{Specification Title Version X.X}

Committee Draft in Progress wd03, {8 July 2010}

Document identifier:

spectools-docbook-template-wd03

Technical committee:

OASIS {official name of technical committee} TC [http://www.oasis-open.org/committees/]

Chairs:

Jon Bosak, Pinax <`bosak@pinax.com`>
Tim McGrath, Document Engineering Ser-
vices <`tim.mcgrath@documentengineeringservices.com`>

Editors:

Mike Grimley, US Navy <`MJGrimley@acm.org`>
Mavis Cournane, Cognitran Limited <`mavis.Cournane@cognitran.com`>

Declared XML Namespaces:

http://docs.oasis-open.org/

Abstract:

{This specification defines...}

Related Work:

This specification replaces or supersedes:

- {specification replaced by this standard}
- {specification replaced by this standard}

This specification is related to:

- {related specifications}
- {related specifications}

Status:

{Describe the status and stability of the specification here.}

This document was last revised or approved by the {TC name | membership of OASIS} on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/{short name}.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/{TC short name}ipr.php).

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/{TC short name}.

Notices:

Table of Contents

1. Introduction

XML is often described as the lingua franca of e-commerce. The implication is that by standardizing on XML, enterprises will be able to trade with anyone, any time, without the need for the costly custom integration work that has been necessary in the past. But this vision of XML-based "plug-and-play" commerce is overly simplistic. Of course XML can be used to create electronic catalogs, purchase orders, invoices, shipping notices, and the other documents needed to conduct business. But XML by itself doesn't guarantee that these documents can be understood by any business other than the one that creates them. XML is only the foundation on which additional standards can be defined to achieve the goal of true interoperability. The Universal Business Language (UBL) initiative is the next step in achieving this goal.

The task of creating a universal XML business language is a challenging one. Most large enterprises have already invested significant time and money in an e-business infrastructure and are reluctant to change the way they conduct electronic business. Furthermore, every company has different requirements for the information exchanged in a specific business process, such as procurement or supply-chain optimization. A standard business language must strike a difficult balance, adapting to the specific needs of a given company while remaining general enough to let different companies in different industries communicate with each other.

The UBL effort addresses this problem by building on the work of the electronic business XML (ebXML) initiative. UBL is organized as an OASIS Technical Committee to guarantee a rigorous, open process for the standardization of the XML business language. The development of UBL within OASIS also helps ensure a fit with other essential ebXML specifications.

This specification documents the rules and guidelines for the naming and design of XML components for the UBL library. ~~It contains only rules that have been agreed on by the OASIS UBL Technical Committee. Consumers of the Naming and Design Rules Specification should consult previous UBL position papers that are available at http://www.oasis-open.org/committees/ubl/ndrsc/. These provide a useful background to the development of the current rule set~~.

1.1. Audiences

This document has several primary and secondary targets that together constitute its intended audience. Our primary target audience is the members of the UBL Technical Committee. Specifically, the UBL Technical Committee uses the rules in this document to create normative form schemas for business transactions. Other XML schema developers may find the rules contained herein sufficiently useful to merit consideration for adoption as, or infusion into, their own approaches to XML schema development.

1.2. Scope

This specification conveys a normative set of XML schema design rules and naming conventions for the creation of UBL schemas for business documents being exchanged between two parties using XML constructs defined in accordance with the ebXML Core Components Technical Specification.

1.3. Terminology and Notation

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in Internet Engineering Task Force (IETF) Request for Comments (RFC) 2119. Non-capitalized forms of these words are used in the regular English sense.

Definition	A formal definition of a term. Definitions are normative.
Example	An example of a definition or a rule. Examples are informative.

Note	Explanatory information. Notes are informative.
RRR*n*	Identifier of a rule to which an XML schema must comply in order to be UBL conformant. The value RRR is a prefix to categorize the type of rule where the value of RRR is as defined in Table 1, “Rule Prefix Value”, and n (1..n) is the sequential number of the rule within its category. To ensure continuity across versions of the specification, rule numbers that are deleted in future versions will not be reissued, and any new rules will be assigned the next higher number — regardless of location in the text. Only rules and definitions are normative; all other text is explanatory.

Table 1. Rule Prefix Value

Rule Prefix	Value
CDL	Code List
CTD	ComplexType Definition
CTN	ComplexType Naming Rules
DOC	Documentation
ELD	Element Declaration
ELN	Element Naming
GNR	General Naming
GTD	General Type Definition
GXS	General XML Schema
MDC	Modeling Constraints
NMC	Naming Constraints
NMS	Namespace
RED	Root Element Declaration
SSM	Schema Structure Modularity
VER	Versioning

The term "XSD" is used throughout this document to refer to Parts 1 and 2 of the W3C *XML Schema Definition Language* (XSD) Recommendation.

1.4. Guiding Principles

The UBL NDR primary objectives are to provide the UBL TC with a set of unambiguous, consistent rules for the development of extensible, reusable UBL schemas.

2. Relationship to ebXML Core Components

UBL employs the methodology and model described in *ISO TS 15000-5:2005 -- ebXML Core Components Technical Specification, Version 2.01 [CCTS]* to build the UBL Component Library. CCTS defines a new paradigm in the design and implementation of reusable, syntactically neutral information building blocks. Syntax-neutral Core Components are intended to form the basis of business information standardization efforts and to be realized in syntactically specific instantiations such as ANSI ASC X12, UN/EDIFACT, and various XML representations such as UBL.

Context-neutral and context-specific building blocks are the essence of the Core Components specification. The context-neutral components are called Core Components. A Core Component is 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". Figure 1 illustrates the various pieces of the overall Core Components metamodel.

The context-specific components are called Business Information Entities (BIEs). A BIE is defined in CCTS as "a piece of business data or a group of pieces of business data with a unique Business Semantic definition". Figure 2 illustrates the various pieces of the overall BIE metamodel and its relationship to the Core Components metamodel. As shown here, there are different types of Core Components and BIEs, each of which has specific relationships to the other components and entities. The context-neutral Core Components establish the formal relationship between the various context-specific BIEs.

Figure 1. Core Components and Datatypes Metamodel

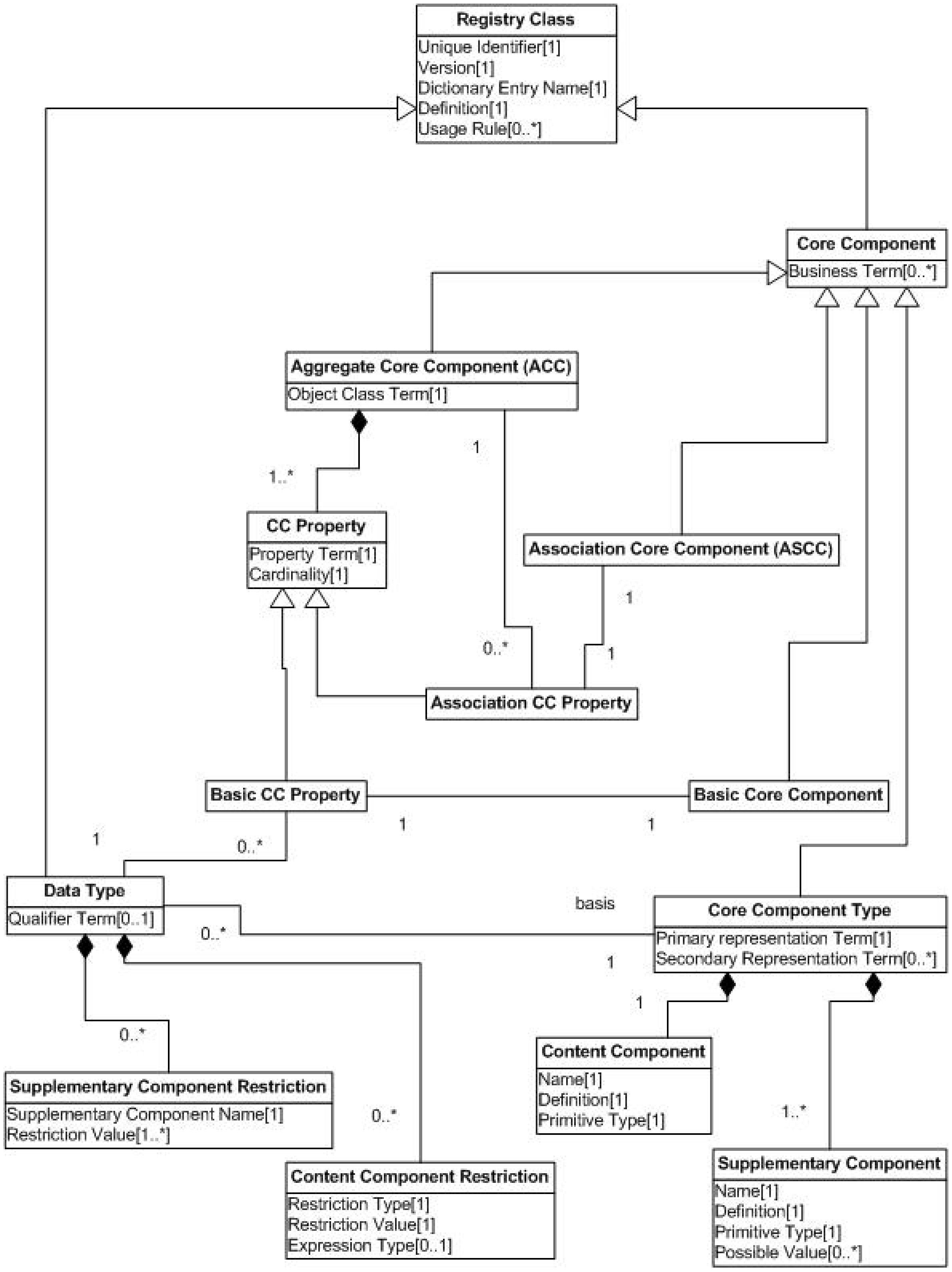
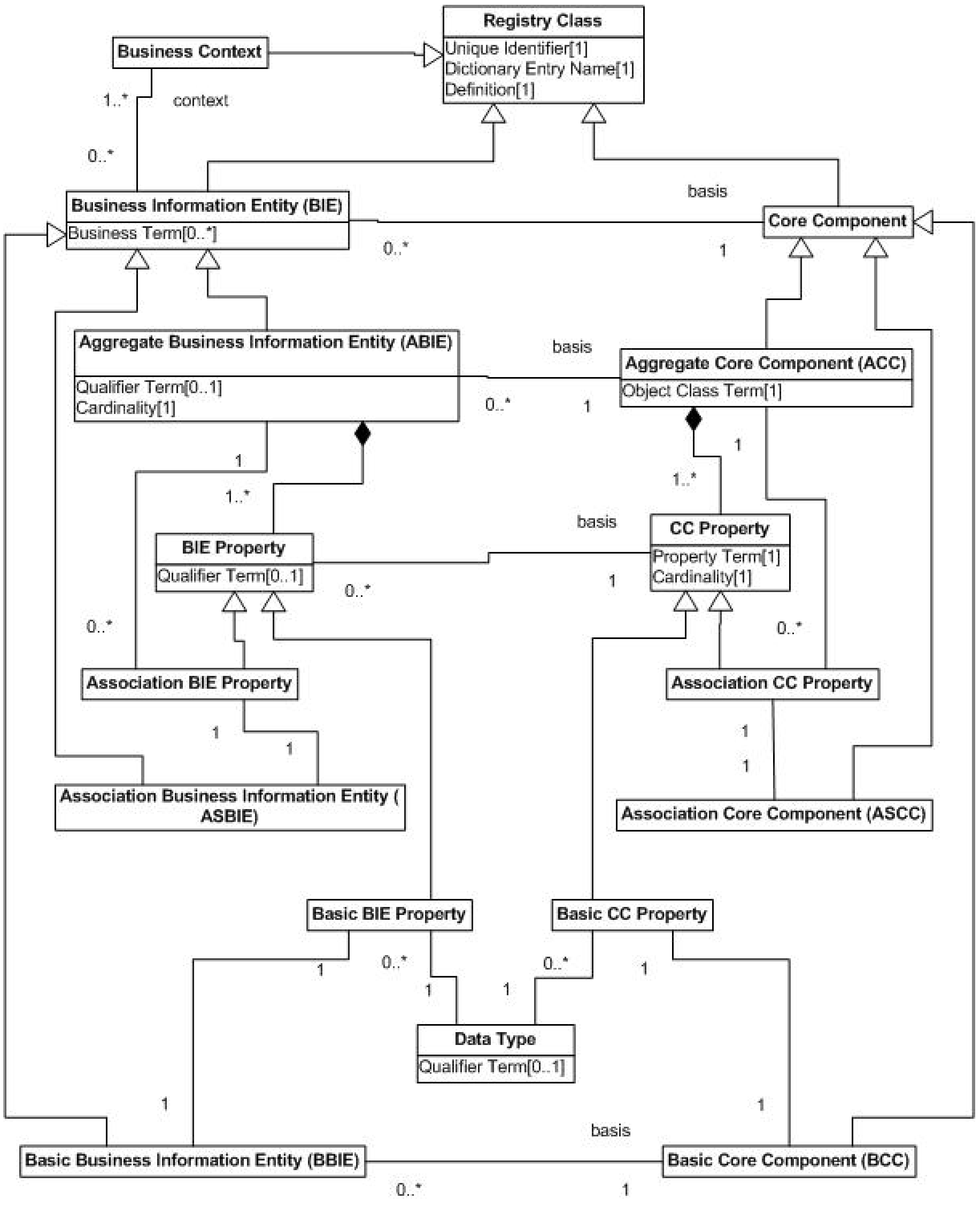


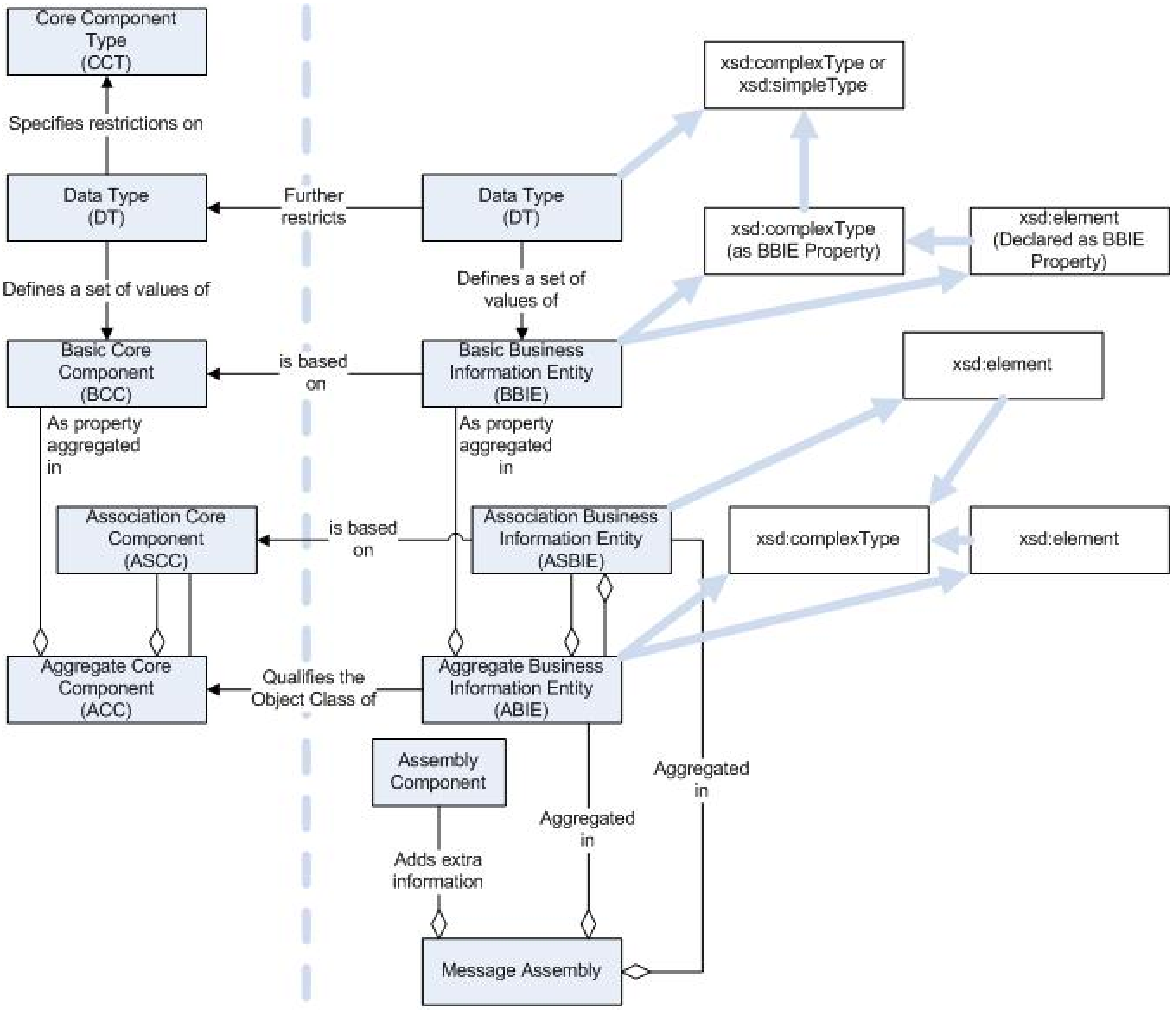
Figure 2. Business Information Entities Basic Definition Model



2.1. Mapping Business Information Entities to XSD

UBL consists of a library of CCTS BIEs, each of which is mapped to an XSD construct (See Figure 3).

Figure 3. UBL Document Metamodel


Core Component Type (CCT)
Specifies restrictions on
Data Type (DT)
Further restricts
Data Type (DT)
xsd:complexType or xsd:simpleType
xsd:complexType (as BBIE Property)
xsd:element (Declared as BBIE Property)
Defines a set of values of
Defines a set of values of
Basic Core Component (BCC)
is based on
Basic Business Information Entity (BBIE)
xsd:element
As property aggregated in
As property aggregated in
Association Core Component (ASCC)
is based on
Association Business Information Entity (ASBIE)
xsd:complexType
xsd:element
Aggregate Core Component (ACC)
Qualifies the Object Class of
Aggregate Business Information Entity (ABIE)
Assembly Component
Aggregated in
Aggregated in
Adds extra information
Message Assembly

A BIE can be a CCTS Aggregate Business Information Entity (ABIE), a CCTS Basic Business Information Entity (BBIE), or a CCTS Association Business Information Entity (ASBIE). In understanding the logic of the UBL binding of BIEs to XSD expressions, it is important to understand the basic constructs of the BIEs and their relationships as shown in Figure 2. The ABIEs are treated as objects and are defined as xsd:complexTypes. The BBIEs are treated as properties of the ABIE and are found in the content model of the ABIE as a referenced xsd:element. The BBIEs are based on reusable CCTS Basic Business Information Entity Properties (BBIE Properties), which are defined as xsd:complexTypes.

A BBIE Property represents an intrinsic property of an ABIE. BBIE Properties are linked to a data type.

CCTS defines an approved set of primary and secondary representation terms. However, these representation terms are simply naming conventions to identify the data type of an object, not actual constructs.

There are two kinds of BIE Properties — Basic and Association. A CCTS Association BIE Property (ASBIE Property) represents an extrinsic property — in other words, an association from one ABIE instance to another ABIE instance. It is the ASBIE Property that expresses the relationship between ABIEs.

Due to their unique extrinsic association role, ASBIEs are not defined as xsd:complexTypes; rather, they are either declared as elements that are then bound to the xsd:complexType of the associated ABIE, or they are reclassified as ABIEs.

BBIEs define the intrinsic structure of an ABIE. These BBIEs are the "leaf" types in the system in that they contain no other BIEs.

A BBIE must have a CCTS Core Component Type. All CCTS Core Component Types are low-level types such as Identifiers and Dates. A CCTS Core Component Type describes these low-level types for use by CCTS Core Components, and (in parallel) a CCTS data type, corresponding to that CCTS Core Component Type, describes these low-level types for use by BBIEs. Every CCTS Core Component Type has a single CCTS Content Component and one or more CCTS Supplementary Components. A CCTS Content Component is of some Primitive Type. All CCTS Core Component Types and their corresponding content and supplementary components are predefined in CCTS.

UBL has developed an XSD schema module that declares each of the predefined CCTS Core Component Types as an xsd:complexType or xsd:simpleType and declares each CCTS Supplementary Component as an xsd:attribute or uses the predefined facets of the built-in XSD datatypes for those that are used as the base expression for an xsd:simpleType.

3. General XML Constructs

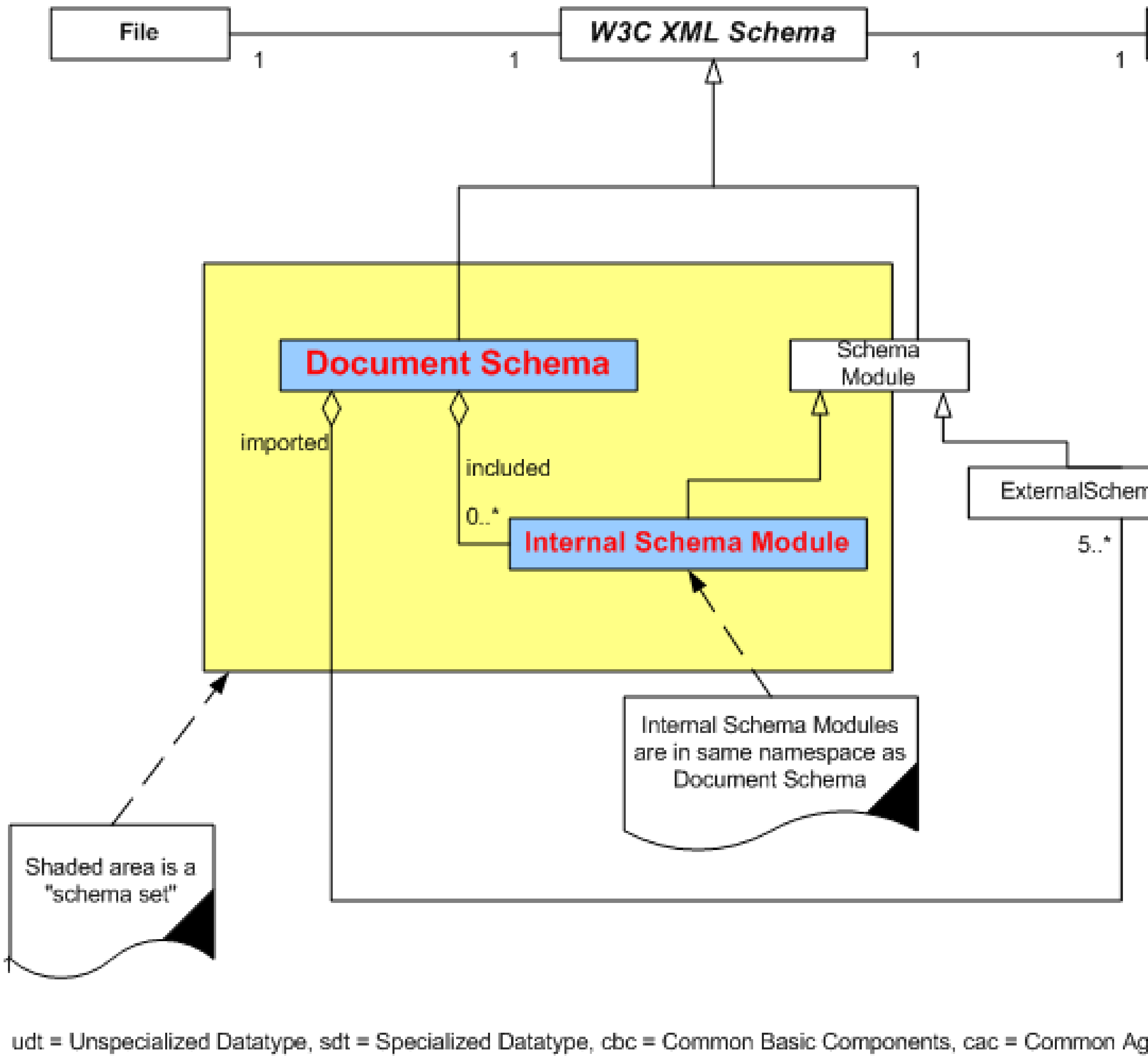
This chapter defines UBL rules related to general XML constructs, including overall schema structure, naming and modeling constraints, reusability, namespaces, versioning, modularity, and documentation.

3.1. Overall Schema Structure

A key aspect of developing standards is to ensure consistency in their implementation. Therefore, it is essential to provide a mechanism that will guarantee that each occurrence of a UBL conformant schema will have the same look and feel.

[*GXS1*] Except in the case of extension, where the "UBL Extensions" element is used, UBL schemas SHOULD conform to the following physical layout as applicable: See Figure 4.

Figure 4. Physical layout


File
1
1
W3C XML Schema
1
1
Document Schema
Schema Module
imported
included
0..*
Internal Schema Module
ExternalSchem
5..*
Internal Schema Modules are in same namespace as Document Schema
Shaded area is a "schema set"
udt = Unspecialized Datatype, sdt = Specialized Datatype, cbc = Common Basic Components, cac = Common Ag

COMMENT:

BH: Is this the correct graphic?

~~As shown above,~~ A UBL schema should contain a comment block at the top of the schema that functions as a "schema header".

3.1.1. Element Declarations within Document Schemas

A document schema is a schema within a specific namespace that conveys the business document functionality of that namespace. The document schema declares a target namespace and is likely to include (xsd:include) internal schema modules or import (xsd:import) external schema modules. Each namespace will have one, and only one, major version of a document schema. ~~as well as any related minor versions.~~

COMMENT:

BH: We have deleted the minor versions in namespaces so there shouldn't be any minor versions in namespaces.

3.1.2. Root Element

In order to facilitate the management and reuse of UBL constructs, all global elements, excluding the root element of the document schema, must be declared in either the Common Aggregate Components (CAC) or Common Basic Components (CBC) schema modules and referenced from within the document schema.

Only a single global element is declared inside a UBL document schema. The single global element is the root element of every conforming instance.

[*RED2*] The root element MUST be the only global element declared in the document schema.

COMMENT:

BH: Should another rule be included here that states: One global complexType MUST be included which with the same name as the Root element with the word "Type" appended to the name. The complexType defines the structure of the document instance.

3.2. Naming and Modeling Constraints

UBL has the following naming and modeling constraints.

3.2.1. Naming Constraints

A primary aspect of the UBL library documentation is its spreadsheet models. The entries in these spreadsheet models fully define the constructs available for use in UBL business documents. The spreadsheet entries contain fully conformant CCTS Dictionary Entry Names (DENs) as well as truncated UBL XML element names developed in conformance with the rules in Section 4. The XML element name is the short form of the DEN. The rules for element naming differ from the rules for DEN naming.

[*NMC1*] Each Dictionary Entry Name MUST define one and only one fully qualified path (FQP) for an element or attribute.

The FQP anchors the use of the element or attribute to a particular location in a business message. Any semantic dependencies that the element or attribute has on other elements and attributes within the UBL library that are not otherwise enforced or made explicit in its structural definition can be found in its prose definition.

3.2.1.1. Modeling Constraints

Modeling constraints are limited to those necessary to ensure consistency in development of the UBL library.

3.2.1.1.1. Defining Classes

UBL is based on instantiating ebXML CCTS BIEs. UBL models and the XML expressions of those models are class driven. Specifically, the UBL library defines classes for each CCTS ABIE and the UBL schemas instantiate those classes. The properties of those classes consist of CCTS BBIEs and ASBIEs.

3.2.1.1.2. Core Component Types

Each BBIE is associated with one of an approved set of CCTS Core Component Types.

> [*MDC1*] UBL libraries and schemas MUST only use CCTS Core Component Types, except in the case of extension, where the UBLExtensions element is used.

3.2.1.1.3. XML Mixed Content

UBL documents are designed to effect data-centric electronic commerce transactions. ~~Including~~Allowing XML mixed content in business documents is undesirable because business transactions are based on exchange of discrete pieces of data. The white space aspects of XML mixed content make processing unnecessarily difficult and add a layer of complexity not desirable in business exchanges.

> [*MDC2*] XML mixed content MUST NOT be used except where contained in an xsd:documentation element.

3.2.1.1.4. Sequencing

COMMENT:

BH: I don't understand what this rule is trying to express?

In the UBL model, the prescribed order for the contents of an ABIE is that ASBIEs follow BBIEs. However, this is, strictly speaking, a rule of the modeling methodology rather than an NDR. The NDR in this case is that the sequential order of entities in the model must be preserved.

> [*MDC0*] The sequence of the business information entities that is expressed in the UBL model MUST be preserved in the schema.

3.3. Reusability Scheme

To promote effective management of the UBL library, all element declarations are unique. Consequently, UBL elements are declared globally.

3.3.1. Reusable Elements

UBL elements are global and qualified. Hence in the example below, the Address element is directly reusable as a modular component.

Example 1.

```
<xsd:element name="Party" type="PartyType"/>
  <xsd:complexType name="PartyType">
    <xsd:annotation>
      <!-- Documentation goes here -->
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="cbc:MarkCareIndicator" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="cbc:MarkAttentionIndicator" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="PartyIdentification" minOccurs="0" maxOccurs="unbounded">
          ...
      </xsd:element>
      <xsd:element ref="PartyName" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="Address" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
          ...
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="Address" type="AddressType"/>

  <xsd:complexType name="AddressType">
      ...
    <xsd:sequence>
      <xsd:element ref="cbc:CityName" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="cbc:PostalZone" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
          ...
    </xsd:sequence>
  </xsd:complexType>
```

Software written to work with UBL's standard library should work with new assemblies of the same components, since global elements will remain consistent and unchanged. The globally declared <Address> element is fully reusable without regard to the reusability of types and provides a solid mechanism for ensuring that extensions to the UBL core library will provide consistency and semantic clarity regardless of their placement within a particular type.

[*ELD2*] All element declarations MUST be global.

3.4. Extension Scheme

Some organizations are required by law to send additional information not covered by the UBL document structure, thus requiring an extension to the UBL message. The xsd:any construct is seen as the most efficient way to implement this requirement.

In general, UBL restricts the use of xsd:any because this feature permits the introduction of unknown elements into an XML instance. However, limiting its use to a single, predefined element mitigates this risk. For meaningful validation of UBL document instances, the value of the xsd:processContents attribute of the element must be set to "skip", thereby removing the potential for errors in the validation layer. Extension imposes cardinality constraints.

The following rules apply in the order below.

COMMENT:

BH: Can someone provide an example of these rules. I don't understand how an extension is built. What is a UBLProfileId and UBLSubsetId. I may be reading it wrong but Rule ELD12 contradicts rule RED1 - where there is only 1 global element declared in a document schema. How can we rectify this in the NDR?

[*ELD12*] The UBL Extensions element MUST be declared as the first child of the document element with xsd:minOccurs="0".

[*ELD13*] The UBLProfileID element MUST be declared immediately following the UBL Extensions element with xsd:minOccurs="0".

[*ELD14*] The UBLSubsetID element MUST be declared immediately following the UBLProfileID element with xsd:minOccurs="0".

3.5. Namespace Scheme

The concept of XML namespaces is defined in the W3C XML namespaces technical specification. The use of XML namespace is specified in the W3C XML Schema (XSD) Recommendation. A namespace is declared in the root element of a schema using a namespace identifier. Namespace declarations can also identify an associated prefix "shorthand identifier" that allows for compression of the namespace name. For each UBL namespace, a normative token is defined as its prefix. These tokens (currently udt, qdt, cac, cbc, ext) are defined in Section 3.7.

3.5.1. Declaring Namespaces

Neither XML 1.0 nor XSD requires the use of namespaces. However, the use of namespaces is essential to managing the complex UBL library. UBL uses UBL-defined schemas (created by the UBL TC) and UBL-used schemas (created by external activities), and both require a consistent approach to namespace declarations.

[*NMS1*] Every UBL-defined or -used schema module, except internal schema modules, MUST declare a namespace using the xsd:targetNamespace attribute.

Each UBL schema module consists of a logical grouping of lower level artefacts that can be used in a variety of UBL schemas. These schema modules are grouped into a schema set. Each schema set is assigned a namespace that identifies that group of schema modules. As constructs are changed, new versions are to be created. The schema set is the versioned entity; all schema modules within that package are of the same version, and each major version has a unique namespace.

Schema set	A collection of schemas that constitute a specific UBL namespace.

Schema validation ensures that an instance conforms to its declared schema. In keeping with Rule NMS1, each UBL schema module is part of a versioned namespace.

[*NMS2*] Every UBL-defined or -used major version schema set MUST have its own unique namespace.

UBL's extension methodology encourages a wide variety in the number of schema modules that are created as derivations from UBL schema modules. Customized schemas should not be confused with those developed by UBL.

[*NMS3*] UBL namespaces MUST only contain UBL developed schema modules.

3.5.2. Namespace Uniform Resource Identifiers

A UBL namespace name must be a URI that conforms to RFC 2396. UBL has adopted the Uniform Resource Name (URN) scheme as the standard for URIs for UBL namespaces, in conformance with IETF's RFC 3121.

Rule NMS2 requires separate namespaces for each UBL major version schema set. In accordance with OASIS procedures, the UBL namespace rules differentiate between committee draft and OASIS Standard status. For each schema holding draft status, a UBL namespace must be declared and named.

> [*NMS4*] The namespace names for UBL schemas holding committee draft status MUST be of the form
>
> urn:oasis:names:tc:ubl:schema:<subtype>:<document-id>

```
xmlns:cac="urn:oasis:names:tc:ubl:schema:xsd:CommonAggregateComponents-2"
```

The format for document-id is found in Section 3.6.

For each UBL schema holding OASIS Committee Specification or Standard status, a UBL namespace must be declared and named using the same notation, but with the value "specification" replacing the value "tc".

> [*NMS5*] The namespace names for UBL schemas holding OASIS Standard status MUST be of the form
>
> urn:oasis:names:specification:ubl:schema:<subtype>:<document-id>

```
xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2"
```

3.5.3. Schema Location

UBL schemas use a URN namespace scheme. In contrast, schema locations are defined as a Uniform Resource Locator (URL). UBL schemas must be available both at design time and run time. Therefore, the UBL schema locations will differ from the UBL namespace declarations. UBL uses an OASIS URL for hosting retrievable copies of UBL schemas.

3.5.4. Persistence

UBL namespaces use URNs to provide name persistence. UBL namespaces must never change once they have been declared. Conversely, changes to a schema may result in a new namespace declaration. Thus, a published schema version and its namespace association will always be inviolate.

> [*NMS6*] UBL published namespaces MUST never be changed.

3.6. Versioning Scheme

UBL distinguishes between major versions and minor versions. Major versions are not backwards compatible. Minor versions do not break backwards compatibility. In other words, a document instance that validates against version 1 of the schema must also validate against version 1.1 of the schema, where version 1.1 is a minor version change based on version 1. However, the same document instances would not necessarily be valid against version 2 of the schema, where version 2 is a major version change.

Versioning information is indicated both in the namespace URI and in the version attribute of the schema module. However, this information is represented somewhat differently in these two locations.

3.6.1. Versioning Information in the Namespace URI

UBL namespaces conform to the OASIS namespace rules defined in RFC 3121 [http://tools.ietf.org/html/rfc3121]. All UBL namespace URIs have the form:

```
urn:oasis:names:specification:ubl:schema:xsd:<modulename>-<major>
```

where <modulename> is the name of the schema module and <major> is a positive integer representing the major version. The field containing <modulename>-<major> is called the *document-id*.

> [*VER2*] Every UBL schema module major version MUST have an RFC 3121 document-id of the form
>
> ```
> <modulename>-<major>
> ```

> [*VER6*] Every UBL schema module major version number MUST be a sequentially assigned integer greater than zero.

The value of <major> is "1" for the first release of a namespace. For example, the namespace URI for the first major release of the Invoice domain has the form:

```
urn:oasis:names:specification:ubl:schema:xsd:Invoice-1
```

Subsequent major releases increment the value by 1. For example, the second major release of the Invoice domain has the URI

```
urn:oasis:names:specification:ubl:schema:xsd:Invoice-2
```

The rule for minor version releases is as follows:

> [*VER4*] Every minor version release of a UBL schema module MUST have a document-id of the form
>
> ```
> <modulename>-<major>
> ```

For example, the fifth minor version of the release based on the second major release mentioned above will have the URI

```
urn:oasis:names:specification:ubl:schema:xsd:Invoice-2
```

As can be seen, both the rule and the example for the minor version releases is exactly the same as that for the major version. There is even a rule stating this directly.

> [*VER5*] For UBL minor version changes, the namespace name MUST not change.

However, minor versioning is handled differently in the xsd:schema element.

3.6.2. Versioning representation in the xsd:schema element

UBL uses the version attribute in the xsd:schema element to convey minor version releases of the schema module.

> [*VER12*] Every major version release of a UBL schema module MUST capture its version number in the xsd:version attribute of the xsd:schema element in the form
>
> ```
> <major>.0
> ```

```
EXAMPLE:  version="2.0"
```

> [*VER14*] Every minor version release of a UBL schema module MUST capture its version information in the xsd:version attribute in the form
>
> ```
> <major>.<non-zero>
> ```

```
EXAMPLE:  version="2.1"
```

[*VER7*] Every UBL schema module minor version number MUST be a sequentially assigned, non-negative integer.

3.6.3. Instance Versioning

UBL version information can also be captured in instances of UBL document schemas via the ubl:UBLVersionID element.

[*VER15*] Every UBL document schema MUST declare an optional element named UBLVersionID immediately following the optional UBL Extensions element.

```
<BillOfLading
      xmlns="urn:oasis:names:specification:ubl:schema:xsd:BillOfLading-2"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   <ID xmlns="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2"/>
</BillOfLading>
```

3.7. Modularity Strategy

There are many possible mappings of XML schema constructs to namespaces and to files. In addition to the logical taming of complexity that namespaces provide, dividing the physical realization of schemas into multiple schema modules provides a mechanism whereby reusable components can be imported as needed without the need to import complete schemas.

[*SSM1*] UBL schema expressions MAY be split into multiple schema modules.

Schema module	A schema document containing type definitions and element declarations intended to be reused in multiple schemas.

3.7.1. UBL Modularity Model

UBL relies extensively on modularity in schema design. There is no single UBL root schema. Rather, there are a number of UBL document schemas used to perform different business functions. UBL is structured so that users can reuse individual document schemas without having to import the entire UBL document schema library. A document schema can import individual modules without having to import all UBL schema modules. Each document schema defines its own dependencies. The UBL schema modularity approach reflects logical associations that exist between document and internal schema modules,and it ensures that individual modules can be reused to the maximum extent possible. If the contents of a namespace are small enough then they can be completely specified within a single document. Document and internal schema modules are shown in Figure 5.

Figure 5. UBL Schema Modularity Model

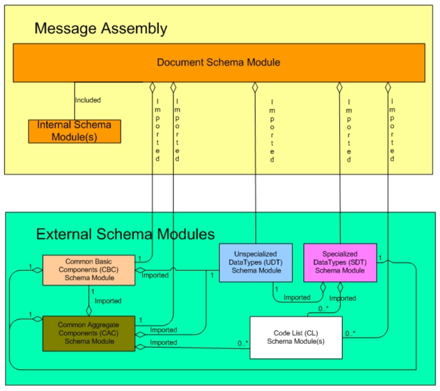
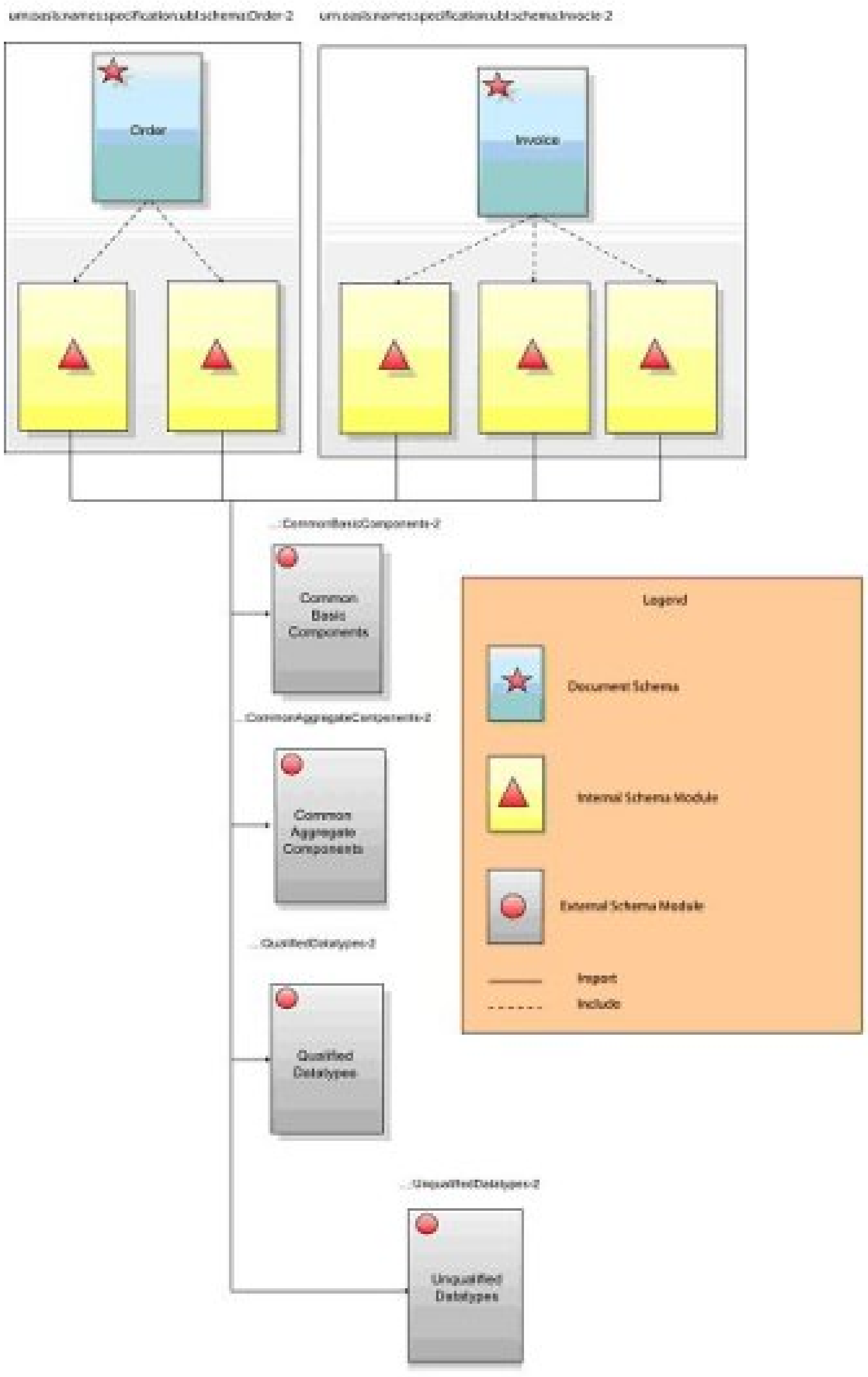

Message Assembly
Document Schema Module
Included
Internal Schema Module(s)
Imported
External Schema Modules
Common Basic Components (CBC) Schema Module
Unspecialized DataTypes (UDT) Schema Module
Specialized DataTypes (SDT) Schema Module
Common Aggregate Components (CAC) Schema Module
Code List (CL) Schema Module(s)
1
0..*

Figure 5 shows the one-to-one correspondence between document schemas and namespaces. It also shows the one-to-one correspondence between files and schema modules. As shown here, there are two types of schemas in the UBL library — document schemas and schema modules. Both types of schemas are conformant with XSD.

Each document schema occupies its own namespace and may include zero or more internal modules. The namespace for a document schema includes any of its internal modules. Schema modules that are not internal to a document occupy a different namespace, as in the qdt, cbc, and cac schema modules.

Figure 6. Schema Modules


urn:oasis:names:specification:ubl:schema:Order-2
urn:oasis:names:specification:ubl:schema:Invoice-2
Order
Invoice
...:CommonBasicComponents-2
Common Basic Components
...:CommonAggregateComponents-2
Common Aggregate Components
...:QualifiedDatatypes-2
Qualified Datatypes
...:UnqualifiedDatatypes-2
Unqualified Datatypes
Legend
Document Schema
Internal Schema Module
External Schema Module
Import
Include

Another way to visualize the structure is by example. Figure 6 depicts instances of the various schema modules from the previous diagram.

Figure 7 shows how the Order and Invoice document schemas import the CommonAggregateComponents and CommonBasicComponents external schema modules. It also shows how the Order document schema may include internal schema modules — modules local to that namespace. The clear boxes show how the various schema modules are grouped into namespaces.

Any UBL schema module, be it a document schema or an internal module, may import other document schemas from other namespaces.

COMMENT:

BH: I am missing this graphic. Does anyone have this or should be it recreated.

Figure 7. Order and Invoice Schema Import of Common Component Schema Modules

If two namespaces are mutually dependent, then importing one will cause the other to be imported as well. For this reason there *must not* exist circular dependencies between UBL schema modules. By extension, there *must not* exist circular dependencies between namespaces. A namespace **called** A **which is** dependent upon type definitions or element declarations defined in another namespace **called** B must import B's document schema.

[*SSM2*] A schema in one UBL namespace that is dependent upon type definitions or element declarations in another schema namespace MUST only import that schema.

An additional rule is necessary to address potentially circular dependencies as well — schema A must not import internal schema modules of schema B.

[*SSM3*] A schema in one UBL namespace that is dependent upon type definitions or element declarations defined in another schema namespace MUST NOT import the internal schema modules of that schema.

3.7.2. Internal and External Schema Modules

As illustrated in figures 5 and 6, UBL schema modules are either internal or external.

3.7.3. Internal Schema Modules

UBL internal schema modules do not declare a target namespace, but instead reside in the namespace of their parent schema. All internal schema modules are accessed using xsd:include.

[*SSM6*] All UBL internal schema modules MUST be in the same namespace as their corresponding document schema.

UBL internal schema modules must have semantically meaningful names. Internal schema module names identify the parent schema module, the internal schema module function, and the schema module itself.

[*SSM7*] Each UBL internal schema module MUST be named <ParentSchemaModuleName><InternalSchemaModuleFunction>

```
Example: ExtensionContentDatatype
```

3.7.4. External Schema Modules

External schema modules are used to group complex types and global elements that are used in multiple document schemas.

[*SSM8*] UBL schema modules MAY be created for reusable components.

UBL external schema modules organize the reusable components into logical groupings. At a minimum, UBL defines the following external schema modules:

1. UBL CommonAggregateComponents
2. UBL CommonBasicComponents
3. UBL Qualified Datatypes

In addition, UBL 2.1 imports the following schema module provided by UN/CEFACT.

1. CCTS Core Component Types

> [*NMS19*] The CCTS Core Component Type schema module must be represented by the namespace prefix "ccts-cct".

Furthermore, where extensions are used, an extension schema module must be provided. This schema module must be named:

COMMENT:

BH: Should the filename be named 'CommonExtensionCompnents' or does this only refer to the internal documentation name?

```
CommonExtensionComponents
```

> [*SSM21*] The UBL extension schema module MUST be identified as CommonExtensionComponents in the document name within the schema header.

> [*SSM22*] The UBL Qualified Datatypes schema module MUST import the UBL Unqualified Datatypes schema module.

To ensure consistency in expressing the CommonExtensionComponents schema module, a namespace prefix that will be used in all UBL schemas must be defined.

> [*NMS18*] The CommonExtensionComponents schema module namespace MUST be represented by the namespace prefix "ext" when referenced in other schemas.

3.7.4.1. UBL Common Aggregate Components Schema Module

The UBL library contains a wide variety of CCTS ABIEs, each defined as an xsd:complexType. Although some of these complex types may be used in only one UBL schema, many will be reused in multiple UBL schema modules. For ease of reuse, all the ABIE xsd:complexType definitions used in more than one UBL schema module are grouped into a single schema module of their own.

> [*SSM9*] A schema module defining all UBL Common Aggregate Components MUST be created.

> [*SSM10*] The UBL Common Aggregate Components schema module MUST be identified as CommonAggregateComponents in the document name within the schema header.

> [*NMS7*] The UBL Common Aggregate Components schema module MUST reside in its own namespace.

[*NMS8*] The UBL Common Aggregate Components schema module namespace MUST be represented by the namespace prefix "cac" when referenced in other schemas.

3.7.4.2. UBL CommonBasicComponents Schema Module

The UBL library contains a wide variety of CCTS BBIEs based on CCTS BBIE Properties. BBIE Properties are reusable in multiple BBIEs, and each is defined as an xsd:complexType. Although some of these complex types may be used in only one UBL schema, many will be reused in multiple UBL schema modules. For ease of reuse, all the BBIE Property xsd:complexType definitions used in more than one UBL schema module are grouped into a single schema module of their own.

[*SSM11*] A schema module defining all UBL Common Basic Components MUST be created.

[*SSM12*] The UBL Common Basic Components schema module MUST be identified as CommonBasicComponents in the document name within the schema header.

[*NMS9*] The UBL Common Basic Components schema module MUST reside in its own namespace.

[*NMS10*] The UBL Common Basic Components schema module namespace MUST be represented by the namespace prefix "cbc" when referenced in other schemas.

3.7.4.3. CCTS CoreComponentType Schema Module

CCTS defines an authorized set of Core Component Types that convey content and supplementary information related to exchanged data. As the basis for all higher level CCTS models, these Core Component Types are reusable in every UBL schema. The complex type definitions for all CCTS Core Component Types are collected in the Core Component Type schema module published by UN/CEFACT.

3.7.4.4. UBL Unqualified Datatypes Schema Module

The UBL Unqualified Datatypes Schema Module imports the CCTS CoreComponentType Schema Module.

[*NMS20*] The UBL Unqualified Datatypes schema module namespace MUST be represented by the prefix "udt" when referenced in other schemas.

3.7.4.5. UBL Qualified Datatypes Schema Module

UBL Qualified Datatypes are not expressed in the schema. Rather, data type qualifcations are expressed in the cva file.

[*SSM18*] A schema module without any declarations must exist.

[*SSM19*] The UBL Qualified Datatypes schema module MUST be identified as QualifiedDatatypes in the document name in the schema header.

[*NMS15*] The UBL Qualified Datatypes schema module MUST reside in its own namespace.

To ensure consistency in expressing the UBL Qualified Datatypes schema module, a namespace prefix that will be used in all UBL schemas must be defined.

[*NMS16*] The UBL Qualified Datatypes schema module namespace MUST be represented by the namespace prefix "qdt" when referenced in other schemas.

4. Modeling

4.1. Data Types for BBIEs

This level goes into the table of contents.

4.2. CCTS 2.01 Core Component Types

This level goes into the table of contents.

4.3. Naming Conventions

This level goes into the table of contents.

4.4. Oxford English

This level goes into the table of contents.

4.5. Acronyms and Abbreviations

This level goes into the table of contents.

4.6. Which and Why

This level goes into the table of contents.

4.7. Singular Nouns Where Applicable

This level goes into the table of contents.

4.8. Namespaces

This level goes into the table of contents.

4.9. Style Guide

This level goes into the table of contents.

4.10. Multiplicity and Preserving Functional Dependency

This level goes into the table of contents.

4.11. Patterns and Prohibitions

This level goes into the table of contents.

5. Spreadsheets

3. Spreadsheets - which columns are mandatory requiring data entry? - which columns are optional not requiring data entry? - which columns are fixed calculations of other columns and have no data entry? - note that eDoCreator has only one such column and I think there are others - which rows are expressed as enumerated code lists?

5.1. Mandatory Columns

This level goes into the table of contents.

5.2. Optional Columns

This level goes into the table of contents.

5.3. Fixed Calculation Columns

This level goes into the table of contents.

5.4. eDoCreator

This level goes into the table of contents.

5.5. Rows with Expressed Enumerated Code Lists

This level goes into the table of contents.

6. Naming and Design Rules

Naming and Design Rules - which aggregate schema declarations are created from the spreadsheets? - which basic schema declarations are created from the spreadsheets? - which document type schema declarations are created from the spreadsheets? - which context value associations are created from the spreadsheets?

6.1. Aggregate Schema Declarations

This level goes into the table of contents.

6.2. Basic Schema Declarations

This level goes into the table of contents.

6.3. Schema Declarations Created from Spreadsheets

This level goes into the table of contents.

6.4. Context Value Associationss Created from Spreadsheets

This level goes into the table of contents.

7. Schema Dependencies

Schema dependencies - which common schema fragments support the declarations from the spreadsheets? (see the SGTG strategy document to populate this section)

7.1. Common Schema Fragments

8. Code Lists

8.1. Genericode

9. Conformance

The last section contains the conformance clauses/statements.

A. Normative Annex

Normative appendices are not marked as non-normative using an attribute.

B. Non-normative Annex (Non-Normative)

Non-normative appendices are marked as such using an attribute.

C. Acknowledgements (Non-Normative)

In a typical OASIS work product one might wish to list committee participants in a non-normative annex (markup shown above in the normative annex example) using wording along the line of "The following individuals have participated in the creation of this specification and are gratefully acknowledged:"

- Mary Baker, Associate Member
- Jane Doe, Example Corporation Member
- John Able, Other Example Corporation Member

Note that the itemized list uses `spacing="compact"` to remove the space between list items in the printed result, not the HTML result).

D. Revision History (Non-Normative)

[optional; should NOT be included in OASIS standards]

Revision 0.5	8 July 2010	gkh
New template structure and filename conventions; mimic latest XHTML template		
Revision 0.4	03 Feb 2006	gkh
New IPR and use of revised 0.4 specification publishing environment; mimic latest Word and Open Office templates		
Revision 03	15 Aug 2002	ndw
Changed copyright holder.		
Revision 02	28 May 2002	ndw
Added IPR section.		
Revision 01	26 Apr 2002	ndw
Reworked after conversations with Eve.		
Revision 00	25 Apr 2002	ndw
First draft.		