OIM Taxonomy

A Modern Semantic Framework for XBRL

Format-agnostic · Object-identified · Semantically rich · Reusable

Agenda

1. **Overall Model Types**
   Report vs Taxonomy, imports, combined models
2. **Unified Object Definitions**
   Registries now defined as standard taxonomies
3. **Simpler Semantic Objects**
   First-class concepts, cubes, networks, groups
4. **Object Identification**
   Every object named via QName / SQName
5. **Enhanced Reuse**
   Selective import — pick exactly what you need
6. **Property Mechanism**
   Declarative properties with inheritance
7. **Format Agnostic**
   Semantic model, currently serialised as JSON
8. **Source Document Linking**
   Link facts to HTML, PDF via interface types
9. **Generic Sources**
   QName references only — data source agnostic
10. **Facts in Cubes**
    Dimension constraints, duplicates, restrictions
11. **Implied Objects**
    LEI, ISIN and other open-world registries

Section 1

Overall Model Types

Report and Taxonomy models — and how they import each other

Two Model Types

xbrl:report and xbrl:taxonomy

xbrl:taxonomy

- Defines reusable concepts, dimensions,
- networks, cubes, properties, and labels
- Can be imported by reports or other
- taxonomies (selective or full import)
- Has modelForm: module or compiled
- Cannot contain facts
- Examples: US-GAAP, IFRS, DEI, UTR

⇄

imports

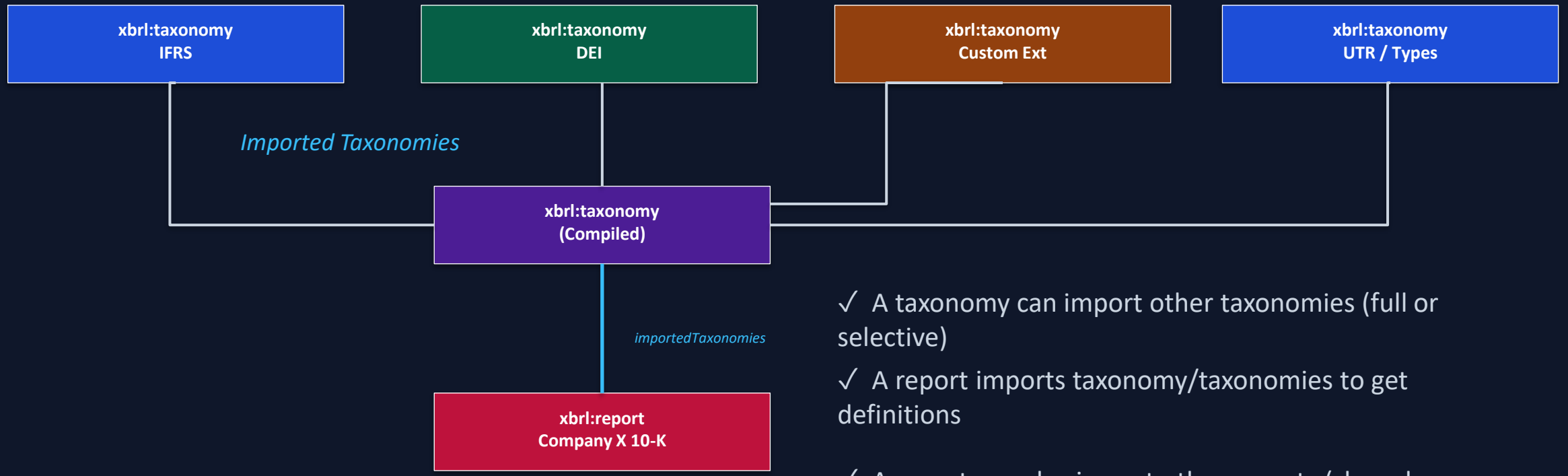
xbrl:report

- Contains the actual fact data for an entity
- Imports one or more taxonomies to get
- concept definitions, labels, structures
- Can also import other reports
- Can define report-specific extensions
- Has modelForm: module or compiled
- Example: Apple 10-K filing (OIM JSON)

xbrl:taxonomy ⇄ **xbrl:report** ⇄ **xbrl:report** ⇄ **xbrl:taxonomy**

Import Architecture

How models reference each other



✓ A taxonomy can import other taxonomies (full or selective)

✓ A report imports taxonomy/taxonomies to get definitions

✓ A report can also import other reports (shared facts/dimensions)

✓ Both forms compile to a single closed, self-contained model

Module vs Compiled

Two forms of an XBRL model

Module Form (modelForm: "module")

- Has documentNamespace property
- Objects must use documentNamespace prefix
- May contain imported Taxonomies references
- May use extendTargetName to augment objects
- Validation requires resolving all imports first
- The authoring format — created by taxonomists

```
{ "modelForm": "module", "documentNamespace": "us-gaap",
  "importedTaxonomies": [{ "xbrlModelName": "dei:DeiModule" }] }
```

Compiled Form (modelForm: "compiled")

- No importedTaxonomies — fully self-contained
- No extendTargetName — all merges resolved
- No documentNamespace — multi-namespace
- All objects from all imports are present
- Can be validated in isolation
- The distribution/consumption format/ archive format

```
{ "modelForm": "compiled" }
  // all objects included inline – no import refs
```

Section 2

Unified Object Definitions

Registries defined as standard xbrl:taxonomy JSON modules

The Existing Approach: Separate Registries

Units, datatypes, transforms — maintained as separate XML artefacts

⚠ **Separate file types**	XBRL 2.1 uses XSD schemas for datatypes and linkbases for linkroles. UTR is a separate XML format entirely.
⚠ **No common toolchain**	Each registry needed bespoke parsers, validators, and tooling — inconsistent across implementations.
⚠ **Hard to extend**	Custom units or transform types required vendor-specific workarounds or couldn't be expressed at all.
⚠ **No semantic links**	Registries could not reference XBRL concepts, dimensions, or other objects — just raw XML.
⚠ **Version fragmentation**	Multiple versions of ISO 4217, UTR, and transforms maintained in separate repositories with no standard import mechanism.

Result: Multiple bespoke tools, inconsistent semantics, difficult maintenance

Registries as Standard Taxonomies

All registries are now xbrl:taxonomy JSON modules — uniform toolchain

iso4217.json

Currency Units

ISO 4217 currencies as xbrl:unit objects
Each unit has a dataType, labels, and references

utr.json

Unit Registry

XBRL UTR physical units as xbrl:unit objects
Grouped by measure type (mass, length, energy...)

transform-types.json

Transforms

iXBRL transform type objects as xbrl:transformType
Each transform maps a source format to a canonical value

types.json

Datatypes

xs:* and xbrl:* data types with constraints
Can be extended with custom types in any taxonomy

ref.json

Reference Parts

Standard reference part definitions
Section, paragraph, footnote etc. as xbrl:referencePart

xbrla.json

Accounting Model

Balance type, period type metadata
Shared accounting properties across US-GAAP/IFRS

✓ **All use the same JSON format, the same import mechanism, and the same toolchain as any other taxonomy**

Registry as Taxonomy — Example

iso4217.json and utr.json define currency and unit objects

Currency Unit (iso4217.json)

```
"units ”: [
          {
            "name": "iso4217:USD",
            "dataType": "xbrl:monetary"
          }
      ],
"labels": [
         { "relatedName": "iso4217:USD",
          "labelType": "xbrl:label",
          "language": "en",
          "value": "US Dollar" }
  ]
```

Transform Type (transform-types.json)

```
{
  "name": "xbrltt:date-day-monthname-en",
  "inputDataType": "xs:string",
  "outputDataType ": "xs:date",
}
```

Key Advantages

✓ Import currency units selectively — only the ones you need

✓ Add labels and references to unit objects — same as concepts

✓ Custom units or transform types use exactly the same syntax

✓ Versioning handled through standard taxonomy import mechanism

✓ Validators, browsers, and tools all work without modification

✓ Registries can import each other (e.g. utr imports xs-types)

Section 3

Simpler Semantic Objects

First-class named objects — no assembly from arcroles

First-Class Semantic Objects

Every structural element is an independently named, referenceable object

Concept

xbrl:conceptObject
Reportable data item with dataType and periodType

Abstract

xbrl:abstractObject
Grouping node (not directly reported)

Member

xbrl:memberObject
Enumeration value in a domain

Cube

xbrl:cubeObject
Multidimensional space containing facts

Dimension

xbrl:dimensionObject
Dimension along which facts vary

Domain

xbrl:domainObject
Set of valid members for a dimension

Network

xbrl:networkObject
Hierarchical presentation structure

Group

xbrl:groupObject
Named set of concept relationships

Label

xbrl:labelObject
Human-readable name for any object

Reference

xbrl:referenceObject
Citation linking objects to source documents

Unit

xbrl:unitObject
Measurement unit (currency, physical unit)

Fact

xbrl:factObject
A single reported value at a dimensional point

Semantic Objects vs XBRL 2.1

From XML assembly constructs to independently addressable objects

XBRL 2.1 (XML Linkbases)

- ✗ Presentation networks assembled from arcroles in linkbases
- ✗ No direct name for a network — identified only by linkrole URI
- ✗ Cubes assembled from hypercubeItem + dimension arcroles
- ✗ Domains are implicit through dimensional constraints
- ✗ Label linkbase connects strings to concepts via arcroles
- ✗ Everything requires parsing multiple XML files to reconstruct

OIM Taxonomy (Semantic Objects)

- ✓ Networks are named objects (exp:IncomeStatementNetwork)
- ✓ Reference networks by QName in relationships and imports
- ✓ Cubes are referenceable objects with explicit dimension lists
- ✓ Domains have names — can add labels, references, properties
- ✓ Labels are objects with relatedName, labelType, language
- ✓ Single JSON file per module — readable by any JSON parser

Section 4

Object Identification

Every object uniquely named by QName or SQName

Object Identification via QName

Every object has a globally unique name — the foundation of the semantic model

QName anatomy: prefix : localName → namespace URI + localName

ifrs : Revenue → http://iasb.org/ifrs/2025 # Revenue

prefix (document-scoped shorthand)

namespace URI + local name = global identity

Concepts	→ exp:Revenue has dataType, periodType, labels, references, properties
Facts	→ Fact's concept dimension is a QName: xbrl:concept = us-gaap:Revenue
Relationships	→ Network nodes reference concepts by QName in a tree structure
Labels	→ Label's relatedName = QName of any object (concept, unit, dimension…)
References	→ Reference's relatedName = QName of any object in the taxonomy
Properties	→ Properties can reference objects by QName value for type safety

SQName & QName

Compact identifier for objects when the namespace is known from context

SQName = prefix:localName resolved within the document's namespaces map

us-gaap:Revenue	*http://iasb.org/ifrs/2025*	**Concept**
dei:EntityName	*http://xbrl.sec.gov/dei/2025*	**Concept**
iso4217:USD	*http://www.xbrl.org/2003/iso4217*	**Unit**
lei:54930084UKLVMY	*http://standards.iso.org/iso/17442*	**Entity**
exp:SalesNetwork	*http://example.com/taxonomy*	**Network**
xbrl:factObject	*https://xbrl.org/2025*	**Object Type**

Prefixes are document-scoped shortcuts only — the namespace URI is the true identity

§4 Object Identification

Section 5

Enhanced Reuse

Selective import — choose precisely which objects to include

Selective Import

Pick exactly the objects you need — not the entire taxonomy

importedTaxonomies → optional selections array

```
"importedTaxonomies": [
  {
    "xbrlModelName": "mini2:MiniTaxonomy2",
    "selections": [
      {
        "objectType": "xbrl:conceptObject",
        "where": [
          {
            "property": "periodType",
            "operator": "==",
            "value": "instant"
          }
        ]
      },
      {
        "objectType": "xbrl:labelObject",
        "where": [
          {
            "property": "relatedName",
            "operator": "==",
            "value": "mini2:Inventory"
          }
        ]
      }
    ],
    "excludeLabels": true
  }
]
```

✓ Smaller compiled models

Only include the concepts, networks, or labels you actually use — compiled model stays lean

✓ Precise control

Filter by any property (periodType, dataType, labelType, relatedName) using comparison operators

✓ Exclude labels

Import concepts without their labels — useful when you provide your own translated labels

✓ Mix and match

One taxonomy import can be selective; another can be full — mixed selections in one model

✓ Sealed taxonomies

finalTaxonomy + finalObjects can prevent others from selectively importing certain objects

Extending Imported Objects

extendTargetName — add relationships to objects you don't own

Extension example: adding members to an existing domain

```
// In importing taxonomy:
{
  "extendTargetName": "us-gaap:StatementGeographicAreaDomain",
  "relationships": [
    { "source": "us-gaap:StatementGeographicAreaDomain",
      "target": "exp:NorthernEurope" },
    { "source": "us-gaap:StatementGeographicAreaDomain",
      "target": "exp:SouthernEurope" }
  ]
}
```

finalTaxonomy — sealing objects against further extension

```
{
  "finalTaxonomy": {
    "finalObjectTypes": ["xbrl:conceptObject"],
    "finalObjects": ["us-gaap:StatementGeographicAreaDomain"]
  }
}
```

- Objects with isExtensible: false reject extension (oimte:cannotExtendObject)
- Multiple extension objects can target the same object — all merged
- Compiled form contains no extendTargetName — all merges resolved
- finalTaxonomy prevents further augmentation from any importer

Section 6

Property Mechanism

Typed, validated properties with inheritance

Property Mechanism

Attach typed, validated metadata to any object via named properties

```
"propertyTypes": [
  {
    "name": "xbrla:balance",
    "dataType": "xs:string",
    "definitional": true,
    "allowedObjects": [
      "xbrl:conceptObject"
    ]
  }
]
```

```
"concepts": [
  {
    "name": "exp:Revenue",
    "dataType": "xbrlr:monetary",
    "periodType": "duration",
    "properties": [{ "property": "xbrla:balance",
        "value": "credit" }]
  }
]
```

§6 Properties

Typed values

Properties have a declared dataType — xs:string, xs:boolean, xs:QName, etc.

Validated

Processor validates property values against declared types and allowedObjects

Definitional

definitional: true marks properties that affect the identity/semantics of an object

Inheritance

Properties can be inherited from a parent network/cube to child relationships

Custom properties

Any taxonomy can declare new propertyTypes — fully extensible

Link properties

allowedAsLinkProperty: true lets a property appear on relationship objects

preferredLabel

xbrl:preferredLabel on a relationship specifies the label type for display

Property Inheritance

Properties cascade from network/cube to relationships to nodes

A preferred label set on a network applies to all its nodes unless overridden on a relationship

```
"networks": [
  {
    "name": "exp:IncomeStatementNetwork",
    "properties": [
      // Network-level: preferredLabel for all nodes
      { "property": "xbrl:preferredLabel",
        "value": "xbrl:label" }
    ],
    "relationships": [
      { "fromName": "exp:StatementOfIncome",
        "toName":   "exp:Revenue" },
      { "fromName": "exp:StatementOfIncome",
        "toName":   "exp:CostOfGoodsSold",
        // Relationship-level override
        "properties": [
          { "property": "xbrl:preferredLabel",
            "value": "xbrl:negatedLabel" }
        ] }
    ]
  }
]
```

Inheritance chain:

networkObject property
Sets default for entire network

cubeDimension property
Sets default for a cube dimension

relationship property
Overrides default for that specific arc

No property set
Falls back to xbrl:label (the default)

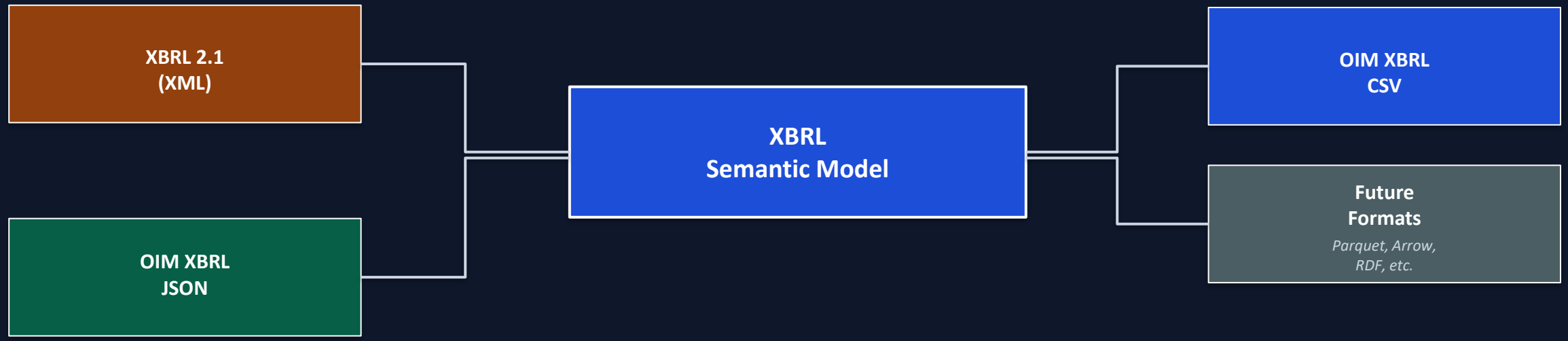
Section 7

Format Agnostic

Semantic model defined independently of serialisation format

Format Agnostic Semantic Model

The model is defined semantically — JSON is the current serialisation



- One semantic spec governs all serialisation formats
- Implementations choose representation; model stays the same
- JSON Schema validation for structural correctness
- Current spec defines JSON as the normative serialisation
- XBRL 2.1 XML converters produce OIM JSON
- CSV and other formats mapped through OIM Common specification

Section 8

Source Document Linking

Facts linked to HTML, PDF, and other source documents

Source Document Linking

Facts carry references back to the source document elements that contain them

The factInterfaceName property on a sourceMapping declares how to locate facts in the document

```
"sourceMappings": [
  {
    "sourceName": "aapl:aapl-20250927Source",
    "url": "../examples/aapl-20250927.htm",
    "factInterfaceName":
       "xbrl:htmlElementLocatorType"
  }
]
```

xbrl:htmlElementLocatorType	**HTML**	Locates facts by HTML element span ID (ix:id attribute)
xbrl:pdfFormFieldLocatorType	**PDF**	Locates facts by PDF form field name
xbrl:pdfMcidLocatorType	**PDF (MCID)**	Locates facts by Marked Content ID in tagged PDF
xbrl:tabularDataLocatorType	**CSV**	Locates facts by row/column path in tabular data

Source interfaces are open — new document types (e-forms, structured PDFs) can define new interface types

HTML Source Linking — Apple 10-K Example

xbrl:htmlSpanId links each fact value to an element in Apple's SEC filing HTML

OIM Taxonomy JSON (aapl-factset.json)

```
{
  "name": "aapl:fs_f-7",
  "factDimensions": {
    "xbrl:concept": "dei:EntityRegistrantName",
    "xbrl:language": "en-US",
    "xbrl:entity": "cik:0000320193",
    "xbrl:period":
      "2024-09-29T00:00:00/2025-09-28T00:00:00"
  },
  "factValues": [
    {
      "name": "aapl:f-7_val",
      "valueSources": [
        {
          "properties": [
            {
              "property": "xbrl:htmlSpanId",
              "value": [ "f-7" ]
            }
          ]
        }
      ]
    }
  ]
}
```

Corresponding HTML element (aapl-20250927.htm)

```
<span id="f-7"
  class="xbrl-fact">  Apple Inc.
</span>

<!-- Fact value "Apple Inc." is read
     from span with id="f-7" -->
```

Key points

- The model never embeds the raw value — it references by ID
- The HTML document is the authoritative source of the value
- Transformation applied to source value gives canonical value
- Multiple valueSources allowed — multi-element facts
- factInterfaceName on sourceMapping defines the protocol
- Works with iXBRL filing structure — no processor changes

PDF & Future Format Linking

The interface pattern extends cleanly to PDF and other document types

PDF Form Field Interface

```
// sourceMapping declares PDF interface
"sourceMappings": [{
  "sourceName": "rpt:AnnualReport",
  "url": "annual-report-2025.pdf",
  "factInterfaceName":
    "xbrl:pdfFormFieldLocatorType"
}]

// Fact value source points to PDF field
"valueSources": [{
  "properties": [{
    "property": "xbrl:pdfFormField",
    "value": ["Revenue_2025"]
  }]
}]
```

PDF MCID (Tagged PDF)

```
"valueSources": [{
  "properties": [
    { "property": "xbrl:pdfPage",    "value": 42 },
    { "property": "xbrl:pdfMcid",    "value": 1203 }
  ]
}]
```

Interface Architecture Advantages

Open for extension

New document types define new interface names — no spec change required

Source document stays

The filing document remains the authoritative source of fact values

Traceability

Every fact value can be traced back to the precise element or field in source

Transformation support

Source values can be transformed to canonical values via transform types

Multiple sources

A single fact can draw its value from multiple source elements (multi-page facts)

Format independence

Model structure identical regardless of whether source is HTML, PDF, or CSV

Section 9

Generic Sources

Model references by QName only — never by URL or file path

Generic Sources — QNames Only

The model references data by QName — never by file path or URL

The importMapping in documentInfo maps model names (QNames) to physical URLs — but the model body only uses QNames

Physical URL mapping (documentInfo only)

```
"importMapping": {
  "aapl:Apple-10K-2025-09-27":
    "../converted-taxonomies/aapl-10K-20250927.json",
  "ixt-sec:TransformsTaxonomyModule":
    "../spec-taxonomies/sec-transform-types.json",
  "xbrltt:TransformsTaxonomyModule":
    "../spec-taxonomies/transform-types.json"
}
```

Model body uses QNames only

```
"importedTaxonomies": [
  { "xbrlModelName": "aapl:Apple-10K-2025-09-27" },
  { "xbrlModelName": "ixt-sec:TransformsTaxonomyModule" },
  { "xbrlModelName": "xbrltt:TransformsTaxonomyModule" }
]
// No URLs here — model identity is the QName
```

✓ URL independence
Move or rename files without touching any model body

✓ Registry lookup
Processors can resolve QNames via a registry rather than local files

✓ Network access
importMapping can point to URIs, local paths, or network endpoints

✓ Environment control
Different importMapping in test vs production — same model body

✓ Compiled models
Compiled form includes all content inline — no runtime resolution needed

✓ Stable identifiers
QNames are permanent identifiers; URLs are transient storage locations

Section 10

Facts in Cubes

Cubes control which facts are valid and how duplicates are handled

Facts in Cubes

Cubes define the dimensional space — facts populate the intersections

```
"cubes": [
  {
    "name": "exp:BalanceSheetCube",
    "cubeDimensions": [
      {
        "dimensionName": "xbrl:concept",
        "domainName": "exp:BalanceSheetDomain"
      },
      {
        "dimensionName": "xbrl:entity",
        "members": ["cik:0000320193"]
      },
      {
        "dimensionName": "xbrl:period",
        "periodType": "instant"
      },
      {
        "dimensionName":
          "us-gaap:StatementEquityComponents",
        "domainName":
          "us-gaap:StatementEquityComponentsDomain",
        "allowDomainFacts": true
      }
    ],
    "duplicateFactsInCube": "consistent duplicates"
  }
]
```

Core dimensions
xbrl:concept, xbrl:entity, xbrl:period, xbrl:unit, xbrl:language

Taxonomy dimensions
Explicitly declared xbrl:dimensionObject objects

allowDomainFacts
true = facts without this dimension are still valid in the cube

excludeCubes
Remove specific subsets of facts using an exclusion cube

Cube per statement
Balance Sheet, Income Statement etc. each defined as a cube

Entity restriction
Constrain cube to specific entities — exclude subsidiaries

Period restriction
instant vs duration enforced at the cube level

Duplicate policies
no / complete / consistent / inconsistent duplicates

Cube Constraints & Duplicate Policies

Fine-grained control over which facts belong and how duplicates are handled

Exclude Cube Pattern

```
"cubes": [
  {
    "name": "exp:ConsolidatedBalanceSheet",
    "cubeDimensions": [...],
    "excludeCubes": [
      "exp:SubsidiaryBalanceSheet"
    ]
    // Facts matching exp:SubsidiaryBalanceSheet
    // are removed from this cube
  }
]
```

Duplicate Fact Policies

"no duplicates"	Error — no two facts may share the same dimensional intersection
"complete duplicates"	Only exact duplicates allowed (identical value and all properties)
"consistent duplicates"	Duplicates allowed if their values are consistent (default for cubes)
"inconsistent duplicates"	Conflicting duplicates are reported as conformance errors

Why Cube Constraints Matter

Precise inclusion

A cube explicitly lists which concepts and members are valid — improper facts are excluded

Entity segmentation

Consolidation cubes can exclude subsidiary-level facts using entity dimension constraints

Period enforcement

Period type (instant/duration) enforced within the cube definition — not just at concept level

Automated QA

Processors check facts against cube definitions and report conformance errors automatically

Regulatory alignment

Regulators can define required cubes — filers that omit required facts fail validation

Section 11

Implied Objects

LEI, ISIN, and other open-world identifiers — no enumeration needed

Implied Objects

Open-world registries without explicit enumeration

For certain namespaces it is impractical to enumerate all valid objects — the population is too large, changes too rapidly, or is defined externally.

```
// In the XBRL model (taxonomy or report)
"namespaces": {
  "lei": "http://standards.iso.org/iso/17442",
  "isin": "https://www.isin.org/isin"
},
"impliedObjects": [
  {
    "namespace":
      "http://standards.iso.org/iso/17442",
    "objectType": "xbrl:memberObject"
  },
  {
    "namespace": "https://www.isin.org/isin",
    "objectType": "xbrl:memberObject"
  }
]
// Now lei:54930084UKLVMY22DS16 is valid
// without explicit definition
```

Processors MUST NOT raise oimte:invalidQNameReference for QNames in implied object namespaces

Legal Entity Identifier (LEI)

ISO 17442 — millions of registered entities, continuously updated.
lei:54930084UKLVMY22DS16 ≡ Apple Inc. LEI

ISIN (Securities)

ISO 6166 — global securities identifiers.
isin:US0378331005 ≡ Apple Inc. common stock

CIK (SEC filers)

SEC Central Index Keys — U.S. company identifiers.
cik:0000320193 ≡ Apple Inc. SEC CIK

Country codes

ISO 3166-1 — 249 current codes + historical codes.
country:US, country:GB — valid without enumerating all

Implied Object Rules

How implied objects interact with the rest of the model

Namespace must be declared

The prefix for an implied object namespace MUST still appear in the namespaces map of documentInfo.
Error oimce:unboundPrefix is raised if a prefix is used without declaration.

Reserved prefix

An implied object namespace prefix is a reserved prefix — it MUST NOT be used to define new objects in a non-system XBRL module (error oimce:invalidURIForReservedAlias).

Treated as fully valid

An implied object is treated as a fully valid object of the declared domain type.
It can appear in dimensions, relationships, labels, and references — just like an explicit object.

Domain type determined by namespace

Each implied object namespace declares one objectType (e.g. xbrl:memberObject).
All QNames in that namespace resolve to that type — no per-QName type declaration needed.

Compiled models

Compiled models may include implied objects as resolved entries in their object lists, or may retain the impliedObjects declaration — both forms are valid.

Summary — OIM Taxonomy Capabilities

1 Model Types	xbrl:taxonomy + xbrl:report; cross-import; module/compiled forms	**2 Unified Definitions**	All registries (currencies, types, transforms) as standard taxonomy JSON
3 Semantic Objects	12 first-class named object types — no assembly from arcroles required	**4 Object ID**	Every object globally identified by QName / SQName
5 Selective Import	Pick exactly the objects you need; filter by any property	**6 Properties**	Declarative, typed, validated properties with inheritance
7 Format Agnostic	Semantic spec independent of serialisation; JSON is primary format	**8 Source Linking**	Facts trace back to HTML/PDF elements via interface types
9 Generic Sources	Model references QNames only; importMapping resolves to files	**10 Facts in Cubes**	Explicit dimension constraints; duplicate and period policies
11 Implied Objects	LEI, ISIN, CIK etc. valid without enumerating every identifier		