XML Naming and Design Rules

Draft 1.1a, 16 February 2005

1 Status of this Documents

This UN/CEFACT Technical Specification has been developed in accordance with the UN/CEFACT/TRADE/22 Open Development Process (ODP) for Technical Specifications. It has been approved by the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) Applied Techniques Group (ATG) for implementation verification in accordance with Step 6 of the ODP.

Distribution of this document is unlimited.

The document formatting is based on the Internet Society's Standard RFC format.

This version:

XML Naming and Design Rules, Version 1.1a of 16 February 2005

Previous version:

NamingAndDesignRules_1.1.doc

Document identifier:

NamingAndDesignRules_1.1a.doc

Location:

http://www.disa.org/cefact-groups/atg/downloads/index.cfm

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:

Mark Crawford	LMI

Editors:

Gunther Stuhec	SAP AG
Paula Heilig	Worldspan
Margaret Pemberton	Diskray
Garret Minakawa	Oracle/OAGI

Contributors:

Hisanao Sugamata	ECOM-Japan
Frank Lin	GCOM
K.K. Suen	EAN Hong Kong
Luc Mouchot	CNAM-TS
Thomas Bikeev	EAN.UCC
Jostein Frømyr	EDISYS
Sue Probert	SEPIA eb
Alain Dechamps	CEN
Michael Dill	GEFEG

2.1 Acknowledgements

The UN/CEFACT - *XML Naming and Design Rules* were developed in close coordination with other XML standards efforts. In particular, the *OASIS Universal Business Language Technical Committee Naming and Design Rules* were instrumental in developing this document. Additionally, contributions were also received from:

SWIFT
U.S. Department of the Navy
U.S. Environmental Protection Agency
U.S. Federal CIO Council XML Working Group
OpenTravel Alliance
Australia Electricity & Gas Industry
CIDX
EAN/UCC
European Transmission System Operators
PIDX

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 re-use 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 information exchange.

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).* The CCTS specification has subsequently been published as ISO/TS 15000-5 ebCCTS ebXML *Electronic Business Extensible Mark-up Language, Part 5: ebCCTS ebXML Core Components Technical Specification, Version 2.01 (2003-11-15).*

4.2 Audience

The primary audience for this UN/CEFACT – *XML Naming and Design Rules* Technical Specification are members of the UN/CEFACT Applied Technologies Group 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.

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.

Section 5 provides information on the guiding principles applied in developing this specification as well as its dependency and relationship to CCTS. Furthermore, this section describes the approach taken to modularity in order to maximize the re-use of business information expressed as XML schema components and the general naming conventions applied. (Normative)

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 ATG Approved Acronyms and Abbreviations (Normative)

Appendix D Core Component Schema Module (Normative)

Appendix E Unqualified Data Type Schema Module (Normative)

Appendix F Annotation Templates (Informative)

Appendix G Mapping of CCTS Representation Terms to CCT and UDT Data Types (Informative)

Appendix H Naming and Design Rules List (Normative)

Appendix I Glossary (Informative)

Appendix J Qualified Data Type Schema Module (Normative)

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.

[R 1]	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:

- Relationship to UMM – UN/CEFACT 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.

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

- ebXML Use – UN/CEFACT XSD Schema and instance documents shall be straightforwardly usable within the ebXML framework and compatible with other frameworks to the maximum extent practicable.
- 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 design rules will be based on Technical Specifications holding the equivalent of W3C recommended status.
- Schema Specification – UN/CEFACT XML 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 aren't precluded.
- Relationship to Other Namespaces - UN/CEFACT XML design rules will be cautious about making dependencies on other namespaces.
- Legacy formats - UN/CEFACT XML design rules are not responsible for sustaining legacy formats.
- Messages must express semantics fully in schema and not rely on well-formedness.

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 Strategy
- Namespace Scheme
- 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.

[R 2]	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 status. Only those W3C specifications holding recommendation status are guaranteed by the W3C to be stable specifications.

[R 3]	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.

[R 4]	UN/CEFACT XSD Schema MUST follow the standard structure defined in Appendix B.

5.2 Relationship to the CCTS

All UN/CEFACT business information 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 2.0 (Second Edition),* UN/CEFACT, 15 November 2003

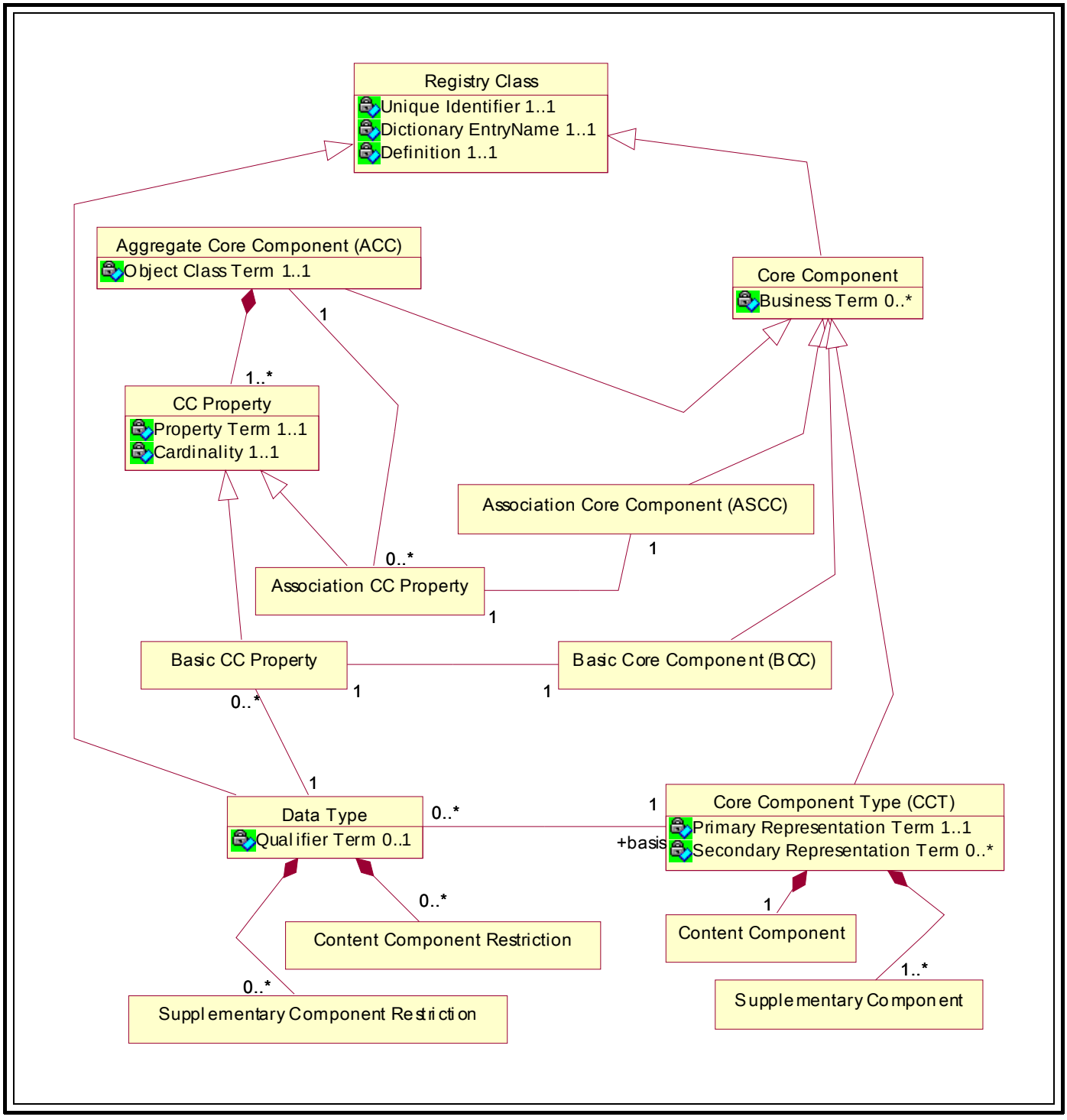


Figure 5-1 Core Component Metamodel

5.2.2 Business Information Entities

In the CCTS model, context neutral core components are instantiated as context specific components for message assembly and model harmonization. The context specific components are defined as Business Information Entities **(ccts:BusinessInformationEntities)**. See CCTS Section 6.2 for a detailed discussion of the ebXML context mechanism.[3] Context specific **ccts:BusinessInformationEntities** are defined in CCTS as "A piece of business data or a group

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

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.

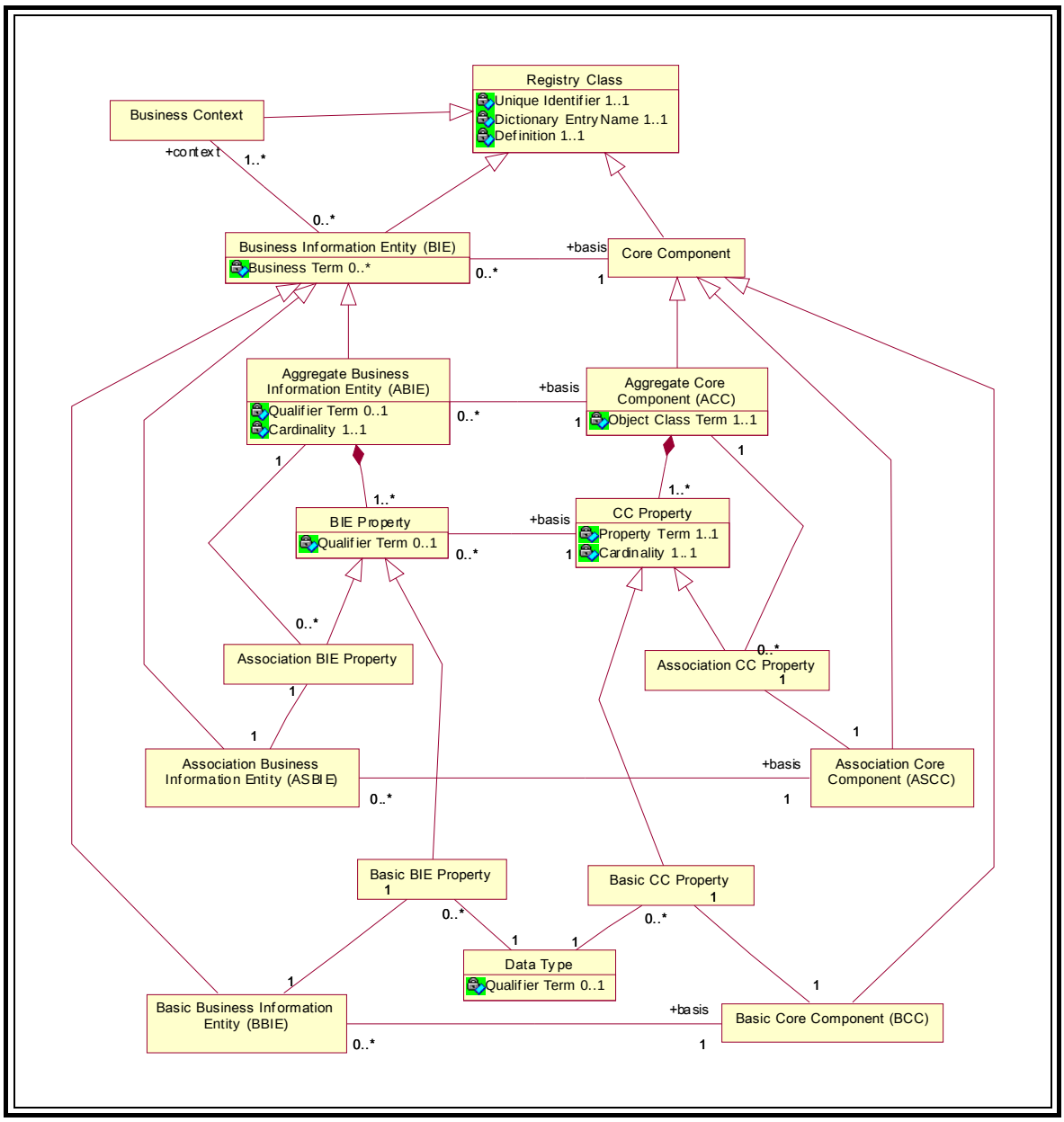


Figure 5-2 Context Specific Business Information Entity Metamodel

5.2.3 The XML Constructs

UN/CEFACT XML design rules will be 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, Business Information Entities), and the other boxes reflect XSD constructs (**`xsd:types, xsd:elements, xsd:attributes`**). The relationships follow the following basic principles:

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

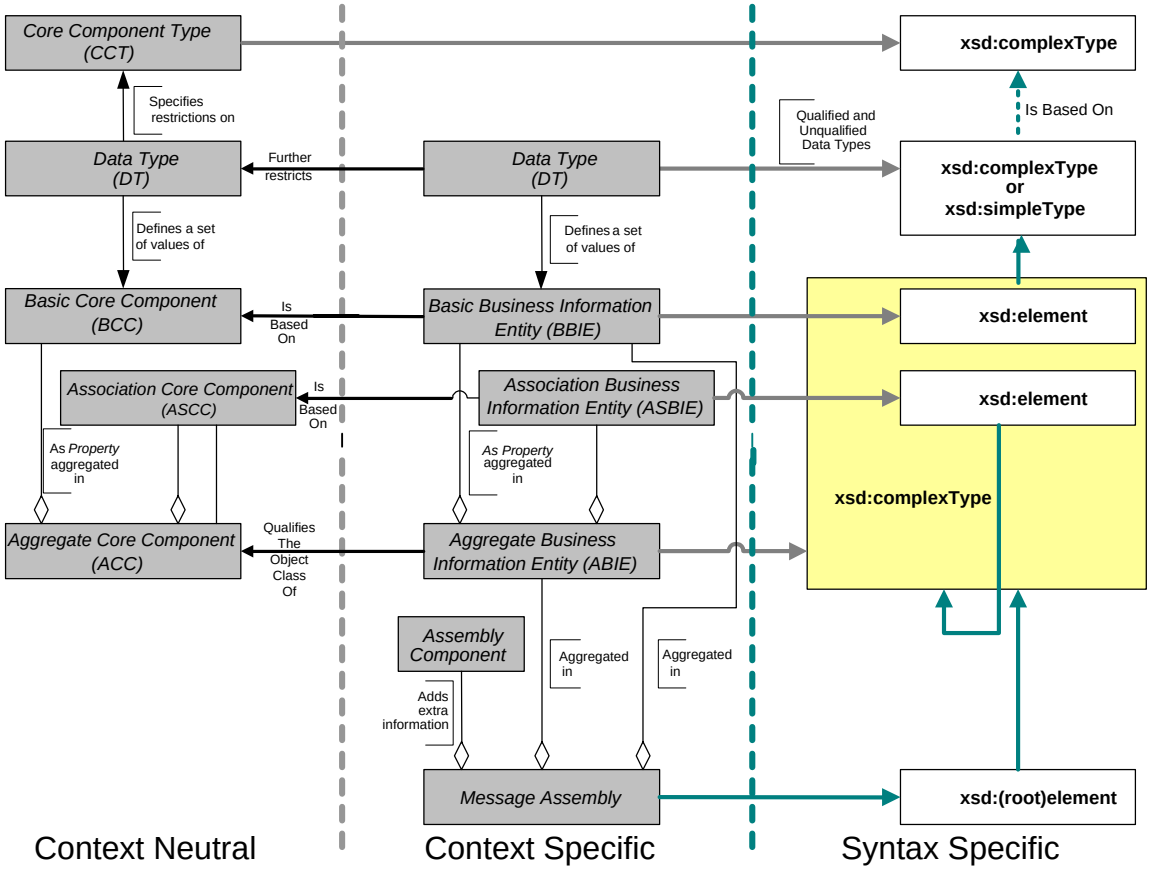


Figure 5-3 Relationship between CCTS and XSD Artefacts in UN/CEFACT XSD Schema

- The 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`**.
- An ASBIE is declared as a local element within the **`xsd:complexType`** representing the associating 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 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:restiction`** because whereas all CCTs are defined as complex types with attributes representing their supplementary components, in several cases we leverage built-in **`xsd:simpleType`** whose facets correspond to the supplementary components. 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 CCTS and UN/CEFACT Modelling Methodology (UMM) process modelling and data analysis. The UN/CEFACT library contains fully conformant CCTS dictionary entry names as well as truncated XML element names developed in conformance with the naming constraint rules specified below. 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.

[R 5]	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

```
Address/Coordinate/Latitude Measure
Organisation/Location/Name
```

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.

[R 6]	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.

[R 7]	Lower camel case (LCC) MUST be used for naming attributes.

Example 5-2: Attribute

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

[R 8]	Upper camel case (UCC) MUST be used for naming elements and types.

Example 5-3: Element

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

Example 5-4: Type

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

[R 9]	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="GoodsQuantity" ...>
```

Not Allowed - Plural:

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

[R 10] Element, attribute and type names MUST be drawn from the following character set: **a-z** and **A-Z**.

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.

[R 11] XML element, attribute and type names constructed from dictionary entry names MUST NOT include periods, spaces, or other separators; or characters not allowed by W3C XML 1.0 for XML names.

Example 5-7: Spaces in Name

Not Allowed

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

[R 12] 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.

[R 13] 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.

[R 14] 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>
```

5.4 Reusability Scheme

UN/CEFACT is committed to transitioning to an object based approach for its process models and core components 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.

[R 15]	All element declarations for BBIEs and ASBIEs MUST be locally declared within the parent ABIE type.

5.4.1 Element Naming Conventions

The fully qualified XPath anchors the use of a construct to a particular location in a business message. 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. As discussed in Section 5.4 above, the dictionary must be carefully controlled to overcome the limitations in control inherent in a local element approach.

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 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 5-4 portrays the CEFACT modularity model. We categorize our modules into message assembly and external schema. The message assembly 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 reside in their own namespace. Dependencies exist amongst the various modules as shown in the figure.

The root schema always includes any internal schema residing in its namespace. It also always imports the ABIE reusable schema, 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 may include other internal schema modules from its own namespace, and may reference – through the root schema – other root schema 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 always imports the unqualified data type schema 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.

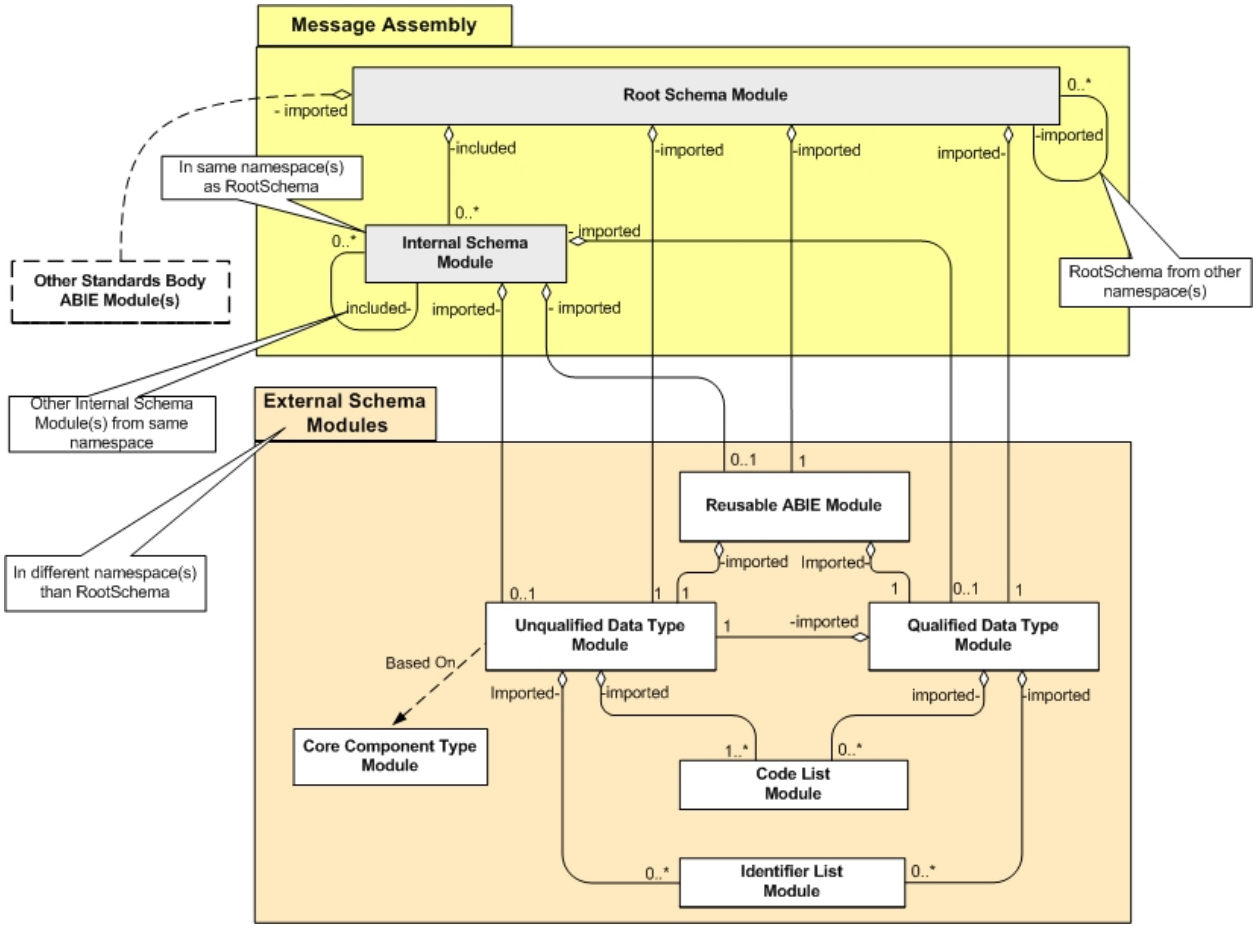


Figure 5-4 UN/CEFACT XSD Schema Modularity Scheme

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
RootSchema	rsm
CCTS CCT	cct
UN/CEFACT Reusable Aggregate Business Information Entity	ram
UN/CEFACT Unqualified Data Type	udt
UN/CEFACT Qualified Data Type	qdt
Code List	clm
Identifier List	ids

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.

[R 16] A root schema MUST be created for each unique business information exchange.

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.

[R 17]	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.

[R 18]	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

Not all ABIEs will be suitable for widespread reuse. Some may be limited to a specific business function or information exchange. These ABIEs will be defined as **`xsd:complexType`** in an internal schema module rather than in the reusable ABIE module, (See Section 5.5.3.4 below). UN/CEFACT XSD Schema may have zero or more internal 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 modules can be reused to the maximum extent possible.

[R 19]	All UN/CEFACT internal schema modules MUST be in the same namespace as their corresponding **`rsm:RootSchema.`**

UN/CEFACT internal schema modules will necessarily 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.

[R 20]	Each UN/CEFACT internal schema module MUST be named <ParentRootSchemaModuleName><InternalSchemaModuleFunction> Schema Module

Example 5-9: UN/CEFACT internal schema module name

```
TravelReservationRequestFlightInformation Schema Module
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:

- Core Component Type
- 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 5-5.

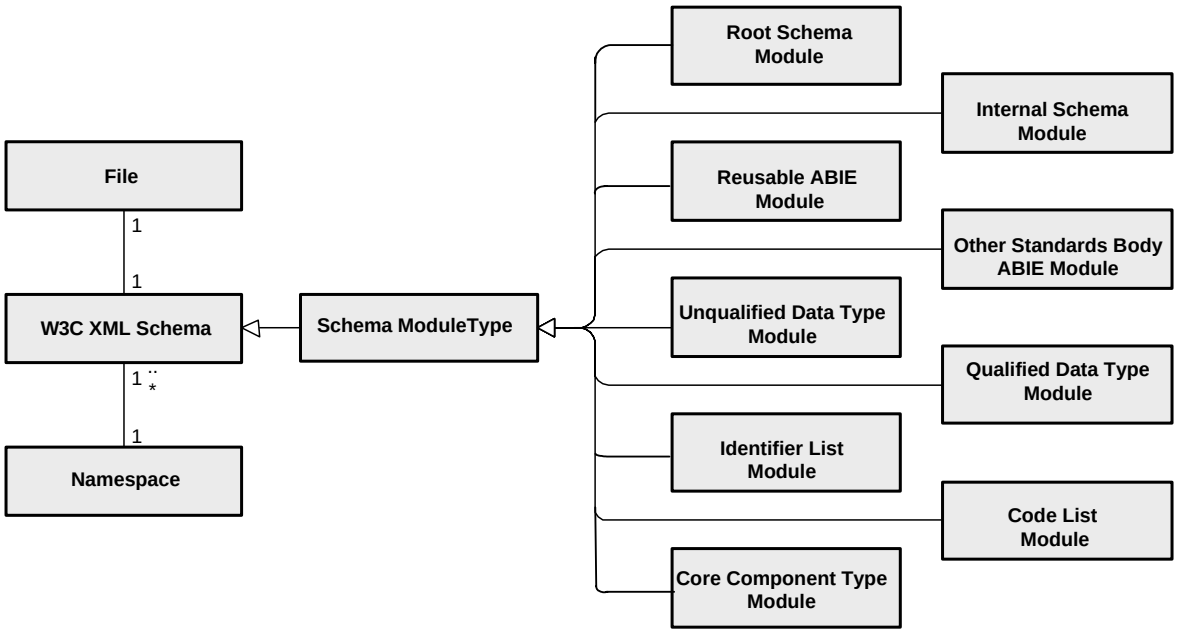


Figure 5-5 UN/CEFACT XSD Schema Modules

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 schema module.

[R 21] 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.

[R 22] The **`cct:CoreComponentType`** schema module MUST be named "CCTS CCT Schema Module"

The current version of the normative UN/CEFACT CCT schema module is contained in Appendix D.

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 built-in **`xsd:simpleType`** rather than their parent CCT **`xsd:complexType`**. As such, the unqualified data type schema module does not import the CCT schema module. The unqualified data types are so named because they contain no additional restrictions on their source CCTs other than those defined in CCTS and agreed upon best practices. An unqualified data type is defined for all approved CCTS primary and secondary representation terms.

[R 23] 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.

[R 24] The **`udt:UnqualifiedDataType`** schema module MUST be named "UN/CEFACT Unqualified Data Type Schema Module"

The current version of the normative UN/CEFACT Unqualified Data Type Schema Module is contained in Appendix E.

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

[R 25]	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.

[R 26]	The **`qdt:QualifiedDataType`** schema module MUST be named "UN/CEFACT Qualified Data Type Schema Module"

The current version of the normative UN/CEFACT Qualified Data Type Schema Module is contained in Appendix E.

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 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 run time performance considerations if necessary. Compression means that a run time 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.

[R 27]	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.

[R 28]	The **`ram:ReusableAggregateBusinessInformationEntity`** schema module MUST be named "UN/CEFACT 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.

[R 29]	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.

[R 30]	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 5-10: Name of UN/CEFACT Account Type Code Schema Module

```
64437 - Code List Schema Module
```

```
where:
6 = Code list agency identifier for UN/CEFACT as defined in UN/CEFACT code
    list 3055
4437 = Code list identification identifier for Account Type Code in UN/CEFACT
    directory
```

Example 5-11: Name for a code using agency name and code list name

```
Security Initiative Document Security Code - Code List Schema Module
```

5.5.3.6 Identifier List Schema Module

In those cases where run time 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 enumeration values for the identifiers.

[R 31] An Identifier List schema module MUST be created to convey enumeration values for each identifier list that requires run time 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.

[R 32] 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 identification scheme
Identifier Scheme Agency Name = Agency that maintains the identifier list
Identifier Scheme Identifier = The identification of the identification scheme
Identification Scheme Name = Name as assigned by the agency that maintains the identifier list

Example 5-12: 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 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 modules when their contents are in strict conformance to the requirements of the CCTS and this specification.

[R 33] Imported schema modules MUST be fully conformant with the *UN/CEFACT XML Naming and Design Rules* Technical Specification and the *Core Components Technical Specification.*

5.6 Namespace Scheme

As defined in the W3C 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 extensive library of reusable types and schema modules. The UN/CEFACT reusability

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

methodology maximizes the reuse of defined named types and locally declared elements and attributes within those types (See Section 5.4). In addition, the modularity approach of multiple reusable schema modules (See Section 5.5) prescribe just such a method. There exists 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 5-4. Accordingly, a sufficiently robust namespace scheme is essential.

5.6.1 UN/CEFACT 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 5-6 reflects such an approach and will be used as the basis for determining the namespace structure and rules that follow.

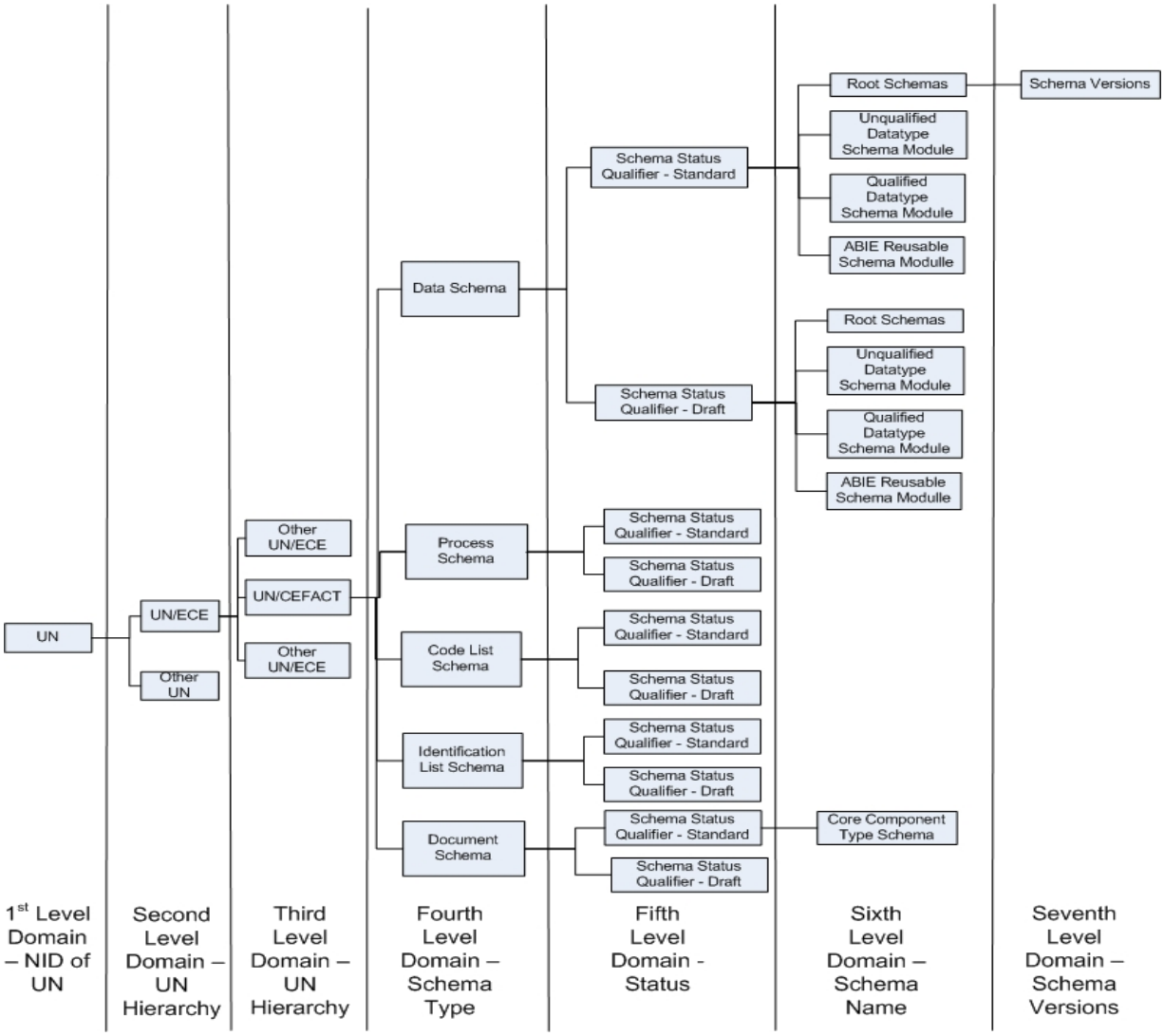


Figure 5-6: UN/CEFACT Namespace Scheme

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.

[R 34]	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 in synchronization. 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.

[R 35] 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.

[R 36] UN/CEFACT published namespace declarations or contents MUST never be changed.

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.

[R 37] 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 if 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 ID 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 module (using upper camel case)
- version = <major>.<minor>.[<revision>]
  - 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. Not applicable for code list or identifier list schema.
  - revision = Sequentially assigned alphanumeric character for each revision of a minor release. Only applicable where status = draft. Not applicable for code list or identifier list schema.

[R 38] The names for namespaces MUST have the following structure while the schema is at draft status:
urn:un:unece:uncefact:<schematype>:draft:<name>:<major>.[<minor>].[<revision]

Where:
schematype = a token identifying the type of schema module:
data|process|codelist|identifierlist|documentation
name = the name of the module (using upper camel case)
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. Not applicable for code list or identifier list schema.
revision = sequentially assigned alphanumeric character for each revision of a minor release. Only applicable where status = draft and schema type does not equal code list or identifier list.

Example 5-13: Namespace Name at Draft Status

```
"urn:un:unece:uncefact:data:draft:UNCEFACTUnqualifiedDataTypeSchemaModule:0.3
.5"
```

[R 39] The namespace names for schema holding specification status MUST be of the form:
urn:un:unece:uncefact:<schematype>:standard:<name>:<major>.[<minor>]

Where:
schematype = a token identifying the type of schema module:
data|process|codelist|identifierlist|documentation
name = the name of the module
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. Not applicable for code list or identifier list schema.

Example 5-14: Namespace Name at Specification Status

```
"urn:un:unece:uncefact:data:standard:UNCEFACTUnqualifiedDataTypeSchemaModule:
1.0"
```

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.

[R 40] UN/CEFACT namespaces MUST only contain UN/CEFACT developed schema modules.

5.6.6 UN/CEFACT XSD Schema Namespace Tokens

A unique token will be defined for each namespace. The exact token for each type of namespace will be defined by 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 the same as their namespaces. Schema locations are typically defined as URL based URI schemes because of resolvability limitations of URN based URI schemes. 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 aligned with the URN based URI scheme used for the namespace declaration as follows:

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

[R 41] The general structure for schema location MUST be:
`http://www.unece.org/uncefact/<schematype>/<name>_<major>.<minor>.[<revision>]_[<status>].xsd`

Where:
schematype = a token identifying the type of schema module:
`data|process|codelist|identifierlist|documentation`
name = the name of the module (using upper camel case)
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.
status = the status of the schema as: **`draft|standard`**

[R 42] Each **`xsd:schemaLocation`** attribute declaration MUST contain a persistent and resolvable URL.

[R 43] Each **`xsd:schemaLocation`** attribute declaration URL MUST contain an absolute path.

5.8 Versioning

A UN/CEFACT namespace URN is divided into three parts. First is the standard UN/CEFACT namespace information. Second is the description of the purpose of the namespace. Third is the version information. The version information will in turn be divided into major (or incompatible) and minor (or compatible) fields. The minor field has an optional revision extension.

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 will experience validation errors. A new major version will be produced when significant and/or non-backwards 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>.0

Where:

- major = the first release starts with the number 1.
- minor = always 0 for major release numbers.

[R 44] Every schema major version MUST have the URI of:
urn:un:unece:uncefact:<schematype>:<status>:<name>:<major>.0.[<revision>]

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.

[R 45] Every UN/CEFACT XSD Schema and schema module major version number MUST be a sequentially assigned incremental integer greater then 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 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

[R 46] Minor versioning MUST be limited to declaring new optional XSD constructs, extending existing XSD constructs and refinements of an optional nature.

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

urn:un:unece:uncefact:<schematype>:<status>:<name>:<major>.<non-zero integer>.[<revision>]

Where:

- major = the major version number, sequentially assigned, first release starting with the number 1
- minor = always positive integer

[R 47] Every UN/CEFACT XSD Schema minor version MUST have the URI of:
urn:un:unece:uncefact:cc:schema:<name>:<major>.<non-zero integer>.[<revision>]

Just like major version numbers, minor 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.

Minor version changes are not allowed to break compatibility with previous minor versions. Compatibility includes consistency in naming of the schema constructs. UN/CEFACT minor version changes will not include renaming the XML construct.

[R 48] For UN/CEFACT minor version changes, the name of the schema construct MUST NOT change.

Semantic compatibility across minor versions is essential.

[R 49] Changes in minor versions MUST NOT break semantic compatibility with prior versions.

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

[R 50] UN/CEFACT minor version schema MUST incorporate all XML constructs from the immediately preceding major or minor version schema.

875 [Note]
876 There has been much discussion surrounding the issue of namespaces and versioning.
877 ATG solicits input from interested parties on the pro's and con's of assigning a unique
878 namespace for each minor version as opposed to assigning a new namespace for only
879 major versions and having all minor versions have the same namespace as its major
880 version.

6 General XML Schema Language Conventions

6.1 Schema Construct

[R 51] The **xsd:elementFormDefault** attribute MUST be declared and its value set to “qualified”.

[R 52] The **xsd:attributeFormDefault** attribute MUST be declared and its value set to “unqualified”.

[R 53] 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

[R 54] The xsi prefix SHALL be used where appropriate for referencing **xsd:schemaLocation** and **xsd:noNamespaceLocation** attributes in instance documents.

[R 55] **xsd:appInfo** MUST NOT be used.

[R 56] **xsd:notation** MUST NOT be used.

[R 57] **xsd:wildcard** MUST NOT be used.

[R 58] The **xsd:any** element MUST NOT be used.

[R 59] The **xsd:any** attribute MUST NOT be used.

[R 60] Mixed content MUST NOT be used (excluding documentation).

[R 61] **xsd:substitutionGroup** MUST NOT be used.

[R 62] **xsd:ID/IDREF** MUST NOT be used.

[R 63] **xsd:key/xsd:keyref** MUST be used for information association.

[R 64] 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, built-in **xsd:attributes** will be used as described elsewhere in this document.

[R 65] 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.

[R 66] An attribute of a supplementary component with variable information MUST be based on the appropriate built-in XSD data type.

[R 67] An attribute of a supplementary component which represents codes MUST be based on the **`xsd:simpleType`** of the appropriate code list

[R 68] An attribute of 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 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.

[R 69] The **`xsd:nillable`** attribute MUST NOT be used.

6.2.2 Elements

6.2.2.1 Usage of Elements

Elements are declared for document level message assembly, BBIEs, and ASBIEs.

6.2.2.2 Element Declaration

[R 70] All element declarations MUST be local except for a root element that must be declared globally.

[R 71] Empty elements MUST NOT be used.

The **`xsd:enumeration`** element may be used within reusable or internal schema modules if the list of enumerated values is less than 10, are not represented by a token, and are considered by TBG to be static and particular to the business processes.

[R 72] The **`xsd:type`** of each leaf element declaration MUST be of the data type of its source business information entity (BBIE) or complex type of its source association business information entity (ASBIE).

Example 6-2:

```
<xsd:complexType name="AccountType">
   <xsd:annotation>
      ...see annotation...
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="ID" type="udt:IdentifierType"
         minOccurs="0" maxOccurs="unbounded">
         <xsd:annotation>
            ...see annotation...
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Status" type="ram:StatusType"
         minOccurs="0" maxOccurs="unbounded">
         <xsd:annotation>
            ...see annotation...
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Name" type="udt:NameType"
         minOccurs="0" maxOccurs="unbounded">
         <xsd:annotation>
            ...see annotation...
         </xsd:annotation>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

6.2.2.3 Constraints on Element Declarations

[R 73] The **`xsd:all`** element MUST NOT be used.

6.3 Type Definitions

6.3.1 Usage of Types

[R 74] All type definitions MUST be named.

Example 6-3:

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

[R 75] Data type definitions MUST NOT duplicate the functionality of an existing data type definition..

6.3.2 Simple Type Definitions

Built-in simple types must always be used where they satisfy the business requirements. Where the 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="DateTimeType">
   <xsd:annotation>
      … see annotation …
   </xsd:annotation>
   <xsd:restriction base="xsd:dateTime"/>
</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 built-in simple 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 "AccountType"

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

6.4 Use 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 TBG groups (including TBG17 - Harmonization) and ICG. 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

[R 76] xsd:extension MUST only be used in the **`cct:CoreComponentType`** schema module and the **`udt:UnqualifiedDataType`** schema module. When used it MUST only extend a built-in XSD datatype.

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.

[R 77] When **`xsd:restriction`** is applied to a **`xsd:simpleType`** or **`xsd:complexType`** the derived construct MUST use a different name.

Example 6-7: Restriction of Simple Type

```
<xsd:simpleType name="IndicatorType">
    <xsd:annotation>
          ... see annotation ...
    </xsd:annotation>
    <xsd:restriction base="xsd:boolean">
       <xsd:pattern value="false"/>
       <xsd:pattern value="true"/>
    </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.

[R 78] 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. 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)
- **Category Code:** The category to which the artefact belongs. (CategoryCode**)**

- **Dictionary Entry Name**: The complete name (not the tag name) of the artefact in the library. (DictionaryEntryName**)**
- **Name:** The name of the message assembly. (Name)
- **Version**: The version of the artefact as assigned by the registry. (VersionID)
- **Definition**: The semantic meaning of the artefact. (Definition)
- **Description**: A brief description of the business information exchange. (Description)
- **Cardinality**: An indication of whether the property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the object. (Cardinality)
- **Business Domain**: The TBG groups(s) that developed the artefact. (BusinessDomain)
- **Object Class Term**: The Object Class represented by the artefact. (ObjectClassTermName)
- **Object Class Qualifier Term:** A term(s) that qualifies the Object Class.. (ObjectClassQualifierTermName)
- **Property Term**: The Property Term represented by the artefact. (PropertyTermName)
- **Property Qualifier Term:** A term(s) that qualifies the Property Term. (PropertyQualifierTermName)
- **Associated Object Class Term**: The Associated Object Class Term represented by the artefact. (AssociatedObjectClassTermName)
- **Associated Object Class Qualifier Term:** A term(s) that qualifies the Associated Object Class Term. (AssociatedObjectClassQualifierTermName)
- **Representation Term**: The Representation Term represented by the artefact. (RepresentationTermName)
- **Data Type Qualifier Term:** A term(s) that qualifies the Data Type Term. (DataTypeQualifierTermName)
- **Primitive Type**: The primitive data type as assigned to the artefact by CCTS. (PrimitiveType)
- **Built In Type**: The XSD built-in data type assigned to the artefact. (BuiltInType)
- **Business Process Context**: A valid value describing the Business Process contexts for which this construct has been designed. Default is “In All Contexts”. (BusinessProcessContext)
- **Geopolitical/Region Context**: A valid value describing the Geopolitical/Region contexts for which this construct has been designed. Default is “In All Contexts”. (GeopoliticalOrRegionContext)
- **Official Constraints Context**: A valid value describing the Official Constraints contexts for which this construct has been designed. Default is “None”. (OfficialConstraintContext)
- **Product Context**: A valid value describing the Product contexts for which this construct has been designed. Default is “In All Contexts”. (ProductContext)
- **Industry Context**: A valid value describing the Industry contexts for which this construct has been designed. Default is “In All Contexts”. (IndustryContext)
- **Business Process Role Context**: A valid value describing the Role contexts for which this construct has been designed. Default is “In All Contexts”. (BusinessProcessRoleContext)
- **Supporting Role Context**: A valid value describing the Supporting Role contexts for which this construct has been designed. Default is “In All Contexts”. (SupportingRoleContext)
- **System Capabilities Context**: A valid value describing the Systems Capabilities contexts for which this construct has been designed. Default is “In All Contexts”. (SystemCapabilitiesContext)
- **Usage Rule:** A constraint that describes specific conditions which are applicable to the artefact. (UsageRuleText)

- **Business Term:** A synonym term under which the artefact is commonly known and used in business. (BusinessTermName)
- **Example:** A possible value for the artefact. (Example)

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

Example 6-8: Example of annotation

```
<xsd:annotation>
 <xsd:documentation xml:lang="en">
    <ccts:UniqueID>UN00000002</ccts:UniqueID>
    <ccts:CategoryCode>BBIE</ccts:CategoryCode>
    <ccts:DictionaryEntryName>Account.
       Identifier</ccts:DictionaryEntryName>
    <ccts:Definition>The identification of a specific
       account.</ccts:Definition>
    <ccts:VersionID>1.0</ccts:VersionID>
    <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
    <ccts:PropertyTermName>Identifier</ccts:PropertyTermName>
 <ccts:RepresentationTermName>Identifier</ccts:RepresentationTermName>
    <ccts:BusinessTermName>Account Number</ccts:BusinessTermName>
 </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.

1128 The following table provides a summary view of the annotation data as defined in section 6.

		rsm:RootSchema	ABIE xsd:complexType	BBIE xsd:element	ASBIE: xsd:element	cct:CoreComponentType	supplementary component	udt:UnqualifiedDataType	qdt:QualifiedDataType
Unique Identifier	UniqueID	M	M	M	M	M	M	M	M
Category Code	CategoryCode	M	M	M	M	M	M	M	M
Dictionary Entry Name	DictionaryEntryName		M	M	M	M	M	M	M
Name	Name	M							
Version	VersionID	M	M	M	M	M		M	M
Definition	Definition		M	M	M	M	M	M	M
Description	Description	M							
Cardinality	Cardinality			M	M				
Business Domain	BusinessDomain	M, R							
Object Class Term	ObjectClassTermName		M	M	M		M		
Object Class Qualifier Term	ObjectClassQualifierTermName		O	O	O				
Property Term	PropertyTermName			M	M		M		
Property Qualifier Term	PropertyQualifierTermName			O	O				
Associated Object Class Term	AssociatedObjectClassTermName				M				
Associated Object Class Qualifier Term	AssociatedObjectClassQualifierTermName				O				
Representation Term	RepresentationTermName			M		M	M	M	M
Data Type Qualifier Term	DataTypeQualifierTermName								M
Primitive Type	PrimitiveType					M	M	M	M
XSD Built-in data type	BuiltInType						C	M	M
Business Process Context	BusinessProcessContext	M, R	O, R	O, R	O, R				O, R
Geopolitical/Region Context	GeopoliticalOrRegionContext	O, R	O, R	O, R	O, R				O, R
Official Constraints Context	OfficialContraintContext	O, R	O, R	O, R	O, R				O, R
Product Context	ProductContext	O, R	O, R	O, R	O, R				O, R
Industry Context	IndustryContext	O, R	O, R	O, R	O, R				O, R
Business Process Role Context	BusinessProcessRoleContext	O, R	O, R	O, R	O, R				O, R
Supporting Role Context	SupportingRoleContext	O, R	O, R	O, R	O, R				O, R
System Capabilities Context	SystemCapabilitiesContext	O, R	O, R	O, R	O, R				O, R
Usage Rule	UsageRuleText		O, R	O, R	O, R	O, R	O, R	O, R	O, R
Business Term	BusinessTermName		O, R	O, R	O, R	O, R			
Example	Example		O, R	O, R	O, R	O, R	O, R	O, R	O, R

1129 Key:

1130 M - mandatory

1131 O - optional

1132 R - repeating

1133 C - conditional

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; [VERSION]                        ===== -->
<!-- ========================================================================== -->
<!--
    Module of      [MODULENAME]
    Agency:        UN/CEFACT
    Version:       0.3 Rev. 6
    Last change:   25 June 2004

    Copyright (C) UN/CEFACT (2004). All Rights Reserved.

... see copyright information ...
-->
<xsd:schema
 targetNamespace="urn:un:unece:uncefact:data:draft:[MODULENAME]:0.3.6"
 ... see namespaces ...
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
 <!-- ===== Includes                                                  ===== -->
 <!-- ========================================================================== -->
 <!-- ===== Include of [MODULENAME] ===== -->
 ... see includes ...
<!-- ===== Imports                                                 ===== -->
 <!-- ========================================================================== -->
 <!-- ===== Import of [MODULENAME] ===== -->
 ... see imports ...

 <!-- ===== Root Element                                              ===== -->
 <!-- ========================================================================== -->
 ... see root element declaraction ...
 <!-- ===== Type Definitions                                          ===== -->
 <!-- ========================================================================== -->
 <!-- ===== Type Definition: [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 ATG to represent the namespace prefix. This token will be prefixed by ‘rsm’.

[R 79] The root schema module MUST be represented by a unique token.

Example 7-2: Structure of Root Schema Module

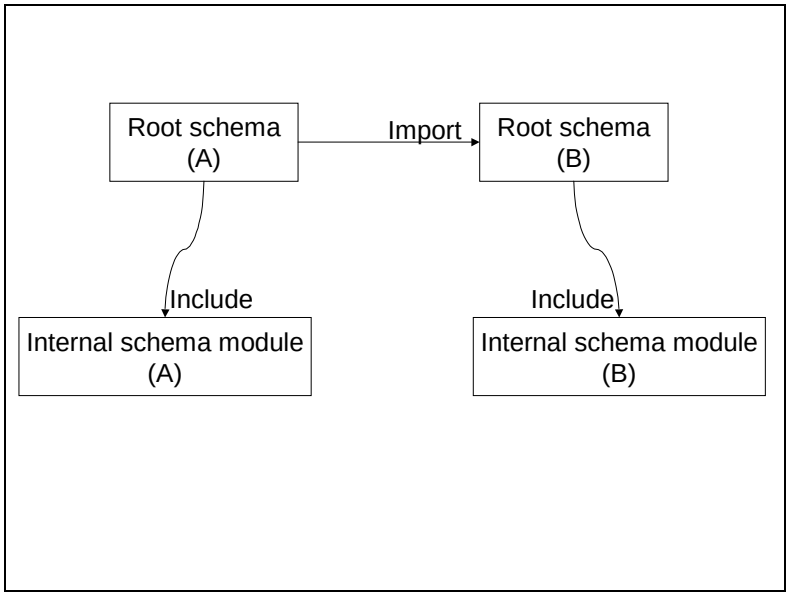
```
xmlns:rsm="urn:un:unece:uncefact:data:draft:UNCEFACTExamplesSchemaModule:0.3.6"
```

> [Note]
>
> Throughout this specification, the token ‘rsm’ is used for the unique root schema token.

7.1.3 Imports and Includes

[R 80] 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 an Order Response message (root schema A) makes use of ABIEs defined as part of the schema definition for an Order 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.



[R 81] 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.

[R 82] A **`rsm:RootSchema`** in one UN/CEFACT namespace that is dependant upon type definitions or element declarations defined in another namespace MUST NOT import Schema Modules from that namespace other than the **`rsm:RootSchema.`**

[R 83] 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 message has a single root element that is globally declared in the root schema.. The root element is named according to the business information exchange that it represents and references the message assembly that contains the actual business information.

[R 84] A single global element known as the root element MUST be globally declared in a **`rsm:RootSchema`**.

[R 85] The name of the root element MUST be the name of the Message Assembly with separators and spaces removed.

Example 7-3: Name of Root Element

```
<!-- ===== Root Element ===== -->
<!-- ============================ -->
<xsd:element name="PurchaseOrder" type="rsm:PurchaseOrderType">
   <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 exchange.

[R 86] Root schema MUST define a single **`xsd:complexType`** that fully describes the business information exchange.

[R 87] The name of the top-level complex type MUST be the name of the root element with the word "Type" appended.

[R 88] The **`xsd:complexType`** of the root element must be the top-level complex type.

Example 7-4: Name of Complex Type Definition

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

7.1.6 Annotations

[R 89] For every **`rsm:RootSchema`** root element declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references the Message Assembly instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be RSM.
- Name (mandatory): The name of the Message Assembly
- VersionID (mandatory): An indication of the evolution over time of a Message Assembly.
- Description (mandatory): A brief description of the business information exchange.
- BusinessDomain (mandatory, repetitive): The TBG group(s) that developed this Message Assembly.
- BusinessProcessContext (mandatory, repetitive): The business process with which this Message Assembly is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this Message Assembly.

- OfficialConstraintContext (optional, repetitive): The official constraint context for this Message Assembly.
- ProductContext (optional, repetitive): The product context for this Message Assembly.
- IndustryContext (optional, repetitive): The industry context for this Message Assembly.
- BusinessProcessRoleContext (optional, repetitive): The role context for this Message Assembly.
- SupportingRoleContext (optional, repetitive): The supporting role context for this Message Assembly.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this Message Assembly.

7.2 Internal Schema

A UN/CEFACT internal schema module will contain schema constructs representing ABIEs that are specific to a given root schema. 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

[R 90]	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`**.

[R 91]	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 Entities

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"?>
<!-- ============================================================== -->
<!-- ===== RAM Reusable ABIEs Schema Module                   ===== -->
<!-- ============================================================== -->
<!--
```

```
   Module of      Reusable ABIEs (Aggregate Business Information Entities)
   Agency:        UN/CEFACT
   Version:       0.3 Rev. 6
   Last change:   25 June 2004

   Copyright (C) UN/CEFACT (2004). All Rights Reserved.

   ... see copyright information ...
-->
<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 defintions ...
</xsd:schema>
```

7.3.2 Namespace Scheme

[R 92] 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:UNCEFACTReusableAggregateBusinessInformationEntitySc
hemaModule:0.3.6"
```

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

```
<xsd:schema
 targetNamespace=
 "urn:un:unece:uncefact:data:draft:UNCEFACTReusableAggregateBusinessInformationEntitySc
hemaModule:0.3.6"
 xmlns:ram=
 "urn:un:unece:uncefact:data:draft:UNCEFACTReusableAggregateBusinessInformationSchemaMo
dule:0.3.6"
```

7.3.3 Imports and Includes

[R 93] 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 (QDT)        ===== -->
 <!-- ================================================================== -->
 <xsd:import
    namespace=
    "urn:un:unece:uncefact:data:draft:UNCEFACTQualifiedDataTypeSchemaModule:0.3.6"
    schemaLocation="http://www.unece.org/uncefact/data/
UNCEFACTQualifiedDataTypeSchemaModule_0.3.6_draft.xsd"/>
    <!-- ================================================================== -->
<!-- ===== Import of Unqualified Data Type Schema Module (UDT)      ===== -->
 <!-- ================================================================== -->
 <xsd:import
    namespace=
       "urn:un:unece:uncefact:data:draft:UNCEFACTUnqualifiedDataTypeSchemaModule:
       0.3.6"

 schemaLocation="http://www.unece.org/uncefact/data/UNCEFACTUnqualifiedDataTypesSchemaM
odule_0.3.6_draft.xsd"/>
```

7.3.4 Type Definitions

[R 94] For every object class (ABIE) identified in the UN/CEFACT syntax-neutral model, a named **`xsd:complexType`** MUST be defined.

[R 95] The name of the ABIE **`xsd:complexType`** MUST be the **`ccts:DictionaryEntryName`** with the separators removed 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 a local element declaration, with its cardinality and sequencing within the schema XSD **`content`** model determined by the details of the source business information entity (ABIE).

[R 96] Every aggregate business information entity (ABIE) **`xsd:complexType`** definition **`xsd:content`** model MUST use the **`xsd:sequence`** and/or **`xsd:choice`** elements with appropriate local element declarations to reflect each property (BBIE or ASBIE) of its class.

[R 97] 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.

Example 7-9: Sequence within an object class

```
<xsd:complexType name="AccountType" >
   <xsd:annotation>
      ...see annotation...
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="ID" type="udt:IdentifierType"
         minOccurs="0" maxOccurs="unbounded">
         <xsd:annotation>
            ...see annotation...
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Status" type="ram:StatusType"
         minOccurs="0" maxOccurs="unbounded">
         <xsd:annotation>
            ...see annotation...
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Name" type="udt:NameType"
         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>
```

[R 98] 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="AccountType" >
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="ID" type="udt:IdentifierType"
         minOccurs="0" maxOccurs="unbounded">
         <xsd:annotation>
            ... see annotation ...
         </xsd:annotation>
      </xsd:element>
      ... see element declaration ....
   </xsd:sequence>
</xsd:complexType>
```

7.3.5 Element Declarations

[R 99] For every attribute of an object class (BBIE) identified in an ABIE, a named **`xsd:element`** MUST be locally declared within the **`xsd:complexType`** representing that ABIE.

[R 100] Each BBIE element name declaration MUST be based on the property term and qualifiers and the representation term of the basic business information entity (BBIE). If there are successive duplicate words in the property term and representation terms of the source dictionary entry name, then the duplicate words MUST be removed.

[R 101] If the representation term of a BBIE is 'text', it MUST be removed.

[R 102] 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.

[R 103] For every association (ASBIE) identified in the UN/CEFACT syntax-neutral model, a named **`xsd:element`** MUST be locally declared within the **`xsd:complexType`** representing the ABIE.

[R 104] Each ASBIE element name declaration MUST be based on the property term and object class of the association business information entity (ASBIE). If there are successive duplicate words in the property term and object class of the associated ABIE, then the duplicate words MUST be removed.

[R 105] The element representing an association business information entity (ASBIE) MUST be of the complex type corresponding to its associated aggregate business information (ABIE).

Example 7-13: Element declaration within an ABIE

```
... see type defintion ...
<xsd:element name="ID" type="udt:IdentifierType"
   minOccurs="0" maxOccurs="unbounded">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
</xsd:element>
<xsd:element name="Status" type="ram:StatusType"
   minOccurs="0" maxOccurs="unbounded">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
</xsd:element>
<xsd:element name="Name" type="udt:NameType"
   minOccurs="0" maxOccurs="unbounded">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
</xsd:element>
<xsd:element name="CurrencyCode" type="qdt:CurrencyCodeType"
   minOccurs="0" maxOccurs="unbounded">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
</xsd:element>
... see type definition ...
```

7.4 Annotation

[R 106] For every ABIE **`xsd:complexType`** definition a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references an ABIE instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be ABIE.

- DictionaryEntryName (mandatory): The official name of an ABIE.
- VersionID (mandatory): An indication of the evolution over time of an ABIE instance.
- Definition (mandatory): The semantic meaning of an ABIE.
- ObjectClassTermName (mandatory): The Object Class Term of the ABIE.
- QualifierTermName (optional): Qualifies the Object Class Term of the ABIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the ABIE.
- BusinessTermName (optional, repetitive): A synonym term under which the ABIE is commonly known and used in the business.
- BusinessProcessContext (optional, repetitive): The business process with which this ABIE is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this ABIE.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this ABIE.
- ProductContext (optional, repetitive): The product context for this ABIE.
- IndustryContext (optional, repetitive): The industry context for this ABIE.
- BusinessProcessRoleContext (optional, repetitive): The role context for this ABIE.
- SupportingRoleContext (optional, repetitive): The supporting role context for this ABIE.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this ABIE.
- Example (optional, repetitive): Example of a possible value of an ABIE.

Example 7-14: Annotation of an ABIE

```
 <xsd:complexType name="AccountType" >
    <xsd:annotation>
       <xsd:documentation xml:lang="en">
          <ccts:UniqueID>UN00000001</ccts:UniqueID>
          <ccts:CategoryCode>ABIE</ccts:CategoryCode>
          <ccts:DictionaryEntryName>Account.
Details</ccts:DictionaryEntryName>
          <ccts:VersionID>1.0</ccts:VersionID>
          <ccts:Definition> A business arrangement whereby debits and/or
credits arising from transactions are recorded. This could be with a bank,
i.e. a financial account, or a trading partner offering supplies or services
'on account', i.e. a commercial account</ccts:Definition>
          <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
       </xsd:documentation>
    </xsd:annotation>
    ...
 </xsd:complexType>
```

[R 107] For every BBIE **`xsd:element`** declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references a BBIE instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be BBIE.
- Dictionary Entry Name (mandatory): The official name of the BBIE.
- VersionID (mandatory): An indication of the evolution over time of a BBIE instance.
- Definition (mandatory): The semantic meaning of the BBIE.

- Cardinality (mandatory): Indication whether the BIE Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the ABIE.
- ObjectClassTermName (mandatory): The Object Class Term of the parent ABIE.
- ObjectClassQualifierTermName (optional): Qualifies the Object Class Term of the parent ABIE.
- PropertyTermName (mandatory): The Property Term of the BBIE.
- PropertyQualifierTermName (optional): Qualifies the Property Term of the BBIE.
- RepresentationTermName (mandatory): The Representation Term of the BBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the BBIE.
- BusinessProcessContext (optional, repetitive): The business process with which this BBIE is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this BBIE.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this BBIE.
- ProductContext (optional, repetitive): The product context for this BBIE.
- IndustryContext (optional, repetitive): The industry context for this BBIE.
- BusinessProcessRoleContext (optional, repetitive): The role context for this BBIE.
- SupportingRoleContext (optional, repetitive): The supporting role context for this BBIE.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this BBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to this BBIE.
- BusinessTermName (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:IdentifierType"
           minOccurs="0" maxOccurs="unbounded">
           <xsd:annotation>
              <xsd:documentation xml:lang="en">
                 <ccts:UniqueID>UN00000002</ccts:UniqueID>
                 <ccts:CategoryCode>BBIE</ccts:CategoryCode>
                 <ccts:DictionaryEntryName>Account.
Identifier</ccts:DictionaryEntryName>
                 <ccts:VersionID>1.0</ccts:VersionID>
                 <ccts:Definition>The identification of a specific account.
                 </ccts:Definition>
                 </ccts:Cardinality>0..n</ccts:Cardinality>
                 <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
                 <ccts:PropertyTermName>Identifier</ccts:PropertyTermName>
        <ccts:RepresentationTermName>Identifier</ccts:RepresentationTermName>
                 <ccts:BusinessTermName>Account Number</ccts:BusinessTermName>
              </xsd:documentation>
           </xsd:annotation>
        </xsd:element>
```

[R 108] For every ASBIE **`xsd:element`** declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references an ASBIE instance in a unique and unambiguous way.

- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be ASBIE.
- DictionaryEntryName (mandatory): The official name of the ASBIE.
- VersionID (mandatory): An indication of the evolution over time of the ASBIE instance.
- Definition (mandatory): The semantic meaning of the ASBIE.
- Cardinality (mandatory): Indication whether the ASBIE Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the ABIE.
- ObjectClassTermName (mandatory): The Object Class Term of the associating ABIE.
- ObjectClassQualifierTermName (Optional): A term that qualifies the Object Class Term of the associating ABIE.
- PropertyTermName (mandatory): The Property Term of the ASBIE.
- PropertyQualifierTermName (Optional): A term that qualifies the Property Term of the ASBIE.
- AssociatedObjectClassTermName (mandatory): The Object Class Term of the associated ABIE.
- AssociatedObjectClassQualifierTermName (optional): Qualifies the Object Class Term of the associated ABIE.
- BusinessProcessContext (optional, repetitive): The business process with which this ASBIE is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this ASBIE.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this ASBIE.
- ProductContext (optional, repetitive): The product context for this ASBIE.
- IndustryContext (optional, repetitive): The industry context for this ASBIE.
- BusinessProcessRoleContext (optional, repetitive): The role context for this ASBIE.
- SupportingRoleContext (optional, repetitive): The supporting role context for this ASBIE.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this ASBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the ASBIE.
- BusinessTermName (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="Status" type="ram:StatusType"
           minOccurs="0" maxOccurs="unbounded">
           <xsd:annotation>
              <xsd:documentation xml:lang="en">
                 <ccts:UniqueID>UN00000003</ccts:UniqueID>
                 <ccts:CategoryCode>ASCC</ccts:CategoryCode>
                 <ccts:DictionaryEntryName>Account.
Status</ccts:DictionaryEntryName>
                 <ccts:VersionID>1.0</ccts:VersionID>
                 <ccts:Definition>Associated status information related to
account details.</ccts:Definition>
              <ccts:Cardinality>0..n</ccts:Cardinality>
                 <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
                 <ccts:PropertyTermName>Status</ccts:PropertyTermName>
                 <ccts:AssociatedObjectClassTermName>Status
                 </ccts:AssociatedObjectClassTermName>
              </xsd:documentation>
```

```
            </xsd:annotation>
        </xsd:element>
```

7.5 Core Component Type

7.5.1 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. The normative formatted schema for the Core Component Type module is contained in Appendix D.

7.5.2 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"?>
<!-- ========================================================================= -->
<!-- ===== CCTS Core Component Type Schema Module                        ===== -->
<!-- ========================================================================= -->
<!--
    Module of     Core Component Type
    Agency:       UN/CEFACT
    Version:      0.3 Rev. 6
    Last change:  25 June 2004

    Copyright (C) UN/CEFACT (2004). All Rights Reserved.

    ... see copyright information ...

-->
<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>
```

7.5.3 Namespace Scheme

[R 109] 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:UNCEFACTCCTSCCTSchemaModule:03.6"
```

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

```
<xsd:schema
  targetNamespace=
  "urn:un:unece:uncefact:documentation:draft:UNCEFACTCCTSCCTSchemaModule:0.3.6
"
  xmlns:cct=
  "urn:un:unece:uncefact:documentation:draft:UNCEFACTCCTSCCTSchemaModule:0.3.6
"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
```

7.5.4 Imports and Includes

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

[R 110] The **cct:CoreCoreComponentType** schema module MUST NOT include or import any other schema modules.

7.5.5 Type Definitions

[R 111] Every **cct:CoreComponentType** MUST be defined as a named **xsd:complexType** in the **cct:CoreComponentType** schema module.

[R 112] The name of each **xsd:complexType** based on a **cct:CoreComponentType** MUST be the dictionary entry name of the core component type (CCT), with the separators and spaces removed.

[R 113] Each **cct:CoreComponentType xsd:complexType** definition MUST contain one **xsd:simpleContent** element.

[R 114] The **cct:CoreComponentType 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 built-in XSD data type required for the CCT content component.

[R 115] Within the **cct:CoreComponentType** xsd:extension element a **xsd:attribute** MUST be declared for each supplementary component pertaining to that **cct:CoreComponentType**.

Example 7-20: Type definition of a CCT

```
<!-- ===== Type Definitions                                        ===== -->
<!-- ========================================================================= -->
<!-- ===== CCT: 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>
```

7.5.6 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, ATG has applied the naming concepts from ISO 11179, part 5. Specifically, ATG 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.

[R 116] Each **cct:CoreComponentType** supplementary component **xsd:attribute** "name" MUST be the CCTS supplementary component dictionary entry name with the separators and spaces removed.

[R 117] 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.

[R 118] 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.

[R 119] 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.

[R 120] The attribute representing as supplementary component MUST be based on the appropriate built-in XSD data type.

Example 7-22: 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 core component type schema module is a generic module that will be restricted in qualified and unqualified data type schema modules.

7.5.8 Annotation

[R 121] For every **`cct:CoreComponentType`** **`xsd:complexType`** definition a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references the Core Component Type instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be CCT.
- DictionaryEntryName (mandatory): The official name of a Core Component Type.
- VersionID (mandatory): An indication of the evolution over time of a Core Component Type instance.
- Definition (mandatory): The semantic meaning of a Core Component Type.
- RepresentationTermName (mandatory): The primary representation term of the Core Component Type.
- PrimitiveType (mandatory): 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.
- BusinessTermName (optional, repetitive): A synonym term under which the Core Component Type is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of a Core Component Type.

Example 7-21: Annotation of a CCT

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

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

[R 122] For every supplementary component **`xsd:attribute`** declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references a Supplementary Component instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be SC.
- DictionaryEntryName (mandatory): The official name of the Supplementary Component.
- Definition (mandatory): The semantic meaning of the Supplementary Component.
- ObjectClassTermName (mandatory): The Object Class of the Supplementary Component.
- PropertyTermName (mandatory): The Property Term of the Supplementary Component.
- RepresentationTermName (mandatory): The Representation term of the Supplementary Component.
- PrimitiveType (mandatory): 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 (optional, repetitive): Example of a possible value of a Basic Core Component.

Example 7-22: Annotation of a supplementary component

```
 ... see attribute declaration ...
 <xsd:annotation>
    <xsd:documentation xml:lang="en">
       <ccts:UniqueID>UNDT000001-SC2</ccts:UniqueID>
       <ccts:CategoryCode>SC</ccts:CategoryCode>
       <ccts:DictionaryEntryName>Amount. Currency.
Identifier</ccts:DictionaryEntryName>
       <ccts:Definition>The currency of the amount.</ccts:Definition>
       <ccts:ObjectClassTermName>Amount</ccts:ObjectClassTermName>
       <ccts:PropertyTermName>Currency</ccts:PropertyTermName>
       <ccts:RepresentationTermName>Identifier</ccts:RepresentationTermName>
       <ccts:PrimitiveType>string</ccts:PrimitiveType>
       <ccts:UsageRule>Reference UNECE Rec 9, using 3-letter
          alphabetic codes.</ccts:UsageRule>
    </xsd:documentation>
 </xsd:annotation>
 ... see attribute declaration ...
```

7.6 Unqualified Data Type

7.6.1 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.

7.6.2 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 7-23: Structure of unqualified data type schema module

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ================================================================== -->
<!-- ===== UDT Unqualified Data Type Schema Module                ===== -->
```

```
<!-- ========================================================================= -->
<!--
    Module of     Unqualified Data Type
    Agency:       UN/CEFACT
    Version:      0.3 Rev.6
    Last change:  25 June 2004

    Copyright (C) UN/CEFACT (2004). All Rights Reserved.

    ... see copyright information ...

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

7.6.3 Namespace Scheme

[R 123] The Unqualified Data Type schema module namespace MUST be represented by the token "udt".

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

```
"urn:un:unece:uncefact:data:draft:UNCEFACTUnqualifiedDataTypeSchemaModule:0.3
.6"
```

Example 7-25: Schema-element of unqualified data type schema module

```
<xsd:schema
  targetNamespace=
  "urn:un:unece:uncefact:data:draft:UNCEFACTUnqualifiedDataTypeSchemaModule:0.
3.6"
  xmlns:udt=
  "urn:un:unece:uncefact:data:draft:UNCEFACTUnqualifiedDataTypeSchemaModule:0.
3.6"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
```

7.6.4 Imports and Includes

The Unqualified Data Type schema will import the required code list and identifier list schema modules.

[R 124] The **udt:UnqualifiedDataType** schema MUST NOT import any other schema modules than the following:
– **ids:IdentifierList** schema modules
– **clm:CodeList** schema modules

Example 7-26: Imports

```
  <!-- ===== Imports                                                  ===== -->
  <!-- ========================================================================= -->
  <!-- ===== Imports of Code Lists                                    ===== -->
  <!-- ========================================================================= -->
  <xsd:import namespace=
     "urn:un:unece:uncefact:codelist:draft:6:3403:D.04A"
  schemaLocation="http://www.unece.org/uncefact/codelist/63403_D.04A_draft.xsd
"/>
  <!-- ===== Imports of Identifier Lists                              ===== -->
  <!-- ========================================================================= -->
```

```
  <xsd:import namespace=
     "urn:un:unece:uncefact:identifierlist:draft:5:3166-1:1977"
  schemaLocation="http://www.unece.org/uncefact/identifierlist/53166-
1.1997_standard.xsd"/>
```

7.6.5 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 defined in the CCTS.

[R 125] A **udt:UnqualifiedDataType** MUST be defined for each approved primary and secondary representation terms identified in the CCTS Permissible Representation Terms table.

[R 126] The name of each **udt:UnqualifiedDataType** MUST be the dictionary entry name of the primary or secondary representation term, with “Type” at the end and the separators and spaces removed.

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 the supplementary components for that data type.

[R 127] For every **udt:UnqualifiedDataType** whose supplementary components map directly to the properties of a built-in **xsd:dataTtpe**, the **udt:UnqualifiedDataType** MUST be defined as a named **xsd:simpleType** in the **udt:UnqualifiedDataType** schema module.

[R 128] Every **udt:UnqualifiedDataType** defined as a **xsd:simpleType** MUST contain one **xsd:restriction** element. This **xsd:restriction** element MUST include an **xsd:base** attribute that defines the specific built-in XSD 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, it will be defined as an **xsd:complexType**.

[R 129] For every **udt:UnqualfiedDataType** whose supplementary components are not equivalent to the properties of a built-in XSD data type, a **udt:UnqualifiedDataType** MUST be defined as an **xsd:complexType** in the **udt:UnqualifiedDataType** schema module.

[R 130] Every **udt:UnqualifiedDataType xsd:complexType** definition MUST contain one xsd:simpleContent element.

[R 131] Every **udt:UnqualifiedDataType 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 built-in XSD datatype required for the content component.

7.6.6 Attribute Declarations

Each core component supplementary component will normally be declared as an attribute of the complex type. However, the namespace scheme for code lists and identification scheme lists has been designed to include some of the supplementary components for the CCTs Code. Type and Identifier. Type. Thus, those attributes that are included in the namespace will not be declared as part of the unqualified data type.

[R 132] Within the **udt:UnqualifiedDataType xsd:complextype xsd:extension** element an **xsd:attribute** MUST be declared for each supplementary component pertaining to the underlying CCT, unless the attribute is contained in the namespace declaration.

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
- simpleTypes of a code list, if the specific supplementary component represents a code value
- simpleTypes 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 will reference these.

[R 133] Each supplementary component **`xsd:attribute`** name MUST be the supplementary component name with the separators and spaces removed.

[R 134] 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.

[R 135] 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.

[R 136] 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 7-27: Type definitions of unqualified data types

```
 <!-- ===== Type Definitions                                        ===== -->
 <!-- ========================================================================= -->
 <!-- ===== Primary RT: Amount. Type                                ===== -->
 <!-- ========================================================================= -->
 <xsd:complexType name="AmountType">
    <xsd:annotation>
       ... see annotation ...
    </xsd:annotation>
    <xsd:simpleContent>
       <xsd:extension base="xsd:decimal">
          <xsd:attribute name="currencyID"
             type="clm54217:CurrencyCodeContentType" use="required">
             <xsd:annotation>
                ... see annotation ...
             </xsd:annotation>
          </xsd:attribute>
       <xsd:attribute name="currencyCodeListVersionID"
type="xsd:normalizedString" use="optional">
             <xsd:annotation>
                ... see annotation ...
             </xsd:annotation>
          </xsd:attribute>
       </xsd:extension>
    </xsd:simpleContent>
 </xsd:complexType>
 <!-- ===== Primary RT: 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:BinaryObjectMimeCodeContentType">
             <xsd:annotation>
                ... see annotation ...
             </xsd:annotation>
```

```
            </xsd:attribute>
         <xsd:attribute name="encodingCode" type="xsd:normalizedString"
use="optional">
            <xsd:annotation>
               ... see annotation ...
            </xsd:annotation>
         </xsd:attribute>
         <xsd:attribute name="characterSetCode" type="xsd:normalizedString"
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. See Appendix G for the mapping of the representation terms to the user defined attributes.

[R 137] If the representation term of the relevant supplementary component is a “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 7-28: Supplementary Component is a Code

```
 <xsd:complexType name="MeasureType">
    …
    <xsd:simpleContent>
       <xsd:extension base="xsd:decimal">
          <xsd:attribute name="unitCode"
             type="clm620:MeasureUnitCodeContent" use="optional">
             …
          </xsd:attribute>
          …
       </xsd:extension>
    </xsd:simpleContent>
 </xsd:complexType>
```

[R 138] If the representation term of the relevant supplementary component is an “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 scheme.

Example 7-29: 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="clm54217:CurrencyIdentifierContentType"
use="required">
                 ...
          </xsd:attribute>
       </xsd:extension>
    </xsd:simpleContent>
 </xsd:complexType>
```

[R 139] If the representation term of the supplementary component is not “Code” or “Identifier”, then the attribute representing this supplementary component MUST be based on the appropriate built-in XSD data type.

Example 7-30: 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>
```

7.6.7 Restriction

The unqualified data types can be further restricted in the qualified data type module.

7.6.8 Annotation

[R 140] For every **`udt:UnqualifiedDataType`** **`xsd:complexType`** or **`xsd:simpleType`** definition a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references an Unqualified Data Type instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be UDT.
- DictionaryEntryName (mandatory): The official name of the Unqualified Data Type.
- VersionID (mandatory): An indication of the evolution over time of the Unqualified Data Type instance.
- Definition (mandatory): The semantic meaning of the Unqualified Data Type.
- RepresentationTermName (mandatory): The primary or secondary representation term of the associated Core Component Type.
- PrimitiveType (mandatory): The primitive data type of the Unqualified Data Type.
- BuiltInType (mandatory): The XSD built-in 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 (optional, repetitive): Example of a possible value of an Unqualified Data Type.

Example 7-31: Annotation of unqualified type definition

```
    .. see complex type definition ...
 <xsd:annotation>
    <xsd:documentation xml:lang="en">
       <ccts:UniqueID>UNDT000001</ccts:UniqueID>
       <ccts:CategoryCode>UDT</ccts:CategoryCode>
       <ccts:DictionaryEntryName>Amount. Type</ccts:DictionaryEntryName>
       <ccts:VersionID>1.0</ccts:VersionID>
       <ccts:Definition>    A number of monetary units specified in a
currency where the unit of the currency is explicit or
implied.</ccts:Definition>
       <ccts:RepresentationTermName>Amount</ccts:RepresentationTermName>
       <ccts:PrimitiveType>decimal</ccts:PrimitiveType>
       <ccts:BuiltInType>decimal</ccts:BuiltIn
Type>
</xsd:documentation>
 </xsd:annotation>
... see complex type definition ...
```

[R 141] For every supplementary component **`xsd:attribute`** declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references a Supplementary Component instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be SC.
- Dictionary Entry Name (mandatory): The official name of the Supplementary Component.
- Definition (mandatory): The semantic meaning of the Supplementary Component.
- ObjectClassTermName (mandatory): The Object Class of the Supplementary Component.
- PropertyTermName (mandatory): The Property Term of the Supplementary Component.
- RepresentationTermName (mandatory): The Representation term of the Supplementary Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Core Component.
- Example (optional, repetitive): Example of a possible value of a Basic Core Component.

Example 7-32: 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:UniqueID>UNDT000001-SC2</ccts:UniqueID>
          <ccts:CategoryCode>SC</ccts:CategoryCode>
          <ccts:DictionaryEntryName>Amount. Currency. Identifier
          </ccts:DictionaryEntryName>
          <ccts:Definition>The currency of the amount.</ccts:Definition>
          </ccts:ObjectClassTermName>Amount</ccts:ObjectClassTermName>
          <ccts:PropertyTermName>Currency</ccts:PropertyTermName>
          <ccts:RepresentationTermName>Identifier</ccts:Representation
       TermName>
          <ccts:PrimitiveType>decimal</ccts:PrimitiveType>
          <ccts:BuiltInType>decimal</ccts:BuiltInType>
          <ccts:UsageRule>ReferenceUNECE Rec 9, using 3-letter alphabetic
codes.</ccts:UsageRule>
       </xsd:documentation>
    </xsd:annotation>
 </xsd:attribute>
 ... see complex type definition ...
```

7.7 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 module naming approach. The qualified data type schema module will be used by the reusable ABIE schema module and all root schema modules.

7.7.1 Use of Qualified Data Type 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. 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 built-in XSD data type.

7.7.2 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 7-33: Structure of qualified data type schema module

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ============================================================== -->
<!-- ===== QDT Qualified Data Type Schema Module                ===== -->
<!-- ============================================================== -->
<!--
    Module of     Qualified Data Type
    Agency:       UN/CEFACT
    Version:      0.3 Rev. 6
    Last change:  25 June 2004


    Copyright (C) UN/CEFACT (2004). All Rights Reserved.

  ... see copyright information ...

-->
<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>
```

7.7.3 Namespace Scheme

[R 142] The UN/CEFACT:QualifiedDataType schema module namespace MUST be represented by the token “qdt”.

Example 7-34: Namespace name

```
"urn:un:unece:uncefact:data:draft:UNCEFACTQualifiedDataTypeSchemaModule:0.3.6
"
```

Example 7-35: Schema element

```
<xsd:schema targetNamespace="urn:un:unece:uncefact:data:draft:
    UNCEFACTQualifiedDataTypeSchemaModule:0.3.6"
  xmlns:udt="urn:un:unece:uncefact:data:draft:
    UNCEFACTUnqualifiedDataTypeSchemaModule:0.3.6"
  xmlns:qdt="urn:un:unece:uncefact:data:draft:
    UNCEFACTQualifiedDataTypeSchemaModule:0.3.6"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
```

7.7.4 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.

[R 143] The **`qdt:QualifiedDataType`** schema module MUST import the **`udt:UnqualifiedDataType`** schema 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.

7.7.5 Type Definitions

[R 144] Where required to change facets of an existing **udt:UnqualifiedDataType**, a new data type MUST be defined in the **qdt:QualifiedDataType** schema module.

[R 145] A **qdt:QualifiedDataType** MUST be based on an **unqualified data type** and add some semantic and/or technical restriction to the unqualified data type

[R 146] The name of a **qdt:QualifiedDataType** MUST be the name of its base **udt:UnqualifiedDataType** with separators and spaces removed and with its qualifier term added.

[R 147] Every **qdt:QualifiedDataType** based on a **udt:UnqualifiedDataType** **xsd:complexType** whose supplementary components map directly to the properties of a built-in **xsd:data type**

- MUST be defined as a **xsd:simpleType**
- MUST contain one **xsd:restriction** element
- MUST include **a xsd:base** attribute that defines the specific built-in XSD data type required for the content component.

[R 148] Every **qdt:QualifiedDataType** based on a **udt:UnqualifiedDataType** **xsd:complexType** whose supplementary components do not map directly to the properties of a built-in **xsd:data type**

- MUST be defined as a **xsd:complexType**
- MUST contain one **xsd:simpleContent** element
- MUST contain one **xsd:extension** element
- MUST include **the udt:UnqualifiedDataType** as its **xsd:base** attribute

[R 149] Every **qdt:QualifiedDataType** based on a **udt:UnqualifiedDataType** **xsd:simpleType**

- MUST contain one **xsd:restriction** element
- MUST include **the udt:UnqualifiedDataType** as its **xsd:base** attribute

> [Note]
>
> If a non-standard variation of the standard date time built-in data types are required, for example year month, then a qualified data type of textType needs to be defined, with the appropriate restriction specified, e.g. as a pattern, to specify the required format.

Example 7-36: Type Definitions

```
<!-- ===== Type Definitions                                        ===== -->
<!-- ================================================================== -->
<!-- ===== Qualified Data Type based on DateTime Type             ===== -->
<!-- ================================================================== -->
<!-- ===== Qualified DT: Day_ Date. Type                          ===== -->
<!-- ================================================================== -->
<xsd:simpleType name="DayDateType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:restriction base="xsd:gDay"/>
</xsd:simpleType>
...
<!-- ===== Qualified Data Type based on Text. Type                ===== -->
<!-- ================================================================== -->
<!-- ===== Qualified DT: Description_ Text. Type                  ===== -->
<!-- ================================================================== -->
<xsd:complexType name="DescriptionTextType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:simpleContent>
      <xsd:restriction base="udt:TextType"/>
   </xsd:simpleContent>
</xsd:complexType>
```

```
...
<!-- ===== Qualified Data Type based on Identifier. Type          ===== -->
<!-- ========================================================================= -->
<!-- ===== Qualified DT: Uniform Resource_ Identifier. Type       ===== -->
<!-- ========================================================================= -->
<xsd:simpleType name="UniformResouceIdentifierType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>
...
<!-- ===== Qualified DT: Country_ Identifier. Type                ===== -->
<!-- ========================================================================= -->
<xsd:simpleType name="CountryIdentifierType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:restriction base="ids53166:CountryCodeContentType"/>
</xsd:simpleType>
...
```

7.7.6 Attribute and Element Declarations

There will be no element declarations in the qualified data type schema module. Attribute names will appear in the qualified data type as defined in the unqualified data type schema module with further restrictions applied as required.

7.7.7 Extension and Restriction

[R 150] The **`qdt:QualifiedDataType`** **`xsd:complexType`** definition **`xsd:simpleContent`** element MUST only restrict attributes declared in its base type, or MUST only restrict facets equivalent to allowed supplementary components.

Example 7-37: Qualified Data Type Restricting an Identification Scheme

```
<xsd:complexType name="PartyIdentifierType">
   <xsd:annotation>
      ... see annotation ...
   </xsd:annotation>
   <xsd:simpleContent>
      <xsd:restriction base="udt:IdentifierType">
         <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>
```

7.7.8 Annotation

[R 151] Every **`qdt:QualifiedDataType`** definition MUST contain a structured set of annotations in the following sequence and pattern:

- UniqueID (mandatory): The identifier that references a Qualified Data Type instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be QDT.
- DictionaryEntryName (mandatory): The official name of the Qualified Data Type.
- VersionID (mandatory): An indication of the evolution over time of the Qualified Data Type instance.
- Definition (mandatory): The semantic meaning of the Qualified Data Type.

- RepresentationTermName (mandatory): The Representation Term of the Qualified Data Type.
- PrimitiveType (mandatory): The primitive data type of the Qualified Data Type.
- BuiltInType (mandatory): The XSD built-in data type of the Qualified Data Type.
- Data Type Qualifier Term (mandatory): A term that qualifies the Representation Term in order to differentiate it from its underlying Unqualified Data Type and other Qualified Data Type.
- BusinessProcessContext (optional, repetitive): The business process context for this Qualified Data Type is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this Qualified Data Type.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this Qualified Data Type.
- ProductContext (optional, repetitive): The product context for this Qualified Data Type.
- IndustryContext (optional, repetitive): The industry context for this Qualified Data Type.
- BusinessProcessRoleContext (optional, repetitive): The role context for this Qualified Data Type.
- SupportingRoleContext (optional, repetitive): The supporting role context for this Qualified Data Type.
- SystemCapabilitiesContext (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 7-38: Annotation of qualified data types

```
... see type definition ...
 <xsd:annotation>
    <xsd:documentation xml:lang="en">
       <ccts:UniqueID/>
       <ccts:CategoryCode>QDT</ccts:CategoryCode>
       <ccts:DictionaryEntryName>Account_ Type_ Code.
Type</ccts:DictionaryEntryName>
       <ccts:VersionID>1.0</ccts:VersionID>
       <ccts:Definition>This code represents the type of an account.
       </ccts:Definition>
      <ccts:RepresentationTermName>Code</ccts:RepresentationTermName>
       <ccts:RepresentationTermQualifier>Account</ccts:
     RepresentationTermQualifier>
       <ccts:RepresentationTermQualifier>Type</ccts:
     RepresentationTermQualifier>
      <ccts:PrimitiveType>string</ccts:PrimitiveType>
      <ccts:BuiltInType>normalizedString</ccts:BuiltInType>
    </xsd:documentation>
 </xsd:annotation>
... see type definition ...
```

[R 152] For every supplementary component **`xsd:attribute`** declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references a Supplementary Component of a Core Component Type instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be QDT.
- Dictionary Entry Name (mandatory): The official name of a Supplementary Component.

- VersionID (mandatory): An indication of the evolution over time of a Supplementary Component instance.
- Definition (mandatory): The semantic meaning of a Supplementary Component.
- Cardinality (mandatory): Indication whether the Supplementary Component Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the Core Component Type.
- PropertyTermName (optional): The Property Term of the associated Supplementary Component.
- RepresentationTermName (optional): The Representation Term of the associated Suppplementary Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Component.
- BusinessProcessContext (optional, repetitive): The business process with which this Supplementary Component is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this Supplementary Component.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this Supplementary Component.
- ProductContext (optional, repetitive): The product context for this Supplementary Component.
- IndustryContext (optional, repetitive): The industry context for this Supplementary Component.
- BusinessProcessRoleContext (optional, repetitive): The role context for this Qualified Data Type.
- SupportingRoleContext (optional, repetitive): The supporting role context for this Supplementary Component.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this Supplementary Component.
- Example (optional, repetitive): Example of a possible value of a Supplementary Component.

7.8 Code Lists

Codes are an integral component of any business to business information flow as they facilitate the ability of the flow to be machine understandable. 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 sectorial 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.

[R 153] 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.

[R 154] Internal code list schema MUST NOT duplicate existing external code list schema when the existing ones are available to be imported.

7.8.1 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 7-39: Structure of code lists

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ============================================================================ -->
<!-- =====  Code List: Account Type Code ; UNECE                          ===== -->
<!-- ============================================================================ -->
<!--
    Codelist of   Account Type Code
    Agency:       UNECE
    Version:      D.01C
    Last change:  25 June 2004

    Copyright (C) UN/CEFACT (2004). All Rights Reserved.

  ... see copyright information ...

-->
<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                                              ===== -->
  <!-- ============================================================================ -->
  <!-- ===== Code List Type Definition: Account Type Code                  ===== -->
  <!-- ============================================================================ -->
    ... see type definition ...
</xsd:schema>
```

7.8.2 Namespace Name for Code Lists

The namespace name for code list is somewhat 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 codelist 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>

[R 155] The names for namespaces MUST have the following structure while the schema is at draft status:

urn:un:unece:uncefact:codelist:draft:<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
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.

Example 7-40: 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 7-41: 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 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
1.2 = the value for Code List. Version. Identifier
```

[R 156] The namespace names for schema holding specification status MUST be of the form:

urn:un:unece:uncefact:codelist:standard:<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
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.

Example 7-42: 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 7-43: Namespace name of proprietary code list at standard status

```
"urn:un:unece:uncefact:codelist:standard:Security_Initiative:Document_Securit
y: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 our control. As 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.

7.8.3 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.

[R 157] 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 7-44: 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 7-45: 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 7-46: 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 7-47.

Example 7-47: 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.

7.8.4 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>

[R 158] The structure for schema location of code lists MUST be:

http://www.unece.org/uncefact/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 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.

[R 159] Each **xsd:schemaLocation** attribute declaration of a code list MUST contain a persistent and resolvable URL.

[R 160] Each **xsd:schemaLocation** attribute declaration URL of a code list MUST contain an absolute path.

7.8.5 Imports and Includes

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

[R 161] Code List schema modules MUST not import or include any other schema modules.

7.8.6 Type Definitions

[R 162] Within each code list module one, and only one, named **xsd:simpleType** MUST be defined for the content component.

[R 163] The name of the **xsd:simpleType** MUST be the name of root element based on the value of the `code list name text` with the word “ContentType” appended.

Example 7-48: Simple type definition of code lists

```
<!-- ===== Type Definitions                                        ===== -->
<!-- ================================================================== -->
<!-- ===== Code List Type Definition: Account Type Code            ===== -->
<!-- ================================================================== -->
<xsd:simpleType name="AccountTypeCodeContentType">
   <xsd:restriction base="xsd:token">
      <xsd:enumeration value="2">
```

```
            ... see enumeration ...
         </xsd:enumeration>
      </xsd:restriction>
   </xsd:simpleType>
```

[R 164] The **`xsd:restriction`** element base attribute value MUST be set to “xsd:token”.

[R 165] 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 7-49: Enumeration facet of code lists

```
... see type defintion ...
 <xsd:enumeration value="2">
    <xsd:annotation>
       ... see annotation
    </xsd:annotation>
 </xsd:enumeration>
 <xsd:enumeration value="15">
    <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. Therefore, no other facet restrictions are allowed.

[R 166] Facets other than **`xsd:enumeration`** MUST NOT be used in the code list schema module.

7.8.7 Element Declarations

Each code list schema module will contain the list of enumerations allowed for a particular element.

[R 167] For each code list a single root element MUST be globally declared.

[R 168] The name of root element MUST be based on the `code list name text` following the naming rules as defined in section 5.3.

[R 169] The root element MUST be of a type representing the actual list of code values.

Example 7-50: Root element declaration of code lists

```
 <!-- ===== Root Element                                          ===== -->
 <!-- ========================================================================= -->
 <xsd:element name="AccountTypeCode"
type="clm64437:AccountTypeCodeContentType"/>
```

7.8.8 Extension and Restriction

Users of the UN/CEFACT library may identify any subset 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 code lists could be

- a combination of several individual code lists using `xsd:union`
- a choice between several code lists, using `xsd:choice`

Both of these can easily be accommodated in this syntax solution as required by the user’s business requirements.

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

Example 7-51: Usage of only one Code List

```
 <xsd:simpleType name="TemperatureMeasureUnitCodeType">
    <xsd:annotation>
       ... see annotation ...
    </xsd:annotation>
    <xsd:restriction base="clm66411:UnitCodeContentType">
       <xsd:length value="3"/>
```

```
      <xsd:enumeration value="BTU">
         <xsd:annotation>
            <xsd:documentation source="code" xml:lang="en">
               <ccts:CodeName>British thermal unit</ccts:CodeName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="CEL">
         <xsd:annotation>
            <xsd:documentation source="code" xml:lang="en">
               <ccts:CodeName>degree Celsius</ccts:CodeName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="FAH">
         <xsd:annotation>
            <xsd:documentation source="code" xml:lang="en">
               <ccts:CodeName>degree Fahrenheit</ccts:CodeName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:enumeration>
   </xsd:restriction>
</xsd:simpleType>
```

Example 7-52: 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>
```

Example 7-53: Combination of Code Lists

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

7.8.9 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.

[R 170] Each **`xsd:enumeration`** MUST include an annotation documentation providing the code name and the code description.

Example 7-54: Annotation of codes

```
<xsd:enumeration value="2">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         <ccts:CodeName>Budgetary account</ccts:CodeName>
         <ccts:CodeDescription>Code identifying a budgetary account.
         </ccts:CodeDescription>
      </xsd:documentation>
   </xsd:annotation>
</xsd:enumeration>...
```

7.9 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.

[R 171] Internal identifier lists schema MUST NOT duplicate existing external identifier list schema when the existing ones are available to be imported.

[R 172] Each UN/CEFACT maintained identifier list MUST be defined in its own schema module.

7.9.1 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 7-55: Structure of identifier lists

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ================================================================== -->
<!-- =====  ISO Country Identifier – Identifier List Schema Module  ===== -->
<!-- ================================================================== -->
<!--
    Identifier of Country Identifier
    Agency:       ISO
    Version:      2
    Last change:  25 June 2004

    Copyright (C) UN/CEFACT (2004). All Rights Reserved.

  ... see copyright information ...

-->
<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                                      ===== -->
  <!-- ================================================================== -->
  <!-- ===== Identifier List Type Definition: Country Identifier
===== -->
  <!-- ================================================================== -->
    ... see type definition ...
</xsd:schema>
```

7.9.2 Namespace Name for Identifier List Schema

The namespace name for identifier list is somewhat 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>

[R 173] The names for namespaces MUST have the following structure while the schema is at draft status:

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

Where:
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.

Example 7-56: 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:5:4217:2001"

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

[R 174] The namespace names for identifier list schema holding specification status MUST be of the form:

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

Where:
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 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.

Example 7-57: Namespace of an identifier list schema with an agency and an identifier list schema identifier at standard status

```
"urn:un:unece:uncefact:identifierlist:standard:5:4217:2001"

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

Versioning for identifier list schemas published by external organisations is outside our control. As 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.

7.9.3 UN/CEFACT XSD Schema 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 directory. The identifier of the identification list will be the identifier as allocated by the identification scheme agency.

[R 175] 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>

Example 7-58: 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 4-37, a namespace declaration for an identifier list would appear as shown in Example 4-38.

Example 7-59: 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.

7.9.4 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>

[R 176] The structure for schema location of identifier lists MUST be:

http://www.unece.org/uncefact/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.

[R 177] Each **xsd:schemaLocation** attribute declaration of an identifier list schema MUST contain a persistent and resolvable URL.

[R 178] Each **xsd:schemaLocation** attribute declaration URL of an identifier list schema MUST contain an absolute path.

7.9.5 Imports and Includes

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

[R 179] Identifier list schema modules MUST NOT import or include any other schema modules.

7.9.6 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.

[R 180] Within each identifier list schema module one, and only one, named **xsd:simpleType** MUST be defined for the content component.

[R 181] The name of the **xsd:simpleType** MUST be the name of root element with the word "ContentType" appended.

Example 7-60: Simple type definition of an identifier list

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

[R 182] The **xsd:restriction** element base attribute value MUST be set to "xsd:token".

[R 183] 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 7-61: Enumeration facet of an identifier list

```
... see type defintion ...
<xsd:enumeration value="AU">
   <xsd:annotation>
      ... see annotation
   </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="US">
   <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.

[R 184] Facets other than **`xsd:enumeration`** MUST NOT be used in the identifier list schema module.

7.9.7 Attribute and Element Declarations

Each identifier list schema module will contain a list of enumerations allowed for a particular element.

[R 185] For each identifier list a single root element MUST be globally declared.

[R 186] The name of the root element MUST be based on the **`identification scheme. name. text`** following the naming rules as defined in section 5.3.

[R 187] The root element MUST be of a type representing the actual list of identifier values.

Example 7-62: Root element declaration of identifier lists

```
  <!-- ===== Root Element                                              ===== -->
  <!-- ================================================================== -->
  <xsd:element name="CountryIdentifier"
type="ids53166:CountryIdentifierContentType"/>
```

7.9.8 Extension and Restriction

Users of the UN/CEFACT library may identify any subset 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`**

Both of these can easily be accommodated in this syntax solution as required by the user's business requirements.

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

Example 7-63: 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>
  ... see type definition ...
```

Example 7-64: Usage of only one identifier scheme

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

Example 7-65: Usage of alternative identifier schemes

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

7.9.9 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.

[R 188] Each **`xsd:enumeration`** MUST include an annotation documentation providing the identifier name and optionally the description of the identifier.

Example 7-66: Annotation of Identifiers

```
<xsd:enumeration value="AU">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         <ccd:IdentifierName>Australia</ccd:IdentifierName>
      </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/IETF/ITU/UNCEFACT Memorandum of Understanding Management Group (MOUMG) Resolution 01/08 (MOU/MG01n83) as agreed to by UN/CEFACT, 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.

[R 189] 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 Empty Content

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

[R 190] UN/CEFACT conformant instance documents MUST NOT contain an element devoid of content.

[R 191] The **`xsi:nil`** attribute MUST NOT appear in any conforming instance.

8.3 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.

[R 192] 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*
- *ebXML Technical Architecture Specification v1.04*
- *OASIS/ebXML Registry Information Model v2.0*
- *ebXML Requirements Specification v1.06*
- *Information Technology - Metadata registries: Framework for the Specification and Standardization of Data Elements, International Standardization Organization, ISO 11179-1*
- *Information Technology - Metadata registries: Classification of Concepts for the Identification of Domains, International Standardization Organization, ISO 11179-2*
- *Information Technology - Metadata registries: Registry Metamodel, International Standardization Organization, ISO 11179-3*
- *Information Technology - Metadata registries: Rules and Guidelines for the Formulation of Data Definitions, International Standardization Organization, ISO 11179-4*
- *Information Technology - Metadata registries: Naming and Identification Principles for Data Elements, International Standardization Organization, ISO 11179-5*
- *Information Technology - Metadata registries: Framework for the Specification and Standardization of Data Elements, International Standardization Organization, ISO 11179-6*

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
- Root element
- Type Definitions

B.1 XML Declaration

A UTF-8 encoding is adopted throughout all UN/CEFACT XML schema.

Example B-1: XML Declaration

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

B.2 Schema Module Identification and Copyright Information

Example B-2: Copyright Information

```
<!-- ================================================================== -->
<!-- =====  Examples Schema Module; 0.3 Rev.6                     ===== -->
<!-- ================================================================== -->
<!--
    Module:        Example
    Agency:        UN/CEFACT
    Version:       0.3 Rev. 6
    Last change:   25 June 2004

    Copyright (C) UN/CEFACT (2004). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its
implementation may be prepared, copied, published and distributed, in whole or in
part, without restriction of any kind, provided that the above copyright notice and
this paragraph are included on all such copies and derivative works. However, this
document itself may not be modified in any way, such as by removing the copyright
notice or references to UN/CEFACT, except as needed for the purpose of developing
UN/CEFACT specifications, in which case the procedures for copyrights defined in the
UN/CEFACT Intellectual Property Rights document must be followed, or as required to
translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by
UN/CEFACT or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and
UN/CEFACT DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
-->
```

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 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
- 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
targetNamespace="urn:un:unece:uncefact:data:draft:UNCEFACTExamplesSchemaModule:0.3.6"
xmlns:rsm="urn:un:unece:uncefact:data:draft:UNCEFACTExamplesSchemaModule:0.3.6"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ram="urn:un:unece:uncefact:data:draft:UNCEFACTReusableAggregateBusinessInformati
onEntitySchemaModule:0.3.6"
xmlns:udt="urn:un:unece:uncefact:data:draft:UNCEFACTUnqualifiedDataTypeSchemaModule:0.
3.6"
xmlns:qdt="urn:un:unece:uncefact:data:draft:UNCEFACTQualifiedDataTypeSchemaModule:0.3.
6"
xmlns:ids53166="urn:un:unece:uncefact:codelist:draft:5:3166-1:1997"
xmlns:ids53166-2="urn:un:unece:uncefact:codelist:draft:5:3166-2:1998"
xmlns:clm65153="urn:un:unece:uncefact:codelist:draft:6:5153:D.01C"
xmlns:clm64405="urn:un:unece:uncefact:codelist:draft:6:4405:D.01C "
xmlns:clm69143="urn:un:unece:uncefact:codelist:draft:6:9143:D.01C "
xmlns:clmPerson_Characteristic_Code63289="urn:un:unece:uncefact:codelist:draft:
6:3289:D.01C" xmlns:clm63479="urn:un:unece:uncefact:codelist:draft:6:3479:D.01C"
xmlns:clm63499="urn:un:unece:uncefact:codelist:draft:6:3499:D.01C"
xmlns:clm1161131="urn:un:unece:uncefact:codelist:draft:11:61131:4031"
xmlns: clm66411="urn:un:unece:uncefact:codelist:draft:6:6411:2001"
xmlns:clm54217="urn:un:unece:uncefact:codelist:draft:5:4217:2001"
xmlns:clm5639="urn:un:unece:uncefact:codelist:draft:5:639:1988"
xmlns:clm64437="urn:un:unece:uncefact:codelist:draft:6:4437:D.01C"

elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

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:draft:UNCEFFACTInternalAggregateBusinessInformat
ionEntitySchemaModule:0.3.6"
schemaLocation="http://www.unece.org/uncefact/data/UNCEFACTInternalAggregateBusinessIn
formationEntitySchemaModule_0.3.6_draft.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 reusable UN/CEFACTAggregate Business Information Entity == -->
<!-- ========================================================== -->
 <xsd:import namespace="
urn:un:unece:uncefact:data:draft:UNCEFACTReusableAggregateBusinessInformationSchemaMod
ule:0.3.6" schemaLocation=" http://www.unece.org/uncefact/data/
UNCEFACTReusableAggregateBusinessInformationEntitySchemaModule_0.3.6_draft.xsd"/>
<!-- ========================================================== -->
<!-- =====  Import of Unqualified Data Type                  ===== -->
<!-- ========================================================== -->
<xsd:import
namespace="urn:un:unece:uncefact:data:draft:UNCEFACTUnqualifiedDataTypeSchemaModule:0.
3.6" schemaLocation=" http://www.unece.org/uncefact/data/
UNCEFACTUnqualifiedDataTypeSchemaModule_0.3.6_draft.xsd"/>
<!-- ========================================================== -->
<!-- =====  Import of Qualified Data Type                    ===== -->
<!-- ========================================================== -->
<xsd:import
namespace="urn:un:unece:uncefact:data:draft:UNCEFACTQualifiedDataTypeSchemaModule:0.3.
6" schemaLocation=" http://www.unece.org/uncefact/data/
UNCEFACTQualifiedDataTypeSchemaModule_0.3.6_draft.xsd"/>
<!-- ========================================================== -->
<!-- =====  Import of Code lists                             ===== -->
<!-- ========================================================== -->
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:6:4437:D.01C"
schemaLocation="http://www.unece.org/uncefact/codelist/64437_D.01C_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:6:6411:2001"
schemaLocation=" http://www.unece.org/uncefact/codelist/66411_2001_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:5:4217:2001"
schemaLocation=" http://www.unece.org/uncefact/codelist/54217_2001_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:5:639-1:1988"
schemaLocation="http://www.unece.org/uncefact/codelist/5639-1.1988_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:11:61131:4031"
schemaLocation="http://www.unece.org/uncefact/codelist/1161131_4031_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:6:3499:D.01C"
schemaLocation="http://www.unece.org/uncefact/codelist/63499_D.01C_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:6:3479:D.01C"
schemaLocation="http://www.unece.org/uncefact/codelist/63479_D.01C_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:6:3289:D.01C"
schemaLocation="http://www.unece.org/uncefact/codelist/63289_D.01C_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:6:9143:D.01C"
schemaLocation="http://www.unece.org/uncefact/codelist/69143_D.01C_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:6:4405:D.01C"
schemaLocation="http://www.unece.org/uncefact/codelist/64405_D.01C_draft.xsd"/>
<xsd:import namespace="urn:un:unece:uncefact:codelist:draft:6:5153:D.01C"
schemaLocation="http://www.unece.org/uncefact/codelist/65153_D.01C_draft.xsd"/>
<!-- ========================================================== -->
<!-- =====  Import of Identifier Schemes                       ===== -->
<!-- ========================================================== -->
 <xsd:import namespace="urn:un:unece:uncefact:codelist:draft:5:3166-1:1997"
schemaLocation="http://www.unece.org/uncefact/identiferlist/53166-1_1997_draft.xsd"/>
 <xsd:import namespace="urn:un:unece:uncefact:codelist:draft:5:3166-2:1998"
schemaLocation="http://www.unece.org/uncefact/identifierlist/53166-2_1998_draft.xsd"/>
```

B.6 Root element

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-6:

```
<!-- ================================================================== -->
<!-- =====  Root element                                          ===== -->
<!-- ================================================================== -->
 <xsd:element name="PurchaseOrder" type="exp:PurchaseOrderType">
    <xsd:annotation>
       <xsd:documentation>
          <ccts:UniqueID>UNM0000001</ccts:UniqueID>
          <ccts:CategoryCode>RSM</ccts:CategoryCode>
          <ccts:Name>PurchaseOrder</ccts:Name>
          <ccts:VersionID>1.0</ccts:VersionID>
          <ccts:Description>A document that contains information directly relating to
             the economic event of ordering products.</ccts:Description>
          <ccts:BusinessDomain>TBG1</ccts:BusinessDomain>
          <ccts:BusinessProcessContext>Purchase Order</ccts:BusinessProcessContext>
       </xsd:documentation>
    </xsd:annotation>
 </xsd:element>
```

B.7 Type Definitions

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

Example B-7:

```
<!-- ================================================================== -->
<!-- =====  Type Definitions                                      ===== -->
<!-- ================================================================== -->
<!-- =====  Type Definitions: Account type                        ===== -->
<!-- ================================================================== -->
 <xsd:complexType name="AccountType">
    <xsd:annotation>
       <xsd:documentation xml:lang="en">
          <ccts:UniqueID>UN00000001</ccts:UniqueID>
          <ccts:CategoryCode>ABIE</ccts:CategoryCode>
          <ccts:DictionaryEntryName>Account. Details</ccts:DictionaryEntryName>
          <ccts:VersionID>1.0</ccts:VersionID>
        <ccts:Definition>A business arrangement whereby debits and/or credits arising
from transactions are recorded. This could be with a bank, i.e. a financial account,
or a trading partner offering supplies or services 'on account', i.e. a commercial
account</ccts:Definition>
       <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
       </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
       <xsd:element name="ID" type="udt:IdentifierType" minOccurs="0"
maxOccurs="unbounded">
          <xsd:annotation>
             <xsd:documentation xml:lang="en">
                <ccts:UniqueID>UN00000002</ccts:UniqueID>
                <ccts:CategoryCode>BBIE</ccts:CategoryCode>
                <ccts:DictionaryEntryName>Account.
Identifier</ccts:DictionaryEntryName>
                <ccts:VersionID>1.0</ccts:VersionID>
                <ccts:Definition>The identification of a specific
account.</ccts:Definition>
                <ccts:Cardinality>0..n</ccts:Cardinality>
                <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
                <ccts:PropertyTermName>Identifier</ccts:PropertyTermName>
                <ccts:RepresentationTermName>Identifier</ccts:RepresentationTermName>
                <ccts:BusinessTermName>Account Number</ccts:BusinessTermName>
             </xsd:documentation>
          </xsd:annotation>
       </xsd:element>
       <xsd:element name="Status" type="ram:StatusType" minOccurs="0"
maxOccurs="unbounded">
          <xsd:annotation>
             <xsd:documentation xml:lang="en">
```

```
               <ccts:UniqueID>UN00000003</ccts:UniqueID>
               <ccts:CategoryCode>ASBIE</ccts:CategoryCode>
               <ccts:DictionaryEntryName>Account. Status</ccts:DictionaryEntryName>
               <ccts:VersionID>1.0</ccts:VersionID>
               <ccts:Definition>Status information related to account
details.</ccts:Definition>
               <ccts:Cardinality>0..n</ccts:Cardinality>
               <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
               <ccts:PropertyTermName>Status</ccts:PropertyTermName>
               <ccts:RepresentationTermName>Code</ccts:RepresentationTermName>

 <ccts:AssociatedObjectClassTermName>Status</ccts:AssociatedObjectClassTermName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Name" type="udt:NameType" minOccurs="0"
maxOccurs="unbounded">
         <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:UniqueID>UN00000004</ccts:UniqueID>
               <ccts:CategoryCode>BBIE</ccts:CategoryCode>
               <ccts:DictionaryEntryName>Account. Name.
Text</ccts:DictionaryEntryName>
               <ccts:VersionID>1.0</ccts:VersionID>
               <ccts:Definition>The text name for a specific account</ccts:Definition>
               <ccts:Cardinality>0..n</ccts:Cardinality>

               <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
               <ccts:PropertyTermName>Name</ccts:PropertyTermName>
               <ccts:RepresentationTermName>Text</ccts:RepresentationTermName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="CurrencyCode" type="qdt:CurrencyCodeType" minOccurs="0"
maxOccurs="unbounded">
         <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:UniqueID>UN00000005</ccts:UniqueID>
               <ccts:CategoryCode>BBIE</ccts:CategoryCode>
               <ccts:DictionaryEntryName>Account. Currency.
Code</ccts:DictionaryEntryName>
               <ccts:VersionID>1.0</ccts:VersionID>
               <ccts:Definition>A code specifying the currency in which monies are
held within the account.</ccts:Definition>
               <ccts:Cardinality>0..n</ccts:Cardinality>
               <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
               <ccts:PropertyTermName>Currency</ccts:PropertyTermName>
               <ccts:RepresentationTermName>Code</ccts:RepresentationTermName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="TypeCode" type="qdt:AccountTypeCodeType" minOccurs="0"
maxOccurs="unbounded">
         <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:UniqueID>UN00000006</ccts:UniqueID>
               <ccts:CategoryCode>BBIE</ccts:CategoryCode>
               <ccts:DictionaryEntryName>Account. Type.
Code</ccts:DictionaryEntryName>
               <ccts:VersionID>1.0</ccts:VersionID>
               <ccts:Definition>This provides the ability to indicate what type of
account this is (checking, savings, etc).</ccts:Definition>
               <ccts:Cardinality>0..1<ccts:Cardinality>
               <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
               <ccts:PropertyTermName>Type</ccts:PropertyTermName>
               <ccts:RepresentationTermName>Code</ccts:RepresentationTermName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Country" type="ram:CountryType" minOccurs="0"
maxOccurs="unbounded">
         <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:UniqueID>UN00000007</ccts:UniqueID>
               <ccts:CategoryCode>ASBIE</ccts:CategoryCode>
               <ccts:DictionaryEntryName>Account. Country</ccts:DictionaryEntryName>
```

```
               <ccts:VersionID>1.0</ccts:VersionID>
               <ccts:Definition>Country information related to account
details.</ccts:Definition>
               <ccts:Cardinality>0..n<ccts:Cardinality>
               <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
               <ccts:PropertyTermName>Country</ccts:PropertyTermName>

 <ccts:AssociatedObjectClassTermName>Country</ccts:AssociatedObjectClassTermName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Person" type="ram:PersonType" minOccurs="0"
maxOccurs="unbounded">
         <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:UniqueID>UN00000008</ccts:UniqueID>
               <ccts:CategoryCode>ASBIE</ccts:CategoryCode>
               <ccts:DictionaryEntryName>Account. Person</ccts:DictionaryEntryName>
               <ccts:VersionID>1.0</ccts:VersionID>
               <ccts:Definition>Associated person information related to account
details.  This can be used to identify multiple people related to an account, for
instance, the account holder.</ccts:Definition>
               <ccts:Cardinality>0..n<ccts:Cardinality>
               <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
               <ccts:PropertyTermName>Person</ccts:PropertyTermName>

 <ccts:AssociatedObjectClassTermName>Person</ccts:AssociatedObjectClassTermName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Organisation" type="ram:OrganisationType" minOccurs="0"
maxOccurs="unbounded">
         <xsd:annotation>
            <xsd:documentation xml:lang="en">
               <ccts:UniqueID>UN00000009</ccts:UniqueID>
               <ccts:CategoryCode>ASBIE</ccts:CategoryCode>
               <ccts:DictionaryEntryName>Account.
Organisation</ccts:DictionaryEntryName>
               <ccts:VersionID>1.0</ccts:VersionID>
               <ccts:Definition>The associated organisation information related to
account details.  This can be used to identify multiple organisations related to this
account, for instance, the account holder.</ccts:Definition>
               <ccts:Cardinality>0..n<ccts:Cardinality>
               <ccts:ObjectClassTermName>Account</ccts:ObjectClassTermName>
               <ccts:PropertyTermName>Organisation</ccts:PropertyTermName>

 <ccts:AssociatedObjectClassTermName>Organisation</ccts:AssociatedObjectClassTermName>
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
   </xsd:sequence>
 </xsd:complexType>
```

Example B-8: Complete Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ================================================================== -->
<!-- =====  [MODULENAME] Schema Module; [VERSION]                 ===== -->
<!-- ================================================================== -->
<!--
   Module:        [MODULENAME]
   Agency:        UN/CEFACT
   Version:       [VERSION]
   Last change:   [DATE OF LAST CHANGE]

   Copyright (C) UN/CEFACT (2004). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its
implementation may be prepared, copied, published and distributed, in whole or in
part, without restriction of any kind, provided that the above copyright notice and
this paragraph are included on all such copies and derivative works. However, this
document itself may not be modified in any way, such as by removing the copyright
notice or references to UN/CEFACT, except as needed for the purpose of developing
UN/CEFACT specifications, in which case the procedures for copyrights defined in the
UN/CEFACT Intellectual Property Rights document must be followed, or as      required
to translate it into languages other than English.
```

```
The limited permissions granted above are perpetual and will not be revoked by
UN/CEFACT or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and
UN/CEFACT DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL  NOT INFRINGE ANY RIGHTS OR
ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
-->
<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="…"/>
<!-- ================================================================== -->
<!-- =====  Root element                                            ===== -->
<!-- ================================================================== -->
 <xsd:element name="[ELEMENTNAME]" type="[TOKEN]:[TYPENAME]>
<!-- ================================================================== -->
<!-- =====  Type Definitions                                        ===== -->
<!-- ================================================================== -->
<!-- =====  Type Definitions: [TYPE]                                ===== -->
<!-- ================================================================== -->
 <xsd:complexType name="[TYPENAME]">
    <xsd:restriction base="xsd:token">
       ... see type definition ....
    </xsd:restriction>
 </xsd:complexType>
</xsd:schema>
```

Appendix C. ATG Approved Acronyms and Abbreviations

The following constitutes a list of ATG 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. Core Component Schema Module

The Core Component Schema Module is published as a separate file – CoreComponentType.xsd.

Appendix E. Unqualified Data Type Schema Module

The Unqualified Data Type Schema Module is published as a separate file – UnqualifiedDataType.xsd.

Appendix F. 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:CategoryCode>RSM</ccts:CategoryCode>
                <ccts:Name></ccts:Name>
                <ccts:VersionID></ccts:VersionID>
                <ccts:Description></ccts:Description>
                <ccts:BusinessDomain></ccts:BusinessDomain>
                <ccts:BusinessProcessContext></ccts:BusinessProcessContext>
                <ccts:GeopoliticalOrRegionContext></ccts:GeopoliticalOrRegionContext>
                <ccts:OfficialConstraintContext></ccts:OfficialConstraintContext>
                <ccts:ProductContext></ccts:ProductContext>
                <ccts:IndustryContext></ccts:IndustryContext>
                <ccts:BusinessProcessRoleContext></ccts:BusinessProcessRoleContext>
                <ccts:SupportingRoleContext></ccts:SupportingRoleContext>
                <ccts:SystemCapabilitiesContext></ccts:SystemCapabilitiesContext>
            </xsd:documentation>
        </xsd:annotation>

<!--  ABIE Documentation -->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:CategoryCode>ABIE</ccts:CategoryCode>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:VersionID></ccts:VersionID>
                <ccts:Definition></ccts:Definition>
                <ccts:ObjectClassTermName></ccts:ObjectClassTermName>
                <ccts:ObjectClassQualifierTermName></ccts:ObjectClassQualifierTermName>
                <ccts:BusinessProcessContext></ccts:BusinessProcessContext>
                <ccts:GeopoliticalOrRegionContext></ccts:GeopoliticalOrRegionContext>
                <ccts:OfficialConstraintContext></ccts:OfficialConstraintContext>
                <ccts:ProductContext></ccts:ProductContext>
                <ccts:IndustryContext></ccts:IndustryContext>
                <ccts:BusinessProcessRoleContext></ccts:BusinessProcessRoleContext>
                <ccts:SupportingRoleContext></ccts:SupportingRoleContext>
                <ccts:SystemCapabilitiesContext></ccts:SystemCapabilitiesContext>
                <ccts:UsageRule></ccts:UsageRule>
                <ccts:BusinessTermName></ccts:BusinessTermName>
                <ccts:Example></ccts:Example>
            </xsd:documentation>
        </xsd:annotation>

<!--  BBIE Documentation -->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:CategoryCode>ABIE</ccts:CategoryCode>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:VersionID></ccts:VersionID>
                <ccts:Definition></ccts:Definition>
                <ccts:Cardinality></ccts: Cardinality>
                <ccts:ObjectClassTermName></ccts:ObjectClassTermName>
                <ccts:ObjectClassQualifierTermName></ccts:ObjectClassQualifierTermName>
```

```
                <ccts:PropertyTermName></ccts:PropertyTermName>
                <ccts:PropertyQualifierTermName></ccts:PropertyQualifierTermName>
                <ccts:RepresentationTermName></ccts:RepresentationTermName
                <ccts:BusinessProcessContext></ccts:BusinessProcessContext>
                <ccts:GeopoliticalOrRegionContext></ccts:GeopoliticalOrRegionContext>
                <ccts:OfficialConstraintContext></ccts:OfficialConstraintContext>
                <ccts:ProductContext></ccts:ProductContext>
                <ccts:IndustryContext></ccts:IndustryContext>
                <ccts:BusinessProcessRoleContext></ccts:BusinessProcessRoleContext>
                <ccts:SupportingRoleContext></ccts:SupportingRoleContext>
                <ccts:SystemCapabilitiesContext></ccts:SystemCapabilitiesContext>
                <ccts:UsageRule></ccts:UsageRule>
                <ccts:BusinessTermName></ccts:BusinessTermName>
                <ccts:Example></ccts:Example>
            </xsd:documentation>
        </xsd:annotation>

<!--  ASBIE Documentation -->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:CategoryCode>ABIE</ccts:CategoryCode>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:VersionID></ccts:VersionID>
                <ccts:Definition></ccts:Definition>
                <ccts:Cardinality></ccts: Cardinality>
                <ccts:ObjectClassTermName></ccts:ObjectClassTermName>
                <ccts:ObjectClassQualifierTermName></ccts:ObjectClassQualifierTermName>
                <ccts:PropertyTermName></ccts:PropertyTermName>
                <ccts:PropertyQualifierTermName></ccts:PropertyQualifierTermName>
                <ccts:AssociatedObjectClassTermName></ccts:AssociatedObjectClassTermName>
                <ccts: AssociatedObjectClassQualifierTermName></ccts:
AssociatedObjectClassQualifierTermName>
                <ccts:BusinessProcessContext></ccts:BusinessProcessContext>
                <ccts:GeopoliticalOrRegionContext></ccts:GeopoliticalOrRegionContext>
                <ccts:OfficialConstraintContext></ccts:OfficialConstraintContext>
                <ccts:ProductContext></ccts:ProductContext>
                <ccts:IndustryContext></ccts:IndustryContext>
                <ccts:BusinessProcessRoleContext></ccts:BusinessProcessRoleContext>
                <ccts:SupportingRoleContext></ccts:SupportingRoleContext>
                <ccts:SystemCapabilitiesContext></ccts:SystemCapabilitiesContext>
                <ccts:UsageRule></ccts:UsageRule>
                <ccts:BusinessTermName></ccts:BusinessTermName>
                <ccts:Example></ccts:Example>
            </xsd:documentation>
                </xsd:annotation>

<!-- Qualified Data Type Documentation -->
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                <ccts:UniqueID></ccts:UniqueID>
                <ccts:CategoryCode>QDT</ccts:CategoryCode>
                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
                <ccts:VersionID></ccts:VersionID>
                <ccts:Definition></ccts:Definition>
                <ccts:RepresentationTermName></ccts:RepresentationTermName>
                <ccts:DataTypeQualifierTermName></ccts:DataTypeQualifierTermName>
                <ccts:PrimitiveType></ccts:PrimitiveType>
                <xsd:BuiltInType></xsd: BuiltInType>
                <ccts:BusinessProcessContext></ccts:BusinessProcessContext>
```

```
3747                <ccts:GeopoliticalOrRegionContext></ccts:GeopoliticalOrRegionContext>
3748                <ccts:OfficialConstraintContext></ccts:OfficialConstraintContext>
3749                <ccts:ProductContext></ccts:ProductContext>
3750                <ccts:IndustryContext></ccts:IndustryContext>
3751                <ccts:BusinessProcessRoleContext></ccts:BusinessProcessRoleContext>
3752                <ccts:SupportingRoleContext></ccts:SupportingRoleContext>
3753                <ccts:SystemCapabilitiesContext></ccts:SystemCapabilitiesContext>
3754                <ccts:UsageRule></ccts:UsageRule>
3755                <ccts:BusinessTermName></ccts:BusinessTermName>
3756                <ccts:Example></ccts:Example>
3757             </xsd:documentation>
3758          </xsd:annotation>
3759
3760    <!-- Unqualified Data Type Documentation-->
3761          <xsd:annotation>
3762             <xsd:documentation xml:lang="en">
3763                <ccts:UniqueID></ccts:UniqueID>
3764                <ccts:CategoryCode>CCT</ccts:CategoryCode>
3765                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
3766                <ccts:VersionID></ccts:VersionID>
3767                <ccts:Definition></ccts:Definition>
3768                <ccts:RepresentationTermName></ccts:RepresentationTermName>
3769                <ccts:PrimitiveType></ccts:PrimitiveType>
3770                <xsd:BuiltInType></xsd:BuiltInType>
3771                <ccts:UsageRule></ccts:UsageRule>
3772                <ccts:BusinessTermName></ccts:BusinessTermName>
3773                <ccts:Example></ccts:Example>
3774             </xsd:documentation>
3775          </xsd:annotation>
3776
3777    <!-- Unqualified Data Type Supplementary Component Documentation-->
3778          <xsd:annotation>
3779             <xsd:documentation xml:lang="en">
3780                <ccts:UniqueID></ccts:UniqueID>
3781                <ccts:CategoryCode>SC</ccts:CategoryCode>
3782                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
3783                <ccts:Definition></ccts:Definition>
3784                <ccts:ObjectClassTermNam></ccts: ObjectClassTermName>
3785                <ccts:PropertyTermName></ccts:PropertyTermName>
3786                <ccts:RepresentationTermName></ccts:RepresentationTermName>
3787                <ccts:ObjectClassTermeName></ccts:ObjectClassTermeName>
3788                <ccts:PrimitiveType></ccts:PrimitiveType>
3789                <xsd:BuiltInType></xsd:BuiltInType>
3790                <ccts:UsageRule></ccts:UsageRule>
3791                <ccts:Example></ccts:Example>
3792             </xsd:documentation>
3793          </xsd:annotation>
3794
3795    <!-- Core Component Type Documentation -->
3796          <xsd:annotation>
3797             <xsd:documentation xml:lang="en">
3798                <ccts:UniqueID></ccts:UniqueID>
3799                <ccts:CategoryCode>CCT</ccts:CategoryCode>
3800                <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
3801                <ccts:VersionID></ccts:VersionID>
3802                <ccts:Definition></ccts:Definition>
3803                <ccts:RepresentationTermName></ccts:RepresentationTermName>
3804                <ccts:PrimitiveType></ccts:PrimitiveType>
3805                <ccts:UsageRule></ccts:UsageRule>
3806                <ccts:BusinessTermName></ccts:BusinessTermName>
```

```
3807                    <ccts:Example></ccts:Example>
3808                </xsd:documentation>
3809            </xsd:annotation>
3810
3811    <!-- Core Component Type Supplementary Component Documentation-->
3812            <xsd:annotation>
3813                <xsd:documentation xml:lang="en">
3814                    <ccts:UniqueID></ccts:UniqueID>
3815                    <ccts:CategoryCode>SC</ccts:CategoryCode>
3816                    <ccts:DictionaryEntryName></ccts:DictionaryEntryName>
3817                    <ccts:Definition></ccts:Definition>
3818                    <ccts:ObjectClassTermNam></ccts: ObjectClassTermName>
3819                    <ccts:PropertyTermName></ccts:PropertyTermName>
3820                    <ccts:RepresentationTermName></ccts:RepresentationTermName>
3821                    <ccts:ObjectClassTermeName></ccts:ObjectClassTermeName>
3822                    <ccts:PrimitiveType></ccts:PrimitiveType>
3823                    <ccts:UsageRule></ccts:UsageRule>
3824                    <ccts:Example></ccts:Example>
3825                </xsd:documentation>
3826            </xsd:annotation>
3827
3828    <!-- Code List / Identification Schema Documentation-->
3829                    <xsd:annotation>
3830                        <xsd:documentation xml:lang="en">
3831                            <ccts:CodeName></ccts:CodeName>
3832                            <ccts:CodeDescription></ccts:CodeDescription>
3833                        </xsd:documentation>
3834                    </xsd:annotation>
```

Appendix G. Mapping of CCTS Representation Terms to CCT and UDT Data Types

The following table represents the mapping between the representation terms as defined in CCTS and their equivalent data types as declared in the CCT schema module and the UDT schema module.

Representation Term	Data Type for CCT	Data Type for UDT
Amount	xsd:decimal	xsd:decimal
Binary Object	xsd:base64Binary	xsd:base64Binary
Graphic		xsd:base64Binary
Sound		xsd:base64Binary
Video		xsd:base64Binary
Code	xsd:normalizedString	xsd:normalizedString
Date Time	xsd:string	xsd:dateTime
Date		xsd:date
Time		xsd:time
Identifier	xsd:normalizedString	xsd:normalizedString
Indicator	xsd:string	xsd:boolean
Measure	xsd:decimal	xsd:decimal
Value		xsd:decimal
Percent		xsd:decimal
Rate		xsd:decimal
Numeric	xsd:string	xsd:decimal
Quantity	xsd:decimal	xsd:decimal
Text	xsd:string	xsd:string
Name		xsd:string

Appendix H. Naming & Design Rules List

[R 1]	Conformance shall be determined through adherence to the content of normative sections, rules and definitions.
[R 2]	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*.
[R 3]	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.
[R 4]	UN/CEFACT XSD Schema MUST follow the standard structure defined in Appendix B.
[R 5]	Each element or attribute XML name MUST have one and only one fully qualified XPath (FQXP).
[R 6]	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.
[R 7]	Lower camel case (LCC) MUST be used for naming attributes.
[R 8]	Upper camel case (UCC) MUST be used for naming elements and types.
[R 9]	Element, attribute and type names MUST be in singular form unless the concept itself is plural.
[R 10]	Element, attribute and type names MUST be drawn from the following character set: a-z and A-Z.
[R 11]	XML element, attribute and type names constructed from dictionary entry names MUST NOT include periods, spaces, or other separators; or characters not allowed by W3C XML 1.0 for XML names.
[R 12]	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.
[R 13]	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.
[R 14]	Acronyms MUST appear in all upper case for all element declarations and type definitions.
[R 15]	All element declarations for BBIEs and ASBIEs MUST be locally declared within the parent ABIE type.
[R 16]	A root schema MUST be created for each unique business information exchange.
[R 17]	A root schema MUST NOT replicate reusable constructs available in schema modules capable of being referenced through `xsd:include` or `xsd:import`.
[R 18]	UN/CEFACT XSD schema modules MUST either be treated as external schema modules or as internal schema modules of the root schema.
[R 19]	All UN/CEFACT internal schema modules MUST be in the same namespace as their corresponding `rsm:RootSchema`.
[R 20]	Each UN/CEFACT internal schema module MUST be named <ParentRootSchemaModuleName><InternalSchemaModuleFunction>SchemaModule
[R 21]	A Core Component Type schema module MUST be created
[R 22]	The `cct:CoreComponentType` schema module MUST be named "CCTS CCT Schema Module"
[R 23]	An Unqualified Data Type schema module MUST be created

[R 24] The udt:UnqualifiedDataType schema module MUST be named "UN/CEFACT Unqualified Data Type Schema Module"

[R 25] A Qualified Data Type schema module MUST be created..

[R 26] The qdt:QualifiedDataType schema module MUST be named "UN/CEFACT Qualified Data Type Schema Module"

[R 27] A Reusable Aggregate Business Information Entity schema module MUST be created

[R 28] The ram:ReusableAggregateBusinessInformationEntity schema module MUST be named "UN/CEFACT Reusable Aggregate Business Information Entity Schema Module"

[R 29] Reusable Code List schema modules MUST be created to convey code list enumerations

[R 30] 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

[R 31] An Identifier List schema module MUST be created to convey enumeration values for each identifier list that requires run time validation .

[R 32] 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 identification scheme Identifier Scheme Agency Name = Agency that maintains the identifier list Identifier Scheme Identifier = The identification of the identification scheme Identification Scheme Name = Name as assigned by the agency that maintains the identifier list

[R 33] Imported schema modules MUST be fully conformant with the *UN/CEFACT XML Naming and Design Rules* Technical Specification and the *Core Components Technical Specification.*

[R 34] Every UN/CEFACT defined or imported schema module MUST have a namespace declared, using the xsd:targetNamespace attribute.

[R 35] Every version of a defined or imported schema module other than internal schema modules MUST have its own unique namespace.

[R 36] UN/CEFACT published namespace declarations or contents MUST never be changed.

[R 37] UN/CEFACT namespaces MUST be defined as Uniform Resource Names

[R 38] The names for namespaces MUST have the following structure while the schema is at draft status: urn:un:unece:uncefact:<schematype>:draft:<name>:<major>.[<minor>].[<revision] Where: schematype = a token identifying the type of schema module: data|process|codelist|identifierlist|documentation name = the name of the module (using upper camel case) 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. Not applicable for code list or identifier list schema. revision = sequentially assigned alphanumeric character for each revision of a minor release. Only applicable where status = draft and schema type does not equal code list or identifier list.

[R 39] The namespace names for schema holding specification status MUST be of the form: urn:un:unece:uncefact:<schematype>:standard:<name>:<major>.[<minor>] Where: schematype = a token identifying the type of schema module: data|process|codelist|identifierlist|documentation name = the name of the module 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. Not applicable for code list or identifier list schema.

[R 40] UN/CEFACT namespaces MUST only contain UN/CEFACT developed schema modules.

[R 41] The general structure for schema location MUST be: `http://www.unece.org/uncefact/<schematype>/<name>_<major>.<minor>.[<revision>]_[<status>].xsd` Where: schematype = a token identifying the type of schema module: `data|process|codelist|identifierlist|documentation` name = the name of the module (using upper camel case) 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. status = the status of the schema as: `draft|standard`

[R 42] Each `xsd:schemaLocation` attribute declaration MUST contain a persistent and resolvable URL.

[R 43] Each `xsd:schemaLocation` attribute declaration URL MUST contain an absolute path.

[R 44] Every schema major version MUST have the URI of: `urn:un:unece:uncefact:<schematype>:<status>:<name>:<major>.0.[<revision>]`

[R 45] Every UN/CEFACT XSD Schema and schema module major version number MUST be a sequentially assigned incremental integer greater then zero.

[R 46] Minor versioning MUST be limited to declaring new optional XSD constructs, extending existing XSD constructs and refinements of an optional nature.

[R 47] Every UN/CEFACT XSD Schema minor version MUST have the URI of: `urn:un:unece:uncefact:cc:schema:<name>:<major>.<non-zero integer>.[<revision>]`

[R 48] For UN/CEFACT minor version changes, the name of the schema construct MUST NOT change.

[R 49] Changes in minor versions MUST NOT break semantic compatibility with prior versions.

[R 50] UN/CEFACT minor version schema MUST incorporate all XML constructs from the immediately preceding major or minor version schema.

[R 51] The `xsd:elementFormDefault` attribute MUST be declared and its value set to “qualified”.

[R 52] The `xsd:attributeFormDefault` attribute MUST be declared and its value set to “unqualified”.

[R 53] 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

[R 54] The xsi prefix SHALL be used where appropriate for referencing `xsd:schemaLocation` and `xsd:noNamespaceLocation` attributes in instance documents.

[R 55] xsd:appInfo MUST NOT be used.

[R 56] xsd:notation MUST NOT be used.

[R 57] xsd:wildcard MUST NOT be used.

[R 58] The `xsd:any` element MUST NOT be used.

[R 59] The `xsd:any` attribute MUST NOT be used.

[R 60] Mixed content MUST NOT be used (excluding documentation).

[R 61] xsd:substitutionGroup MUST NOT be used.

[R 62] xsd:ID/IDREF MUST NOT be used.

[R 63] xsd:key/xsd:keyref MUST be used for information association.

[R 64] The absence of a construct or data MUST NOT carry meaning.

[R 65] User declared attributes MUST only be used to convey core component type (CCT) supplementary component information.

[R 66] An attribute of a supplementary component with variable information MUST be based on the appropriate built-in XSD data type.

[R 67] An attribute of a supplementary component which represents codes MUST be based on the `xsd:simpleType` of the appropriate code list

[R 68] An attribute of a supplementary component which represents identifiers MUST be based on the `xsd:simpleType` of the appropriate identifier scheme.

[R 69] The `xsd:nillable` attribute MUST NOT be used.

[R 70] All element declarations MUST be local except for a root element that must be declared globally.

[R 71] Empty elements MUST NOT be used.

[R 72] The `xsd:type` of each leaf element declaration MUST be of the data type of its source business information entity (BBIE) or complex type of its source association business information entity (ASBIE).

[R 73] The `xsd:all` element MUST NOT be used.

[R 74] All type definitions MUST be named.

[R 75] Data type definitions MUST NOT duplicate the functionality of an existing data type definition..

[R 76] xsd:extension MUST only be used in the `cct:CoreComponentType` schema module and the `udt:UnqualifiedDataType` schema module. When used it MUST only extend a built-in XSD datatype.

[R 77] When `xsd:restriction` is applied to a `xsd:simpleType` or `xsd:complexType` the derived construct MUST use a different name.

[R 78] Each UN/CEFACT defined or declared construct MUST use the `xsd:annotation` element for required CCTS documentation.

[R 79] The root schema module MUST be represented by a unique token.

[R 80] The `rsm:RootSchema` MUST import the following schema modules: – `ram:ReusableABIE` Schema Module – `udt:UnqualifiedDataType` Schema Module – `qdt:QualifiedDataType` Schema Module

[R 81] 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.

[R 82] A `rsm:RootSchema` in one UN/CEFACT namespace that is dependant upon type definitions or element declarations defined in another namespace MUST NOT import Schema Modules from that namespace other than the `rsm:RootSchema.`

[R 83] The `rsm:RootSchema` MUST include any internal schema modules that reside in the root schema namespace.

[R 84] A single global element known as the root element MUST be globally declared in a `rsm:RootSchema`.

[R 85] The name of the root element MUST be the name of the Message Assembly with separators and spaces removed.

[R 86] Root schema MUST define a single `xsd:complexType` that fully describes the business information exchange.

[R 87] The name of the top-level complex type MUST be the name of the root element with the word “Type” appended.

[R 88] The `xsd:complexType` of the root element must be the top-level complex type.

[R 89] For every `rsm:RootSchema` root element declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references the Message Assembly instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be RSM.
- Name (mandatory): The name of the Message Assembly
- VersionID (mandatory): An indication of the evolution over time of a Message Assembly.
- Description (mandatory): A brief description of the business information exchange.
- BusinessDomain (mandatory, repetitive): The TBG group(s) that developed this Message Assembly.
- BusinessProcessContext (mandatory, repetitive): The business process with which this Message Assembly is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this Message Assembly.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this Message Assembly.
- ProductContext (optional, repetitive): The product context for this Message Assembly.
- IndustryContext (optional, repetitive): The industry context for this Message Assembly.
- BusinessProcessRoleContext (optional, repetitive): The role context for this Message Assembly.
- SupportingRoleContext (optional, repetitive): The supporting role context for this Message Assembly.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this Message Assembly.

[R 90] All UN/CEFACT internal schema modules MUST be in the same namespace as their corresponding `rsm:RootSchema`.

[R 91] The internal schema module MUST be represented by the same token as its `rsm:RootSchema`.

[R 92] The Reusable Aggregate Business Information Entity schema module MUST be represented by the token “ram”.

[R 93] The `ram:ReusableAggregateBusinessInformationEntity` schema MUST import the following schema modules: – `udt:UnqualifiedDataType` Schema Module – `qdt:QualifiedDataType` Schema Module

[R 94] For every object class (ABIE) identified in the UN/CEFACT syntax-neutral model, a named `xsd:complexType` MUST be defined.

[R 95] The name of the ABIE `xsd:complexType` MUST be the `ccts:DictionaryEntryName` with the separators removed and with the "Details" suffix replaced with "Type".

[R 96] Every aggregate business information entity (ABIE) `xsd:complexType` definition `xsd:content` model MUST use the `xsd:sequence` and/or `xsd:choice` elements with appropriate local element declarations to reflect each property (BBIE or ASBIE) of its class.

[R 97] Recursion of `xsd:sequence` and/or `xsd:choice` MUST NOT occur.

[R 98] 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.

[R 99] For every attribute of an object class (BBIE) identified in an ABIE, a named `xsd:element` MUST be locally declared within the `xsd:complexType` representing that ABIE.

[R 100] Each BBIE element name declaration MUST be based on the property term and qualifiers and the representation term of the basic business information entity (BBIE). If there are successive duplicate words in the property term and representation terms of the source dictionary entry name, then the duplicate words MUST be removed.

[R 101] If the representation term of a BBIE is 'text', it MUST be removed.

[R 102] 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.

[R 103] For every association (ASBIE) identified in the UN/CEFACT syntax-neutral model, a named `xsd:element` MUST be locally declared within the `xsd:complexType` representing the ABIE.

[R 104] Each ASBIE element name declaration MUST be based on the property term and object class of the association business information entity (ASBIE). If there are successive duplicate words in the property term and object class of the associated ABIE, then the duplicate words MUST be removed.

[R 105] The element representing an association business information entity (ASBIE) MUST be of the complex type corresponding to its associated aggregate business information (ABIE).

[R 106] For every ABIE `xsd:complexType` definition a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references an ABIE instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be ABIE.
- DictionaryEntryName (mandatory): The official name of an ABIE.
- VersionID (mandatory): An indication of the evolution over time of an ABIE instance.
- Definition (mandatory): The semantic meaning of an ABIE.
- ObjectClassTermName (mandatory): The Object Class Term of the ABIE.
- QualifierTermName (optional): Qualifies the Object Class Term of the ABIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the ABIE.
- BusinessTermName (optional, repetitive): A synonym term under which the ABIE is commonly known and used in the business.
- BusinessProcessContext (optional, repetitive): The business process with which this ABIE is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this ABIE.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this ABIE.
- ProductContext (optional, repetitive): The product context for this ABIE.
- IndustryContext (optional, repetitive): The industry context for this ABIE.
- BusinessProcessRoleContext (optional, repetitive): The role context for this ABIE.
- SupportingRoleContext (optional, repetitive): The supporting role context for this ABIE.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this ABIE.
- Example (optional, repetitive): Example of a possible value of an ABIE.

[R 107] For every BBIE `xsd:element` declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references a BBIE instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be BBIE.
- Dictionary Entry Name (mandatory): The official name of the BBIE.
- VersionID (mandatory): An indication of the evolution over time of a BBIE instance.
- Definition (mandatory): The semantic meaning of the BBIE.

- Cardinality (mandatory): Indication whether the BIE Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the ABIE.
- ObjectClassTermName (mandatory): The Object Class Term of the parent ABIE.
- ObjectClassQualifierTermName (optional): Qualifies the Object Class Term of the parent ABIE.
- PropertyTermName (mandatory): The Property Term of the BBIE.
- PropertyQualifierTermName (optional): Qualifies the Property Term of the BBIE.
- RepresentationTermName (mandatory): The Representation Term of the BBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the BBIE.
- BusinessProcessContext (optional, repetitive): The business process with which this BBIE is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this BBIE.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this BBIE.
- ProductContext (optional, repetitive): The product context for this BBIE.
- IndustryContext (optional, repetitive): The industry context for this BBIE.
- BusinessProcessRoleContext (optional, repetitive): The role context for this BBIE.
- SupportingRoleContext (optional, repetitive): The supporting role context for this BBIE.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this BBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to this BBIE.
- BusinessTermName (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.

[R 108] For every ASBIE xsd:element declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references an ASBIE instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be ASBIE.
- DictionaryEntryName (mandatory): The official name of the ASBIE.
- VersionID (mandatory): An indication of the evolution over time of the ASBIE instance.
- Definition (mandatory): The semantic meaning of the ASBIE.
- Cardinality (mandatory): Indication whether the ASBIE Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the ABIE.
- ObjectClassTermName (mandatory): The Object Class Term of the associating ABIE.
- ObjectClassQualifierTermName (Optional): A term that qualifies the Object Class Term of the associating ABIE.
- PropertyTermName (mandatory): The Property Term of the ASBIE.
- PropertyQualifierTermName (Optional): A term that qualifies the Property Term of the ASBIE.
- AssociatedObjectClassTermName (mandatory): The Object Class Term of the associated ABIE.
- AssociatedObjectClassQualifierTermName (optional): Qualifies the Object Class Term of the associated ABIE.
- BusinessProcessContext (optional, repetitive): The business process with which this ASBIE is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this ASBIE.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this ASBIE.
- ProductContext (optional, repetitive): The product context for this ASBIE.
- IndustryContext (optional, repetitive): The industry context for this ASBIE.
- BusinessProcessRoleContext (optional, repetitive): The role context for this ASBIE.

- SupportingRoleContext (optional, repetitive): The supporting role context for this ASBIE.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this ASBIE.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the ASBIE.
- BusinessTermName (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.

[R 109] The core component type (CCT) schema module MUST be represented by the token "cct".

[R 110] The `cct:CoreCoreComponentType` schema module MUST NOT include or import any other schema modules.

[R 111] Every `cct:CoreComponentType` MUST be defined as a named `xsd:complexType` in the `cct:CoreComponentType` schema module.

[R 112] The name of each `xsd:complexType` based on a `cct:CoreComponentType` MUST be the dictionary entry name of the core component type (CCT), with the separators and spaces removed.

[R 113] Each `cct:CoreComponentType` `xsd:complexType` definition MUST contain one `xsd:simpleContent` element.

[R 114] The `cct:CoreComponentType` `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 built-in XSD data type required for the CCT content component.

[R 115] Within the `cct:CoreComponentType` xsd:extension element a `xsd:attribute` MUST be declared for each supplementary component pertaining to that `cct:CoreComponentType`.

[R 116] Each `cct:CoreComponentType` supplementary component `xsd:attribute` "name" MUST be the CCTS supplementary component dictionary entry name with the separators and spaces removed.

[R 117] 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.

[R 118] 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.

[R 119] 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.

[R 120] The attribute representing as supplementary component MUST be based on the appropriate built-in XSD data type.

[R 121] For every `cct:CoreComponentType` `xsd:complexType` definition a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references the Core Component Type instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be CCT.
- DictionaryEntryName (mandatory): The official name of a Core Component Type.
- VersionID (mandatory): An indication of the evolution over time of a Core Component Type instance.
- Definition (mandatory): The semantic meaning of a Core Component Type.
- RepresentationTermName (mandatory): The primary representation term of the Core Component Type.

- PrimitiveType (mandatory): 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.
- BusinessTermName (optional, repetitive): A synonym term under which the Core Component Type is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of a Core Component Type.

[R 122] For every supplementary component `xsd:attribute` declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references a Supplementary Component instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be SC.
- DictionaryEntryName (mandatory): The official name of the Supplementary Component.
- Definition (mandatory): The semantic meaning of the Supplementary Component.
- ObjectClassTermName (mandatory): The Object Class of the Supplementary Component.
- PropertyTermName (mandatory): The Property Term of the Supplementary Component.
- RepresentationTermName (mandatory): The Representation term of the Supplementary Component.
- PrimitiveType (mandatory): 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 (optional, repetitive): Example of a possible value of a Basic Core Component.

[R 123] The Unqualified Data Type schema module namespace MUST be represented by the token "udt".

[R 124] The `udt:UnqualifiedDataType` schema MUST NOT import any other schema modules than the following: – `ids:IdentifierList` schema modules – `clm:CodeList` schema modules

[R 125] A `udt:UnqualifiedDataType` MUST be defined for each approved primary and secondary representation terms identified in the CCTS Permissible Representation Terms table.

[R 126] The name of each `udt:UnqualifiedDataType` MUST be the dictionary entry name of the primary or secondary representation term, with “Type” at the end and the separators and spaces removed.

[R 127] For every `udt:UnqualifiedDataType` whose supplementary components map directly to the properties of a built-in `xsd:dataTtpe`, the `udt:UnqualifiedDataType` MUST be defined as a named `xsd:simpleType` in the `udt:UnqualifiedDataType` schema module.

[R 128] Every `udt:UnqualifiedDataType` defined as a `xsd:simpleType` MUST contain one `xsd:restriction` element. This `xsd:restriction` element MUST include an `xsd:base` attribute that defines the specific built-in XSD data type required for the content component.

[R 129] For every `udt:UnqualfiedDataType` whose supplementary components are not equivalent to the properties of a built-in XSD data type, a `udt:UnqualifiedDataType` MUST be defined as an `xsd:complexType` in the `udt:UnqualifiedDataType` schema module.

[R 130] Every `udt:UnqualifiedDataType xsd:complexType` definition MUST contain one xsd:simpleContent element.

[R 131] Every `udt:UnqualifiedDataType 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 built-in XSD datatype required for the content component.

[R 132] Within the `udt:UnqualifiedDataType xsd:complextype xsd:extension` element an `xsd:attribute` MUST be declared for each supplementary component pertaining to the underlying CCT, unless the attribute is contained in the namespace declaration.

[R 133] Each supplementary component `xsd:attribute` name MUST be the supplementary component name with the separators and spaces removed.

[R 134] 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.

[R 135] 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.

[R 136] 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.

[R 137] If the representation term of the relevant supplementary component is a "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.

[R 138] If the representation term of the relevant supplementary component is an "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 scheme.

[R 139] If the representation term of the supplementary component is not "Code" or "Identifier", then the attribute representing this supplementary component MUST be based on the appropriate built-in XSD data type.

[R 140] For every `udt:UnqualifiedDataType` `xsd:complexType` or `xsd:simpleType` definition a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references an Unqualified Data Type instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be UDT.
- DictionaryEntryName (mandatory): The official name of the Unqualified Data Type.
- VersionID (mandatory): An indication of the evolution over time of the Unqualified Data Type instance.
- Definition (mandatory): The semantic meaning of the Unqualified Data Type.
- RepresentationTermName (mandatory): The primary or secondary representation term of the associated Core Component Type.
- PrimitiveType (mandatory): The primitive data type of the Unqualified Data Type.
- BuiltInType (mandatory): The XSD built-in 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 (optional, repetitive): Example of a possible value of an Unqualified Data Type.

[R 141] For every supplementary component `xsd:attribute` declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references a Supplementary Component instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be SC.
- Dictionary Entry Name (mandatory): The official name of the Supplementary Component.
- Definition (mandatory): The semantic meaning of the Supplementary Component.
- ObjectClassTermName (mandatory): The Object Class of the Supplementary Component.
- PropertyTermName (mandatory): The Property Term of the Supplementary Component.
- RepresentationTermName (mandatory): The Representation term of the Supplementary Component.

- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Core Component.
- Example (optional, repetitive): Example of a possible value of a Basic Core Component.

[R 142] The UN/CEFACT:QualifiedDataType schema module namespace MUST be represented by the token “qdt”.

[R 143] The `qdt:QualifiedDataType` schema module MUST import the `udt:UnqualifiedDataType` schema module

[R 144] Where required to change facets of an existing `udt:UnqualifiedDataType`, a new data type MUST be defined in the `qdt:QualifiedDataType` schema module.

[R 145] A `qdt:QualifiedDataType` MUST be based on an unqualified data type and add some semantic and/or technical restriction to the unqualified data type

[R 146] The name of a `qdt:QualifiedDataType` MUST be the name of its base `udt:UnqualifiedDataType` with separators and spaces removed and with its qualifier term added.

[R 147] Every `qdt:QualifiedDataType` based on a `udt:UnqualifiedDataType` `xsd:complexType` whose supplementary components map directly to the properties of a built-in `xsd:data type` MUST be defined as a `xsd:simpleType` MUST contain one `xsd:restriction` element MUST include a `xsd:base` attribute that defines the specific built-in XSD data type required for the content component.

[R 148] Every `qdt:QualifiedDataType` based on a `udt:UnqualifiedDataType` `xsd:complexType` whose supplementary components do not map directly to the properties of a built-in `xsd:data type` MUST be defined as a `xsd:complexType` MUST contain one `xsd:simpleContent` element MUST contain one `xsd:extension` element MUST include `the udt:UnqualifiedDataType` as its `xsd:base` attribute

[R 149] Every `qdt:QualifiedDataType` based on a `udt:UnqualifiedDataType` `xsd:simpleType` MUST contain one `xsd:restriction` element MUST include `the udt:UnqualifiedDataType` as its `xsd:base` attribute

[R 150] The `qdt:QualifiedDataType` `xsd:complexType` definition `xsd:simpleContent` element MUST only restrict attributes declared in its base type, or MUST only restrict facets equivalent to allowed supplementary components.

[R 151] Every `qdt:QualifiedDataType` definition MUST contain a structured set of annotations in the following sequence and pattern:

- UniqueID (mandatory): The identifier that references a Qualified Data Type instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be QDT.
- DictionaryEntryName (mandatory): The official name of the Qualified Data Type.
- VersionID (mandatory): An indication of the evolution over time of the Qualified Data Type instance.
- Definition (mandatory): The semantic meaning of the Qualified Data Type.
- RepresentationTermName (mandatory): The Representation Term of the Qualified Data Type.
- PrimitiveType (mandatory): The primitive data type of the Qualified Data Type.
- BuiltInType (mandatory): The XSD built-in data type of the Qualified Data Type.
- Data Type Qualifier Term (mandatory): A term that qualifies the Representation Term in order to differentiate it from its underlying Unqualified Data Type and other Qualified Data Types.
- BusinessProcessContext (optional, repetitive): The business process context for this Qualified Data Type is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this Qualified Data Type.

- OfficialConstraintContext (optional, repetitive): The official constraint context for this Qualified Data Type.
- ProductContext (optional, repetitive): The product context for this Qualified Data Type.
- IndustryContext (optional, repetitive): The industry context for this Qualified Data Type.
- BusinessProcessRoleContext (optional, repetitive): The role context for this Qualified Data Type.
- SupportingRoleContext (optional, repetitive): The supporting role context for this Qualified Data Type.
- SystemCapabilitiesContext (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.

[R 152] For every supplementary component `xsd:attribute` declaration a structured set of annotations MUST be present in the following pattern:

- UniqueID (mandatory): The identifier that references a Supplementary Component of a Core Component Type instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be QDT.
- Dictionary Entry Name (mandatory): The official name of a Supplementary Component.
- VersionID (mandatory): An indication of the evolution over time of a Supplementary Component instance.
- Definition (mandatory): The semantic meaning of a Supplementary Component.
- Cardinality (mandatory): Indication whether the Supplementary Component Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the Core Component Type.
- PropertyTermName (optional): The Property Term of the associated Supplementary Component.
- RepresentationTermName (optional): The Representation Term of the associated Suppplementary Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Supplementary Component.
- BusinessProcessContext (optional, repetitive): The business process with which this Supplementary Component is associated.
- GeopoliticalorRegionContext (optional, repetitive): The geopolitical/region contexts for this Supplementary Component.
- OfficialConstraintContext (optional, repetitive): The official constraint context for this Supplementary Component.
- ProductContext (optional, repetitive): The product context for this Supplementary Component.
- IndustryContext (optional, repetitive): The industry context for this Supplementary Component.
- BusinessProcessRoleContext (optional, repetitive): The role context for this Qualified Data Type.
- SupportingRoleContext (optional, repetitive): The supporting role context for this Supplementary Component.
- SystemCapabilitiesContext (optional, repetitive): The system capabilities context for this Supplementary Component.
- Example (optional, repetitive): Example of a possible value of a Supplementary Component.

[R 153] Each UN/CEFACT maintained code list MUST be defined in its own schema module.

[R 154] Internal code list schema MUST NOT duplicate existing external code list schema when the existing ones are available to be imported.

[R 155] The names for namespaces MUST have the following structure while the schema is at draft status: `urn:un:unece:uncefact:codelist:draft:<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 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.

[R 156] The namespace names for schema holding specification status MUST be of the form: urn:un:unece:uncefact:codelist:standard:<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 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.

[R 157] 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.

[R 158] The structure for schema location of code lists MUST be: http://www.unece.org/uncefact/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 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.

[R 159] Each xsd:schemaLocation attribute declaration of a code list MUST contain a persistent and resolvable URL.

[R 160] Each xsd:schemaLocation attribute declaration URL of a code list MUST contain an absolute path.

[R 161] Code List schema modules MUST not import or include any other schema modules.

[R 162] Within each code list module one, and only one, named xsd:simpleType MUST be defined for the content component.

[R 163] The name of the xsd:simpleType MUST be the name of root element based on the value of the code list name text with the word “ContentType” appended.

[R 164] The xsd:restriction element base attribute value MUST be set to “xsd:token”.

[R 165] 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.

[R 166] Facets other than xsd:enumeration MUST NOT be used in the code list schema module.

[R 167] For each code list a single root element MUST be globally declared.

[R 168] The name of root element MUST be based on the code list name text following the naming rules as defined in section 5.3.

[R 169] The root element MUST be of a type representing the actual list of code values.

[R 170] Each xsd:enumeration MUST include an annotation documentation providing the code name and the code description.

[R 171] Internal identifier lists schema MUST NOT duplicate existing external identifier list schema when the existing ones are available to be imported.

[R 172] Each UN/CEFACT maintained identifier list MUST be defined in its own schema module.

[R 173] The names for namespaces MUST have the following structure while the schema is at draft status: urn:un:unece:uncefact:identifierlist:draft:<Identifier Scheme. Agency Identifier|Identifier Scheme Agency Name Text>:<Identifier Scheme Identifier|Identifier Scheme Name Text>:<Identifier Scheme Version Identifier> Where: 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.

[R 174] The namespace names for identifier list schema holding specification status MUST be of the form: urn:un:unece:uncefact:identifierlist:standard:<Identifier Scheme. Agency Identifier|Identifier Scheme Agency Name Text>:<Identifier Scheme Identifier|Identifier Scheme Name Text>:<Identifier Scheme. Version Identifier> Where: 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 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.

[R 175] 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>

[R 176] The structure for schema location of identifier lists MUST be: http://www.unece.org/uncefact/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.

[R 177] Each xsd:schemaLocation attribute declaration of an identifier list schema MUST contain a persistent and resolvable URL.

[R 178] Each xsd:schemaLocation attribute declaration URL of an identifier list schema MUST contain an absolute path.

[R 179] Identifier list schema modules MUST NOT import or include any other schema modules.

[R 180] Within each identifier list schema module one, and only one, named xsd:simpleType MUST be defined for the content component.

[R 181] The name of the `xsd:simpleType` MUST be the name of root element with the word "ContentType" appended.

[R 182] The `xsd:restriction` element base attribute value MUST be set to "xsd:token".

[R 183] 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.

[R 184] Facets other than `xsd:enumeration` MUST NOT be used in the identifier list schema module.

[R 185] For each identifier list a single root element MUST be globally declared.

[R 186] The name of the root element MUST be based on the `identification scheme. name. text` following the naming rules as defined in section 5.3.

[R 187] The root element MUST be of a type representing the actual list of identifier values.

[R 188] Each `xsd:enumeration` MUST include an annotation documentation providing the identifier name and optionally the description of the identifier.

[R 189] 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.

[R 190] UN/CEFACT conformant instance documents MUST NOT contain an element devoid of content.

[R 191] The `xsi:nil` attribute MUST NOT appear in any conforming instance.

[R 192] The `xsi:type` attribute MUST NOT be used.

Appendix I. 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*.

Assembly Rules - Assembly Rules group sets of unrefined *Business Information Entities* into larger structures. *Assembly Rules* are more fully defined and explained in the *Assembly Rules* Supplemental Document.

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.*

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 Catalogue of Common Business Processes.

Business Process Context – The Business Process name(s) as described using the UN/CEFACT Catalogue of Common Business Processes as extended by the user.

Business Process Role Context – The actors conducting a particular Business Process, as identified in the UN/CEFACT Catalogue of Common Business Processes.

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.

Catalogue of Business Information Entities – This represents the approved set of *Business Information Entities* from which to choose when applying the *Core Component* discovery process

Catalogue of Core Components – see Core Component Catalogue.

CCL – see Core Component Library.

Child Core Component – A *Core Component* used as part of a larger aggregate construct.

Classification Scheme – This is an officially supported scheme to describe a given *Context Category*.

Constraint Language – A formal expression of actions occurring in specific *Context*s to assemble, structurally refine, and semantically qualify *Core Components*. The result of applying the *Constraint Language* to a set of *Core Components* in a specific *Context* is a set of *Business Information Entities*.

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.

Context Rules Construct – The overall expression of a single set of rules used to apply *Context* to *Core Components*.

Controlled Vocabulary – A supplemental vocabulary used to uniquely 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 Catalogue – The temporary collection of all metadata about each *Core Component* discovered during the development and initial testing of this *Core Component Technical Specification*, pending the establishment of a permanent Registry/repository.

Core Component Dictionary – An extract from the *Core Component Catalogue* that provides a ready reference of the *Core Component* through its *Dictionary Entry Name*, component parts, and definition.

Core Component Library – The Core Component Library is the part of the registry/repository in which Core Components shall be stored as Registry Classes. The Core Component Library will contain all the Core Component Types, Basic Core Components, Aggregate Core Components, Basic Business Information Entities and Aggregate Business Information Entities.

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*.

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 dictionary.

Geopolitical Context – Geographic factors that influence *Business Semantics* (e.g., the structure of an address).

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).

Order – In the *Constraint Language*, the *Property* on the *ContextRules Construct* that applies a sequence to the application of a set of rules. Two Rule constructs cannot have the same value for the *Property Order*.

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 Compont 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 generic term used to identify the family of grammar based XML document structure validation languages to include the more formal W3C XML Schema Technical Specification, Document Type Definition, Schematron, Regular Language Description for XML (RELAX), and the OASIS RELAX NG.

Appendix J. Qualified Data Type Schema Module

The Qualified Data Type Schema Module is published as a separate file – QualifiedDataType.xsd.