XBRL GL Next Taxonomy Framework — Part 2: Tuple Palette Taxonomy

Document status: Draft — For Collaborative Review
Date: **2026-11-06**
Author: Nobuyuki SAMBUICHI
Role: Advisor to XBRL Japan
Project: XBRL GL Next
Distribution: Public collaborative review draft.
Repository version: public collaborative review draft.
Editors: SAMBUICHI, Nobuyuki
Contributors: invited project participants (individual names not yet confirmed for publication).

Status

This document is a draft prepared by Nobuyuki SAMBUICHI, Advisor to XBRL Japan, as part of the XBRL GL Next project. It is made available through a public GitHub repository for collaborative review. It has not been approved or endorsed as an official specification by XBRL Japan, XBRL Europe, XBRL International, or any other organisation. Copyright and licence terms remain under review.

Abstract

This Part specifies the XBRL 2.1 Tuple palette taxonomy realisation of XBRL GL Next from formal, root-specific HMD-for-taxonomy inputs governed by Part 1. It preserves the approved semantic hierarchy, occurrence conditions and Attribute meanings by mapping Classes and References to Tuple structures and Attributes to XBRL item concepts.

It defines reusable module declarations, HMD-specific structural content schemas, labels and presentation networks, message entry points, root-specific Discoverable Taxonomy Sets (DTSs), packaging and validation requirements. The Accounting Entries and Business Transactions examples illustrate independent HMD roots and the controlled reuse of module-owned declarations.

The Tuple binding is distinct from the OIM and proposed Tavi realisations in Parts 3 and 5. Differences in physical structures do not authorise changes to the agreed business semantics or the authoritative HMDs.

Table of Contents

Status 1

Abstract 1

1 Language independence 4

2 Introduction 4
- **2.1 Scope** 4
- **2.2 Normative language** 4
- **2.3 Relationship to the other Parts** 4
- **2.4 Implementation independence** 5

3 Terms and definitions 6

4 Authoritative input and validation contract 8
- **4.1 Input authority** 8
- **4.2 Required 18-column order** 8
- **4.3 XPath and QName authority** 8
- **4.4 Row validation** 8
- **4.5 Non-use subtrees** 9
- **4.6 Cross-HMD shared declarations** 9

5 Tuple concept and content-model mapping 9
- **5.1 Module declarations** 9
- **5.2 Class and Reference rows** 9
- **5.2.1 Class hierarchy and occurrence identity** 10
- **5.3 Attribute rows** 10
- **5.4 Direct-child sequence** 10
- **5.5 Shared `gen` module and Value Domain realisation** 11

6 Palette taxonomy 12
- **6.1 Tuple module schema** 12
- **6.2 HMD content schemas** 12
- **6.3 Tuple message entry point** 12
- **6.4 DTS assembly and validation boundary** 12
- **6.5 DTS constraints** 13

7 Labels and presentation networks 13
- **7.1 Labels** 13
- **7.2 QName identity and locator resolution** 13
- **7.3 Module presentation linkbase** 13
- **7.4 Presentation Linkbase Set and message selection** 13
- **7.5 Design rationale: module-level presentation (informative)** 14
- **7.6 Linkroles and extended link roles** 14

8 accountingEntries baseline and businessTransactions extension example (informative) 14
- **8.1 accountingEntries HMD** 14
- **8.2 businessTransactions HMD** 14
- **8.3 Required result** 15

9 Formal package assembly 15
- **9.1 Formal package tree** 15
- **9.2 Version and namespaces** 16
- **9.3 Deterministic HMD identifier** 16
- **9.4 All-or-nothing publication** 16

10 Validation and conformance 16
- **10.1 Conformant taxonomy generator** 16
- **10.2 Minimum test set** 16

11 Tuple instance examples (informative) 17
11.1 Purpose 17
11.2 Common XBRL instance components 17
11.3 Accounting Entries example 17
11.4 Business Transactions example 19
11.5 Relationship to presentation 21
Appendix A (informative) Designer guidance for the Palette taxonomy 22
A.1 Validation boundary 22
A.2 Palette assembly overview 22
A.3 Presentation forest 22
A.4 Designer cautions 22
A.5 Suggested validation procedure 22
Appendix B References 23
Appendix C Intellectual property status 24
Appendix D Document history 25
Appendix E Errata corrections in this document 28
Appendix F (informative) Correspondence with XBRL GL Next Requirements 29

1 Language independence

This collaborative review draft is written in English. The preferred spelling convention is UK English. XML names, QNames, namespace URIs, and normative taxonomy filenames are case-sensitive and are not translated.

2 Introduction

2.1 Scope

This Part defines the XBRL 2.1/XML Tuple binding of the XBRL GL Next Taxonomy Framework. It specifies deterministic taxonomy generation from the **HMD for taxonomy** input set, the Tuple Palette module components, root-specific **DTS** assembly, labels and presentation networks, validation requirements, and informative instance examples. It does not permit semantic-model correction during taxonomy generation and does not define the OIM dimensional binding governed by Part 3.

2.2 Normative language

The key words “MUST”, “MUST NOT”, “SHOULD”, “SHOULD NOT”, and “MAY” in this document are to be interpreted as described in BCP 14 (RFC 2119 and RFC 8174) when, and only when, they appear in all capitals.

“MUST” and “MUST NOT” indicate mandatory requirements and prohibitions, respectively. “SHOULD” and “SHOULD NOT” indicate recommendations for which deviations may be justified in particular circumstances, provided that their implications are carefully considered. “MAY” indicates a permitted option.

The normative or informative status of provisions is determined by the document's stated conventions and the context in which they appear.

2.3 Relationship to the other Parts

The XBRL GL Next Requirements Specification provides the background, design objectives, and requirements for the Framework.

The XBRL GL Next Taxonomy Framework consists of the following Parts:

- XBRL GL Next Taxonomy Framework — Part 1: Semantic Modelling and Taxonomy Foundations;
- XBRL GL Next Taxonomy Framework — Part 2: Tuple Palette Taxonomy;
- XBRL GL Next Taxonomy Framework — Part 3: OIM Palette Taxonomy;
- XBRL GL Next Taxonomy Framework — Part 4: Aligned Pool and Extension Governance;
- XBRL GL Next Taxonomy Framework — Part 5: Tavi Palette Taxonomy.

Part 1 defines the serialisation-independent semantic pipeline and the formal HMD contract. This Part 2 defines the XBRL 2.1 Tuple realisation generated from formal HMDs. Part 3 defines the OIM realisation generated from the same HMDs. Part 4 governs extension classification, registration, alignment and publication. Part 5 defines purpose-specific HMD views and their Tavi taxonomy and Table Template realisation.

This Part 2 specifies how formal HMDs are realised as XBRL 2.1 Tuple taxonomies, including reusable module declarations, message entry points, structural relationships, and the corresponding taxonomy generation requirements.

Parts 2, 3 and 5 provide alternative taxonomy realisations of the common semantic framework established by Part 1, under the extension governance defined in Part 4. Each HMD serves as a purpose-specific palette, selecting reusable semantics from COR, BTX and AAA, which constitute the semantic paint boxes.

2.4 Implementation independence

The requirements in this Part define the HMD-for-taxonomy input and the observable properties of the Tuple realisation. The sole maintained reference Generator authority is `tools/taxonomy/xBRLGL_TaxonomyGenerator.py`. Datatype mapping and overrides, EE1 Value Domain handling, selector-effective multiplicity, occurrence handling, Tuple/OIM generation, manifest output, and deterministic generation are handled by the Generator and its supporting modules. `tools/taxonomy/datatype_binding.py` is a required runtime dependency.

3 Terms and definitions

For the purposes of this document, the terms and definitions given in **Part 1** and the following apply. Unless otherwise specified, these terms constitute the common terminology used in this Part and the other Parts of the XBRL GL Next Taxonomy Framework.

HMD root
the root Class selected from an **HMD for taxonomy** to serve as the root of a root-specific syntax binding. In the semantic model it is identified by its Class semantic-model identifier; the selected syntax binding assigns the applicable QName.

formal HMD
one **HMD_for_taxonomy** CSV produced by Post-Graph Walk and containing exactly one effective level-1 root.

HMD identifier
the stable filesystem identifier obtained from the **HMD root** module identifier, an underscore, and the root local name; for example cor_accountingEntries.

conformant taxonomy generator
the software implementation that processes one conforming **HMD for taxonomy** input set and generates the complete formal Tuple and OIM package required by Parts 2 and 3.

shared palette declaration
one module-owned global element declaration used in one or more **HMDs** with an identical expanded QName and concept-defining properties. For a Class or Reference declaration, the ordered direct-child QName and multiplicity signature is also identical wherever the declaration is reused.

Tuple module schema
the versioned <module>-<V>.xsd document that owns reusable global Attribute item declarations and Class/Reference Tuple declarations for one module namespace. Structural complexTypes referenced by Class and Reference declarations are supplied by **HMD**-specific **content schemas**, so the module schema is not a standalone validation entry point.

content schema
an **HMD**-specific <module>-content-<V>.xsd document in the same module namespace that includes the **Tuple module schema** and supplies the structural complexTypes required by the selected **HMD DTS**.

module presentation linkbase
the binding-specific presentation linkbase owned by a module. For the Tuple binding it is named <module>-pre-<V>.xml and contains one or more reusable presentation trees rooted at Class or Reference concepts owned by that module. Tree traversal may cross module boundaries.

Presentation Linkbase Set
the binding-specific set of **module presentation linkbase**s published with the taxonomy package. The Tuple set contains one <module>-pre-<V>.xml for each published module. A root-specific Tuple **message entry point** discovers the member corresponding to its root module; that member contains the complete presentation tree reachable from the selected **HMD root** and may also contain other reusable trees owned by the same module.

Discoverable Taxonomy Set
DTS
the set of taxonomy schemas and linkbases discovered from an XBRL entry point in accordance with XBRL discovery rules. In this Part, the root-specific Tuple **message entry point** defines the validation and discovery boundary for one **HMD**.

message entry point
: the root-specific schema through which one **HMD** Tuple **DTS** is discovered.

concept locator
: a link:loc whose xlink:href identifies a global concept declaration by the URI of its owning **Tuple module schema** followed by the concept XML id.

formal package
: the complete generated taxonomy package containing the binding-specific module palettes and one Tuple and one OIM **message entry point** for every **formal HMD** in the accepted **HMD for taxonomy** input set.

linkrole
: the role URI carried by xlink:role on an extended link; it identifies the network to which the relationships in that extended link belong.

extended link role
ELR
: a standard or custom **linkrole** used to distinguish one label, presentation, calculation, definition, or reference network from another.

`gen` taxonomy module
: the one shared generated taxonomy module containing binding-neutral reusable semantic definitions and used by both Tuple and OIM taxonomy assemblies.

Value Domain binding
: the explicit association of a semantic Attribute occurrence with a separately identified governed permitted value set and its approved XBRL realisation, independent of datatype binding.

4 Authoritative input and validation contract

4.1 Input authority

A conformant taxonomy generator MUST accept the **HMD for taxonomy** input set defined in Part 1. The set contains one or more formal 18-column HMD_for_taxonomy CSV files. Each **formal HMD** MUST be an unmodified output of Post-Graph Walk and is authoritative for the hierarchy, QNames, semantic datatypes, multiplicities, labels, definitions, and occurrence order used by the Tuple binding. The XBRL item type is selected by the separate explicit datatype-binding operation in 5.3.

Each **formal HMD** MUST contain exactly one effective level-1 root. Every **formal HMD** in the input set MUST satisfy Clause 4 before any **formal package** is published. Implementation-specific manifests, execution records, and diagnostics are outside the normative taxonomy-generation input contract.

4.2 Required 18-column order

```
sequence,module,level,type,identifier,name,datatype,multiplicity,value_domain,definit
ion,label_local,definition_local,source_bsm_id,semantic_path,associated_module,class_
term,local_name,xpath
```

A conforming **formal HMD** MUST contain exactly the 18 columns shown above in that order. A byte-order mark MAY precede the first header name. Empty data rows MAY be ignored.

4.3 XPath and QName authority

The xpath column is authoritative for the physical hierarchy. Every XPath MUST start with /xbrli:xbrl/ and every subsequent step MUST be a lexical QName of the form gl-<module>:<local-name>. The number of QName steps after xbrli:xbrl MUST equal level.

For each row, the terminal XPath step MUST resolve to the namespace for module and have the local part given by local_name. A mismatch is an error.

HMD	Root module	Root local_name	Normative root XPath	HMD directory identifier	Root prefix <P>
Accounting Entries	cor	accountingEntries	/xbrli:xbrl/ gl-cor:accountingEntries	cor_accountingEntries	cor
Business Transactions	btx	businessTransactions	/xbrli:xbrl/ gl-btx:businessTransactions	btx_businessTransactions	btx

4.4 Row validation

- sequence MUST be an integer and input order MUST determine the order of direct children within a content model.
- level MUST be a positive integer and MUST equal the number of physical QName steps in xpath.
- type MUST be C, A, or R. C denotes Class, A denotes Attribute, and R denotes Reference or other structural Association occurrence.
- module and local_name MUST each be a valid XML Schema NCName.
- multiplicity MUST be one of 0..0, 1, 1..1, 0..1, 0..*, or 1..*.
- The semantic datatype value MUST be empty for C and R rows. For every A row, an approved explicit override or default semantic-datatype mapping MUST resolve an XBRL item type; an unknown or `review-required` result is an error.
- semantic_path MUST be unique within the **HMD**.

- Every non-root physical parent XPath MUST precede its child in the same **HMD**.
- Every occurrence of the same expanded QName MUST have consistent row type and datatype definitions.

4.5 Non-use subtrees

A row with multiplicity 0..0 and every physical descendant of that row MUST be excluded from generated taxonomy content. Normally Post-Graph Walk has already removed the non-use subtree from the **formal HMD**.

4.6 Cross-HMD shared declarations

The conforming result MUST be independent of HMD input order. Cross-HMD equality requirements apply only to declarations deliberately governed as shared. A deliberately shared Attribute QName MUST have identical concept-defining values. A deliberately shared Class or Reference global declaration MUST have the same expanded QName, substitution group, nillability, XML id, and named type QName. HMD-specific structural complexType content is validated within its root-specific DTS and MAY differ between independent HMD DTSs.

Within one assembled Tuple DTS, each expanded element QName and expanded type QName MUST resolve exactly once, and each referenced structural complexType MUST be consistent within that DTS. A conflict in a deliberately shared declaration is an error. Mere reuse of a QName in another independent root-specific DTS does not by itself create a cross-DTS content-model equality requirement.

5 Tuple concept and content-model mapping

5.1 Module declarations

For each effective A row, the conformant taxonomy generator MUST derive the expanded item QName from the terminal XPath step. The global Attribute item declaration and its item type MUST be owned by the corresponding reusable module schema <module>/<module>-<V>.xsd. Every effective C and R QName MUST likewise have its global Tuple element declaration in the owning reusable module schema. The HMD-specific content schema supplies the named structural complexTypes referenced by those C/R declarations.

5.2 Class and Reference rows

Every effective C and R row MUST correspond to one global XBRL Tuple element declaration in the reusable module schema for its module, with substitutionGroup xbrli:tuple, nillable=false, and type <local_name>ComplexType in the same module namespace. The corresponding structural complexType MUST be defined in the HMD-specific content schema for that module. The global declaration is reusable module content; the structural type definition is root-specific DTS content.

```
<xs:element name="accountingEntries" id="cor_accountingEntries" type="cor:accountingEntriesComplexType"
substitutionGroup="xbrli:tuple" nillable="false"/>
```

The HMD content schema uses the same module namespace as the reusable module schema and includes that module schema. This combines module-owned global Attribute, Class, and Reference declarations with HMD-specific structural complexTypes. Independent HMD DTSs MAY therefore use the same structural type QName with different effective content models, provided those content schemas are never assembled into one DTS.

5.2.1 Class hierarchy and occurrence identity

Tuple structure MUST remain faithful to every effective Class in the authoritative HMD hierarchy. A non-root Class with multiplicity `0..1` or `1..1` remains a Tuple and complexType in its semantic position even though Part 1 does not classify it as an occurrence-key Class.

Occurrence-key classification is relevant to Structured CSV and the OIM dimensional realisation; it does not remove or flatten Tuple Classes. Tuple nesting itself represents structural occurrence and containment. In the hierarchy `Invoice / InvoiceLine [1..*] / ItemInformation [1..1] / ItemAttributes [0..*]`, all four Classes remain in the Tuple content model.

For nested repeated Tuple Classes such as `Parent [0..*] / Child [0..*]`, each Child Tuple is physically nested in its Parent Tuple. This preserves parent-scoped Child occurrence identity without adding an OIM typed dimension to the Tuple content model.

5.3 Attribute rows

Every A row MUST be declared as an XBRL item with substitutionGroup xbrli:item, nillable=true, and xbrli:periodType=instant. Its HMD `datatype` remains a semantic representation term. The XBRL item type MUST be selected deterministically by applying an approved explicit occurrence or concept override first and the approved default semantic-datatype mapping second. If neither resolves the semantic datatype, or if the applicable result is `review-required`, generation MUST fail. The generator MUST NOT use a silent `xbrli:stringItemType` fallback or infer the type from `name`, `local_name`, a lexical value, or a suffix. An approved named type in the shared **`gen` taxonomy module** MAY be selected by that binding operation.

HMD datatype	XBRL item type
String	xbrli:stringItemType
Boolean	xbrli:booleanItemType
Date	xbrli:dateItemType
Date Time	xbrli:dateTimeItemType
Decimal	xbrli:decimalItemType
Integer	xbrli:integerItemType
Monetary	xbrli:monetaryItemType
Pure	xbrli:pureItemType
Token	xbrli:tokenItemType
QName	xbrli:QNameItemType

5.4 Direct-child sequence

The complex type of each C or R declaration MUST contain an xs:sequence of its immediate physical children in HMD order, with occurrence constraints derived from multiplicity.

Multiplicity	minOccurs	maxOccurs
1 or 1..1	1 (default)	1 (default)
0..1	0	1 (default)
0..*	0	unbounded
1..*	1 (default)	unbounded

Each generated structural complexType MUST be defined in the appropriate **HMD**-specific <module>-content-<V>.xsd and MUST permit an optional XML id attribute of type ID. A direct child from the same module MUST be expressed with xs:element ref to the declaration included from the module schema. A cross-module direct child MUST be expressed with xs:element ref to the QName in the imported module namespace. Two definitions for the same expanded type QName within one assembled **DTS** are an error. Different **content schemas** belonging to independent **HMD DTSs** MAY supply different effective content models for the same named type, provided that they are never assembled into the same **DTS**.

5.5 Shared `gen` module and Value Domain realisation

The Tuple taxonomy assembly MUST use the same shared generated **`gen` taxonomy module** as the OIM assembly. The `gen` module contains binding-neutral reusable semantic definitions; Tuple content schemas, Tuple declarations, and Tuple hierarchy MUST remain in the Tuple modules and MUST NOT be introduced into `gen`.

Datatype binding and **Value Domain binding** are separate operations. Datatype binding selects the XBRL item type without changing the HMD semantic datatype. **Value Domain binding** selects a separately identified governed permitted value set and MUST NOT be inferred from an Attribute name, `local_name`, lexical form, or datatype alone.

Legacy, standard, and user-defined **Value Domains** MUST remain separately identifiable. XBRL GL 2003-compatible legacy **Value Domains** MUST remain available for backward compatibility and MUST NOT be silently replaced or translated to standard **Value Domains**. A user-defined Value Domain retains separate governance and identity even if some values match an external standard.

A standard **Value Domain** MAY reference an authoritative external code list. Its authority, vocabulary identifier, edition or version, subset, provenance, and rights status belong to the **Value Domain binding** and taxonomy realisation. A crosswalk does not replace either **Value Domain**, and any crosswalk translation is an explicit transformation.

Where an approved `value_domain` requires an extensible enumeration, Extensible Enumerations 1.0 SHALL be used so that one item fact reports one governed Value Domain member. The Taxonomy Generator deterministically generates one shared domain head, the permitted member concepts, the domain-member relationships, a deterministic Value Domain linkrole, and the applicable concept binding from the authoritative Value Domain, Value, overlay, and datatype inputs. Existing reusable or legacy enumeration content in `gen` remains separately identifiable and MUST NOT be silently replaced or translated.

For an EE1-bound fact concept, the generated XBRL declaration MUST use the Extensible Enumerations 1.0 enumeration item type, identify the generated domain head and Value Domain linkrole, and set the domain head as not directly usable (`enum:headUsable="false"`). The reported fact value is the QName of one permitted member. Extensible Enumerations 2.0 multiple-value semantics are not required for Framework Value Domain facts.

The formal HMD retains exactly the 18 columns in 4.2 with `value_domain` in column 9; no `domain_name` column is added. The approved LHM reviewed already contains `value_domain` in column 9, and `post_graphwalk` preserves it unchanged into the HMD for taxonomy. Value Domain binding MUST NOT be inferred from `name`, `local_name`, datatype, lexical content, or the former candidate-LHM `association_role`.

6 Palette taxonomy

The Tuple Palette separates reusable module-owned global declarations and presentation networks from **HMD**-specific structural type completion.

6.1 Tuple module schema

For each selected module, the generator MUST write one **Tuple module schema** at <module>/<module>-<V>.xsd. It MUST contain every reusable Attribute item declaration and every Class or Reference Tuple declaration owned by the module exactly once. Each declaration MUST have a stable XML id of the form <module>_<local_name>.

Every C or R declaration MUST reference the named <local_name>ComplexType in the same module namespace. The structural complexType itself MUST NOT be defined in the **Tuple module schema**. The module schema is therefore a reusable declaration palette and is intentionally incomplete for standalone structural type validation.

6.2 HMD content schemas

For every module selected into an **HMD** Tuple **DTS**, the generator MUST write one **content schema** at tuple/<HMD>/<module>-content-<V>.xsd. The **content schema** MUST have the same target namespace as the corresponding **Tuple module schema**, MUST use xs:include to incorporate that module schema, and MUST define every structural complexType needed to resolve the included C/R declarations for the selected **DTS**.

A **content schema** MUST use xs:import for each different module namespace required by its direct-child references. The complete set of **HMD content schemas** reached from the entry point MUST provide one consistent definition for every expanded type QName used in the **DTS**.

6.3 Tuple message entry point

Each **formal HMD** MUST have one filesystem-safe directory identifier <HMD> derived from its root module and root local_name. Let <P> denote the root module. The Tuple **message entry point** MUST be written at tuple/<HMD>/<P>-all-<V>.xsd.

The entry point MUST import the root-module **content schema**, MUST discover the applicable module label linkbases, and MUST discover the Tuple **module presentation linkbase** <P>/<P>-pre-<V>.xml corresponding to the **HMD root** module.

6.4 DTS assembly and validation boundary

A **DTS** is not one physical schema document. The Tuple **message entry point** establishes the discovery boundary from which the applicable **content schemas**, included **Tuple module schemas**, imported module dependencies, labels, and presentation linkbase are assembled.

```
Tuple message entry point
  -> root <module>-content-<V>.xsd
       -> xs:include <module>/<module>-<V>.xsd
       -> xs:import dependent <module>-content-<V>.xsd
  -> module labels
  -> root-module <module>-pre-<V>.xml
```

The intentionally incomplete **Tuple module schema** becomes valid in the assembled schema component set because the HMD **content schema** supplies the referenced structural complexTypes. XML Schema and XBRL conformance MUST therefore be assessed from the root-specific **message entry point**, not from the isolated **Tuple module schema**.

6.5 DTS constraints

Within one assembled HMD **DTS**, every global element QName, type QName, element reference, linkbase locator, and role reference MUST resolve exactly once. Tuple entry points MUST NOT discover OIM module schemas or OIM-specific p_, h_, or d_ declarations.

7 Labels and presentation networks

7.1 Labels

Each effective Tuple concept MUST have a standard label in the base language. A local-language label linkbase MUST be generated when local labels are supplied. Documentation labels MAY be generated from definition and definition_local. Tuple label locators MUST identify the concept declaration in its owning <module>-<V>.xsd by schema URI and XML id.

7.2 QName identity and locator resolution

An XBRL linkbase relationship addresses a taxonomy concept, not an HMD occurrence or semantic_path. A Tuple locator therefore resolves to one global xs:element declaration in the owning **Tuple module schema**. The declaration QName is the concept identity used by Tuple content models, labels, presentation relationships, and instance elements.

Within one **HMD DTS**, conflicting declarations or type definitions for the same expanded QName are prohibited. Across HMDs, a shared declaration is reusable only when the conditions in 4.6 are satisfied.

7.3 Module presentation linkbase

For each module M, the generator MUST write one Tuple *module presentation linkbase* at M/M-pre-<V>.xml. Every effective C or R concept owned by M is a presentation-tree starting candidate. From each starting candidate, the generator MUST represent the complete reachable parent-child hierarchy in HMD order. Traversal MUST continue when descendant concepts belong to other modules.

A **module presentation linkbase** MAY therefore contain multiple reusable presentation trees. The number of serialized trees is not required to equal the number of C/R concepts because a C/R concept that is already reachable within another tree need not cause the same descendant subtree to be serialized again. Distinct incoming parent-child relationships MUST nevertheless be preserved.

Every presentation locator MUST resolve to the global declaration in the concept-owning <module>-<V>.xsd. A cross-module locator therefore refers to another module schema by relative URI. A presentation locator MUST NOT target an HMD-specific <module>-content-<V>.xsd. Within one presentationLink, duplicate locators and duplicate presentationArc relationships MUST NOT be serialized.

7.4 Presentation Linkbase Set and message selection

The Tuple **Presentation Linkbase Set** comprises the **module presentation linkbases** published for the package. For an **HMD rooted** in module P, the **message entry point** discovers P/P-pre-<V>.xml. That root-module linkbase contains the complete presentation tree reachable from the **HMD root**, including cross-module descendants, and may also contain other reusable trees rooted in C/R concepts owned by P.

Other **module presentation linkbases** remain independently reusable for message **DTSs** whose roots are owned by those modules. The physical distribution of **module presentation linkbases** does not change the HMD hierarchy defined by the **formal HMD** and the Tuple content model.

7.5 Design rationale: module-level presentation (informative)

A complete presentation view could in principle be packaged separately for every HMD. XBRL GL Next instead publishes presentation trees as module components so that the same module-owned presentation structures can be reused by different message assemblies. The selected HMD still has a complete root view because traversal of the root-module tree can cross module boundaries.

Design aspect	HMD-specific packaging	Module-level Palette packaging
Physical unit	one presentation file per HMD	one presentation linkbase per module
Reuse	relationships may be repeated for each HMD	module-owned trees are reusable
Cross-module descendants	represented inside the HMD file	traversal continues through cross-module locators
Message selection	message file itself carries the view	entry point selects the root-module presentation member
Palette flexibility	assembly-oriented	module-oriented and reusable

7.6 Linkroles and extended link roles

A **linkrole** is the URI in xlink:role on an extended link. Relationships in different **extended link roles** belong to different networks even when they use the same arcrole and concepts. The arcrole identifies relationship semantics; for example, http://www.xbrl.org/2003/arcrole/parent-child identifies a presentation relationship.

Every custom presentation role URI MUST have one discoverable link:roleType declaration whose link:usedOn identifies link:presentationLink. A linkbase that uses the role MUST make the declaration discoverable through the **DTS**. Role URIs and roleType declarations MUST be deterministic within a package version.

8 accountingEntries baseline and businessTransactions extension example (informative)

8.1 accountingEntries HMD

The accountingEntries HMD is rooted at gl-cor:accountingEntries and has **HMD identifier** cor_accountingEntries. Its Tuple entry point is tuple/cor_accountingEntries/cor-all-<V>.xsd. The cor **content schema** completes the structural complexTypes required by the cor declarations used in this **DTS**, while imported **content schemas** complete the selected dependent modules.

The entry point discovers cor/cor-pre-<V>.xml. The presentation tree rooted at cor:accountingEntries continues through cor-, bus-, lnk-, taf-, or other module concepts by cross-module locators as required by the HMD.

8.2 businessTransactions HMD

The businessTransactions HMD is independently rooted at gl-btx:businessTransactions and has **HMD identifier** btx_businessTransactions. Its Tuple entry point is tuple/btx_businessTransactions/btx-all-<V>.xsd. The btx **content schema** and its imported content-schema dependencies provide the complete Tuple content model for that root.

The entry point discovers btx/btx-pre-<V>.xml. The btx root tree may reference cor, bus, lnk, taf, and other module declarations through locators to the owning **Tuple module schemas**. The addition of this independent HMD does not place businessTransactions beneath accountingEntries.

8.3 Required result

The **formal package** MUST preserve the independence of the accountingEntries and businessTransactions message **DTSs** while reusing consistent module declarations, labels, and module presentation components. The same **formal HMD** input set and namespace version MUST produce the same package independently of HMD input order.

For validation and comparison, one **HMD for taxonomy** defines one Tuple DTS. A separate root-specific HMD is validated as a separate DTS even where module QNames are reused. Cross-DTS comparison is required only for deliberately shared declarations governed as identical.

9 Formal package assembly

9.1 Formal package tree

```
taxonomy/
  gen/
    gl-gen-<V>.xsd
  <M>/
    <M>-<V>.xsd
    <M>-pre-<V>.xml
    <M>-oim-<V>.xsd
    <M>-oim-pre-<V>.xml
    label/
      <M>-lab-en-<V>.xml
      <M>-lab-<language>-<V>.xml
      <M>-oim-lab-en-<V>.xml
      <M>-oim-lab-<language>-<V>.xml
  tuple/
    <HMD>/
      <P>-all-<V>.xsd
      <M>-content-<V>.xsd
  oim/
    <HMD>/
      <P>-all-oim-<V>.xsd
      <P>-all-dim-<V>.xml
```

NOTE In this directory tree, terms enclosed in angle brackets are metavariables and are replaced as follows:

- <M> is the module identifier of a taxonomy module, for example cor, btx, bus, lnk, or taf.
- <HMD> is the deterministic HMD identifier formed from the root module identifier, an underscore, and the root local_name, for example cor_accountingEntries or btx_businessTransactions.
- <P> is the root-module identifier used as the filename prefix for the HMD entry point and its HMD-specific linkbases, for example cor for cor_accountingEntries and btx for btx_businessTransactions.
- <V> is the taxonomy version expressed as a date in YYYY-MM-DD form, for example 2026-12-31.
- <language> is the language identifier used for an additional-language label linkbase, for example ja.

The angle brackets are not part of the actual directory or file names.

The Tuple files governed by this Part are <module>-<V>.xsd, <module>-pre-<V>.xml, the Tuple label linkbases, tuple/<HMD>/<P>-all-<V>.xsd, and tuple/<HMD>/<module>-content-<V>.xsd. The OIM-specific module schemas, presentations, labels, and HMD artefacts are governed by Part 3.

9.2 Version and namespaces

The package MUST be an explicit date in YYYY-MM-DD form. The namespace for module M MUST be https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/<M>. The palette namespace used by binding-specific HMD entry points MUST be https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/plt. A lexical prefix is not part of QName identity.

9.3 Deterministic HMD identifier

For a root XPath terminal gl-M:L, the **HMD identifier** MUST be M_L and unique within the **HMD for taxonomy** input set. The current examples therefore use cor_accountingEntries and btx_businessTransactions.

9.4 All-or-nothing publication

A **formal package** is conformant only when every **formal HMD** is valid and all required shared and HMD-specific components form complete valid **DTSs**. The generated result MUST be deterministic and MUST NOT be published as a conforming **formal package** if any required component or reference is unresolved.

10 Validation and conformance

10.1 Conformant taxonomy generator

A conformant taxonomy generator MUST implement Clauses 4 to 9 and generate the complete formal package from the selected HMD-for-taxonomy input set. In the maintained reference implementation, this responsibility is solely `tools/taxonomy/xBRLGL_TaxonomyGenerator.py`, together with its required supporting module `tools/taxonomy/datatype_binding.py`. Every generated Tuple message DTS MUST satisfy the applicable XBRL 2.1 and XML Schema requirements.

10.2 Minimum test set

Conformance testing SHOULD cover at least the following conditions.

- one accountingEntries HMD and one businessTransactions HMD;
- the same HMD set supplied in different input orders;
- valid and conflicting shared A/C/R declarations and structural signatures;
- each supported multiplicity and 0..0 exclusion;
- same-namespace xs:include and cross-namespace xs:import assembly;
- successful validation from each root-specific Tuple entry point;
- C and R presentation-tree starting candidates;
- cross-module presentation traversal and locator resolution;
- no presentation locator to a **content schema**;
- no duplicate locator or duplicate presentationArc within a presentationLink;
- Tuple/OIM **DTS** separation; and

- representative Tuple instance validation for each current **HMD root**.

11 Tuple instance examples (informative)

11.1 Purpose

The example instances illustrate how a root-specific Tuple message **DTS** is used in practice. They are informative and do not add requirements to the **formal HMD** or taxonomy-generation contracts.

The current example package contains the following XML instances.

Example	Instance file	Tuple taxonomy entry point
Accounting Entries journal entry	ids/tuple/ accountingEntries-journalEntry.xml	taxonomy/tuple/ cor_accountingEntries/ cor-all-<V>.xsd
Business Transactions vendor invoice	ids/tuple/ businessTransactions-vendorInvoice.xml	taxonomy/tuple/ btx_businessTransactions/ btx-all-<V>.xsd

11.2 Common XBRL instance components

Each example is an xbrli:xbrl document containing a link:schemaRef to exactly one root-specific Tuple entry point, an entity/period context, applicable units, and one root Tuple occurrence. Attribute facts use contextRef and, where applicable, unitRef and decimals. Structural Classes and References are serialized as nested Tuple elements according to the HMD-specific xs:sequence.

```
<link:schemaRef xlink:type="simple"
  xlink:href="../../taxonomy/tuple/cor_accountingEntries/cor-all-<V>.xsd"/>
...
<cor:accountingEntries id="AE-001">
  <cor:entryHeader id="AE-HDR-001">
    <cor:entryDetail id="AE-DET-1"> ... </cor:entryDetail>
  </cor:entryHeader>
</cor:accountingEntries>
```

11.3 Accounting Entries example

The accountingEntries example records journal entry JE-2026-0811-001 for vendor invoice INV-2026-0811-001. It contains three entryDetail occurrences: a JPY 10,000 expense debit, a JPY 1,000 input-tax debit, and a JPY 11,000 accounts-payable credit. totalDebits and totalCredits are each JPY 11,000. The repeated entryDetail Tuple occurrences demonstrate HMD multiplicity and physical Tuple nesting.

```
<?xml version='1.0' encoding='UTF-8'?>
<xbrli:xbrl xmlns:xbrli="http://www.xbrl.org/2003/instance"
xmlns:link="http://www.xbrl.org/2003/linkbase"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:iso4217="http://www.xbrl.org/2003/iso4217"
xmlns:cor="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/cor"
xmlns:bus="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/bus"
xmlns:btx="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/btx"
xmlns:lnk="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/lnk"
xmlns:taf="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/taf"
xmlns:plt="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/plt">
  <link:schemaRef xlink:type="simple"
xlink:href="../../taxonomy/tuple/cor_accountingEntries/cor-all-2026-12-31.xsd"/>
  <xbrli:context id="c1">
```

```
      <xbrli:entity>
        <xbrli:identifier scheme="https://example.com/entity">EXAMPLE-BUYER-
001</xbrli:identifier>
      </xbrli:entity>
      <xbrli:period>
        <xbrli:instant>2026-08-11</xbrli:instant>
      </xbrli:period>
    </xbrli:context>
    <xbrli:unit id="JPY">
      <xbrli:measure>iso4217:JPY</xbrli:measure>
    </xbrli:unit>
    <xbrli:unit id="pure">
      <xbrli:measure>xbrli:pure</xbrli:measure>
    </xbrli:unit>
    <cor:accountingEntries id="AE-001">
      <cor:documentInformation id="AE-DOC-001">
        <cor:documentType contextRef="c1">journalEntry</cor:documentType>
        <cor:creationDate contextRef="c1">2026-08-11T00:00:00</cor:creationDate>
        <cor:creator contextRef="c1">XBRL GL Next sample</cor:creator>
        <cor:documentComment contextRef="c1">Vendor invoice recognition
sample</cor:documentComment>
        <cor:defaultCurrency contextRef="c1">iso4217:JPY</cor:defaultCurrency>
      </cor:documentInformation>
      <cor:entityParty id="AE-ENTITY-001">
        <cor:entityPartyName contextRef="c1">Example Buyer Co.</cor:entityPartyName>
        <cor:entityPartyId contextRef="c1">EXAMPLE-BUYER-001</cor:entityPartyId>
      </cor:entityParty>
      <cor:entryHeader id="AE-HDR-001">
        <cor:datePosted contextRef="c1">2026-08-11T00:00:00</cor:datePosted>
        <cor:headerEntryDate contextRef="c1">2026-08-
11T00:00:00</cor:headerEntryDate>
        <cor:sourceJournal contextRef="c1">AP</cor:sourceJournal>
        <cor:entryId contextRef="c1">JE-2026-0811-001</cor:entryId>
        <cor:entryDescription contextRef="c1">Record vendor invoice INV-2026-0811-
001</cor:entryDescription>
        <cor:totalDebits contextRef="c1" unitRef="JPY"
decimals="0">11000</cor:totalDebits>
        <cor:totalCredits contextRef="c1" unitRef="JPY"
decimals="0">11000</cor:totalCredits>
        <cor:entryDetail id="AE-DET-1">
          <cor:lineNumber contextRef="c1">1</cor:lineNumber>
          <cor:postingDate contextRef="c1">2026-08-11T00:00:00</cor:postingDate>
          <cor:detailDocumentType contextRef="c1">invoice</cor:detailDocumentType>
          <cor:detailDocument id="AE-DOCREF-1">
            <cor:detailDocumentNumber contextRef="c1">INV-2026-0811-
001</cor:detailDocumentNumber>
          </cor:detailDocument>
          <cor:documentDate contextRef="c1">2026-08-11T00:00:00</cor:documentDate>
          <cor:detailDescription contextRef="c1">Office supplies
expense</cor:detailDescription>
          <cor:monetaryAmount contextRef="c1" unitRef="JPY"
decimals="0">10000</cor:monetaryAmount>
          <cor:currency contextRef="c1">iso4217:JPY</cor:currency>
          <cor:debitCreditIndicator contextRef="c1">D</cor:debitCreditIndicator>
        </cor:entryDetail>
        <cor:entryDetail id="AE-DET-2">
          <cor:lineNumber contextRef="c1">2</cor:lineNumber>
          <cor:postingDate contextRef="c1">2026-08-11T00:00:00</cor:postingDate>
          <cor:detailDocumentType contextRef="c1">invoice</cor:detailDocumentType>
```

```
        <cor:detailDocument id="AE-DOCREF-2">
          <cor:detailDocumentNumber contextRef="c1">INV-2026-0811-
001</cor:detailDocumentNumber>
        </cor:detailDocument>
        <cor:documentDate contextRef="c1">2026-08-11T00:00:00</cor:documentDate>
        <cor:detailDescription contextRef="c1">Input consumption
tax</cor:detailDescription>
        <cor:monetaryAmount contextRef="c1" unitRef="JPY"
decimals="0">1000</cor:monetaryAmount>
        <cor:currency contextRef="c1">iso4217:JPY</cor:currency>
        <cor:debitCreditIndicator contextRef="c1">D</cor:debitCreditIndicator>
      </cor:entryDetail>
      <cor:entryDetail id="AE-DET-3">
        <cor:lineNumber contextRef="c1">3</cor:lineNumber>
        <cor:postingDate contextRef="c1">2026-08-11T00:00:00</cor:postingDate>
        <cor:detailDocumentType contextRef="c1">invoice</cor:detailDocumentType>
        <cor:detailDocument id="AE-DOCREF-3">
          <cor:detailDocumentNumber contextRef="c1">INV-2026-0811-
001</cor:detailDocumentNumber>
        </cor:detailDocument>
        <cor:documentDate contextRef="c1">2026-08-11T00:00:00</cor:documentDate>
        <cor:detailDescription contextRef="c1">Accounts
payable</cor:detailDescription>
        <cor:monetaryAmount contextRef="c1" unitRef="JPY"
decimals="0">11000</cor:monetaryAmount>
        <cor:currency contextRef="c1">iso4217:JPY</cor:currency>
        <cor:debitCreditIndicator contextRef="c1">C</cor:debitCreditIndicator>
      </cor:entryDetail>
    </cor:entryHeader>
  </cor:accountingEntries>
</xbrli:xbrl>
```

11.4 Business Transactions example

The businessTransactions example represents vendor invoice INV-2026-0811-001. Its transactionHeader contains a tax structure with JPY 1,000 tax on a JPY 10,000 basis, a header amount structure of JPY 11,000, and a transactionDetail with a JPY 10,000 line amount. The invoice number provides an informative linkage to the corresponding Accounting Entries example without requiring the two roots to share one Tuple **DTS**.

```
<?xml version='1.0' encoding='UTF-8'?>
<xbrli:xbrl xmlns:xbrli="http://www.xbrl.org/2003/instance"
xmlns:link="http://www.xbrl.org/2003/linkbase"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:iso4217="http://www.xbrl.org/2003/iso4217"
xmlns:cor="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/cor"
xmlns:bus="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/bus"
xmlns:btx="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/btx"
xmlns:lnk="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/lnk"
xmlns:taf="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/taf"
xmlns:plt="https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/plt">
  <link:schemaRef xlink:type="simple"
xlink:href="../../taxonomy/tuple/btx_businessTransactions/btx-all-2026-12-31.xsd"/>
  <xbrli:context id="c1">
    <xbrli:entity>
      <xbrli:identifier scheme="https://example.com/entity">EXAMPLE-BUYER-
001</xbrli:identifier>
    </xbrli:entity>
    <xbrli:period>
```

```
        <xbrli:instant>2026-08-11</xbrli:instant>
      </xbrli:period>
    </xbrli:context>
    <xbrli:unit id="JPY">
      <xbrli:measure>iso4217:JPY</xbrli:measure>
    </xbrli:unit>
    <xbrli:unit id="pure">
      <xbrli:measure>xbrli:pure</xbrli:measure>
    </xbrli:unit>
    <btx:businessTransactions id="BT-001">
      <cor:transactionDocumentInformation id="BT-DOC-001">
        <cor:transactionDocumentType
contextRef="c1">vendorInvoice</cor:transactionDocumentType>
        <cor:transactionCreationDate contextRef="c1">2026-08-
11T00:00:00</cor:transactionCreationDate>
        <cor:transactionCreator contextRef="c1">XBRL GL Next
sample</cor:transactionCreator>
        <cor:transactionDocumentComment contextRef="c1">Vendor invoice sample
corresponding to JE-2026-0811-001</cor:transactionDocumentComment>
        <cor:transactionDefaultCurrency
contextRef="c1">iso4217:JPY</cor:transactionDefaultCurrency>
      </cor:transactionDocumentInformation>
      <cor:entityParty id="BT-ENTITY-001">
        <cor:entityPartyName contextRef="c1">Example Buyer Co.</cor:entityPartyName>
        <cor:entityPartyId contextRef="c1">EXAMPLE-BUYER-001</cor:entityPartyId>
      </cor:entityParty>
      <btx:transactionHeader id="BT-HDR-001">
        <btx:datePosted contextRef="c1">2026-08-11T00:00:00</btx:datePosted>
        <btx:headerDocumentType contextRef="c1">invoice</btx:headerDocumentType>
        <btx:headerInvoiceType contextRef="c1">vendorInvoice</btx:headerInvoiceType>
        <btx:headerDocumentNumber contextRef="c1">INV-2026-0811-
001</btx:headerDocumentNumber>
        <btx:headerMaturityDateOrDateDue contextRef="c1">2026-09-
10T00:00:00</btx:headerMaturityDateOrDateDue>
        <btx:headerPaymentTerms contextRef="c1">Net 30 days</btx:headerPaymentTerms>
        <btx:headerTax id="BT-TAX-001">
          <btx:taxAuthority contextRef="c1">JP</btx:taxAuthority>
          <btx:amountOfTaxes contextRef="c1" unitRef="JPY"
decimals="0">1000</btx:amountOfTaxes>
          <btx:basisForTaxation contextRef="c1" unitRef="JPY"
decimals="0">10000</btx:basisForTaxation>
          <btx:taxPercentageRate contextRef="c1" unitRef="pure"
decimals="2">0.10</btx:taxPercentageRate>
          <btx:taxCategory contextRef="c1">standard</btx:taxCategory>
        </btx:headerTax>
        <btx:headerAmountStructure id="BT-HAMT-001">
          <btx:headerAmount contextRef="c1" unitRef="JPY"
decimals="0">11000</btx:headerAmount>
          <btx:headerAurrencyCode contextRef="c1">JPY</btx:headerAurrencyCode>
          <btx:headerAmountType contextRef="c1">invoiceTotal</btx:headerAmountType>
        </btx:headerAmountStructure>
        <btx:transactionDetail id="BT-DET-001">
          <btx:lineNumber contextRef="c1">1</btx:lineNumber>
          <btx:documentType contextRef="c1">invoice</btx:documentType>
          <cor:document id="BT-DOCREF-001">
            <cor:documentNumber contextRef="c1">INV-2026-0811-
001</cor:documentNumber>
          </cor:document>
          <btx:documentDate contextRef="c1">2026-08-11T00:00:00</btx:documentDate>
```

```
          <btx:detailDescription contextRef="c1">Office
  supplies</btx:detailDescription>
          <btx:detailAmountStructure id="BT-DAMT-001">
            <btx:amount contextRef="c1" unitRef="JPY" decimals="0">10000</btx:amount>
            <btx:currencyCode contextRef="c1">JPY</btx:currencyCode>
            <btx:detailAmountType
  contextRef="c1">lineNetAmount</btx:detailAmountType>
          </btx:detailAmountStructure>
        </btx:transactionDetail>
      </btx:transactionHeader>
    </btx:businessTransactions>
  </xbrli:xbrl>
```

11.5 Relationship to presentation

The XML Tuple hierarchy is controlled by the HMD-specific **content schemas** and their xs:sequence definitions. The **Presentation Linkbase Set** provides a navigational presentation view of the same concepts but does not determine the XML instance nesting or multiplicity. A conforming processor validates the instance against the **DTS** reached from schemaRef.

Appendix A
(informative)
Designer guidance for the Palette taxonomy

A.1 Validation boundary

A **Tuple module schema** owns reusable A/C/R declarations but is intentionally incomplete for structural type resolution. The root-specific **content schemas** complete the named structural complexTypes. XML Schema and XBRL validity are therefore assessed from tuple/<HMD>/<P>-all-<V>.xsd and its discovered **DTS**.

A.2 Palette assembly overview

```
Tuple Palette module M
  M-<V>.xsd              global A/C/R declarations
  M-pre-<V>.xml          reusable presentation forest
  label/...              reusable labels

HMD-specific assembly
  tuple/<HMD>/M-content-<V>.xsd
      xs:include ../../M/M-<V>.xsd
      structural ComplexTypes
  tuple/<HMD>/<P>-all-<V>.xsd
      imports root content schema
      discovers labels and P-pre-<V>.xml
```

A.3 Presentation forest

A **module presentation linkbase** can contain several trees rooted at C/R concepts owned by the module. A tree can continue through concepts owned by other modules. The cross-module relationship is represented by a locator to the owning module schema; it does not transfer concept ownership.

A.4 Designer cautions

- Do not move HMD-specific structural complexTypes into the **Tuple module schema** merely to make the module schema validate in isolation.
- Do not replace a same-namespace xs:include with xs:import.
- Do not point a presentation locator at an HMD **content schema**.
- Do not infer the Tuple XML hierarchy from presentationArc relationships; xs:sequence is authoritative for instance nesting.
- Do not load OIM module schemas into a Tuple **DTS**.

A.5 Suggested validation procedure

1. Open tuple/<HMD>/<P>-all-<V>.xsd as the XML Schema and XBRL **DTS** entry point.
2. Follow xs:import to the root **content schema** and its transitive content-schema dependencies.
3. Follow same-namespace xs:include to each **Tuple module schema**.
4. Confirm that every global element type QName and xs:element ref resolves exactly once.
5. Confirm that every presentation locator resolves to a concept in the owning **Tuple module schema** and that duplicate locators/arcs are absent.
6. Validate a representative instance through schemaRef to the same entry point.

Appendix B
References

(Reference list under collaborative review. Official XBRL specifications are available at https://specifications.xbrl.org/.)

RFC2119	The Internet Engineering Task Force (IETF). "Key words for use in RFCs to Indicate Requirement Levels, March 1997" Scott Bradner. (See http://www.ietf.org/rfc/rfc2119.txt)
SCHEMA-1	World Wide Web Consortium (W3C). "XML Schema Part 1: Structures" (See http://www.w3.org/TR/xmlschema-1/)
XPath	World Wide Web Consortium (W3C), XML Path Language (XPath) 2.0 (Second Edition), W3C Recommendation, 14 December 2010. (See https://www.w3.org/TR/xpath20/)
TRP	XBRL International Inc. (XII). "XBRL Taxonomy Recognition Process, 28 October 2004" Peter Calvert, and Joseph MacDonald. (See https://xbrl.org/taxonomyrecognition/taxonomy-recognition-acknowledgement-process-2013-04-09.htm)
XBRL	XBRL International Inc. (XII). "Extensible Business Reporting Language (XBRL) 2.1 Specification with corrected errata dated 2 July 2008" Phillip Engel, Walter Hamscher, Geoff Shuetrim, David vun Kannon, and Hugh Wallis. (See https://specifications.xbrl.org/)
XLINK	World Wide Web Consortium (W3C). "XML Linking Language (XLink) Version 1.0." Steve DeRose, Eve Maler, and David Orchard. (See http://www.w3.org/TR/xlink/)
GXLTFTA	XBRL International Inc. (XII). “XBRL GL Taxonomy Framework Technical Architecture 2015” Gianluca Garbellotto, Eric E. Cohen, Masatomo Goto, Toshimitsu Suzuki (See https://www.xbrl.org/int/gl/2015-03-25/GLTFTA-REC-2015-03-25.html)
XBRL GL Framework	XBRL International Inc. (XII). “XBRL Global Ledger Taxonomy Framework 2015“ Eric E. Cohen, Masatomo Goto, Toshimitsu Suzuki (See https://www.xbrl.org/int/gl/2015-03-25/gl-framework-REC-2015-03-25.html)

Appendix C
Intellectual property status

Copyright and license terms are under review. This draft is made available through the public project repository for collaborative review. Publication of this draft does not establish additional redistribution or derivative-use rights.

References to external specifications and other third-party materials remain subject to the terms of their respective rights holders.

This draft does not claim approval, endorsement, publication, or official status by XBRL Japan, XBRL Europe, XBRL International, or any other organization.

Appendix D
Document history

2026-09-15 accepted alignment: This section records the current accepted project specification. It is an internal project decision and does not state or imply recognition, approval, or publication by XBRL International, XBRL Japan, or another external organisation.

Topic	Accepted specification and boundary
Project module namespace	The namespace of module M is exactly https://www.xbrl.or.jp/taxonomy/xbrl-gl-next/M. The URI contains neither an experimental segment nor a date. Namespace prefixes, versioned physical filenames, and directory paths are separate identifiers and do not alter the namespace URI. External XML, XBRL, XLink, ISO, and historical 2015 XBRL GL namespaces remain unchanged.
Model authority and pipeline	The authoritative authored source is the FSM. Specialisation deterministically produces the BSM. Graph Walk produces the candidate LHM. Human review controls the reviewed LHM. Post-Graph Walk produces the root-specific HMDs. The HMD semantic_path, hierarchy, datatype, multiplicity, and value_domain govern Tuple and OIM generation.
Root-specific DTSs	Accounting Entries and Business Transactions are expanded from different root HMDs. Each root is used and validated through its matching Tuple or OIM entry point. The split layout is mandatory; same-named files are not copied over one another and the two DTSs are not mixed without a separate integration design.
Accepted structural differences	Fourteen tracked differences among same-module and same-name declarations are accepted as ROOT_HMD_SPECIFIC_STRUCTURAL_DIFFERENCE. Exact equality of root-derived type, structure, or multiplicity is not required across AE and BT. Each declaration remains subject to its own accepted root HMD and DTS.
Tuple binding	A Class hierarchy in one selected root HMD is realised as Tuple elements, complex types, sequence, and occurrence constraints. The message entry point selects the required module schemas and root-specific content schema.
Cross-root boundary	Tuple declarations are assessed against the selected root HMD. Root-derived structural equality with the other HMD is not a conformance requirement. A consumer selects either the Accounting Entries or Business Transactions Tuple entry point.

12 August 2026 v14: Defined the initial-edition Tuple module Palette, **DTS** assembly boundary, **module Presentation Linkbase Set**, C/R-rooted reusable presentation forests, binding-specific Tuple/OIM presentation separation, final package naming, and informative Accounting Entries and Business Transactions Tuple instance examples.

20 August 2026 v15: Clarified semantic datatype versus XBRL item type binding; added the shared **`gen` taxonomy module** and **Value Domain** realisation; preserved distinct legacy, standard, and user-defined **Value Domains**; and stated the transitional implementation boundary.

21 August 2026 v16: Preserved all singular and repeated Classes in the Tuple content model while distinguishing occurrence-key classification as a Structured CSV/OIM concern; added parent-scoped nested-repeat guidance; and made the pending authoritative **Value Domain**-to-EE 1.0 generation boundary explicit.

Date	Details
8 August 2026	v3: Added normative explanations of module-schema linkbase discovery and **concept locators**; clarified xs:include and xs:import assembly, HMD-specific content complex types, palette extensibility, QName uniqueness, businessTransactions labels and presentation extension, and **linkrole**/extended-link-role use.
8 August 2026	v2: Rewritten to define deterministic generation from formal 18-column HMDs; added accountingEntries baseline, businessTransactions additional-HMD example, HMD-directed package assembly, shared-QName rules, presentation differences, and all-or-nothing publication.
3 August 2026	Previous collaborative review draft.
8 August 2026	v4: Clarified formal **DTS** discovery, module-schema locators, same-namespace include and cross-namespace import, the logical placement of structural complex types, shared label reuse, businessTransactions presentation extension, and the distinct functions of roleType, roleRef, **linkrole**, and arcrole.
8 August 2026	v5: Reused and updated the supplied taxonomy figures to explain HMD-specific entry points, same-namespace inclusion, cross-module imports, QName-based Tuple content models, and the businessTransactions extension without restoring Profile-dependent assembly.
8 August 2026	v6: Added the document date and a focused Figure explaining how cor.xsd owns reusable concept identity, cor-content.xsd defines the HMD-specific Tuple content model through xs:include and QName references, and cor-pre.xml defines presentation relationships through locators to cor.xsd.
9 August 2026	v7: Revised Figure 1 and Clause 6 to explain the palette principle using independent HMD entry points, HMD-specific content-model assembly, reusable module concept schemas, cross-module imports, linkbaseRef discovery, and URI#id **concept locators**, without restoring the former Profile or plt-root mechanism.
9 August 2026	v8: Restored the defining Palette taxonomy separation: module schemas contain stable global element declarations but no structural complexTypes and are intentionally incomplete alone; HMD-specific **content schemas** define effective complexTypes and complete the schema through xs:include. Revised Figures 1 and 2 in grayscale, clarified linkbase extensibility, per-**DTS** QName uniqueness, and permitted independent HMD content-model variation.
9 August 2026	v9: Reorganised Clause 6 for XML Schema practitioners. Distinguished a physical schema document, the assembled XML Schema component set, and the XBRL **DTS**; explained why palette module schemas are intentionally incomplete in isolation, why same-namespace xs:include is required, where per-**DTS** QName uniqueness applies, how independent HMD content models vary, and how to validate the package without mistaking the Palette taxonomy design for a schema-generation defect.
9 August 2026	v10: Replaced the generic output tree with the agreed taxonomy directory layout. Restored gen/gen-<V>.xsd; version-before-purpose module linkbase names; lang label directories; **HMD root**-module directories under plt and oim; plt-all, plt-oim, plt-def and plt-pre filenames; and aligned Clauses 6–9, Figures 1 and 2, examples, and the normative output-name summary.
9 August 2026	v11: Adopted the generic plt/<HMD> and oim/<HMD> directory convention, with cor and btx as the current filesystem-safe **HMD identifiers**. Rewrote Clause 6 as MUST/MUST NOT/MAY requirements only; moved XML Schema explanations, Figures 1 and 2, common misinterpretations, and the suggested validation procedure to informative Annex B; and generalised Clause 9.3 and Annex A to the adopted output names.
10 August 2026	v12: Generalised the **conformant taxonomy generator** as an implementation-independent processor of one or more **formal HMD**s; removed requirements for commands, staging, diagnostics, execution records, checksums, exit status, and ancillary output; replaced redundant prohibitions with concise result-oriented requirements; retained the defining rule that palette module schemas contain no HMD-specific structural complexType definitions; and aligned Figure 1 with the generalised package model.
11 August 2026	v13: Aligned Part 2 with the accepted taxonomy package: common LHM-for-taxonomy input, reusable module schemas containing Attribute items only, HMD-specific Tuple

Date	Details
	declarations and structural complexTypes under tuple/<HMD>, binding-specific HMD presentation extensions, and the accepted gl-gen, label, entry-point, and content-schema filenames. Clarified that only Attribute item declarations are shared across HMDs, while Class/Reference Tuple declarations and structural complexTypes are HMD-specific and may differ across independent **DTSs**.
28 August 2026	Phase 2C: aligned the HMD header with `value_domain`, identified the sole self-contained Taxonomy Generator authority, classified datatype/selector helpers as optional tools, and stated one-HMD/one-Tuple-DTS validation scope.
29 August 2026	Aligned Tuple Palette generation with the implemented EE1 and reviewed-Value-Domain pipeline; removed obsolete NOT-YET-IMPLEMENTED and companion-binding wording; restored module-owned global C/R declarations with HMD-specific structural complexTypes; and limited cross-DTS equality to deliberately shared declarations.
15 September 2026	Restored the accepted 2026-08-29 successor as the base; aligned project module namespaces with the stable URI; recorded root-HMD-specific AE/BT split conformance and the accepted 14 structural differences; aligned the UADC interface boundary and current semantic items where applicable.
6 November 2026	Revised the correspondence with the XBRL GL Next Requirements Specification to reflect the updated requirements.

Appendix E
Errata corrections in this document

This appendix records editorial corrections incorporated into this collaborative review draft. The entries have not been approved or endorsed by XBRL Japan, XBRL Europe, XBRL International, or any other organization.

No errata have been incorporated into this document.

Appendix F
(informative)
Correspondence with XBRL GL Next Requirements

Table F.1 maps the applicable requirements of the XBRL GL Next Requirements Specification to their implementation in this Part. The requirement identifiers and names are reproduced from that Specification. This correspondence is informative and does not create additional requirements or assert implementation conformance.

Table F.1 — Mapping of XBRL GL Next Requirements

Requirement ID	Requirement Name	Implementation in Part 2
GL-REQ-0001	Consistent business semantics across representations	Uses the accepted root-specific 18-column HMD as the binding input and retains its semantic_path, source_bsm_id, hierarchy and reviewed values without making XML Tuple structure the semantic authority (Clauses 4–5).
GL-REQ-0107	Business completeness and data quality	Defines HMD row checks, QName and datatype resolution, exclusion of non-use subtrees, assembled-DTS integrity and representative Tuple instance validation (Clauses 4, 6 and 10).
GL-REQ-0108	Business transaction coverage and linkage	Defines independent Tuple entry points for accountingEntries and businessTransactions and demonstrates their separate root-specific content models and DTS boundaries (Clauses 6.3 and 8).
GL-REQ-0201	Reusable business scope	Builds reusable module-owned Tuple concept declarations, labels and presentation linkbases, then assembles them with HMD-specific structural content schemas and message entry points (Clauses 6, 7 and 9).
GL-REQ-0201	Reusable business scope	Derives a separate message entry point and content-schema assembly for each selected HMD root, so the included module combinations follow that root without changing another root (Clauses 6.3 and 8).
GL-REQ-0301	Exchange-format independence	Defines the XBRL 2.1/XML Tuple realisation of the common HMD input alongside, but separate from, the OIM realisation specified by Part 3 (Clauses 2.1, 5 and 9).
GL-REQ-0303	Identity and relationships of repeated business information	Preserves the source HMD semantic paths, concept QNames, child order and occurrence multiplicities through deterministic Tuple declarations, xs:sequence content models and root-specific validation (Clauses 4–5 and 10).
GL-REQ-0401	Accessible examples and independent inspection	Supplies representative Accounting Entries and Business Transactions Tuple instance examples and identifies the validation scope and checks for generated DTSs (Clauses 10–11).
GL-REQ-0402	Demonstrable business interoperability	Defines testable evidence for deterministic generation, root-specific DTS validity, namespace resolution, presentation linkages, multiplicities and representative instance conformance (Clause 10).
GL-REQ-0501	Business code values and enumerations	Binds approved enumerated Value Domains to governed EE1 domain heads, members, relationships and fact concepts, preserving one scalar member per reported fact (Clause 5.5).

GL-REQ-0502	Reference to externally governed codes	Keeps legacy, standard and user-defined Value Domains separately identifiable, records external authority and rights, and treats any crosswalk translation as an explicit operation (Clause 5.5).