UBL Deprecation Policy version 1.0

1. Table of Contents

2. Introduction

This policy defines the guidelines and procedures for deprecating components within UBL. Deprecation is the process of marking components within UBL as obsolete, signaling to users that use of these components is no longer recommended and that they will be removed in future versions. Components covered by this policy are:

- Document types
- Business Information Entities (BIEs)
- BIE cardinalities
- BIE definitions in the semantic library
- Supplementary components

Deprecating UBL components is sometimes necessary. It allows for the removal of obsolete or problematic components, improving clarity, reducing ambiguity, enhancing usability and accommodating the evolving needs of the user community. For example, when there are ambiguous or redundant information items in the standard or when improved interoperability between different UBL implementations is desired. However, deprecating a component is a delicate decision that demands careful analysis and a clear understanding of the impact on existing users. It is important that deprecation of a component is done thoughtfully and with clear justification, and in a way that minimizes disruptions and confusion for users.

This policy exists to facilitate a more efficient and interoperable UBL ecosystem while taking into account the concerns of existing users and stakeholders.

3. Assessing component deprecation

The deprecation of a component may be warranted when one or more of the following situations are met and the correction results in breaking backwards compatibility:

- **Consolidation**
    - In cases where data redundancy or fragmentation needs to be eliminated for better data management and consistency, for example removing redundant components.
- **Ambiguity**
    - When the use of a component is unclear or can lead to conflicting interpretations, for example because of a vague or imprecise semantic definition.
- **Rectification**
    - To correct an error introduced in a previous UBL version, for example a spelling error in a component name.
- **Interoperability**
    - When a component is creating interoperability issues between users, for example when providing different ways of achieving the same functionality.
- **Feedback and user needs**
    - When feedback from users and stakeholders reveal that a component is causing confusion, inefficiency, or other issues.
- **No Longer Relevant**
    - When a component no longer serves a purpose or is no longer used in UBL, for example orphaned ABIEs in the semantic library.
- **Better Alternatives**
    - When there are better or more efficient ways to achieve the same functionality and it is in the interest of UBL to encourage the use of these alternatives.

When deciding to deprecate a UBL component, it should also be assessed how users can migrate from the deprecated component to another component in UBL. This is especially important when BIEs and Supplementary Components are deprecated because of redundancy or ambiguity.

4. Vetting process

Any member of the UBL TC may initiate the process for deprecating a UBL component. This process is initiated by submitting a written proposal to the TC mailing list, clearly identifying which component or components are being proposed and providing the rationale for the proposed deprecation. Once received, the proposal will be triaged to determine whether the TC will continue working on the proposal. If the TC cannot reach a consensus regarding the deprecation of a component, a majority vote among TC voting members is conducted to make the final decision.

5. Documentation

5.1. General

Each deprecation of a UBL component should be extensively documented to provide clarity and guidance for users. This documentation should include:

- a clear explanation of why the component has been deprecated,
- alternatives considered,
- instructions on how to transition from the deprecated component to a recommended alternative,
- expected impact on existing implementations,
- any dependencies to other components and/or document types that rely on the deprecated component.

The following general requirements apply when documenting a deprecated component:

- Documentation must contain information about the UBL version number from which the component was deprecated.
- The deprecation of the component must be mentioned in the release notes of the UBL specification.
- Deprecated components must be clearly labeled in all documentation, libraries, schemas, and other relevant artefacts.
- A warning message must be added to all applicable documentation and artefacts, encouraging users to stop using the component.

In addition, specific documentation requirements may apply depending on the type of component being deprecated, as outlined in the following sections.

5.2. Deprecating document types

When a UBL document type is deprecated, the specification and documentation must be updated in the following ways:

- All mentions of the deprecated document type and of its declared namespace must clearly label it as deprecated.
- The document schema documentation (currently under section 3.2) must be replaced with a warning that the document type is deprecated as well as any helpful background information and migration guidance.
- All processes and process diagrams that involve the deprecated document type must be either removed or updated to use a viable alternative document type.
- Technical artefacts, such as XSDs and other schemas, must carry a warning message that the document type is deprecated.

- All occurrences of the document type in the common library and document models must clearly identify the document type as deprecated.
- All examples using the deprecated document type must either be removed or changed to use an alternative document type.

5.3. Deprecating Business Information Entities (BIEs)

When deprecating a BBIE or an ABIE, the specification and documentation must be updated in the following ways:

- All mentions of the deprecated BIE must clearly label it as deprecated.
- Technical artefacts, such as XSDs and other schemas, must carry a warning message that the BIE is deprecated.
- All occurrences of the BIE in the common library and document models must clearly identify the BIE as deprecated.
- For ABIEs, all associations to the ABIE (i.e., ASBIEs) must also clearly identify the BIE as deprecated.
- If an alternative BIE or other functionality is being recommended, then the recommended approach should be available everywhere in the common library and document models where the deprecated BIE was previously available.
- The deprecated BIE must either be removed from the example UBL instances, or the examples must be changed to use an alternative recommended approach.
- Where feasible, validation artefacts should show a warning message when encountering use of the deprecated BIE.

5.4. Deprecating cardinalities

Changing the cardinality of a BIE from optional to mandatory or from unbounded to bounded is sometimes desirable, however, it is not feasible between minor versions of UBL because of backwards compatibility. Instead, the cardinality can be deprecated, informing users that a cardinality change is planned for a future UBL release.

When deprecating the cardinality of a BIE, the specification and documentation must be updated in the following ways:

- In the common library or document model, the BIE must show both the deprecated and the new cardinality, with a clear warning that the old cardinality is deprecated.
- Technical artefacts, such as XSDs and other schemas, must carry a warning message that use of the old cardinality is deprecated.
- Example UBL instances must be updated to avoid using the deprecated cardinality.
- Where feasible, validation artefacts should show a warning message when encountering use of the deprecated cardinality.

5.5. Deprecating semantic definitions

Improving the semantic definition of a BIE or a document type within the common library or the document models can be desirable, for example to clarify its intended use. However, if such a change carries the risk of invalidating previous interpretations or uses of the BIE, it cannot be changed within minor versions of UBL as this would break backwards compatibility. Instead, the old semantic definition must be deprecated, and a new definition must be communicated to users.

When deprecating a semantic definition, the specification and documentation must be updated in the following ways:

- In the common library or document model, the BIE must show both the deprecated and the new definition, with a clear warning that the old definition is deprecated.
- Technical artefacts, such as XSDs and other schemas, must carry a warning message that use of the old definition is deprecated.
- Example UBL instances must be updated to ensure that example values are conformant with the new definition.

5.6. Deprecating supplementary components

When deprecating the use of a CCTS supplementary component in UBL, the specification and documentation must be updated in the following ways:

- All mentions of the deprecated supplementary component must clearly label it as deprecated.
- The documentation of data type qualifications (currently appendix D of the UBL specification) must clearly label the supplementary components as deprecated as well as provide helpful background information and migration guidance.
- Technical artefacts, such as XSDs and other schemas, must carry a warning message that the supplementary component is deprecated.
- All common library and document models documentation must clearly identify the supplementary component as deprecated.
- The deprecated supplementary component must either be removed from the example UBL instances, or the examples must be changed to use an alternative recommended approach.
- Where feasible, validation artefacts should show a warning message when encountering use of the deprecated supplementary component.

6. Removal of deprecated components

6.1. Removing components

Deprecated components will only be removed from the UBL library when transitioning to a new major version of UBL. This approach ensures that backward compatibility is maintained within each major version. Consequently, when moving to the next minor version within the same major version, deprecated components will not be removed. This strategy aims to avoid breaking existing systems and to allow a smoother transition for users, as they can continue using deprecated components within the same major version without any disruption.

6.2. Future syntax bindings

When incorporating new syntax bindings into UBL, the question of backwards compatibility is not a concern. As a result, new syntax specifications may selectively omit the use of deprecated components, irrespective of the major or minor version of the UBL semantic library used when introducing the new syntax binding.