice>
      <xsd:element ref="clm54217-N:CurrencyCode"/>
      <xsd:element ref="clm54217-A:CurrencyCode"/>
   </xsd:choice>
 </xsd:complexType>
```

7.8.8 Extension and Restriction

Users of the UN/CEFACT library may identify any subset or superset they wish from a specific identifier list for their own trading community requirements by defining a qualified data type.

Representation of a qualified data type of identifier lists could be

- a combination of several individual identifier lists using **xsd:union**
- a choice between several identifier lists, using **xsd:choice**
- subsetting an existing code list using **xsd:restriction**

Each of these can easily be accommodated in this syntax solution as required by the user's business requirements. Appendix D provides detailed examples of the various identifier list options.

XML declarations for using identifier lists in qualified data types are shown in the following examples.

Example 7-66: Enumeration facet of identifier scheme

```
... see type definition ...
 <xsd:enumeration value="AD">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
 </xsd:enumeration>
 <xsd:enumeration value="AE">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
 </xsd:enumeration>
 <xsd:enumeration value="AF">
```

```
      <xsd:annotation>
         ... see annotation ...
      </xsd:annotation>
   </xsd:enumeration>
```

Example 7-67: Usage of only one identifier scheme

```
   <xsd:simpleType name="CountryIDType">
      <xsd:annotation>
         ... see annotation ...
      </xsd:annotation>
      <xsd:restriction base="ids53166:CountryIDContentType"/>
   </xsd:simpleType>
```

Example 7-68: Usage of alternative identifier schemes

```
   <xsd:complexType name="GeopoliticalIDType">
      <xsd:annotation>
         ... see annotation ...
      </xsd:annotation>
      <xsd:choice>
         <xsd:element ref="ids53166:CountryCode"/>
         <xsd:element ref="ids53166-2:RegionCode"/>
      </xsd:choice>
     </xsd:complexType>
```

7.8.9 Annotation

In order to facilitate a clear and unambiguous understanding of the list of allowable identifiers within an element, annotations will be provided for each enumeration to provide the name, and optionally a description of, the identifier.

[R197] Each **`xsd:enumeration`** MAY contain a structured set of annotations in the following sequence and pattern:

- Name (required): The name of the identifier.
- Description (optional): Descriptive information concerning the identifier.

Example 7-69: Annotation of Identifiers

```
   <xsd:enumeration value="1">
      <xsd:annotation>
         <xsd:documentation xml:lang="en">
            <ccts:Name>Draft(s) drawn on issuing bank</ccts:Name>
            <ccts:Description>Draft(s) must be drawn on the issuing
            bank.</ccts:Description>
         </xsd:documentation>
      </xsd:annotation>
   </xsd:enumeration>
```

8 XML Instance Documents

In order to be UN/CEFACT conformant, an instance document must be valid against the relevant UN/CEFACT compliant XML schema. The XML instance documents should be readable and understandable by both humans and applications, and should enable reasonably intuitive interactions. It should represent all truncated tag names as described in section 7. A XPath navigation path should describe the complete semantic understanding by concatenating the nested elements. This navigation path should also reflect the meaning of each dictionary entry name of a BBIE or ASBIE.

8.1 Character Encoding

In conformance with ISO/IEC/ITU/UNECE Memorandum of Understanding Management Group (MOUMG) Resolution 01/08 (MOU/MG01n83) all UN/CEFACT XML will be instantiated using UTF. UTF-8 is the preferred encoding, but UTF-16 may be used where necessary to support other languages.

[R198] All UN/CEFACT XML MUST be instantiated using UTF. UTF-8 should be used as the preferred encoding. If UTF-8 is not used, UTF-16 MUST be used.

8.2 xsi:schemaLocation

The **`xsi:schemaLocation`** and **`xsi:noNamespaceLocation`** attributes are part of the XML schema instance namespace (http://www.w3.org/2001/XMLSchema-instance). To ensure consistency, the token **`xsi`** will be used to represent the XML schema instance namespace.

[R199] The **`xsi`** prefix MUST be used where appropriate for referencing **`xsd:schemaLocation`** and **`xsd:noNamespaceLocation`** attributes in instance documents.

8.3 Empty Content

Empty elements do not provide the level of assurance necessary for business information exchanges and as such, will not be used.

[R200] UN/CEFACT conformant instance documents MUST NOT contain an element devoid of content.

[R201] The **`xsi:nil`** attribute MUST NOT appear in any conforming instance.

8.4 xsi:type

The **`xsi:type`** attribute allows for substitution during an instantiation of a xml document. In the same way that substitution groups are not allowed, the **`xsi:type`** attribute is not allowed.

[R202] The **`xsi:type`** attribute MUST NOT be used

Appendix A Related Documents

The following documents provided significant levels of influence in the development of this document:

- UN/CEFACT Core Components Technical Specification, Part 8 of the ebXML Framework Version 2.01 with corrigenda.
- UN/CEFACT Core Components Business Document Assembly Technical Specification.

Appendix B Overall Structure

The structure of an UN/CEFACT compliant XML schema must contain one or more of the following sections as relevant. Relevant sections must appear in the order given:

- XML Declaration
- Schema Module Identification and Copyright Information
- Schema Start-Tag
- Includes
- Imports
- Element
- Root Element
- Global Elements
- Type Definitions

B.1 XML Declaration

A UTF-8 encoding is adopted throughout all UN/CEFACT XML schema, unless characters are required that are not in UTF-8, in which case UTF-16 can be used.

Example B-1: XML Declaration

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

B.2 Schema Module Identification and Intellectual Property Disclaimer

Example B-2: Schema Module Identification and Intellectual Property Disclaimer

```
<!-- ================================================================== -->
      <!-- =====  Example - Schema Module Name                        ===== -->
      <!-- ================================================================== -->
      <!--
         Schema agency:              UN/CEFACT
         Schema version:             2.0
         Schema date:                [SCHEMADATE]



       ECE draws attention to the possibility that the practice or implementation of its
       outputs (which include but are not limited to Recommendations, norms, standards,
       guidelines and technical specifications) may involve the use of a claimed intellectual
       property right.
       Each output is based on the contributions of participants in the UN/CEFACT process, who
       have agreed to waive enforcement of their intellectual property rights pursuant to the
       UN/CEFACT IPR Policy (document ECE/TRADE/C/CEFACT/2010/20/Rev.2 available at
       http://www.unece.org/cefact/cf_docs.html or from the ECE secretariat). ECE takes no
       position concerning the evidence, validity or applicability of any claimed intellectual
       property right or any other right that might be claimed by any third parties related to
       the implementation of its outputs. ECE makes no representation that it has made any
       investigation or effort to evaluate any such
       rights.
       Implementers of UN/CEFACT outputs are cautioned that any third-party intellectual
       property rights claims related to their use of a UN/CEFACT output will be their
       responsibility and are urged to ensure that their use of UN/CEFACT outputs does not
       infringe on an intellectual property right of a third party.
       ECE does not accept any liability for any possible infringement of a claimed
       intellectual property right or any other right that might be claimed to relate to the
       implementation of any of its outputs.
      -->
```

B.3 Schema Start Tag

The Schema Start-Tag section of an UN/CEFACT compliant XML schema must contain one or more of the below declarations as relevant. Relevant declarations must appear in the order given:

- Version
- Namespaces
  - targetNamespace attribute xmlns:xsd attribute
  - namespace declaration for current schema
  - namespace declaration for reusable ABIEs actually used in the schema
  - namespace declaration for unqualified data types actually used in the schema
  - namespace declaration for qualified data types actually used in the schema
  - namespace declaration for code lists actually used in the schema
  - namespace declaration for identifier schemes actually used in the schema
  - namespace declaration for CCTS
- Form Defaults elementFormDefault attributeFormDefault
- Others
  - other schema attributes with schema namespace
  - other schema attributes with non-schema namespace

Example B-3: XML Schema Start Tag

```
<xsd:schema
xmlns:ccts="urn:un:unece:uncefact:documentation:standard:CoreComponentsTechnicalSpecificat
ion:2"
xmlns:clm6Recommendation20="urn:un:unece:uncefact:codelist:standard:6:Recommendation20:8"
xmlns:clm60133="urn:un:unece:uncefact:codelist:standard:6:0133:40106"
xmlns:clm5ISO42173A="urn:un:unece:uncefact:codelist:standard:5:ISO42173A:2012-08-31"
xmlns:ids5ISO316612A="urn:un:unece:uncefact:identifierlist:standard:5:ISO316612A:SecondEdi
tion2006VI-13"
xmlns:clmIANAMIMEMediaType="urn:un:unece:uncefact:codelist:standard:IANA:MIMEMediaType:201
3-01-03"
xmlns:clmIANACharacterSetCode="urn:un:unece:uncefact:codelist:standard:IANA:CharacterSetCo
de:2013-01-08" xmlns:clm63055="urn:un:unece:uncefact:codelist:standard:6:3055:D12A"
xmlns:udt="urn:un:unece:uncefact:data:standard:UnqualifiedDataType:13"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:un:unece:uncefact:data:standard:UnqualifiedDataType:13"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
version="13.0">
```

B.4 Includes

The Include section of an UN/CEFACT compliant XML schema must contain one or more of the below declarations as relevant. Relevant declarations must appear in the order given:

- Inclusion of the internal ABIE schema module if used

Example B-4: Includes

```
<!-- ================================================================== -->
<!-- =====  Include                                                 ===== -->
<!-- ================================================================== -->
<!-- =====  Inclusion of internal ABIE                              ===== -->
<!-- ================================================================== -->
 <xsd:include
namespace="urn:un:unece:uncefact:data:standard:CIIAggregateBusinessInformationEntity
:1"
schemaLocation="http://www.unece.org/uncefact/data/standard/CIIAggregateBusinessInformationE
ntity_1.xsd"/>
```

B.5 Imports

The Import section of an UN/CEFACT compliant XML schema must contain one or more of the below declarations as relevant. Relevant declarations must appear in the order given:

- Import of the reusable ABIE schema module if used
- Import of the unqualified data type schema module if used
- Import of the qualified data type schema module if used
- Import of code list schema modules actually used
- Import of identifier list schema modules actually used

Example B-5: Imports

```
<!-- ========================================================================= -->
<!-- =====  Imports                                                        ===== -->
<!-- ========================================================================= -->
<!-- =====  Import of Unqualified Data Type                                ===== -->
<!-- ========================================================================= -->
<xsd:import namespace="urn:un:unece:uncefact:data:standard:UnqualifiedDataType:14"
schemaLocation="http://www.unece.org/uncefact/data/standard/UnqualifiedDataType_14.xsd "/>
<!-- ========================================================================= -->
<!-- =====  Import of Qualified Data Type                                  ===== -->
<!-- ========================================================================= -->
<xsd:import namespace="urn:un:unece:uncefact:data:standard:QualifiedDataType:14"
schemaLocation="http://www.unece.org/uncefact/data/standard/QualifiedDataType_14.xsd"/
>
<!-- ========================================================================= -->
<!-- =====  Import of Reusable Aggregate Business Information Entity Schema Module -->
<!-- ========================================================================= -->
<xsd:import
namespace="urn:un:unece:uncefact:data:standard:ReusableAggregateBusinessInformationEn
tity:13" schemaLocation="ReusableAggregateBusinessInformationEntity_13p0.xsd"/>
```

B.6 Root Element

The root element is declared first when needed in schema that are used to support instance documents. Global elements are then declared following the root element when it is present.

Example B-6:

```
<!-- ========================================================================= -->
<!-- =====  Element Declarations                                           ===== -->
<!-- ========================================================================= -->
<!-- =====  Root element                                                   ===== -->
<!-- ========================================================================= -->
 <xsd:element name="[ELEMENTNAME]" type="[TOKEN]:[TYPENAME]>
<!-- ========================================================================= -->
<!-- =====  Global Element Declarations                                    ===== -->
<!-- ========================================================================= -->
 <xsd:element name="[ELEMENTNAME]" type="[TOKEN]:[TYPENAME]>
<!-- ========================================================================= -->
```

The root element's type definition is defined immediately following the definition of the global root element to provide clear visibility of the root element's type, of which this particular schema is all about.

Example B-7:

```
<!-- ========================================================================= -->
<!-- =====  Root element                                                   ===== -->
<!-- ========================================================================= -->
 <xsd:element name="CrossIndustryInvoice" type="rsm: CrossIndustryInvoiceType">
    <xsd:annotation>
       <xsd:documentation>
          <ccts:UniqueID>CII</ccts:UniqueID>
          <ccts:Acronym>RSM</ccts:Acronym>
          <ccts:Name>CrossIndustryInvoice</ccts:Name>
          <ccts:Version>1.0</ccts:Version>
          <ccts:Definition>A message used as a request for payment, or modification of
             a request for payment, for the supply of goods or services ordered,
             delivered, received, consumed.</ccts:Definition>
          <ccts:BusinessProcessContextValue>PurchaseOrder</ccts:BusinessProcessContextValue>
          <ccts:GeopoliticalOrRegionContextValue>In All
          Contexts</ccts:GeopoliticalOrRegionContextValue>
          <ccts:OfficialConstraintContextValue>None</ccts:OfficialConstraintContextValue>
          <ccts:ProductContextValue>In All Contexts</ccts:ProductContextValue>
          <ccts:IndustryContextValue>In All Contexts</ccts:IndustryContextValue>
          <ccts:BusinessProcessRoleContextValue>In All
          Contexts</ccts:BusinessProcessRoleContextValue>
          <ccts:SupportingRoleContextValue>In All
          Contexts</ccts:SupportingRoleContextValue>
          <ccts:SystemCapabilitiesContextValue>In All
          Contexts</ccts:SystemCapabilitiesContextValue>
       </xsd:documentation>
    </xsd:annotation>
 </xsd:element>
```

Example B-8: Global elements

```
<!-- ================================================================== -->
<!-- ===== Global element                                          ===== -->
<!-- ================================================================== -->
 <xsd:element name="AdministrativeCountrySubDivision"
 type="ram:AdministrativeCountrySubDivisionType"/>
    <xsd:annotation>
       <xsd:documentation>
          <ccts:UniqueID>UN01009362</ccts:UniqueID>
          <ccts:Acronym>ABIE</ccts:Acronym>
          <ccts:DictionaryEntryName>Administrative_ Country Subdivision.
          Details</ccts:DictionaryEntryName>
          <ccts:Version>1.0</ccts:Version>
          <ccts:Definition>An area which is an administrative sub-division within a
          country, such as a state, a county, a canton or a province.</ccts:Definition>
          <ccts:ObjectClassTerm>Country Subdivision<ccts:ObjectClassTerm>
          <ccts:QualifierTerm>Administrative<ccts:QualifierTerm>
        </xsd:documentation>
    </xsd:annotation>
 </xsd:element>
```

B.7 Type Definitions

- Definition of types for Aggregate Business Information Entities in alphabetical order, if applicable.
- Definition of types for Basic Business Information Entities in alphabetical order, if applicable.

Example B-9: Type Definitions

```
<!-- ================================================================== -->
<!-- =====  Type Definitions                                       ===== -->
<!-- ================================================================== -->
<!-- =====  Type Definition: Administrative SubDivision type       ===== -->
<!-- ================================================================== -->
 <xsd:complexType name="AdministrativeCountrySubDivisionType">
    <xsd:annotation>
       <xsd:documentation xml:lang="en">
          <ccts:UniqueID>UN01009362</ccts:UniqueID>
          <ccts:Acronym>ABIE</ccts:Acronym>
          <ccts:DictionaryEntryName>Administrative_ Country Subdivision.
          Details</ccts:DictionaryEntryName>
          <ccts:Version>1.0</ccts:Version>
          <ccts:Definition>An area which is an administrative subdivision within a
             country, such as a state, a county, a canton or a
             province.</ccts:Definition>
          <ccts:ObjectClassTerm>Country Subdivision</ccts:ObjectClassTerm>
          <ccts:ObjectClassTermQualifier>Administrative</ccts:ObjectClassTermQualifier>
       </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
       <xsd:element name="ID" type="udt:IDType ">
          <xsd:annotation>
               <xsd:documentation xml:lang="en">
                <ccts:UniqueID>UN01009363</ccts:UniqueID>
                <ccts:Acronym>BBIE</ccts:Acronym>
                <ccts:DictionaryEntryName>Administrative_ Country Subdivision.
                    Identification. Identifier</ccts:DictionaryEntryName>
                <ccts:Version>1.0</ccts:Version>
                <ccts:Definition>The identifier for this administrative country
                    subdivision.</ccts:Definition>
                <ccts:Cardinality>1</ccts:Cardinality>
                <ccts:ObjectClassTerm>Country Subdivision</ccts:ObjectClassTerm>
                <ccts:ObjectClassTermQualifier>Administrative</ccts:ObjectClassTermQualifi
                er>
                <ccts:PropertyTerm>Identification</ccts:PropertyTerm>
                 <ccts:PrimaryRepresentationTerm>Identifier</ccts:PrimaryRepresen
          tationTerm>
              </xsd:documentation>
          </xsd:annotation>
       </xsd:element>
       <xsd:element name="Description" type="udt:TextType" minOccurs="0
">
          <xsd:annotation>
```

```
                <xsd:documentation xml:lang="en">
                   <ccts:UniqueID>UN01009364</ccts:UniqueID>
                   <ccts:Acronym>BBIE</ccts:Acronym>
                   <ccts:DictionaryEntryName>Administrative_ Country Subdivision. Description.
                   Text</ccts:DictionaryEntryName>
                   <ccts:Version>1.0</ccts:Version>
                   <ccts:Definition>The textual description for this
                       administrative country subdivision.</ccts:Definition>
                   <ccts:Cardinality>0..1</ccts:Cardinality>
                   <ccts:ObjectClassTerm>Country Subdivision</ccts:ObjectClassTerm>
                   <ccts:ObjectClassTermQualifier>Administrative</ccts:ObjectClassTermQualifier>
                   <ccts:PropertyTerm>Description</ccts:PropertyTerm>
                   <ccts:PrimaryRepresentationTerm>Text</ccts:PrimaryRepresentationTerm>
                </xsd:documentation>
             </xsd:annotation>
          </xsd:element>
```

Example B-10: Complete Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ================================================================== -->
<!-- =====  [SCHEMA MODULE TYPE] Schema Module                    ===== -->
<!-- ================================================================== -->
<!--
    Schema agency:              [SCHEMA AGENCY NAME]
    Schema version:             [SCHEMA VERSION] Schema
    date:                       [DATE OF SCHEMA]

    [Code list name:]           [NAME OF CODE LIST] [Code
    list agency:]               [CODE LIST AGENCY] [Code
    list version:]              [VERSION OF CODE LIST]
    [Identifier list name:]     [NAME OF IDENTIFIER LIST]
    [Identifier list agency:]   [IDENTIFIER LIST AGENCY]
    [Identifier list version:]  [VERSION OF IDENTIFIER LIST]


    ECE draws attention to the possibility that the practice or implementation of its
    outputs (which include but are not limited to Recommendations, norms, standards,
    guidelines and technical specifications) may involve the use of a claimed intellectual
    property right.

    Each output is based on the contributions of participants in the UN/CEFACT process, who
    have agreed to waive enforcement of their intellectual property rights pursuant to the
    UN/CEFACT IPR Policy (document ECE/TRADE/C/CEFACT/2010/20/Rev.2 available at
    http://www.unece.org/cefact/cf_docs.html or from the ECE secretariat). ECE takes no
    position concerning the evidence, validity or applicability of any claimed intellectual
    property right or any other right that might be claimed by any third parties related to
    the implementation of its outputs. ECE makes no representation that it has made any
    investigation or effort to evaluate any such rights.

    Implementers of UN/CEFACT outputs are cautioned that any third-party intellectual
    property rights claims related to their use of a UN/CEFACT output will be their
    responsibility and are urged to ensure that their use of UN/CEFACT outputs does not
    infringe on an intellectual property right of a third party.

    ECE does not accept any liability for any possible infringement of a claimed
    intellectual property right or any other right that might be claimed to relate to the
    implementation of any of its outputs.-->

<xsd:schema
targetNamespace="urn:un:unece:uncefact:data:draft:[MODULENAME]:[VERSION"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
… FURTHER NAMESPACES ….
elementFormDefault="qualified" attributeFormDefault="unqualified">
<!-- ================================================================== -->
<!-- =====  Include                                               ===== -->
<!-- ================================================================== -->
<!-- =====  Inclusion of [TYPE OF MODULE]                         ===== -->
<!-- ================================================================== -->
 <xsd:include namespace="…" schemaLocation="…"/>
<!-- ================================================================== -->
<!-- =====  Imports                                               ===== -->
<!-- ================================================================== -->
<!-- =====  Import of [TYPE OF MODULE]                            ===== -->
<!-- ================================================================== -->
<xsd:import namespace="…" schemaLocation="…"/>
<!-- ================================================================== -->
```

```
<!-- =====  Element Declarations                                  ===== -->
<!-- ========================================================== -->
<!-- =====  Root element                                         ===== -->
<!-- ========================================================== -->
 <xsd:element name="[ELEMENTNAME]" type="[TOKEN]:[TYPENAME]>
<!-- ========================================================== -->
<!-- =====  Global Element Declarations                          ===== -->
<!-- ========================================================== -->
 <xsd:element name="[ELEMENTNAME]" type="[TOKEN]:[TYPENAME]>
<!-- ========================================================== -->
<!-- =====  Type Definitions                                     ===== -->
<!-- ========================================================== -->
<!-- =====  Type Definition: [TYPE]                              ===== -->
<!-- ========================================================== -->

 <xsd:complexType name="[TYPENAME]">
    <xsd:restriction base="xsd:token">
          ... see type definition ....
    </xsd:restriction>
 </xsd:complexType>
```

Appendix C BPS Approved Acronyms and Abbreviations

The following constitutes a list of BPS approved acronyms and abbreviations which must be used within tag names when these words are part of the dictionary entry name:

ID – **Identifier**

URI – **Uniform Resource Identifier**

Appendix D Common Use Cases for Code Lists and Identifier Lists

Code lists and identifier lists provide mechanisms for conveying data in a consistent fashion where all parties to the information – originator, sender, receiver, processor – fully understand the purpose, use, and meaning of the data. The UN/CEFACT XML NDRs support flexible use of code and identifier lists. This section details the mechanisms for such use.

D.1 The Use of Code Lists within XML Schemas

The UN/CEFACT XML NDRs allow for five alternative uses for code lists:

- Referencing a predefined standard code list, such as ISO 4217 currency codes as a supplementary component in an unqualified data type, such as udt:AmountType.
- Referencing any code list, standard or proprietary, by providing the required identification as attributes in the unqualified data type udt:CodeType.
- Referencing a predefined code list by declaring a specific qualified data type.
- Choosing or combining values from several code lists.
- Restricting the set of allowed code values from an established code list.

The following Code Use Example Schema is used as the basis for examples that illustrate how to implement each of these alternatives.

Example D-1: Code Use Example Schema

```
<xsd:schema xmlns:ram="urn:un:unece:cefact:ram:0p1"
xmlns:udt="urn:un:unece:uncefact:data:draft:UnqualifiedDataTypeSchemaModule:1"
xmlns:qdt="urn:un:unece:uncefact:data:draft:QualifiedDataTypeSchemaModule:1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:un:unece:cefact:ram:1p1"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <!-- Imports -->
 <xsd:import
namespace="urn:un:unece:uncefact:data:draft:UnqualifiedDataTypeSchemaModule:1"
schemaLocation=" http://www.unece.org/uncefact/data/draft/unqualifieddatatype_1.xsd"/>
 <xsd:import
namespace="urn:un:unece:uncefact:data:draft:QualifiedDataTypeSchemaModule:1"
schemaLocation=" http://www.unece.org/uncefact/data/draft/qualifieddatatype_1.xsd"/>
 <!-- Root element -->
 <xsd:element name="PurchaseOrderRequest" type="ram:PurchaseOrderRequestType"/>
 <!-- Messase type declaration -->
 <xsd:complexType name="PurchaseOrderRequestType">
    <xsd:sequence>
       <xsd:element name="Product" type="ram:ProductType"/>
    </xsd:sequence>
 </xsd:complexType>
 <!-- The below type declaration would normally appear in a separate schema module for all
reusable components (ABIE) but is included here for completeness  -->
 <xsd:complexType name="ProductType">
    <xsd:sequence>
       <xsd:element name="TotalAmount" type="udt:AmountType"/>
       <xsd:element name="TaxCurrencyCode" type="udt:CodeType"/>
       <xsd:element name="ChangeCurrencyCode" type="qdt:CurrencyCodeType"/>
       <xsd:element name="CalculationCurrencyCode"
type="qdt:CalculationCurrencyCodeType"/>
       <xsd:element name=”RestrictedCurrencyCode” type=”qdt:RestrictedCurrencyCodeType”/>
    </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

This schema example imports:

- the schema module of all unqualified data types, such as, **udt:AmountType, udt:CodeType, udt:QuantityType**.
- the schema module of all qualified data types, in which the two specific data types **CurrencyCodeType** and **CalculationCurrencyCodeType** are defined.

Within the **xsd:complexType** of **ProductType**, five local elements are declared. Each of these elements represent one of the five different code list options.

D.1.1 Referencing a Predefined Standard Code List in and Unqualified Data Type

In the Code Use Example Schema, the element **TotalAmount** is declared as:

```
<xsd:element name="TotalAmount" type="udt:AmountType"/>
```

As shown in the element declaration, **TotalAmount** is of the CCTS unqualified data type **udt:AmountType** which has been defined in the UN/CEFACT unqualified data type schema module (See Section 7.6). The **udt:AmountType** declaration in the unqualified schema module is as follows:

```
<xsd:schema
targetNamespace="urn:un:unece:uncefact:data:draft:UnqualifiedDataTypeSchemaModule:1"
xmlns:clm54217="urn:un:unece:uncefact:codelist:draft:5:4217:2001" …
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <!-- ========================================================================= -->
 <!-- ===== Imports                                                       ===== -->
 <!-- ========================================================================= -->
 <!-- ===== Imports of Code Lists                                         ===== -->
 <!-- ========================================================================= -->
 <xsd:import namespace="urn:un:unece:uncefact:codelist:draft:5:4217:2001"
schemaLocation=" http://www.unece.org/uncefact/codelist/draft/5/4217_2001_.xsd "/>
 <!-- ========================================================================= -->
 <!-- ===== Type Definitions                                              ===== -->
 <!-- ========================================================================= -->
 <!-- ===== Amount. Type                                                  ===== -->
 <!-- ========================================================================= -->
 <xsd:complexType name="AmountType">
    <xsd:simpleContent>
       <xsd:extension base="xsd:decimal">
          <xsd:attribute name="currencyID" type="clm54217:CurrencyCodeContentType"
use="required"/>
       </xsd:extension>
    </xsd:simpleContent>
 </xsd:complexType>
```

This **udt:AmountType** has attributes declared that represent the supplementary components defined in CCTS for this data type. These attributes include **currencyCode** for the supplementary component of **Amount. Currency. Code**. This **currencyCode** attribute is declared to be of the **xsd:simpleType clm54217:CurrencyCodeContentType.** The **clm54217:CurrencyCodeContentType** has been declared in the code list schema module for ISO Currency Codes, and the allowed code values for the **currencyCode** attribute have been defined as enumeration facets in the **clm54217:CurrencyCodeContentType** type definition.

An extract of the code list schema module for ISO Currency Codes is as follows:

```
<!-- ========================================================================= -->
<!-- =====   Root Element Declarations                                   ===== -->
<!-- ========================================================================= -->
<xsd:element name="CurrencyCode" type="clm54217:CurrencyCodeContentType"/>
<!-- ========================================================================= -->
<!-- ===== Type Definitions                                              ===== -->
<!-- ========================================================================= -->
<!-- =====  Code List Type Definition: Country Codes                     ===== -->
<!-- ========================================================================= -->
<xsd:simpleType name="CurrencyCodeContentType">
   <xsd:restriction base="xsd:token">
      <xsd:enumeration value="AED">
         <xsd:annotation>
            <xsd:documentation>
               <CodeName>Dirham</CodeName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="AFN">
         <xsd:annotation>
            <xsd:documentation>
```

```
                    <CodeName>Afghani</CodeName>
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
    </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

The **currencyCode** attribute has a fixed value of ISO 4217 Currency Code as defined in CCTS. Thus, only code values from this code list are allowed in a CEFACT conformant instance document. In such an instance document, actual conveyance of a currency code value would be represented as:

```
    <TotalAmount currencyID="AED">3.14</TotalAmount>
```

It should be noted that when using this option, no information about the code list being used is carried in the instance document as this information is already defined in the underlying XML schema.

D.1.2 Referencing Any Code List Using the Unqualified Data Type udt:CodeType

The second element in this example message – **TaxCurrencyCode** – is of the unqualified data type **udt:CodeType**.

```
    <xsd:element name="TaxCurrencyCode" type="udt:CodeType"/>
```

This **udt:CodeType** data type includes a number of supplementary components required in order to uniquely identify the code list to be used for validation.

The **udt:CodeType** is declared in the unqualified schema module as:

```
    <xsd:complexType name="CodeType">
        <xsd:simpleContent>
            <xsd:extension base="xsd:token">
                <xsd:attribute name="listID" type="xsd:token" use="optional"/>
                <xsd:attribute name="listName" type="xsd:string" use="optional"/>
                <xsd:attribute name="listAgencyID" type="xsd:token" use="optional"/>
                <xsd:attribute name="listAgencyName" type="xsd:string" use="optional"/>
                <xsd:attribute name="listVersionID" type="xsd:token" use="optional"/>
                <xsd:attribute name="listURI" type="xsd:anyURI" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
```

When the **udt:CodeType** is used, either the **listURI** (which will point uniquely to the code list) should be used, or a combination of the other attributes should be used. Thus, it is possible to refer to the code list relevant attributes either by the specific attributes for the explicit display of supplementary components, or by the list URI in which the value is based on the namespace name conventions, such as: **urn:un:unece:uncefact:codelist:draft:5:4217:2001**.

The association to the specific namespace must be defined during runtime. In an instance document this element could be represented as:

```
    <TaxCurrencyCode listName="ISO Currency Code" listAgencyName="ISO" listID="ISO 4217"
    listVersionID="2001" listAgencyID="5>AED</TaxCurrencyCode>
```

or

```
    <TaxCurrencyCode
    listURI="urn:un:unece:uncefact:codelist:draft:5:4217:2001">AED</TaxCurrencyCode>
```

It should be noted that when applying this option, validation of code values in the instance document will not be done by the XML parser.

D.1.3 Referencing a Predefined Code List by Declaring a Specific Qualified Data Type

The third element in our example message **ChangeCurrencyCode** is based on the qualified data type **qdt:CurrencyCodeType**.

```
        <xsd:element name="ChangeCurrencyCode" type="qdt:CurrencyCodeType"/>
```

The **qdt:CurrencyCodeType** would be defined in the qualified data type schema module as:

```
<xsd:simpleType name="CurrencyCodeType">
   <xsd:restriction base="clm54217-A:CurrencyCodeContentType"/>
</xsd:simpleType>
```

This means that the value of the **ChangeCurrencyCode** element can only have code values from the identified ISO 4217 code list. In an instance document this element would be represented as:

```
<ChangeCurrencyCode>AED</ChangeCurrencyCode>
```

It should be noted that when using this option no information about the code list to be used is carried in the instance document as this is already defined in the XML schema.

D.1.4 Choosing or Combining Values from Different Code Lists

The fourth option is to chose or combine values from diverse code lists by using either the **xsd:choice** or **xsd:union** elements.

D.1.4.1 Choice

In the Code Use Example Schema, the element **CalculationCurrencyCode** is declared as:

```
<xsd:element name="CalculationCurrencyCode" type="qdt:CalculationCurrencyCodeType"/>
```

The **CalculationCurrencyCode** element is of qualified data type **qdt:CalculationCurrencyCodeType**.

The **qdt:CalculationCurrencyCodeType** is defined in the qualified data type module as:

```
<xsd:complexType name="CalculationCurrencyCodeType">
    <xsd:choice>
      <xsd:element ref="clm54217-N:CurrencyCode"/>
      <xsd:element ref="clm54217-A:CurrencyCode"/>
    </xsd:choice>
</xsd:complexType>
```

The **xsd:choice** element provides a choice of values from either the **clm54217-N:CurrencyCode** or from **clm54217-A:CurrencyCode**. The schema module for **clm54217-A:CurrencyCode** is the same as the one used in section 9.1.1 above. The sample schema module for **clm54217-N:CurrencyCode** is as follows:

Example D-2: Sample clm54217-N:CurrencyCode Schema Module:

```
<!-- ================================================================== -->
<!-- =====  Root Element Declarations                           ===== -->
<!-- ================================================================== -->
<xsd:element name="CurrencyCode" type="clm54217-N:CurrencyCodeContentType"/>
<!-- ===== Type Definitions                                      ===== -->
<!-- ================================================================== -->
<!-- =====  Code List Type Definition: 4217-N Currency Codes     ===== -->
<!-- ================================================================== -->
<xsd:simpleType name="CurrencyCodeContentType">
   <xsd:restriction base="xsd:token">
      <xsd:enumeration value="840">
         <xsd:annotation>
            <xsd:documentation>
               <CodeName>US Dollar</CodeName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="978">
         <xsd:annotation>
            <xsd:documentation>
               <CodeName>Euro</CodeName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:enumeration>
   </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

This **xsd:choice** option allows for the use of code values from different pre-defined code lists in the instance document. The specific code list being used in the instance document will be represented by the namespace prefix (**clm54217-A** or **clm54217-N**) being used for the namespace declaration of the imported code list and for the **CurrencyCode** element:

```
<PurchaseOrder … xmlns:clm54217-N="urn:un:unece:uncefact:codelist:draft:5:4217-N:2001"
… >
    <CalculationCurrencyCode>
       <clm54217-N:CurrencyCode>840</clm54217-N:CurrencyCode>
    </CalculationCurrencyCode>
    …
</PurchaseOrder>
```

The namespace prefix unambiguously identifies to the recipient of the instance from which code list each code value is defined.

D.1.4.2 Union

The **xsd:union** code list approach is similar to that for the **xsd:choice** approach in that multiple code lists are being used. The element declaration in the schema would be identical to that for choice in that the element **CalculationCurrencyCode** is still based on the qualified data type **qdt:CalculationCurrencyCodeType**.

```
<xsd:element name="CalculationCurrencyCode" type="qdt:CalculationCurrencyCodeType"/>
```

The difference is that the **qdt:calculationCurrencyCodeType** would be defined in the qualified data type module using an **xsd:union** element rather than an **xsd:choice** element:

```
<xsd:simpleType name="CalculationCurrencyCodeType">
    <xsd:union memberTypes="clm54217-N:CurrencyCodeContentType clm54217-
                A:CurrencyCodeContentType"/>
</xsd:simpleType>
```

Here the declaration enables the instance to select a choice of values from either the **clm54217-N:CurrencyCodeContentType** or from the **clm54217-A:CurrencyCodeContentType**. The code list schema module for **clm54217-A:CurrencyCodeContentType** is the same as the one used in Section D.1.1 above. The code list schema module for **clm54217-N:CurrencyCodeContentType** is the same as the one used in Section D.1.4.1.

This **xsd:union** option allows for the use of code values from different pre-defined code lists in the instance document. The code lists must be imported once in the XML schema module and must be shown once in the XML instance. The specific code list will be represented by the namespace prefix (**clm54217-A** or **clm54217-N**), but unlike the choice option, the element in the instance document will not have the specific code list token conveyed as the first part of the element name. The recipient of the instance does not know unambiguously in which code list each code value is defined. This is because a reference to the specific code lists comes from different code list schema modules, such as, **clm54217- N** and **clm54217-A**.

In an instance document this element could be represented as:

```
<PurchaseOrder >
    …
    <CalculationCurrencyCode>840</CalculationCurrencyCode>
    …
</PurchaseOrder>
```

The advantage of the **xsd:union** approach is that attributes can make use of these code lists. For example, it may make sense for an implementation to standardize across the board on two currency code lists and have those apply to all of the data types, like **udt:AmountType** and its **currencyID** attribute.

D.1.5 Restricting Allowed Code Values

This option is used when it is desired to reduce the number of allowed code values from an existing code list. For example, a trading partner community may only recognize certain code values from the ISO 4217 Currency Code list. To accomplish this, three options exist:

- Use **xsd:substitutionGroup** to replace the simple type that conveys the enumerated list of codes

- Use **xsd:redefine** to replace the simple type that conveys the enumerated list of codes
- Create a new **xsd:simpleType** with the restricted set of value declarations

The **xsd:substitutionGroup** and **xsd:redefine** features are specifically prohibited in the UN/CEFACT XML NDR due to issues associated with authentication, non-repudiation, ease of understanding, and tool support. Accordingly, when a user community wishes to restrict the allowed code values expressed in an existing schema, a new qualified datatype will be created in the QDT schema module, a new restricted codelist schema module will be created, and a new **xsd:simpleType** will be defined. This new **xsd:simpleType** will contain a complete list of allowed enumerations.

In the example in section D.1.1, a **CurrencyID** element was declared and this element was of the **xsd:simpleType qdt:CurrencyCodeContentType** defined for currency code:

If we wished to restrict the allowed values of **qdt:CurrencyCodeType**, we will have to define a new restricted datatype. For our example, this is the **qdt:RestrictedCurrencyCodeType**. Although in our data model this is a restriction of the **qdt:CurrencyCodeType**, this new datatype's restriction declaration will have a base value of **xsd:token** rather than **CurrencyCodeType** because of XSD limitations. In XSD, enumerations are repeating facets and the nature of **xsd:restriction** is such that the set of facets in a restricted type is the sum of the facets for the original type and the restricted type – actually resulting in an extension rather than restriction. For our example, the **new xsd:simpleType** definition would occur in a new code list schema module:

```
<xsd:simpleType name="RestrictedCurrencyCodeContentType">
   <xsd:restriction base="xsd:token">
      <xsd:enumeration value="AED">
         <xsd:annotation>
            <xsd:documentation>
               <CodeName>Dirham</CodeName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:enumeration>
   </xsd:restriction>
</xsd:simpleType>
```

In the instance documents, allowed values of the element **RestrictedCurrencyCode** are limited to those contained in the restricted code list schema module.

D.2 The Use of Identifier Schemes within XML Schemas

The UN/CEFACT XML NDR allows for five alternative uses for identifier schemes:

- Referencing a predefined standard identifier scheme, such as agency identifiers according to DE 3055, as a supplementary component in an unqualified data type, such as **udt:codeType**.
- Referencing any identifier scheme, standard or proprietary, by providing the required identification as attributes in the unqualified data type **udt:IdentifierType**
- Referencing a predefined identifier scheme by declaring a specific qualified data type
- Choosing or combining values from several identifier schemes
- Restricting allowed identifier values

The rules for identifier schemes are the same as those for code lists, thus the examples found in D.1 also apply to identifier lists

Appendix E Annotation Templates

The following templates define the annotation for each of the schema modules.

```
<!-- Root Schema Documentation -->
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            <ccts:UniqueID></ccts:UniqueID>
            <ccts:Acronym>RSM</ccts:Acronym>
            <ccts:Name></ccts:Name>
            <ccts:Version></ccts:Version>
            <ccts:Definition></ccts:Definition>
            <ccts:BusinessProcessContextValue></ccts:BusinessProcessContextValue>
            <ccts:GeopoliticalOrRegionContextValue></ccts:GeopoliticalOrRegionContextValue>
            <ccts:OfficialConstraintContextValue></ccts:OfficialConstraintContextValue>
            <ccts:ProductContextValue></ccts:ProductContextValue>
            <ccts:IndustryContextValue></ccts:IndustryContextValue>
            <ccts:BusinessProcessRoleContextValue></ccts:BusinessProcessRoleContextValue>
            <ccts:SupportingRoleContextValue></ccts:SupportingRoleContextValue>
            <ccts:SystemCapabilitiesContextValue></ccts:SystemCapabilitiesContextValue>
        </xsd:documentation>
    </xsd:annotation>

<!-- ABIE Documentation -->
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            <ccts:UniqueID></ccts:UniqueID>
            <ccts:Acronym>ABIE</ccts:Acronym>
            <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
            <ccts:Version></ccts:Version>
            <ccts:Definition></ccts:Definition>
            <ccts:ObjectClassTerm></ccts:ObjectClassTerm>
            <ccts:ObjectClassQualifierTerm></ccts:ObjectClassQualifierTerm>
            <ccts:BusinessProcessContextValue></ccts:BusinessProcessContextValue>
            <ccts:GeopoliticalOrRegionContextValue></ccts:GeopoliticalOrRegionContextValue>
            <ccts:OfficialConstraintContextValue></ccts:OfficialConstraintContextVale>
            <ccts:ProductContextValue></ccts:ProductContextValue>
            <ccts:IndustryContextValue></ccts:IndustryContextValue>
            <ccts:BusinessProcessRoleContextValue></ccts:BusinessProcessRoleContextValue>
            <ccts:SupportingRoleContextValue></ccts:SupportingRoleContextValue>
            <ccts:SystemCapabilitiesContextValue></ccts:SystemCapabilitiesContextValue>
            <ccts:UsageRule></ccts:UsageRule>
            <ccts:BusinessTerm></ccts:BusinessTerm>
            <ccts:Example></ccts:Example>
        </xsd:documentation>
    </xsd:annotation>

<!-- BBIE Documentation -->
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            <ccts:UniqueID></ccts:UniqueID>
            <ccts:Acronym>ABIE</ccts:Acronym>
            <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
            <ccts:Version></ccts:Version>
            <ccts:Definition></ccts:Definition>
            <ccts:Cardinality></ccts: Cardinality>
            <ccts:ObjectClassTerm></ccts:ObjectClassTerm>
            <ccts:ObjectClassQualifierTerm></ccts:ObjectClassQualifierTerm>
            <ccts:PropertyTerm></ccts:PropertyTerm>
            <ccts:PropertyQualifierTerm></ccts:PropertyQualifierTerm>
```

```
                <ccts:PrimaryRepresentationTerm></ccts:PrimaryRepresentationTerm
                <ccts:BusinessProcessContextValue></ccts:BusinessProcessContextValue>
                <ccts:GeopoliticalOrRegionContextValue></ccts:GeopoliticalOrRegionContextValue>
                <ccts:OfficialConstraintContextValue></ccts:OfficialConstraintContextValue>
                <ccts:ProductContextValue></ccts:ProductContextValue>
                <ccts:IndustryContextValue></ccts:IndustryContextValue>
                <ccts:BusinessProcessRoleContextValue></ccts:BusinessProcessRoleContextValue>
                <ccts:SupportingRoleContextValue></ccts:SupportingRoleContextValue>
                <ccts:SystemCapabilitiesContextValue></ccts:SystemCapabilitiesContextValue>
                <ccts:UsageRule></ccts:UsageRule>
                <ccts:BusinessTerm></ccts:BusinessTerm>
                <ccts:Example></ccts:Example>
            </xsd:documentation>
        </xsd:annotation>
<!--  ASBIE Documentation -->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:Acronym>ABIE</ccts:Acronym>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:Version></ccts:Version>
                <ccts:Definition></ccts:Definition>
                <ccts:Cardinality></ccts: Cardinality>
                <ccts:ObjectClassTerm></ccts:ObjectClassTerm>
                <ccts:ObjectClassQualifierTerm></ccts:ObjectClassQualifierTerm>
                <ccts:PropertyTerm></ccts:PropertyTerm>
                <ccts:PropertyQualifierTerm></ccts:PropertyQualifierTerm>
                <ccts:AssociatedObjectClassTerm></ccts:AssociatedObjectClassTerm>
                <ccts:AssociatedObjectClassQualifierTerm></ccts: AssociatedObjectClassQualifierTerm>
                <ccts:AssociationType></ccts:AssociationType>
                <ccts:BusinessProcessContexValuet></ccts:BusinessProcessContextValue>
                <ccts:GeopoliticalOrRegionContextValue></ccts:GeopoliticalOrRegionContextValue>
                <ccts:OfficialConstraintContextValue></ccts:OfficialConstraintContextValue>
                <ccts:ProductContextValue></ccts:ProductContextValue>
                <ccts:IndustryContextValue></ccts:IndustryContextValue>
                <ccts:BusinessProcessRoleContextValue></ccts:BusinessProcessRoleContextValue>
                <ccts:SupportingRoleContextValue></ccts:SupportingRoleContextValue>
                <ccts:SystemCapabilitiesContextValue></ccts:SystemCapabilitiesContextValue>
                <ccts:UsageRule></ccts:UsageRule>
                <ccts:BusinessTerm></ccts:BusinessTerm>
                <ccts:Example></ccts:Example>
            </xsd:documentation>
        </xsd:annotation>
<!-- Qualified Data Type Documentation -->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:Acronym>QDT</ccts:Acronym>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:Version></ccts:Version>
                <ccts:Definition></ccts:Definition>
                <ccts:PrimaryRepresentationTerm></ccts:PrimaryRepresentationTerm>
                <ccts:DataTypeQualifierTerm></ccts:DataTypeQualifierTerm>
                <ccts:PrimitiveType></ccts:PrimitiveType>
                <ccts:BusinessProcessContextValue></ccts:BusinessProcessContextValue>
                <ccts:GeopoliticalOrRegionContextValue></ccts:GeopoliticalOrRegionContextValue>
                <ccts:OfficialConstraintContextValue></ccts:OfficialConstraintContextValue>
```

```
                <ccts:ProductContextValue></ccts:ProductContextValue>
                <ccts:IndustryContextValue></ccts:IndustryContextValue>
                <ccts:BusinessProcessRoleContextValue></ccts:BusinessProcessRoleContextValue>
                <ccts:SupportingRoleContextValue></ccts:SupportingRoleContextValue>
                <ccts:SystemCapabilitiesContextValue></ccts:SystemCapabilitiesContextValue>
                <ccts:UsageRule></ccts:UsageRule>
                <ccts:BusinessTerm></ccts:BusinessTerm>
                <ccts:Example></ccts:Example>
            </xsd:documentation>
        </xsd:annotation>
<!-- Qualified Data Type Supplementary Component Documentation-->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:Acronym>SC</ccts:Acronym>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:Definition></ccts:Definition>
                <ccts:Cardinality></ccts: Cardinality>
                <ccts:ObjectClassTerm></ccts:ObjectClassTerm>
                <ccts:PropertyTerm></ccts:PropertyTerm>
                <ccts:PrimaryRepresentationTerm></ccts:PrimaryRepresentationTerm
                <ccts:PrimitiveType></ccts:PrimitiveType>
                <ccts:UsageRule></ccts:UsageRule>
            </xsd:documentation>
        </xsd:annotation>
<!-- Unqualified Data Type Documentation-->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:Acronym>UDT</ccts:Acronym>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:Version></ccts:Version>
                <ccts:Definition></ccts:Definition>
                <ccts:PrimaryRepresentationTerm></ccts:PrimaryRepresentationTerm
                <ccts:PrimitiveType></ccts:PrimitiveType>
                <ccts:UsageRule></ccts:UsageRule>
            </xsd:documentation>
        </xsd:annotation>
<!-- Unqualified Data Type Supplementary Component Documentation-->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:Acronym>SC</ccts:Acronym>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:Definition></ccts:Definition>
                <ccts:Cardinality></ccts: Cardinality>
                <ccts:ObjectClassTerm></ccts:ObjectClassTerm>
                <ccts:PropertyTerm></ccts:PropertyTerm>
                <ccts:PrimaryRepresentationTerm></ccts:PrimaryRepresentationTerm
                <ccts:PrimitiveType></ccts:PrimitiveType>
                <ccts:UsageRule></ccts:UsageRule>
            </xsd:documentation>
        </xsd:annotation>
<!-- Core Component Type Documentation -->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
```

```
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:Acronym>CCT</ccts:Acronym>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:Version></ccts:Version>
                <ccts:Definition></ccts:Definition>
                <ccts:PrimaryRepresentationTerm></ccts:PrimaryRepresentationTerm>
                <ccts:PrimitiveType></ccts:PrimitiveType>
                <ccts:UsageRule></ccts:UsageRule>
            </xsd:documentation>
        </xsd:annotation>
<!-- Core Component Type Supplementary Component Documentation-->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:Acronym>SC</ccts:Acronym>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:Definition></ccts:Definition>
                <ccts:Cardinality></ccts:Cardinality>
                <ccts:ObjectClassTerm></ccts:ObjectClassTerm>
                <ccts:PropertyTerm></ccts:PropertyTerm>
                <ccts:PrimaryRepresentationTerm></ccts:PrimaryRepresentationTerm>
                <ccts:PrimitiveType></ccts:PrimitiveType>
                <ccts:UsageRule></ccts:UsageRule>
            </xsd:documentation>
        </xsd:annotation>
<!-- Code List / Identification Schema Documentation-->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:Name></ccts:Name>
                <ccts:Description></ccts:Description>
            </xsd:documentation>
        </xsd:annotation>
```

Appendix F Naming and Design Rules Checklist

[R1] Conformance shall be determined through adherence to the content of normative sections, rules and definitions.

[R2] All UN/CEFACT XSD Schema design rules MUST be based on the *W3C XML SchemaRecommendations: XML Schema Part 1: Structures and XML Schema Part 2: Data Types*

[R3] All UN/CEFACT XSD Schema and UN/CEFACT conformant XML instance documents MUSTbe based on the W3C suite of technical specifications holding recommendation status.

[R4] UN/CEFACT XSD Schema MUST follow the standard structure defined in Appendix B

[R5] Each element or attribute XML name MUST have one and only one fully qualified XPath(FQXP)

[R6] Element, attribute and type names MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary.

[R7] Lower camel case (LCC) MUST be used for naming attributes

[R8] Upper camel case (UCC) MUST be used for naming elements and types.

[R9] Element, attribute and type names MUST be in singular form unless the concept itself is plural.

[R10] Element, attribute and type names MUST be drawn from the following character set: **`a-z`** and **`A-Z.`** Any special characters such as spaces, underscores, and periods that exist in the underlying Dictionary Entry Names MUST be removed.

[R11] This rule has been combined with [R10].

[R12] XML element, attribute and type names MUST NOT use acronyms, abbreviations, or other word truncations, except those included in the UN/CEFACT controlled vocabulary or listed in Appendix C.

[R13] The acronyms and abbreviations listed in Appendix C MUST always be used.

[R14] Acronyms and abbreviations at the beginning of an attribute declaration MUST appear in all lower case. All other acronym and abbreviation usage in an attribute declaration must appear in upper case.

[R15] Acronyms MUST appear in all upper case for all element declarations and type definitions.

[R16] The schema module file name for modules other than code lists or identifier lists MUST of the form **`<SchemaModuleName>_<Version>.xsd`**, with periods, spaces, or other separators and the words Schema Module removed.

[R17] The schema module file name for code lists and identifier lists, MUST be of the form **`<AgencyName>_<ListName>_<Version>.xsd`**, with periods, spaces, or other separators removed.

[R18] In representing versioning schemes in file names, only the major version should be included.

[R19] A root schema MUST be created for each unique business information payload.

[R20] Each UN/CEFACT root schema module MUST be named <BusinessInformationPayload> Schema Module.

[R21] A root schema MUST NOT replicate reusable constructs available in schema modules capable of being referenced through **`xsd:include`** or **`xsd:import`**.

[R22] UN/CEFACT XSD schema modules MUST either be treated as external schema modules, or as internal schema modules of the root schema.

[R23] All UN/CEFACT internal schema modules MUST be in the same namespace as their corresponding **`rsm:RootSchema.`**

[R24] Each UN/CEFACT internal schema module MUST be named **`<ParentRootSchemaModuleName><InternalSchemaModuleFunction> Schema Module`**

[R25] A Core Component Type schema module MUST be created.

[R26] The **`cct:CoreComponentType`** schema module MUST be named ‘Core Component Type Schema Module’.

[R203] An Unqualified Data Type MUST NOT contain any restriction on their source CCTs other than those defined in CCTS and agreed upon best practices.

[R27] An Unqualified Data Type schema module MUST be created

[R28] The **`udt:UnqualifiedDataType`** schema module MUST be named ’Unqualified Data TypeSchema Module’

[R29] A Qualified Data Type schema module MUST be created.

[R30] The **qdt:QualifiedDataType** schema module MUST be named 'Qualified Data Type Schema Module'.

[R31] A Reusable Aggregate Business Information Entity schema module MUST be created.

[R32] The **ram:ReusableAggregateBusinessInformationEntity** schema module MUST be named 'Reusable Aggregate Business Information Entity Schema Module'.

[R33] Reusable Code List schema modules MUST be created to convey code list enumerations

[R34] The name of each **clm:CodeList** schema module MUST be of the form: **<Code List Agency Identifier|Code List Agency Name><Code List Identification Identifier|Code List Name> - Code List Schema Module**

Where:

Code List Agency Identifier = Identifies the agency that maintains the code list

Code List Agency Name = Agency that maintains the code list

Code List Identification Identifier = Identifies a list of the respective corresponding codes

Code List Name = The name of the code list as assigned by the agency that maintains the code list

[R35] An identifier list schema module MUST be created to convey enumerated values for each identifier list that requires runtime validation.

[R36] The name of each **ids:IdentifierList** schema module MUST be of the form: **<Identifier Scheme Agency Identifier|Identifier Scheme Agency Name><Identifier Scheme Identifier|Identifier Scheme Name> - Identifier List Schema Module**

Where:

Identifier Scheme Agency Identifier = The identification of the agency that maintains the identifier list

Identifier Scheme Agency Name = Agency that maintains the identifier list

Identifier Scheme Identifier = The identification of the identifier list

Identification Scheme Name = Name as assigned by the agency that maintains the identifier list

[R37] Imported schema modules MUST be fully conformant with the UN/CEFACT *XML Naming and Design Rules* Technical Specification and the UN/CEFACT *Core Components Technical Specification.*

[R38] Every UN/CEFACT defined or imported schema module MUST have a namespace declared, using the **xsd:targetNamespace** attribute.

[R39] Every version of a defined or imported schema module other than internal schema modules MUST have its own unique namespace.

[R40] UN/CEFACT published namespace declarations MUST NOT be changed, and its contents MUST NOT be changed unless such change does not break backward compatibility.

[R41] UN/CEFACT namespaces MUST be defined as Uniform Resource Names.

[R42] The names for namespaces MUST have the following structure while the schema is at draft status:

urn:un:unece:uncefact:<schematype>:<status>:<name>:<major>

Where:

schematype = a token identifying the type of schema module:

data|process|codelist|identifierlist|documentation

status = a token identifying the standards status of the schema module: `draft|standard`

name = the name of the schema module (using upper camel case) with periods, spaces, or other separators and the words 'schema module' removed.

major = the major version number. Sequentially assigned, first release starting with the number 1.

[R43] This rule was combined with [R42].

[R44] UN/CEFACT namespace values will only be assigned to UN/CEFACT developed objects.

[R45] The general structure for schema location MUST be:

../<schematype>/<status>/<name>_<major>.<minor>[p <revision>].xsd

Where:

schematype = a token identifying the type of schema module:

data|process|codelist|identifierlist|documentation

status = the status of the schema as: draft|standard

name = the name of the schema module (using upper camel case) with periods, spaces, or other separators and the words 'schema module' removed.
major = the major version number, sequentially assigned, first release starting with the number 1.
minor = the minor version number within a major release, sequentially assigned, first release starting with the number 0.
revision = sequentially assigned alphanumeric character for each revision of a minor release. Only applicable where status = draft.

[R46] Each **xsd:schemaLocation** attribute declaration MUST contain a resolvable URL, and in the case of an absolute path, a persistent URL.

[R47] This rule has been removed.

[R48] The **xsd:schema** version attribute MUST always be declared.

[R49] The **xsd:schema** version attribute MUST use the following template:
<xsd:schema ... version="<major>.<minor>">

[R50] Every schema version namespace declaration MUST have the URI of:
urn:un:unece:uncefact:<schematype>:<status>:<name>:<major>

[R51] Every UN/CEFACT XSD Schema and schema module major version number MUST be a sequentially assigned incremental integer greater then zero.

[R52] Minor versioning MUST be limited to declaring new optional XSD constructs, extending existing XSD constructs, or refinements of an optional nature.

[R53] For UN/CEFACT minor version changes, the name of the schema construct MUST NOT change.

[R54] Changes in minor versions MUST NOT break semantic compatibility with prior versions having the same major version number.

[R55] UN/CEFACT minor version schema MUST incorporate all XML constructs from the immediately preceding major or minor version schema.

[R56] The **xsd:elementFormDefault** attribute MUST be declared and its value set to **qualified**.

[R57] The **xsd:attributeFormDefault** attribute MUST be declared and its value set to **unqualified**.

[R58] The **xsd** prefix MUST be used in all cases when referring to http://www.w3.org/2001/XMLSchema as follows:
xmlns:xsd=http://www.w3.org/2001/XMLSchema.

[R59] **xsd:appInfo** MUST NOT be used.

[R60] **xsd:notation** MUST NOT be used.

[R61] **xsd:wildcard** MUST NOT be used.

[R62] The **xsd:any** element MUST NOT be used.

[R63] The **xsd:any** attribute MUST NOT be used.

[R64] Mixed content MUST NOT be used (excluding documentation).

[R65] **xsd:substitutionGroup** MUST NOT be used.

[R66] **xsd:ID/xsd:IDREF** MUST NOT be used.

[R67] **xsd:key/xsd:keyref** MUST be used for information association.

[R68] The absence of a construct or data MUST NOT carry meaning.

[R69] User declared attributes MUST only be used to convey core component type (CCT) supplementary component information.

[R70] A **xsd:attribute** that represents a supplementary component with variable information MUST be based on the appropriate XSD built-in data type.

[R71] A **xsd:attribute** that represents a supplementary component which represents codes MUST be based on the **xsd:simpleType** of the appropriate code list.

[R72] A **xsd:attribute** that represents a supplementary component which represents identifiers MUST be based on the **xsd:simpleType** of the appropriate identifier scheme.

[R73] The **xsd:nillable** attribute MUST NOT be used.

[R74] Empty elements MUST NOT be used.

[R75] Every BBIE leaf element declaration MUST be of the **udt:UnqualifiedDataType** or **qdt:QualifiedDataType** that represents the source basic business information entity (BBIE) data type.

[R76] The **xsd:all** element MUST NOT be used.

[R77] All type definitions MUST be named.

[R78] Data type definitions with the same semantic meaning MUST NOT have an identical set of facet restrictions.

[R79] **xsd:extension** MUST only be used in the **cct:CoreComponentType** schema module and the **udt:UnqualifiedDataType** schema module. When used it MUST only be used for declaring **xsd:attributes** to accommodate relevant supplementary components.

[R80] When **xsd:restriction** is applied to a **xsd:simpleType** or **xsd:complexType** that represents a data type the derived construct MUST use a different name.

[R81] Each UN/CEFACT defined or declared construct MUST use the **xsd:annotation** element for required CCTS documentation.

[R82] The root schema module MUST be represented by a unique token.

[R83] The **rsm:RootSchema** MUST import the following schema modules:
– **ram:ReusableABIE** Schema Module
– **udt:UnqualifiedDataType** Schema Module
– **qdt:QualifiedDataType** Schema Module

[R84] A **rsm:RootSchema** in one UN/CEFACT namespace that is dependent upon type definitions or element declaration defined in another namespace MUST import the **rsm:RootSchema** from that namespace.

[R85] A **rsm:RootSchema** in one UN/CEFACT namespace that is dependent upon type definitions or element declarations defined in another namespace MUST NOT import Schema Modules from that namespace other than the **rsm:RootSchema.**

[R86] The **rsm:RootSchema** MUST include any internal schema modules that reside in the root schema namespace.

[R87] A single global element known as the root element, representing the business information payload, MUST be declared in a **rsm:RootSchema**.

[R88] The name of the root element MUST be the name of the business information payload with separators and spaces removed.

[R89] The root element declaration must be of **xsd:complexType** that represents the business information payload.

[R90] Root schema MUST define a single **xsd:complexType** that fully describes the business information payload.

[R91] The name of the root schema**xsd:complexType** MUST be the name of the root element with the word ‘Type’ appended.

[R92] The **rsm:RootSchema** root element declaration MAY have a structured set of annotations present in the following pattern:

- UniqueID (required): The identifier that references the business information payload instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be RSM.
- Name (required): The name of the business information payload.
- Version (required): An indication of the evolution over time of a business information payload.
- Definition (required): A brief description of the business information payload.
- BusinessProcessContextValue (required, repetitive): The business process with which this business information is associated.
- GeopoliticalorRegionContextValue (optional, repetitive): The geopolitical/region contexts for this business information payload.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this business information payload.
- ProductContextValue (optional, repetitive): The product context for this business information payload.
- IndustryContextValue (optional, repetitive): The industry context for this business information payload.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this business information payload.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this business information payload.

- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this business information payload.

[R93] All UN/CEFACT internal schema modules MUST be in the same namespace as their corresponding **rsm:RootSchema**.

[R94] The internal schema module MUST be represented by the same token as its **rsm:RootSchema**.

[R95] The Reusable Aggregate Business Information Entity schema module MUST be represented by the token **ram**.

[R96] The **ram:ReusableAggregateBusinessInformationEntity** schema MUST import the following schema modules:
– **udt:UnqualifiedDataType** Schema Module
– **qdt:QualifiedDataType** Schema Module

[R97] For every object class (ABIE) identified in the UN/CEFACT syntax-neutral model, a named **xsd:complexType** MUST be defined.

[R98] The name of the ABIE **xsd:complexType** MUST be the **ccts:DictionaryEntryName** with the spaces and separators removed, approved abbreviations and acronyms applied, and with the 'Details' suffix replaced with 'Type'.

[R99] Every aggregate business information entity (ABIE) **xsd:complexType** definition content model MUST use the **xsd:sequence** and/or **xsd:choice** elements to reflect each property (BBIE or ASBIE) of its class.

[R100] Recursion of **xsd:sequence** and/or **xsd:choice** MUST NOT occur.

[R101] The order and cardinality of the elements within an ABIE **xsd:complexType** MUST be according to the structure of the ABIE as defined in the model.

[R102] For each ABIE, a named **xsd:element** MUST be globally declared.

[R103] The name of the ABIE **xsd:element** MUST be the **ccts:DictionaryEntryName** with the separators and 'Details' suffix removed and approved abbreviations and acronyms applied.

[R104] Every ABIE global element declaration MUST be of the **xsd:complexType** that represents the ABIE.

[R105] For every BBIE identified in an ABIE, a named **xsd:element** MUST be locally declared within the `xsd:complexType` representing that ABIE.

[R106] Each BBIE element name declaration MUST be the property term and qualifiers and the representation term of the basic business information entity (BBIE). Where the word 'identification' is the final word of the property term and the representation term is 'identifier', the term 'identification' MUST be removed. Where the word 'indication' is the final word of the property term and the representation term is 'indicator', the term 'indication' MUST be removed from the property term.

[R107] If the representation term of a BBIE is 'text', 'text' MUST be removed.

[R108] The BBIE element MUST be based on an appropriate data type that is defined in the UN/CEFACT **qdt:QualifiedDataType** or **udt:UnqualifiedDataType** schema modules.

[R109] For every ASBIE whose **ccts:AssociationType** is a composition, a named **xsd:element** MUST be locally declared.

[R110] For each locally declared ASBIE, the element name MUST be the ASBIE property term and qualifier term(s) and the object class term and qualifier term(s) of the associated ABIE.

[R111] For each locally declared ASBIE, the element declaration MUST be of the **sd:complexType** that represents its associated ABIE.

[R112] For every ASBIE whose **ccts:AssociationType** is not a composition, the globally declared element for the associated ABIE must be referenced using **xsd:ref**.

[R113] For every ABIE **xsd:complexType** and `xsd:element` definition a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references an ABIE instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be ABIE.
- DictionaryEntryName (required): The official name of an ABIE.
- Version (required): An indication of the evolution over time of an ABIE instance.

- Definition (required): The semantic meaning of an ABIE.
- ObjectClassTerm (required): The Object Class Term of the ABIE.
- ObjectClassQualifierTerm (optional): Qualifies the Object Class Term of the ABIE.
- BusinessProcessContextValue (optional, repetitive): The business process with which this ABIE is associated.
- GeopoliticalorRegionContexValue (optional, repetitive): The geopolitical/region contexts for this ABIE.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this ABIE.
- ProductContextValue (optional, repetitive): The product context for this ABIE.
- IndustryContextValue (optional, repetitive): The industry context for this ABIE.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this ABIE.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this ABIE.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this ABIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the ABIE.
- BusinessTerm (optional, repetitive): A synonym term under which the ABIE is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of an ABIE.

[R114] This rule was combined with [R113].

[R115] For every BBIE **`xsd:element`** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references a BBIE instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be BBIE.
- DictionaryEntryName (required): The official name of the BBIE.
- VersionID (required): An indication of the evolution over time of a BBIE instance.
- Definition (required): The semantic meaning of the BBIE.
- Cardinality (required): Indication whether the BIE Property represents a not-applicable, optional, required and/or repetitive characteristic of the ABIE.
- ObjectClassTerm (required): The Object Class Term of the parent ABIE.
- ObjectClassQualifierTerm (optional): Qualifies the Object Class Term of the parent ABIE.
- PropertyTerm (required): The Property Term of the BBIE.
- PropertyQualifierTerm (optional): Qualifies the Property Term of the BBIE.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the BBIE.
- BusinessProcessContextValue (optional, repetitive): The business process with which this BBIE is associated.
- GeopoliticalorRegionContextValue (optional, repetitive): The geopolitical/region contexts for this BBIE.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this BBIE.
- ProductContextValue (optional, repetitive): The product context for this BBIE.
- IndustryContextValue (optional, repetitive): The industry context for this BBIE.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this BBIE.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this BBIE.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this BBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to this BBIE.

- BusinessTerm (optional, repetitive): A synonym term under which the BBIE is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of a BBIE.

[R116] For every ASBIE **`xsd:element`** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references an ASBIE instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be ASBIE.
- DictionaryEntryName (required): The official name of the ASBIE.
- Version (required): An indication of the evolution over time of the ASBIE instance.
- Definition (required): The semantic meaning of the ASBIE.
- Cardinality (required): Indication whether the ASBIE Property represents a not-applicable, optional, required and/or repetitive characteristic of the ABIE.
- ObjectClassTerm (required): The Object Class Term of the associating ABIE.
- ObjectClassQualifierTerm (optional): A term that qualifies the Object Class Term of the associating ABIE.
- PropertyTerm (required): The Property Term of the ASBIE.
- PropertyQualifierTerm (Optional): A term that qualifies the Property Term of the ASBIE.
- AssociatedObjectClassTerm (required): The Object Class Term of the associated ABIE.
- AssociatedObjectClassQualifierTerm (optional): Qualifies the Object Class Term of the associated ABIE.
- AssociationType (required): The Association Type of the ASBIE.
- BusinessProcessContextValue (optional, repetitive): The business process with which this ASBIE is associated.
- GeopoliticalorRegionContextValue (optional, repetitive): The geopolitical/region contexts for this ASBIE.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this ASBIE.
- ProductContextValue (optional, repetitive): The product context for this ASBIE.
- IndustryContextValue (optional, repetitive): The industry context for this ASBIE.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this ASBIE.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this ASBIE.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this ASBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the ASBIE.
- BusinessTerm (optional, repetitive): A synonym term under which the ASBIE is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of an ASBIE.

[R117] The core component type (CCT) schema module MUST be represented by the token cct.

[R118] The **`cct:CoreCoreComponentType`** schema module MUST NOT include or import any other schema modules.

[R119] Every core component type MUST be defined as a named **`xsd:complexType`** in the **`cct:CoreComponentType`** schema module.

[R120] The name of each **`xsd:complexType`** based on a core component type MUST be the dictionary entry name of the core component type (CCT), with the separators and spaces removed and approved abbreviations applied.

[R121] Each core component type **xsd:complexType** definition MUST contain one **xsd:simpleContent** element.

[R122] The core component type **xsd:complexType** definition **xsd:simpleContent** element MUST contain one **xsd:extension** element. This **xsd:extension** element must include an XSD based attribute that defines the specific XSD built-in data type required for the CCT content component.

[R123] Within the core component type **xsd:extension** element a **xsd:attribute** MUST be declared for each supplementary component pertaining to that core component type.

[R124] Each core component type supplementary component **xsd:attribute** name MUST be the CCTS supplementary component dictionary entry name with the separators and spaces removed.

[R125] If the object class of the supplementary component dictionary entry name contains the name of the representation term of the parent CCT, the duplicated object class word or words MUST be removed from the supplementary component **xsd:attribute** name.

[R126] If the object class of the supplementary component dictionary entry name contains the term 'identification', the term 'identification' MUST be removed from the supplementary component **xsd:attribute** name.

[R127] If the representation term of the supplementary component dictionary entry name is 'text', the representation term MUST be removed from the supplementary component **xsd:attribute** name.

[R128] The attribute representing the supplementary component MUST be based on the appropriate XSD built-in data type.

[R129] For every core component type **xsd:complexType** definition a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references the Core Component Type instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. . In this case the value will always be CCT.
- DictionaryEntryName (required): The official name of a Core Component Type.
- Version (required): An indication of the evolution over time of a Core Component Type instance.
- Definition (required): The semantic meaning of a Core Component Type.
- PrimaryRepresentationTerm (required): The primary representation term of the Core Component Type.
- PrimitiveType (required): The primitive data type of the Core Component Type.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Core Component Type.

[R130] For every supplementary component **xsd:attribute** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (optional): The identifier that references the Supplementary Component instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of Supplementary Component. In this case the value will always be SC.
- DictionaryEntryName (required): The official name of the Supplementary Component.
- Definition (required): The semantic meaning of the Supplementary Component.
- Cardinality (required): The cardinality of the Supplementary Component.
- ObjectClassTerm (required): The Object Class of the Supplementary Component.
- PropertyTerm (required): The Property Term of the Supplementary Component.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the Supplementary Component.
- PrimitiveType (required): The primitive data type of the Supplementary Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Core Component.

[R131] The Unqualified Data Type schema module namespace MUST be represented by the token **udt**.

[R132] The **udt:UnqualifiedDataType** schema MUST only import the following schema modules: – **ids:IdentifierList** schema modules – **clm:CodeList** schema modules

[R133] An unqualified data type MUST be defined for each approved primary and secondary representation terms identified in the CCTS Permissible Representation Terms table.

[R134] The name of each unqualified data type MUST be the dictionary entry name of the primary or secondary representation term, with the word 'Type' appended, the separators and spaces removed and approved abbreviations applied.

[R135] For every unqualified data type whose supplementary components map directly to the properties of a XSD built-in data type, the unqualified data type MUST be defined as a named **`xsd:simpleType`** in the **`udt:UnqualifiedDataType`** schema module.

[R136] Every unqualified data type **`xsd:simpleType`** MUST contain one **`xsd:restriction`** element. This **`xsd:restriction`** element MUST include an **`xsd:base`** attribute that defines the specific XSD built-in data type required for the content component.

[R137] For every unqualified data type whose supplementary components are not equivalent to the properties of a XSD built-in data type, the unqualified data type MUST be defined as an **`xsd:complexType`** in the **`udt:UnqualifiedDataType`** schema module.

[R138] Every unqualified data type **`xsd:complexType`** definition MUST contain one **`xsd:simpleContent`** element.

[R139] Every unqualified data type **`xsd:complexType xsd:simpleContent`** element MUST contain one **`xsd:extension`** element. This **`xsd:extension`** element must include an **`xsd:base`** attribute that defines the specific XSD built-in data type required for the content component.

[R204] When a combination of the complex and simple types are necessary to support business requirements, the element MUST be declared as an `xsd:complexType` with an `xsd:choice` between elements declared as the two different alternatives.

[R140] Within the unqualified data type **`xsd:complextype xsd:extension`** element an **`xsd:attribute`** MUST be declared for each supplementary component pertaining to the underlying CCT.

[R141] Each supplementary component **`xsd:attribute`** name MUST be the supplementary component name with the separators and spaces removed, and approved abbreviations and acronyms applied.

[R142] If the object class of the supplementary component dictionary entry name contains the name of the representation term, the duplicated object class word or words MUST be removed from the supplementary component **`xsd:attribute`** name.

[R143] If the object class of the supplementary component dictionary entry name contains the term 'identification', the term 'identification' MUST be removed from the supplementary component **`xsd:attribute`** name.

[R144] If the representation term of the supplementary component dictionary entry name is 'text', the representation term MUST be removed from the supplementary component **`xsd:attribute`** name.

[R145] If the representation term of the supplementary component is 'Code' and validation is required, then the attribute representing this supplementary component MUST be based on the defined **`xsd:simpleType`** of the appropriate external imported code list.

[R146] If the representation term of the supplementary component is 'Identifier' and validation is required, then the attribute representing this supplementary component MUST be based on the defined **`xsd:simpleType`** of the appropriate external imported identifier list.

[R147] If the representation term of the supplementary component is other than 'Code' or 'Identifier', then the attribute representing this supplementary component MUST be based on the appropriate XSD built-in data type.

[R148] For every unqualified data type **`xsd:complexType`** or **`xsd:simpleType`** definition a structured set of annotations MAY be present in the following pattern:

- UniqueID (required): The identifier that references an Unqualified Data Type instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be UDT.
- DictionaryEntryName (required): The official name of the Unqualified Data Type.
- Version (required): An indication of the evolution over time of the Unqualified Data Type instance.

- Definition (required): The semantic meaning of the Unqualified Data Type.
- PrimaryRepresentationTerm (required): The primary representation term of the Unqualified Data Type.
- PrimitiveType (required): The primitive data type of the Unqualified Data Type.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Unqualified Data Type.

[R149] For every supplementary component **`xsd:attribute`** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (optional): The identifier that references a Supplementary Component instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be SC.
- Dictionary Entry Name (required): The official name of the Supplementary Component.
- Definition (required): The semantic meaning of the Supplementary Component.
- Cardinality (required): The cardinality of the Supplementary Component.
- ObjectClassTerm (required): The Object Class of the Supplementary Component.
- PropertyTerm (required): The Property Term of the Supplementary Component.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the Supplementary Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Component.

[R150] The Qualified Data Type schema module namespace MUST be represented by the token **`qdt`**.

[R151] The **`qdt:QualifiedDataType`** schema module MUST import the **`udt:UnqualifiedDataType`** **schema module.**

[R205] If a coupled design approach is used, then the qdt:QualifiedDataType schema module MUST import all code list and identifier scheme schemas used in the module..

[R152] Where required to change facets of an existing unqualified data type, a new data type MUST be defined in the **`qdt:QualifiedDataType`** schema module.

[R153] A qualified data type MUST be based on an unqualified or qualified data type and add some semantic and/or technical restriction to the base data type.

[R154] The name of a qualified data type MUST be the name of its base unqualified or qualified data type with separators and spaces removed and with its qualifier term added.

[R155] When a qualified data type is based on an unqualified data type that contains an `xsd:choice` element, then the qualified data type MUST be based on on or the other of the elements, but not both.

[R156] Every qualified data type based on an unqualified or qualified data type **`xsd:complexType`** whose supplementary components do not map directly to the properties of a XSD built-in data type

MUST be defined as a **`xsd:complexType`**
MUST contain one **`xsd:simpleContent`** element
MUST contain one **`xsd:restriction`** element
MUST include the unqualified data type as its **`xsd:base`** attribute.

[R157] Every qualified data type based on an unqualified or qualified data type **`xsd:simpleType`**

MUST contain one **`xsd:restriction`** element
MUST include the unqualified data type as its **`xsd:base`** attribute or if the facet restrictions can be achieved by use of a XSD built-in data type, then that XSD built-in data type may be used as the **`xsd:base`** attribute.

[R158] Every qualified data type based on a single codelist or identifier list **`xsd:simpleType`** MUST contain one **`xsd:restriction`** element or **`xsd:union`** element. When using the **`xsd:restriction`** element, the **`xsd:base`** attribute MUST be set to the code list or identifier list schema module defined simple type with appropriate namespace qualification. When using the **`xsd:union`** element, the **`xsd:member`** type attribute MUST be set to the code list or identifier list schema module defined simple types with appropriate namespace qualification.

[R159] Every qualified data type that has a choice of two or more code lists or identifier lists MUST be defined as an **`xsd:complexType`**

MUST contain the **xsd:choice** element whose content model must consist of element references for the alternative code lists or identifier lists to be included with appropriate namespace qualification.

[R160] The qualified data type **xsd:complexType** definition **xsd:simpleContent** element MUST only restrict attributes declared in its base type, or MUST only restrict facets equivalent to inherited supplementary components.

[R161] Every qualified data type definition MAY contain a structured set of annotations in the following sequence and pattern:

- UniqueID (required): The identifier that references a Qualified Data Type instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be QDT.
- DictionaryEntryName (required): The official name of the Qualified Data Type.
- Version (required): An indication of the evolution over time of the Qualified Data Type instance.
- Definition (required): The semantic meaning of the Qualified Data Type.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the Qualified Data Type.
- DataTypeQualifierTerm (required): A term that qualifies the Representation Term in order to differentiate it from its underlying Unqualified Data Type and other Qualified Data Type.
- PrimitiveType (required): The primitive data type of the Qualified Data Type.
- BusinessProcessContextValue (optional, repetitive): The business process context for this Qualified Data Type is associated.
- GeopoliticalorRegionContextValue (optional, repetitive): The geopolitical/region contexts for this Qualified Data Type.
- OfficialConstraintContextValue (optional, repetitive): The official constraint context for this Qualified Data Type.
- ProductContextValue (optional, repetitive): The product context for this Qualified Data Type.
- IndustryContextValue (optional, repetitive): The industry context for this Qualified Data Type.
- BusinessProcessRoleContextValue (optional, repetitive): The role context for this Qualified Data Type.
- SupportingRoleContextValue (optional, repetitive): The supporting role context for this Qualified Data Type.
- SystemCapabilitiesContextValue (optional, repetitive): The system capabilities context for this Qualified Data Type.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Qualified Data Type.
- Example (optional, repetitive): Example of a possible value of a Qualified Data Type.

[R162] For every supplementary component **xsd:attribute** declaration a structured set of annotations MAY be present in the following pattern:

- UniqueID (optional): The identifier that references a Supplementary Component of a Core Component Type instance in a unique and unambiguous way.
- Acronym (required): The abbreviation of the type of component. In this case the value will always be SC.
- DictionaryEntryName (required): The official name of a Supplementary Component.
- Definition (required): The semantic meaning of a Supplementary Component.
- Cardinality (required): Indication whether the Supplementary Component Property represents a not-applicable, optional, required and/or repetitive characteristic of the Core Component Type.
- ObjectClassTerm (required): The Object Class Term of the associated Supplementary Component.
- PropertyTerm (required): The Property Term of the associated Supplementary Component.
- PrimaryRepresentationTerm (required): The Primary Representation Term of the associated Supplementary Component.
- PrimitiveType (required): The Primitive Type of the associated Supplementary Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Component.

[R163] Each UN/CEFACT maintained code list MUST be defined in its own schema module.

[R164] Internal code list schema MUST NOT duplicate existing external code list schema when the existing ones are available to be imported.

[R165] The namespace names for code list schemas MUST have the following structure:

urn:un:unece:uncefact:codelist:<status>:<Code List Agency Identifier|Code List Agency Name Text>:<Code List Identification. Identifier|Code List Name Text>:<Code List Version. Identifier>

Where:

codelist = this token identifying the schema as a code list

status = a token identifying the standards status of this code list: `draft|standard`

Code List Agency Identifier = identifies the agency that manages a code list. The default agencies used are those from DE 3055 but roles defined in DE 3055 cannot be used.

Code List Agency Name Text = the name of the agency that maintains the code list.

Code List Identification Identifier = identifies a list of the respective corresponding codes. listID is only unique within the agency that manages this code list. Code List Name Text = the name of a list of codes.

Code List Version Identifier = identifies the version of a code list.

[R166] This rule was combined with [R165].

[R167] Each UN/CEFACT maintained code list schema module MUST be represented by a unique token constructed as follows:

clm[Qualified data type name]<Code List Agency Identifier|Code List Agency Name Text><Code List Identification Identifier|Code List Name Text>

with any repeated words eliminated.

[R168] The structure for schema location of code lists MUST be:

../codelist/<status>/<Code List. Agency Identifier|Code List Agency Name Text>/<Code List Identification Identifier|Code List Name Text>_<Code List Version Identifier>.xsd

Where:

schematype = a token identifying the type of schema module: `codelist`

status = the status of the schema as: `draft|standard`

Code List Agency Identifier = identifies the agency that manages a code list. The default agencies used are those from DE 3055. Code List Agency Name Text = the name of the agency that maintains the code list.

Code List Identification Identifier = identifies a list of the respective corresponding codes. listID is only unique within the agency that manages this code list.

Code List Name Text = the name of a list of codes.

Code List Version Identifier = identifies the version of a code list.

[R169] Each **xsd:schemaLocation** attribute declaration of a code list MUST contain a resolvable URL, and if an absolute path is used, it MUST also be persistent.

[R170] This rule has been removed.

[R171] Code List schema modules MUST not import or include any other schema modules.

[R172] Within each code list module one, and only one, named **xsd:simpleType** MUST be defined for the content component.

[R173] The name of the **xsd:simpleType** MUST be the name of code list root element with the word 'ContentType' appended.

[R174] The **xsd:restriction** element base attribute value MUST be set to **xsd:token**.

[R175] Each code in the code list MUST be expressed as an **xsd:enumeration**, where the **xsd:value** for the enumeration is the actual code value.

[R176] For each code list a single root element MUST be globally declared.

[R177] The name of the code list root element MUST be the name of the code list following the naming rules as defined in section 5.3.

[R178] The code list root element MUST be of a type representing the actual list of code values.

[R179] Each code list **xsd:enumeration** MAY contain a structured set of annotations in the following sequence and pattern:

- Name (required): The name of the code.
- Description (optional): Descriptive information concerning the code.

[R180] Internal identifier lists schema MUST NOT duplicate existing external identifier list schema when the existing ones are available to be imported.

[R181] Each UN/CEFACT maintained identifier list MUST be defined in its own schema module.

[R182] The names for namespaces MUST have the following structure:

urn:un:unece:uncefact:identifierlist:<status>:<Identifier Scheme. Agency Identifier|Identifier Scheme Agency Name Text>:<Identifier Scheme Identifier|Identifier Scheme Name Text>:<Identifier Scheme Version Identifier>

Where:

status = the token identifying the publication status of this identifier scheme schema = draft|standard

identifierlist = this token identifying the schema as an identifier scheme

Identifier Scheme Agency Identifier = the identification of the agency that maintains the identification scheme.

Identifier Scheme Agency Name. Text = the name of the agency that maintains the identification list.

Identifier Scheme Identifier = the identification of the identification scheme.

Identifier Scheme Name. Text = the name of the identification scheme.

Identifier Scheme Version. Identifier = the version of the identification scheme.

[R183] This rule was combined with [R182].

[R184] Each UN/CEFACT maintained identifier list schema module MUST be represented by a unique token constructed as follows:

ids[Qualified data type name]<Identification Scheme Agency Identifier><Identification Scheme Identifier>

with any repeated words eliminated.

[R185] The structure for schema location of identifier lists MUST be:

[../identifierlist/<status>/<Identifier **Scheme Agency Identifier|Identifier Scheme Agency Name Text>/< Identifier Scheme Identifier|Identifier Scheme Name Text>_< Identifier Scheme Version Identifier>.xsd**

Where:

schematype = a token identifying the type of schema module: identifierlist

status = the status of the schema as: **draft|standard**

Identifier Scheme. Agency Identifier = the identification of the agency that maintains the identification scheme.

Identifier Scheme. Agency Name. Text = the name of the agency that maintains the identification scheme.

Identifier Scheme. Identifier = the identification of the identification scheme.

Identifier Scheme. Name. Text = the name of the identification scheme.

Identifier Scheme. Version. Identifier = the version of the identification scheme.

[R186] Each **xsd:schemaLocation** attribute declaration of an identifier list schema MUST contain a resolvable URL, and if an absolute path is used, it MUST also be persistent.

[R187] This rule has been removed.

[R188] Identifier list schema modules MUST NOT import or include any other schema modules.

[R189] Within each identifier list schema module one, and only one, named **xsd:simpleType** MUST be defined for the content component.

[R190] The name of the **xsd:simpleType** MUST be the name of the identifier list root element with the word 'ContentType' appended.

[R191] The **xsd:restriction** element base attribute value MUST be set to **xsd:token**.

[R192] Each identifier in the identifier list MUST be expressed as an **xsd:enumeration**, where the **xsd:value** for the enumeration is the actual identifier value.

[R193] Facets other than **xsd:enumeration** MUST NOT be used in the identifier list schema module.

[R194] For each identifier list a single root element MUST be globally declared.

[R195] The name of the identifier list root element MUST be the name of the identifier list following the naming rules as defined in section 5.3.

[R196] The identifier list root element MUST be of a type representing the actual list of identifier values.

[R197] Each **`xsd:enumeration`** MAY contain a structured set of annotations in the following sequence and pattern:

- Name (required): The name of the identifier.
- Description (optional): Descriptive information concerning the identifier.

[R198] All UN/CEFACT XML MUST be instantiated using UTF. UTF-8 should be used as the preferred encoding. If UTF-8 is not used, UTF-16 MUST be used.

[R199] The **`xsi`** prefix MUST be used where appropriate for referencing **`xsd:schemaLocation`** and **`xsd:noNamespaceLocation`** attributes in instance documents.

[R200] UN/CEFACT conformant instance documents MUST NOT contain an element devoid of content.

[R201] The **`xsi:nil`** attribute MUST NOT appear in any conforming instance.

[R202] The **`xsi:type`** attribute MUST NOT be used

Appendix G: Glossary

Aggregate Business Information Entity (ABIE) – A collection of related pieces of business information that together convey a distinct business meaning in a specific *Business Context*. Expressed in modelling terms, it is the representation of an *Object Class*, in a specific *Business Context*.

Aggregate Core Component - (*ACC*) – A collection of related pieces of business information that together convey a distinct business meaning, independent of any specific *Business Context*. Expressed in modelling terms, it is the representation of an *Object Class*, independent of any specific *Business Context*.

Aggregation – An *Aggregation* is a special form of *Association* that specifies a whole-part relationship between the aggregate (whole) and a component part.

Association Business Information Entity (ASBIE) - A *Business Information Entity* that represents a complex business characteristic of a specific *Object Class* in a specific *Business Context*. It has a unique *Business Semantic* definition. An *Association Business Information Entity* represents an *Association Business Information Entity Property* and is therefore associated to an *Aggregate Business Information Entity*, which describes its structure. An *Association Business Information Entity* is derived from an *Association Core Component*.

Association Business Information Entity Property - A *Business Information Entity Property* for which the permissible values are expressed as a complex structure, represented by an *Aggregate Business Information Entity*.

Association Core Component (ASCC) - A *Core Component* which constitutes a complex business characteristic of a specific *Aggregate Core Component* that represents an *Object Class*. It has a unique *Business Semantic* definition. An *Association Core Component* represents an *Association Core Component Property* and is associated to an *Aggregate Core Component*, which describes its structure.

Association Core Component Property – A *Core Component Property* for which the permissible values are expressed as a complex structure, represented by an *Aggregate Core Component.*

Association Type – The association type of the *Association Business Information Entity.*

Attribute – A named value or relationship that exists for some or all instances of some entity and is directly associated with that instance.

Basic Business Information Entity (BBIE) – A *Business Information Entity* that represents a singular business characteristic of a specific *Object Class* in a specific *Business Context*. It has a unique *Business Semantic* definition. A *Basic Business Information Entity* represents a *Basic Business Information Entity Property* and is therefore linked to a *Data Type*, which describes it values. A Basic *Business Information Entity* is derived from a *Basic Core Component*.

Basic Business Information Entity Property – A *Business Information Entity Property* for which the permissible values are expressed by simple values, represented by a *Data Type*.

Basic Core Component (BCC) – A *Core Component* which constitutes a singular business characteristic of a specific *Aggregate Core Component* that represents a *Object Class*. It has a unique *Business Semantic* definition. A *Basic Core Component* represents a *Basic Core Component Property* and is therefore of a *Data Type*, which defines its set of values. *Basic Core Components* function as the properties of *Aggregate Core Components*.

Basic Core Component (CC) Property – A *Core Component Property* for which the permissible values are expressed by simple values, represented by a *Data Type*.

Business Context – The formal description of a specific business circumstance as identified by the values of a set of *Context Categories*, allowing different business circumstances to be uniquely distinguished.

Business Information Entity (BIE) – A piece of business data or a group of pieces of business data with a unique *Business Semantic* definition. A *Business Information Entity* can be a Basic *Business Information Entity* (BBIE), an *Association Business Information Entity* (ASBIE), or an *Aggregate Business Information Entity* (ABIE).

Business Information Entity (BIE) Property – A business characteristic belonging to the *Object Class* in its specific *Business Context* that is represented by an *Aggregate Business Information Entity*.

Business Libraries – A collection of approved process models specific to a line of business (e.g., shipping, insurance).

Business Process – The *Business Process* as described using the UN/CEFACT Modelling Methodology.

Business Process Context – The *Business Process* name(s) as described using an appropriate list of relevant business processes.

Business Process Role Context – The actor(s) conducting a particular Business Process.

Business Semantic(s) – A precise meaning of words from a business perspective.

Business Term – This is a synonym under which the *Core Component* or *Business Information Entity* is commonly known and used in the business. A *Core Component* or *Business Information Entity* may have several *Business Terms* or synonyms.

Cardinality – An indication whether a characteristic is optional, mandatory and/or repetitive.

CCL – see *Core Component Library*.

Classification Scheme – This is an officially supported scheme to describe a given *Context Category*.

Composition – A form of aggregation which requires that a part instance be included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. *Composition* may be recursive.

Content Component – Defines the *Primitive Type* used to express the content of a *Core Component Type*.

Content Component Restrictions – The formal definition of a format restriction that applies to the possible values of a *Content Component*.

Context – Defines the circumstances in which a *Business Process* may be used. This is specified by a set of *Context Categories* known as *Business Context.*

Context Category – A group of one or more related values used to express a characteristic of a business circumstance.

Controlled Vocabulary – A supplemental vocabulary used to define potentially ambiguous words or *Business Terms*. This ensures that every word within any of the core component names and definitions is used consistently, unambiguously and accurately.

Core Component (CC) – A building block for the creation of a semantically correct and meaningful information exchange package. It contains only the information pieces necessary to describe a specific concept.

Core Component Library –The *Core Component Library* will contain all the *Core Component Types, Basic Core Components, Aggregate Core Components, Basic Business Information Entities, Aggregate Business Information Entities, and Data Types*.

Core Component Property – A business characteristic belonging to the *Object Class* represented by an

Aggregate Core Component.

Core Component Type (CCT) – A *Core Component*, which consists of one and only one *Content Component*, that carries the actual content plus one or more *Supplementary Components* giving an essential extra definition to the *Content Component*. *Core Component Types* do not have *Business Semantics*.

Data Type – Defines the set of valid values that can be used for a particular *Basic Core Component Property* or *Basic Business Information Entity Property*. It is defined by specifying restrictions on the *Core Component Type* that forms the basis of the *Data Type*.

Decoupling – The term "Decoupling" refers to decoupling of unqualified data type schema from the data type catalogue (CCT) and/or decoupling a particular qualified data type from a set of value enumerations.

Definition – This is the unique semantic meaning of a *Core Component, Business Information Entity, Business Context* or *Data Type*.

Dictionary Entry Name – This is the unique official name of a *Core Component, Business Information Entity, Business Context* or *Data Type* in the library.

Geopolitical Context – A combination of political and geographic factors influencing or delineating a country or region.

Industry Classification Context – Semantic influences related to the industry or industries of the trading partners (e.g., product identification schemes used in different industries).

Information Entity – A reusable semantic building block for the exchange of business-related information.

Lower-Camel-Case (LCC) – a style that capitalizes the first character of each word except the first word and compounds the name.

Naming Convention – The set of rules that together comprise how the *Dictionary Entry Name* for *Core Components* and *Business Information Entities* are constructed.

Object Class – The logical data grouping (in a logical data model) to which a data element belongs (ISO11179). The *Object Class* is the part of a *Core Component*'s *Dictionary Entry Name* that represents an activity or object in a specific *Context*.

Object Class Term – A component of the name of a *Core Component* or *Business Information Entity* which represents the *Object Class* to which it belongs.

Official Constraints Context – Legal and governmental influences on semantics (e.g. hazardous materials information required by law when shipping goods).

Primitive Type – Used for the representation of a value. Possible values are String, Decimal, Integer, Boolean, Date and Binary.

Product Classification Context – Factors influencing semantics that are the result of the goods or services being exchanged, handled, or paid for, etc. (e.g. the buying of consulting services as opposed to materials)

Property – A peculiarity common to all members of an *Object Class*.

Property Term – A semantically meaningful name for the characteristic of the *Object Class* that is represented by the *Core Component Property*. It shall serve as basis for the *Dictionary Entry Name* of the *Basic* and *Association Core Components* that represents this *Core Component Property*.

Qualifier Term – A word or group of words that help define and differentiate an item (e.g. a *Business Information Entity* or *a Data Type)* from its associated items (e.g. from a *Core Component*, a *Core Component Type,* another *Business Information Entity* or another *Data Type*).

Registry Class – The formal definition of all the information necessary to be recorded in the Registry about a *Core Component*, a *Business Information Entity*, a *Data Type* or a *Business Context*.

Representation Term – The type of valid values for a *Basic Core Component* or *Business Information Entity*.

Supplementary Component – Gives additional meaning to the *Content Component* in the *Core Component Type*.

Supplementary Component Restrictions – The formal definition of a format restriction that applies to the possible values of a *Supplementary Component*.

Supporting Role Context – Semantic influences related to non-partner roles (e.g., data required by a third-party shipper in an order response going from seller to buyer.)

Syntax Binding – The process of expressing a *Business Information Entity* in a specific syntax.

System Capabilities Context – This *Context category* exists to capture the limitations of systems (e.g. an existing back office can only support an address in a certain form).

UMM Information Entity – A *UMM Information Entity* realizes structured business information that is exchanged by partner roles performing activities in a business transaction. Information entities include or reference other information entities through associations."

Unique Identifier – The identifier that references a *Registry Class* instance in a universally unique and unambiguous way.

Upper-Camel-Case (UCC) – a style that capitalizes the first character of each word and compounds the name.

Usage Rules – *Usage Rules* describe how and/or when to use the *Registry Class.*

User Community – A *User Community* is a group of practitioners, with a publicised contact address, who may define *Context* profiles relevant to their area of business. Users within the community do not create, define or manage their individual *Context* needs but conform to the community's standard. Such a community should liase closely with other communities and with general standards-making bodies to avoid overlapping work. A community may be as small as two consenting organisations.

Version – An indication of the evolution over time of an instance of a *Core Component*, *Data Type*, *Business Context*, or *Business Information Entity*.

XML schema – A Recommendation of the World Wide Web Consortium (W3C), which specifies how to formally describe the elements in an Extensible Markup Language (XML) document. This description can be used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed.

Intellectual Property Disclaimer

ECE draws attention to the possibility that the practice or implementation of its outputs (which include but are not limited to Recommendations, norms, standards, guidelines and technical specifications) may involve the use of a claimed intellectual property right.

Each output is based on the contributions of participants in the UN/CEFACT process, who have agreed to waive enforcement of their intellectual property rights pursuant to the UN/CEFACT IPR Policy (document ECE/TRADE/C/CEFACT/2010/20/Rev.2 available at http://www.unece.org/cefact/cf_docs.html or from the ECE secretariat). ECE takes no position concerning the evidence, validity or applicability of any claimed intellectual property right or any other right that might be claimed by any third parties related to the implementation of its outputs. ECE makes no representation that it has made any investigation or effort to evaluate any such rights.

Implementers of UN/CEFACT outputs are cautioned that any third-party intellectual property rights claims related to their use of a UN/CEFACT output will be their responsibility and are urged to ensure that their use of UN/CEFACT outputs does not infringe on an intellectual property right of a third party.

ECE does not accept any liability for any possible infringement of a claimed intellectual property right or any other right that might be claimed to relate to the implementation of any of its outputs.