UBL Guidelines for Customization

v 0.6

26 July 2008

Table of Contents

Table of Figures

1 Introduction

The OASIS Universal Business Language (UBL) Technical Committee (TC) has produced a vocabulary that, for many user communities, can be used "as is." However, the TC also recognizes that some user communities must address use cases whose requirements are not met by the UBL off-the-shelf solution. These Guidelines are intended to aid such users in developing custom solutions based on UBL.

To assist with the scoping of this document, let us begin with some definitions:

- **Customization**: The alteration of something in order to better fit requirements.
- **UBL Customization**: The description of XML instances, or XML-based applications acting on those instances, that are somehow based on or derived from the UBL Standard.

The goal of UBL customization is to maximize interoperability so that all parties understand the meaning of information in the documents being exchanged.

The determining factors governing when to customize may be business-driven, technically driven, or both. The decision should driven by real world needs balanced against perceived economic benefits.

1.1 Conformance vs. Compatibility

Once the need to customize UBL has been determined, designers must decide whether the result will be UBL *conformant* or UBL *compatible*. Although the UBL TC will not be involved in determining whether customizations are conformant, compatible or otherwise, we supply these definitions as a point of reference for those who might.

1.1.1 UBL Conformance

UBL conformance at the instance and schema level means there are no constraint violations when validating the instance against a standard UBL schema. A *UBL conformant instance* is an instance that validates against a standard UBL document schema. A *UBL conformant schema* is a schema that will validate only UBL conformant instances.

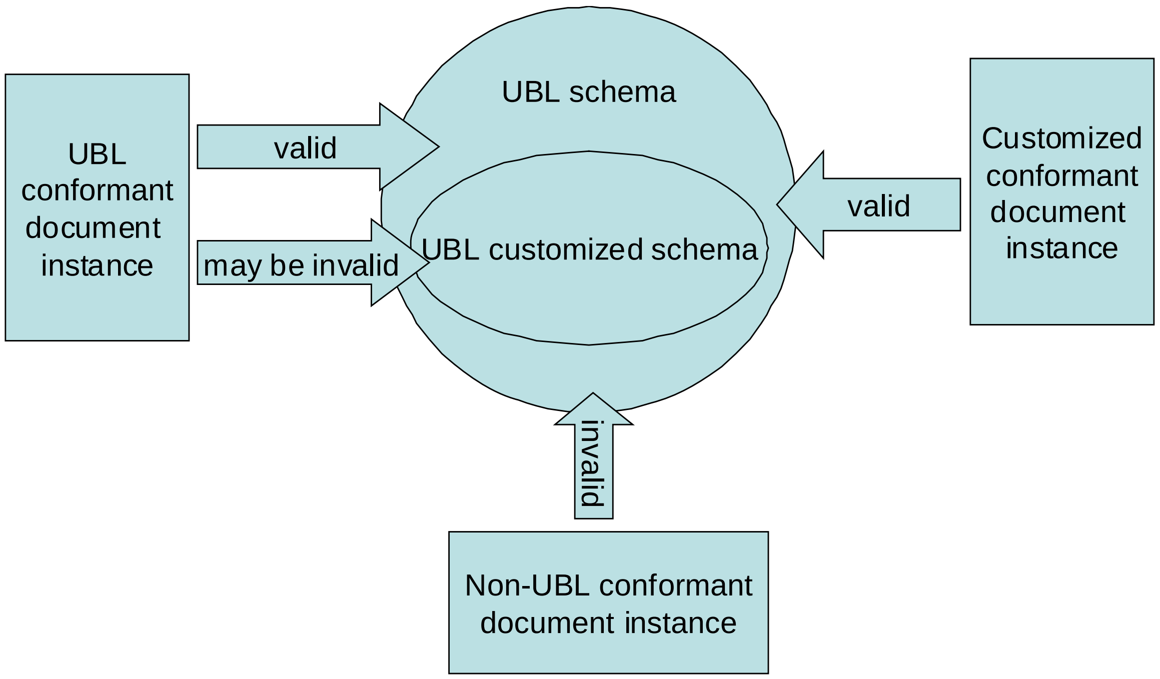

UBL conformant document instance
valid
may be invalid
UBL schema
UBL customized schema
valid
Customized conformant document instance
invalid
Non-UBL conformant document instance

Figure 1. Conformant Schemas and Document Instances

A major advantage of UBL conformance is that it minimizes the need for custom software or modifications to UBL applications.

1.1.2 UBL Compatibility

To be UBL compatible means to be consistent with the principles behind UBL's models or their development. These principles are defined in the ebXML Core Component Technical Specification (CCTS) and the UBL Naming and Design Rules (NDR). While we cannot assume conformance and interoperability of these customized documents, we can expect some degree of familiarity through the re-use of common objects.

1.2 Customization Overview

The UBL library and document schemas have been developed from conceptual models based on the principles of the ebXML Core Component Technical Specification. These are then expressed in W3C XML Schema, based upon the UBL Naming and Design Rules. It is these schemas that may be used to both specify and validate UBL documents.

It is recommended that a similar approach be followed when customizing UBL. Therefore, the following sections discuss conceptual design, then the specification of XML documents, and finally the validation aspects of customization.

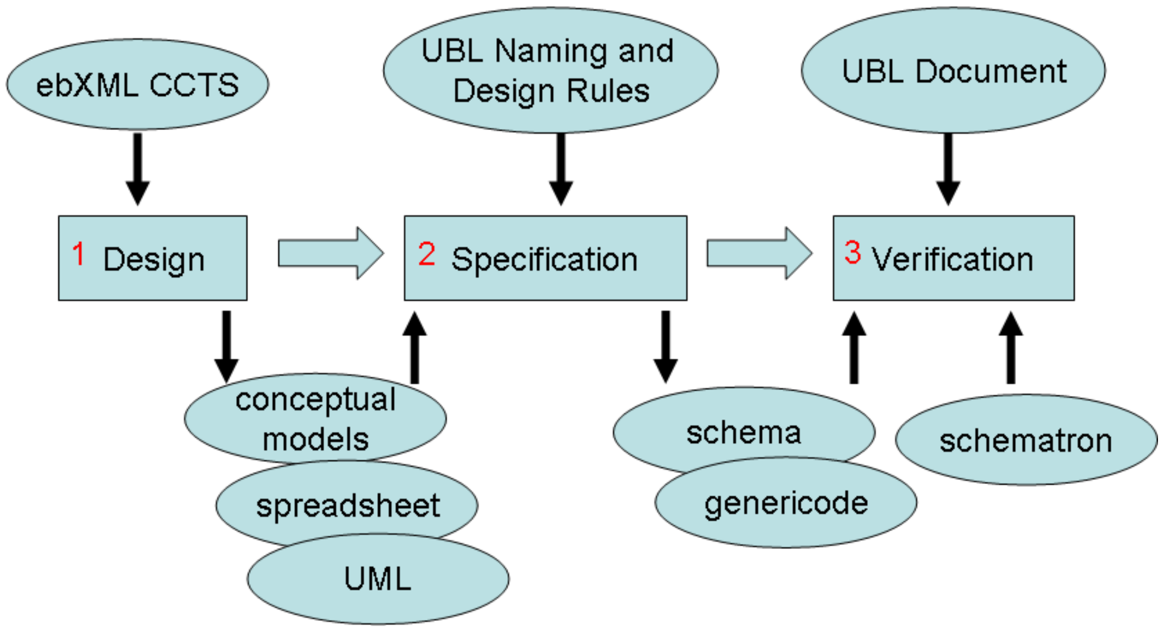


Figure 2. Customization Overview

1.3 Acknowledging OASIS Copyright

OASIS policies support implementations, subsets, and extensions of OASIS works as long as they acknowledge derivation from OASIS works and do not incorrectly claim compliance with or identity with an OASIS work. If you modify the UBL Invoice schema, for example, you cannot claim that it is still the UBL 2.0 Invoice schema, but you should acknowledge that the new work was derived from the UBL 2.0 Invoice schema.

Specifications and models published for use by others that incorporate OASIS work should include the following in an appropriate place, usually near the author's own copyright notice:

Portions copyright (c) OASIS Open 200[8]. All Rights Reserved.

This text can be followed by the OASIS policy URI if the author wishes to provide that reference:

http://www.oasis-open.org/who/intellectualproperty.php

Those who publish such works should take note of the rights available under the OASIS IPR Policy and their limitations, including any notices posted with respect to a specific work. In specific cases there may be parties other than OASIS who, from time to time, post assertions that a license is needed. For IPR notices relating to UBL, see

http://www.oasis-open.org/committees/ubl/ipr.php

OASIS generally welcomes the creation of derivative works, and in appropriate cases, OASIS may assist in publicizing the work in its own channels.

2 Designing for UBL Customization

The design of the conceptual models for UBL and its customizations is not affected by the syntactical issues of XML, schema lnguages, or validation tools. The UBL Technical Committee uses spreadsheets and UML models for their design, but this is not mandatory.

The approach to customization will reflect the chosen goal of either conformance or compatibility. Designing a customization may involve:

- Adding information items to meet requirements of a specific business context
- Omitting information items not needed in a specific context
- Refining the meaning of information items
- Creating constraints on possible values for information entities (such as code lists)
- Combining (or recombining) and assembling information items into new aggregations or documents

2.1 Designing for Conformance

When designing for conformance (see 1.1.1), the key objective is to create custom schemas that can be used to generate UBL-conformant instances. Consequently, with one exception, conformance only allows for restrictions:

- Subsets of the document model – restricting the number of entities in a document
- Constraints on document content – restricting the possible values an entity can have

In either case, the restriction may be accomplished either by removing optional objects from the UBL model or by checking for their existence in the value validation phase. Minimums can be increased, maximums can be decreased and data types may be refined but not extended. Hence, all schema-valid instances of a conformant customization are schema-valid instances of UBL as well; however, this is not true the other way around. Not all schema-valid instances of a UBL document will conform to every customization.

The one exception to this conformance “rule” is when the UBLExtension element is used. If new objects are added to an existing document type exclusively in the extension area, instances validating against the extended schema are still UBL conformant. But in these cases, schema validation cannot ensure the semantic integrity of the new objects.

2.1.1 Subsets of the Document Model

The standard schemas have been designed to accommodate the broadest possible range of business requirements. If all optional elements in a particular UBL document type were instantiated, the resulting instance would be extremely verbose. For example, if a UBL Order document contained just one instance of all the possible elements allowed by the UBL Order schema, that document would contain approximately 800,000 elements. Communities may not need all the information items provided by the standard schema. The use of subsets allows for the removal from a document model of any optional information items that are not needed to satisfy business requirements.

It must be noted that subsetting can only be used to remove optional elements or change cardinality in ways that do not reduce the required minimum number of occurrences or extend the permitted maximum number of occurrences. Thus,

- 0..1 can become 1..1 or 0..0 (but not, for example, 1..2)
- 0..n can become 0..1, 1..n, m..n, or 0..0
- 1..n can become 1..1 or 1..m (where m<n)
- 1..1 cannot be changed

2.1.2 Code List Constraints on Document Content

Using a code list (or an enumerated list) for an information entity is a common customization. Such lists impose value constraints. For example, "the Currency Code must be expressed using ISO 4217 codes" is a constraint on the possible values for Currency Code.

In UBL, there are two levels of constraints for codes:

- Code lists without defined values

  These are not empty lists, they are lists without constraints — in effect, infinite lists of values constrained only by their lexical form.
- Code lists with defined values

  These are explicit lists that constrain possible values for the content.

2.1.3 Other Constraints on Document Content

There are other cases in which the treatment of UBL instances requires customization in order to limit or restrict content values. For example:

- "The Total Value of an Order cannot exceed $100,000."
- "The length of an Address Line cannot exceed 40 characters."

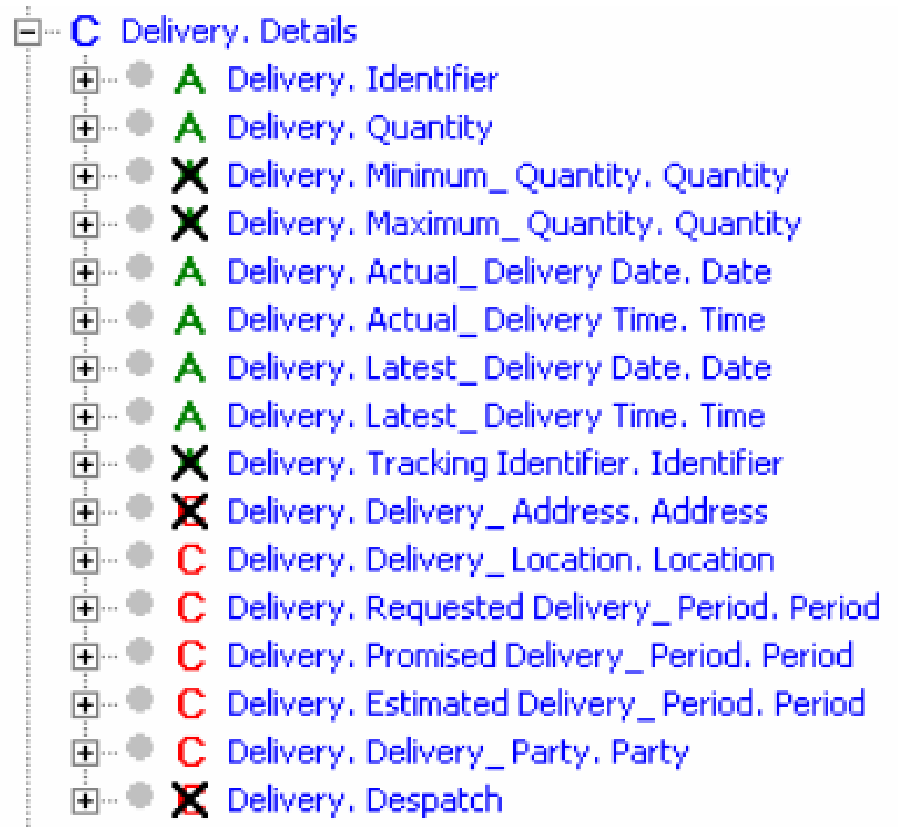
Additionally, there are other use cases relating to the dependencies between values of components which also necessitate customization. For example:

- "The Shipping Address must be the same as the Billing Address."
- "The Start Date must be earlier than the End Date."

Methods for specifying and validating such constraints are discussed in Sections 3 and 4.

2.1.4 Examples of Conformant Customizations

The Northern European Subset group (NES) also produce subset models of UBL 2.0 documents by selectively excluding components in the UBL library as described in the following graphic.



C Delivery. Details
- A Delivery. Identifier
- A Delivery. Quantity
- X Delivery. Minimum_ Quantity. Quantity
- X Delivery. Maximum_ Quantity. Quantity
- A Delivery. Actual_ Delivery Date. Date
- A Delivery. Actual_ Delivery Time. Time
- A Delivery. Latest_ Delivery Date. Date
- A Delivery. Latest_ Delivery Time. Time
- X Delivery. Tracking Identifier. Identifier
- X Delivery. Delivery_ Address. Address
- C Delivery. Delivery_ Location. Location
- C Delivery. Requested Delivery_ Period. Period
- C Delivery. Promised Delivery_ Period. Period
- C Delivery. Estimated Delivery_ Period. Period
- C Delivery. Delivery_ Party. Party
- X Delivery. Despatch

Figure 3. NES Subset of the UBL Delivery ABIE

As an example of a subset document, the following Notification document is a true subset of the UBL Receipt Advice document.

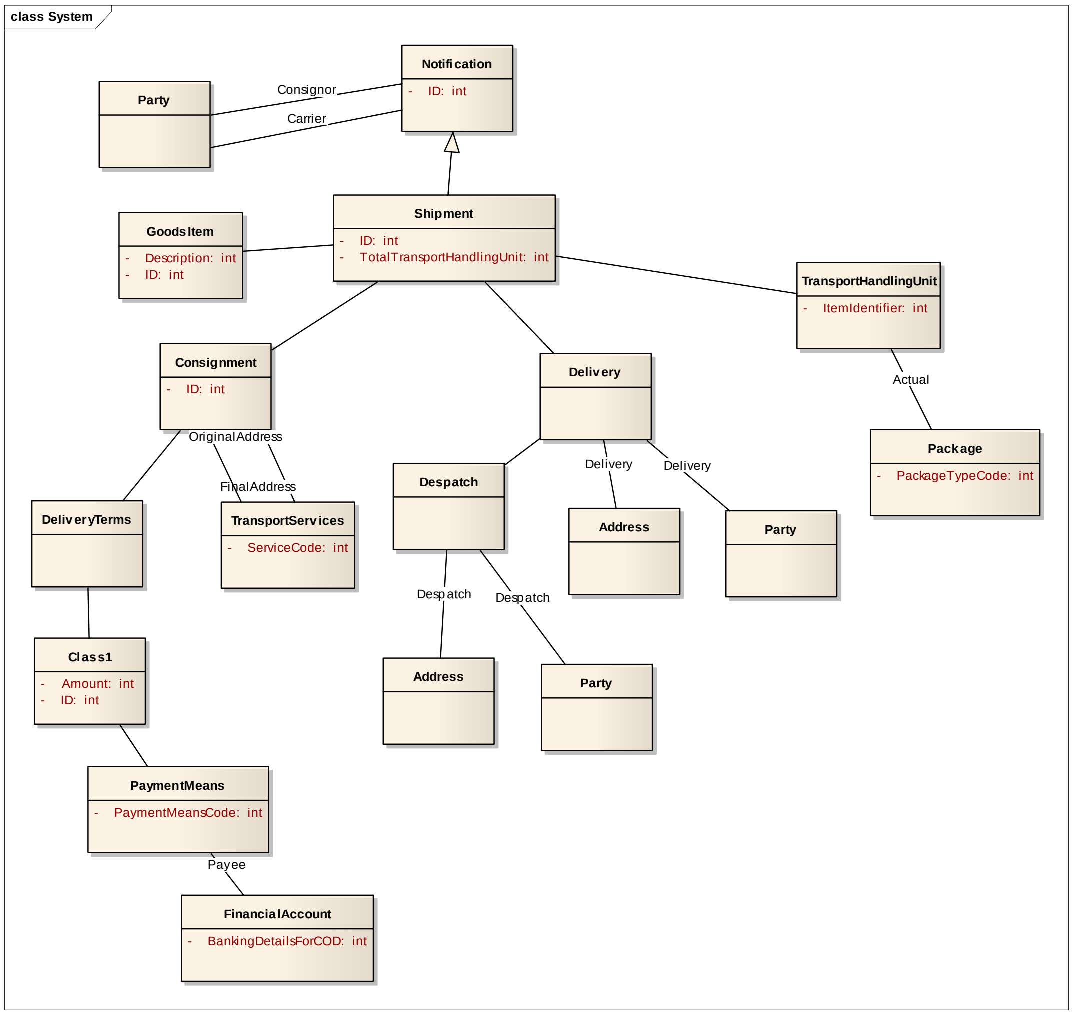


Figure 4. A Document as a Subset of a UBL Document

2.2 Designing for Compatibility

When designing for compatibility (see 1.1.2), the key objective is to re-use as much of the UBL model as possible. Where this is not possible, the guiding principles of the UBL model should be followed. Schema-valid instances of a compatible customization are not necessarily schema-valid instances of UBL. However, schema-valid instances of UBL may be schema-valid instances of a compatible customization. Unlike a conformant design, a compatible design allows for extensions (supersets). One may add to the model any UBL objects that are needed to satisfy business requirements.

2.2.1 Reuse of UBL Objects

Two categories of UBL objects are candidates for re-use:

- Business Information Entities (BIEs)

  A key goal of compatibility is to re-use existing UBL BIEs at the highest possible level. For example, it is better to re-use the UBL-declared BuyerParty element than to create a competing element with a similar content model. Re-using standard UBL constructs keeps customization as closely aligned with UBL 2.0 as possible and prevents an unnecessary proliferation of BIEs requiring maintenance.

- Data Types

  The ebXML CCTS defines a set of Core Component Types that should be the basis for all data types.

2.2.2 Compatible Extension of the UBL Model

If re-use of existing UBL constructs is not feasible, it is possible to customize by extending the UBL model. Extension may be required in a case where the context of use for a particular object differs from the UBL model. Indicating context of use is supported in the ebXML Core Component Technical Specification by qualifying the Property Terms of Dictionary Entry Names.

Example

In UBL, Address. Country Subentity Code. Code could be qualified as Address. Canadian_ Country Subentity Code. Code could be a qualification denoting the context of use is Canada.

If the new object has the same structure as the original object, it shares the same type. The qualifying terms used to name the new object should describe the role of the new object

Example

If an Address is required for a Party's local address that uses the normal address structure, it could be modelled as Party. Local_ Address.

If the new object does not have the same structure as the original object, the new object should include the original object as a child. The new object has a new name, not a qualified name. The other children of the new object are the additional information items needed to describe the new object.

Example

If an Address has additional properties when the Address is in Japan, then a new structure called Japanese Address could be created. This is not a qualification, but a new term. Ideally this should contain the original Address structure by association, plus the new properties.

Changing the meaning of any object's definition changes the object. Therefore, a new object must be defined

Example

In UBL, Communication. Channel. Text is defined as "The method of communication expressed as text." If an entity is required to define the Skype name as a specific communication channel then a new entity (perhaps called Communication. Skype Name. Text) should be defined.

2.2.2.1 New BBIEs

In certain scenarios, a user community may require new properties for their Basic Business Information Entities (BBIEs). This requires an extension of the existing UBL model to either create a new property based on an existing UBL data type or to create a completely new data type. Any new BBIE will result in a new ABIE container. See the sections below for further details.

2.2.2.1.1 Original Data Types

In cases where the representation term matches one of the existing UBL data types, a new property can be created based on this data type.

2.2.2.1.2 Refined Data Types

In cases where the representation term does not match an existing UBL data type, a new qualified data type may be required. You can create new qualified data types based on UBL qualified data types or UN/CEFACT unqualified data types.

{jb: has the following comment been dealt with?} *[TM: E.g.: currency code … is a CC and also a qualified datatype, but its CC type is Code… we are further qualifying a qualified datatype to make e.g. european currency code. At implementation level, we are creating a new XML datatype (limited set of values). So "refining a ccts dt" and creating a new XML datatype are really the same thing. No one will need to create a new CCTS datatype (name, code, text, etc.); every element is one of those. We are always "refining," never "creating new".]*

2.2.2.1.3 Refined Code Types

In UBL, a BBIE with a representation term of Code can have two data types assigned:

- Without defined values (the unqualified code data type)

  For example, CountrySubentityCode (in Address) is assigned the CodeType data type.
- With defined values (the code data type is qualified)

  For example, IdentificationCode (in Country) is assigned the CountryIdentificationCodeType data type.

Assigning a qualified code list to a BBIE that was previously unqualified restricts the infinite list into a finite list.   This restriction on possible content values defines a subset. Therefore, assigning a qualified code list to a BBIE that was previously unqualified is a *conformant* restriction.

Assigning a new qualified code data type to a BBIE already having assigned values will only be a conformant customization if the new qualified code list values are a subset of original qualified data type.

2.2.2.2 New ASBIEs

Aggregate Business Information Entities (ABIEs) are included in a document by associating them with their parent ABIE. This means defining a new Association Business Information Entity (ASBIE).

If the required aggregation has the same structure as an existing ABIE, a new ASBIE should be created with the existing ABIE.

The new ASBIE represents a new use of the ABIE and so qualifying terms can be used to describe the new role.

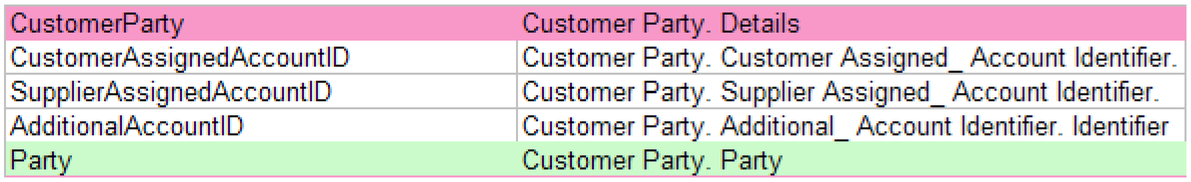
For example, Address is re-used in contexts such as Postal_ Address, Delivery_ Address, and Pickup_ Address. They all share the same structure as Address with “Postal,” “Delivery,” and “Pickup” providing the qualifying terms.

2.2.2.3 New ABIEs

If the required aggregation is an extension of an existing ABIE, making it no longer conformant, a new ABIE should be created with a new name (not a qualified name). The new ABIE includes the extended ABIE as a child (by association) with additional BIEs where required (see 2.2.2 above).

Example

In UBL, CustomerParty is a new ABIE that has a different structure than Party. The Party structure is re-used by inclusion in the CustomerParty ABIE. In addition, CustomerParty also contains additional BIEs. In this case, the name CustomerParty is not a qualification of the name Party, but an addition to the UBL model to create a new ABIE.



CustomerParty	Customer Party. Details
CustomerAssignedAccountID	Customer Party. Customer Assigned_ Account Identifier.
SupplierAssignedAccountID	Customer Party. Supplier Assigned_ Account Identifier.
AdditionalAccountID	Customer Party. Additional_ Account Identifier. Identifier
Party	Customer Party. Party

Figure 5. Extending an ABIE

2.2.2.4 New Data Types

Qualification of data types is another example of re-use by association. Qualified data types can be based on CCTS Unqualified data types or UBL qualified data types. For example, Currency_ Code. Type is a restriction on the Code data type which qualifies a CCTS unqualified data type. European Currency_ Code. Type is a restriction on the Currency_ Code Data Type which qualifies a UBL unqualified data type. ***{jb: The word “which” in the final two sentences introduces an ambiguity. Does the first sentence mean: “Currency_ Code. Type is a restriction on the Code data type, and the Code data type qualifies a CCTS unqualified data type,” or does it mean “Currency_ Code. Type, which qualifies a CCTS unqualified data type, is a restriction on the Code data type”? Does the second sentence mean “European Currency_ Code. Type is a restriction on the Currency_ Code Data Type, and the Currency_ Code Data Type qualifies a UBL unqualified data type,” or does it mean “European Currency_ Code. Type, which qualifies a UBL unqualified data type, is a restriction on the Currency_ Code Data Type”?}***

2.2.2.5 New Document Models

Where existing UBL document models do not meet requirements, it is necessary to create a new document model. The key steps in new document assembly are to select/create the document ABIE and assemble the required BBIEs and ASBIEs, applying cardinality constraints. The process then continues recursively through other BIEs.

2.2.3 Use UBL Principles for New BIEs

The minimum requirement for compatibility is to adhere to the UBL principles when creating or defining new BIEs.

- **Creating aggregates**

  When different BIEs have shared functional dependencies, they should be combined into aggregates. This means that the only things that belong in an ABIE are ASBIEs or BBIEs that are functionally dependent on it.

  For example, the description of an item depends on what that item is. If the item changes, then the description changes. We then say the description is functionally dependent on the item, and in this case, the BBIE Description should be aggregated into the ABIE Item.

  If the price of a cup of coffee is based on whether it is to take out, drink at the table or drink at the bar, then we say the price is functionally dependent on the location. In this case, the BBIE Price should be aggregated into an ABIE called PriceLocation.

- **Re-use common BIEs**

  Aggregates should re-use common ABIEs (as ASBIEs) and Data Types (as BBIEs).

- **Re-use patterns**

  Aggregates should re-use existing patterns for common structures.

- **Use CCTS**

  CCTS should be used when defining BIEs [check the first citation of CCTS]

- **Use UBL NDR for any schema**

  The UBL Naming and Design Rules should be used when implementing the model as an XML schema.

2.2.4 The Customization Ripple Effect

{jb: The following paragraph has been substantially edited and needs technical review.} The creation of a new BIE or data type affects all BIEs and data types in its parental path. This is known as the ripple effect. Every UBL construct has a distinct, unique identity; any change made within it changes the identity of the whole construct and everything above it.

For example, a UBL Address is always the same structure. If any BIE is added to, or required BIE is removed from, a UBL Address, it can no longer be identified as the UBL Address. And this change of identity bubbles or ripples upward through any parent of Address. This rule guarantees that UBL-consuming code is never “surprised” by an unexpected difference hiding inside an incoming data structure wrongly identified as standard UBL. This difference is generally signaled by a change in XML namespace.

2.2.4.1 Custom ABIEs using subsetting

Consider the following diagram of a UBL content model, which will be used to illustrate the ripple effect. Every construct is in the ubl: namespace.

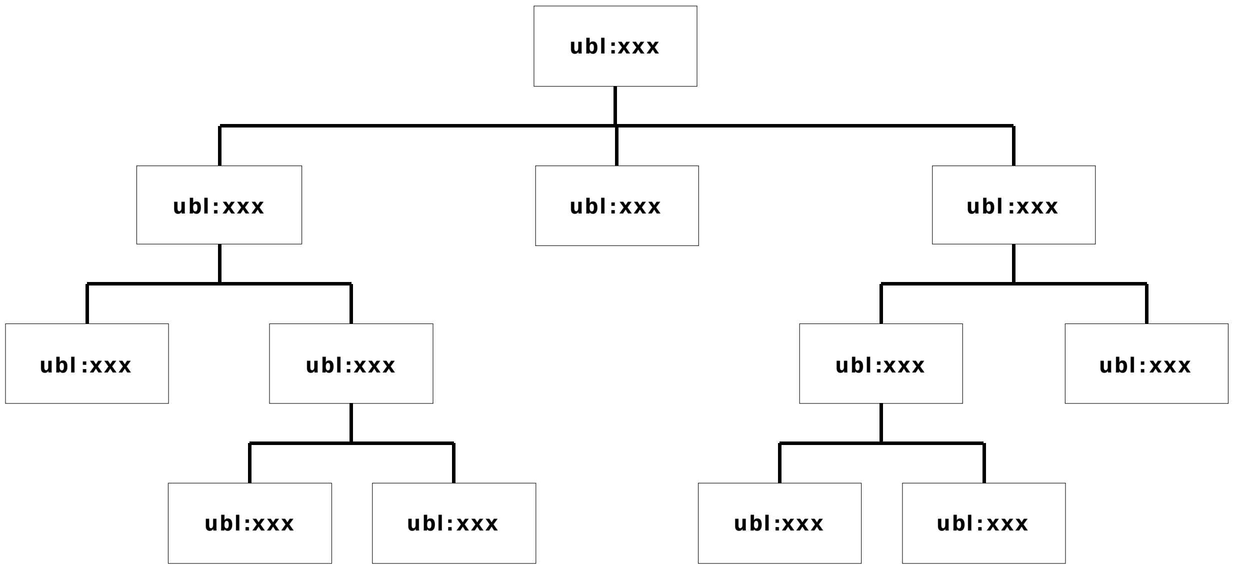


Figure 6. Schematic of a UBL Content Model

When a customization is a proper subset of UBL document model (only *optional* objects are removed), there is no ripple effect; everything keeps the ubl: namespace.

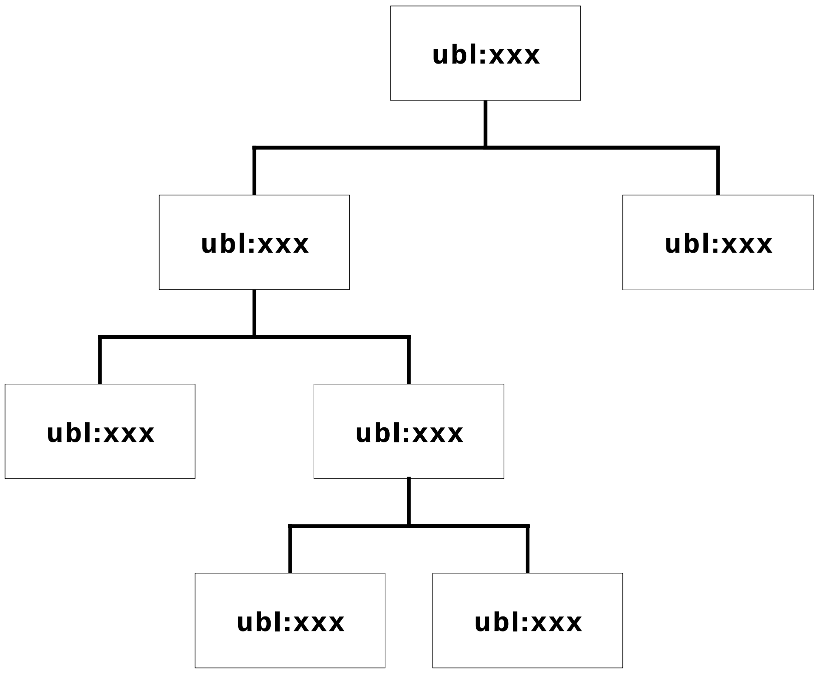


Figure 7. Conformant Subsetting (No Changes in Namespace)

2.2.4.2 Custom ABIE Using Standard UBL Properties

When a new ABIE/ASBIE is added to a customization, all of its ancestors must also be modified to reflect the new information item. In the example below, a custom ABIE is created using standard UBL properties. Its parent must then be customized to allow this custom ABIE in its content model. Accordingly, the document ABIE must also be customized.

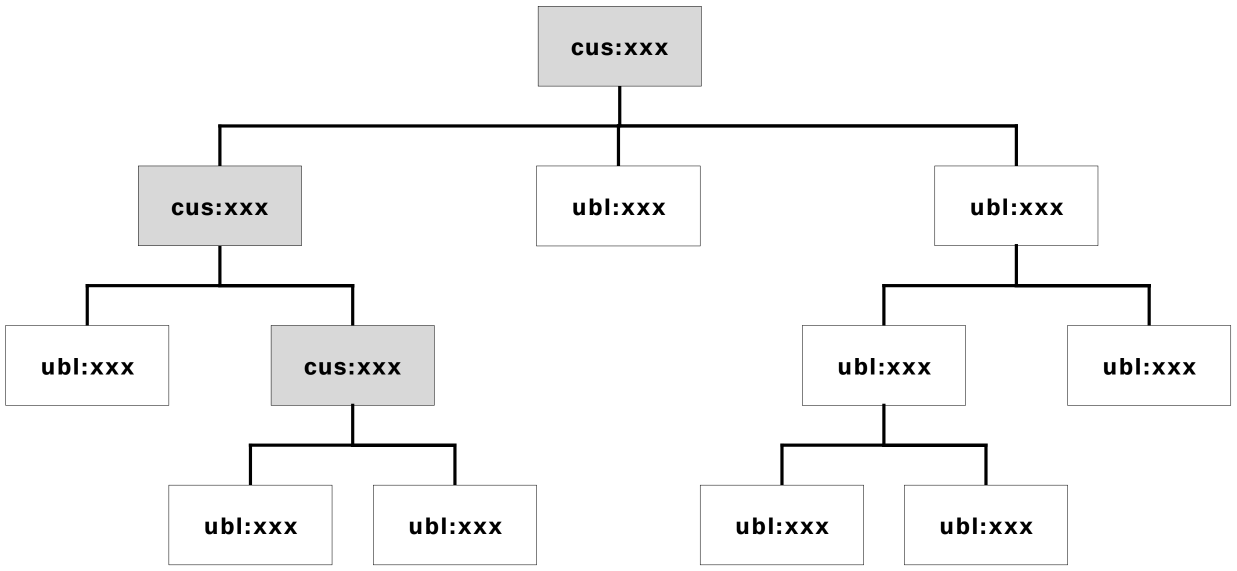


Figure 8. Ripple Effect — Custom ABIE/ASBIE

2.2.4.3 Custom ABIE Using Custom Properties

When a new BBIE is added to a customization, all of its ancestors must also be modified to reflect the new information item. In the example below, a customized ABIE is created by adding a custom BBIE. Its parent must then be customized to allow this custom BBIE in its content model. Accordingly, the document ABIE must also be customized.

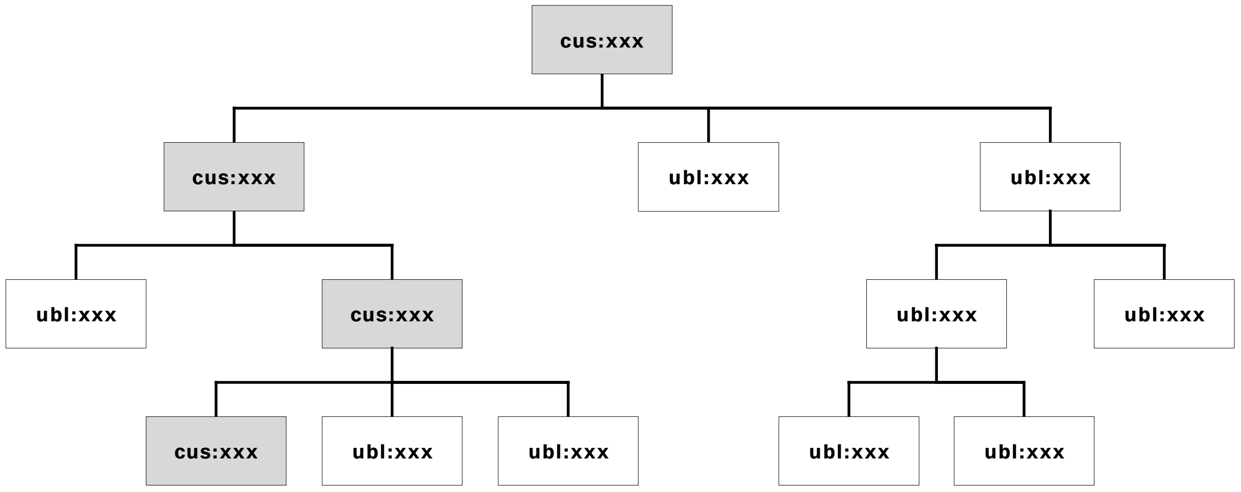


Figure 9. Ripple Effect — Custom BBIE

To sum up:

- Customizing a Data Type creates a new BBIE
- Customizing a BBIE creates a new ABIE

- Customizing an ABIE means creating a new ABIE and new ASBIEs that refer to it
- Customizing an ASBIE creates a new ABIE
- Any new ABIE means a new document model

3 Specification [placeholder for section to be written]

This section will discuss the options for the specification of a customization.

3.1 XML Schema Extension/Restriction [placeholder]

This section will describe the original UBL customization method.

[[MJG: Moved from 3.2.1.2 – didn't want to lose it…]]

For example, a customization may contain a purchaser object instead of the UBL BuyerParty object. For compatibility, at a minimum, the UBL CAC BuyerPartyType should be the basis for deriving your PurchaserType. The advantage of re-using UBL constructs is that there is a semblance of traceability back to the original UBL model.

3.2 Subset Schema [placeholder]

This section will describe how to create a schema that is a proper subset of the UBL schema.

3.3 UBLExtension Element [placeholder]

Note that if new items are added to an existing document type only in the extension area, instances validating against the extended schema are still UBL conformant (not just UBL compatible). [etc.]

3.4 XPath [placeholder]

This section will explain the XPath approach to customization specification.

4 Validation [placeholder for section to be written]

This section will discuss validation.

5 UBL Systems [placeholder for section to be written]

Although UBL will not be involved in determining whether systems are conformant, compatible or otherwise, we supply these definitions as a point of reference for those who might.

5.1 Producer Systems [placeholder]

The system will produce an instance that will validate against any UBL schema whose minor version number (within the indicated major range) is *equal to or greater than* the version to which the system claims conformance.

5.2 Consumer Systems [placeholder]

The system will accept instances that validate against any UBL schema whose minor version number (within the indicated major range) is *equal to or less than* the version to which the system claims conformance.