edi3 Model Interchange 1.0 Specification

## Status

raw

raw

## Licence

All material published on edi3.org including all parts of this specification are the intellectual property of the UN as per the [UN/CEFACT IPR Policy](https://www.unece.org/fileadmin/DAM/cefact/cf_plenary/plenary12/ECE_TRADE_C_CEFACT_2010_20_Rev2E_UpdatedIPRpolicy.pdf).

This Specification is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. See http://www.gnu.org/licenses.

## Change Process

This document is governed by the [2/COSS](http://rfc.unprotocols.org/spec:2/COSS/) (COSS).

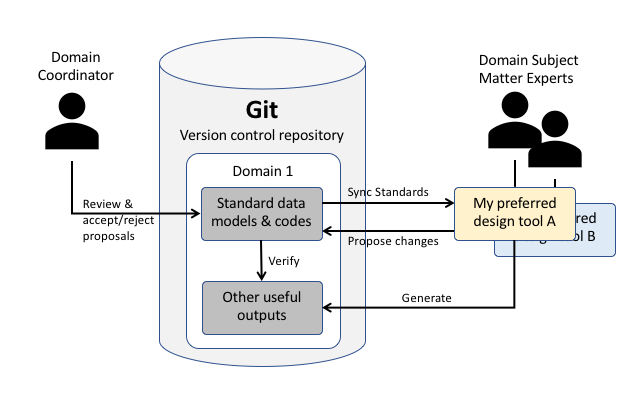
## Language

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

# Introduction

This specification provides a standard JSON structure/schema for the representation of semantic library content and API models. This is used to

* import semantic models into any conformant modelling tool, and to
* interchange semantic models between conformant modelling tools.



use case

This specification will have achieved its purpose when at least two different modelling tools (UML based or otherwise) can successfully interchange any of the three model types.

# Scope

This specification is intended to provide a very simple and standard reporesantation of data models, state charts, and code lists which are the core semantic standard output of organisations such as UN/CEFACT. It is not intended to support internal processes within standards bodies such as the UN/CEFACT core componment library management and harmonisation processes.

## In Scope

* A simple JSON schema for the interchange of business domain models (eg the key resources in the multi-modal transport domain)
* A simple JSON schema for interchange of data models (eg the detailed data model of a supply chain consignment)
* A simple JSON schema for the interchange of state charts (eg the allowed states and transitions of an invoice)
* a simple JSON schema for the interchange of code lists (eg the UNECE Rec 20 units of measure standard code list)

## Out of Scope

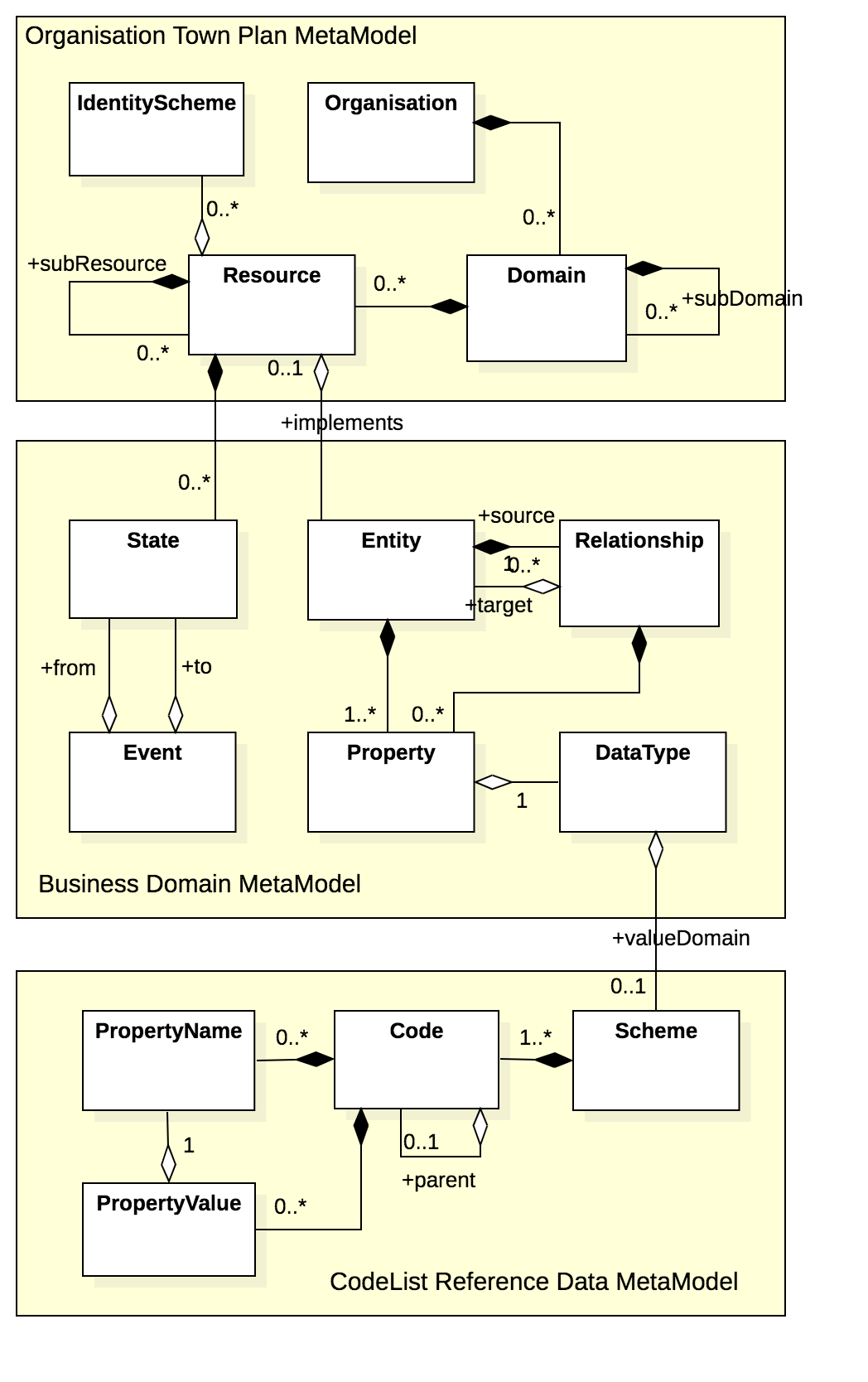
* data elements that are used to support internal harmonisation and library maintenance processes such as those defined by the UN/CEFACT Core Component Technical Specification (CCTS).
* data elements that are used to support complex multi-layered business process modelling such as defined by the UN/CEFACT Modelling Methodology (UMM).
* complete UML model interchange because that is already supported via standards such as XMI (XML Model Interchange)

# Metamodel Overview

A DSL (domain specific language) approach using simple JSON schema is preferred here because it will be simpler and more stable than XMI (interchange standard for UML tools) and will allow non-UML based tools to participate equally in the market.

The interchange specification is broken into three parts, each with a dedicated metamodel and each representing models that can be interchanged independently.

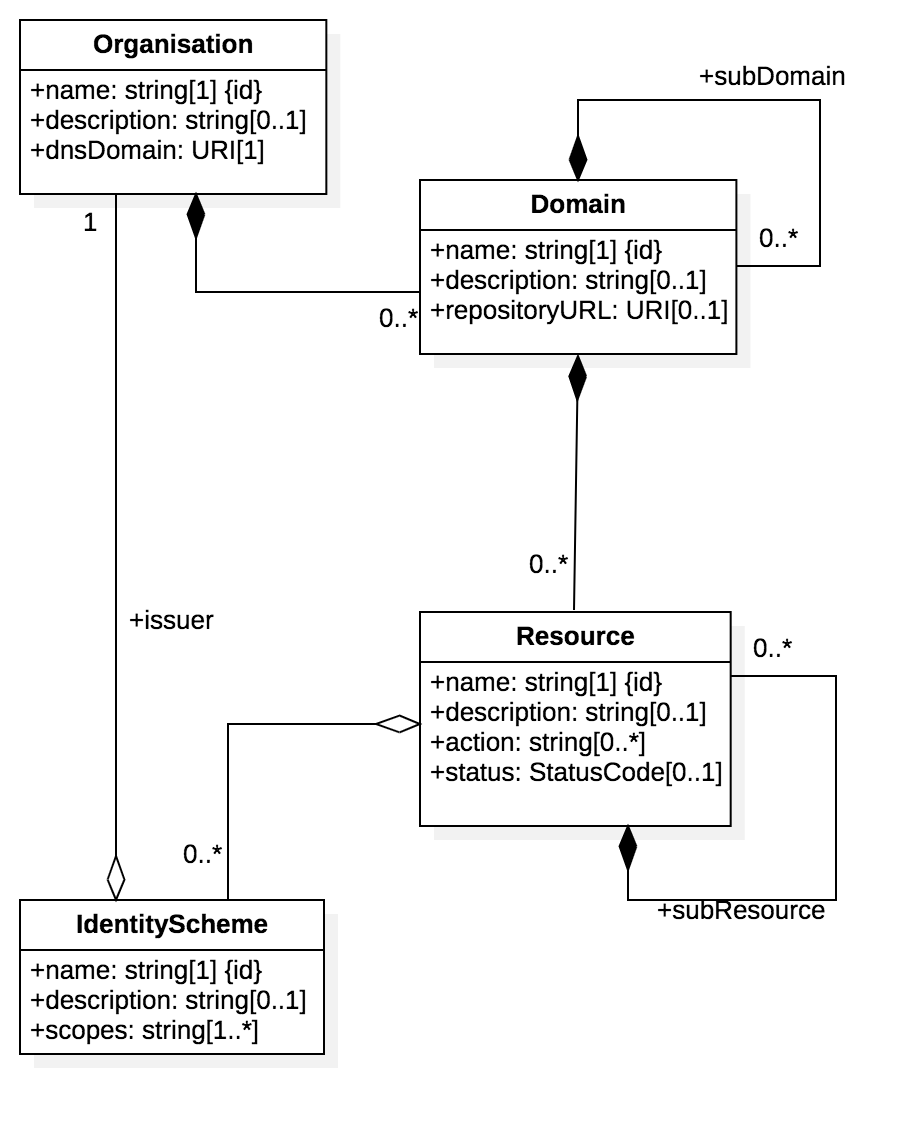
* The town plan model provides the top level organising framework for API resources in multiple business domains. There will typically be just one town plan file version current at any givent time.
* The resource model details the state lifecycle and information model of a specific API resource. There will be typically be one file per domain or subdomain.
* the CodeList model represents code list schemes and the flat or hierarchical set of code values. There will typically be one file per code list.



metamodel

# Town Plan Model Specification

## Logical Model



town plan

The town plan provides a way to break down a complex organisation (suhc as UN/CEFACT) into distinct domains, each responsible for the API resources relevant to the domain. UN/CEFACT domains may include

* International Trade
* Multi Modal Transport
* Regulatory
* Agriculture
* Finance and Accounitng
* Travel and Tourism

Within each domain, there will be a set of well defined API resources. for example, Multi-Model Transport would incluse resources such as

* Consigments - with sub-resource ConssigmentItems
* TransportMeans - with specialised types Vessel, Vehicle, Aircraft
* TransportMovement - with specialised types Voyage, Journey, Flight
* TransportEquipment - with specialised types SeaContainer, AiurULD, etc

Each resource has one or more standard identifier schemes that support the globally unique identity for the resource and also the end-point discoverability of the resource. For example

* A Vessel is identified by an IMO number and details are maintained on the IMO register of vessels
* A SeaContainer is identified by a BIC code and details are maintaine in the international BIC code register.

Structured representations of the town plan can be used to generate browsable sites that navigate the more complex models contained within the town plan. They can also be used to validate that the API specifications published by a domain do conform to the organisation’s town plan (ie they exist on the plan therefore are approved for development). To a large extent, the town plan model is a governance tool to manage the consistent development and publishing of API standards by business domains.

## JSON Schema

[town plan schema](townplan-schema.json)

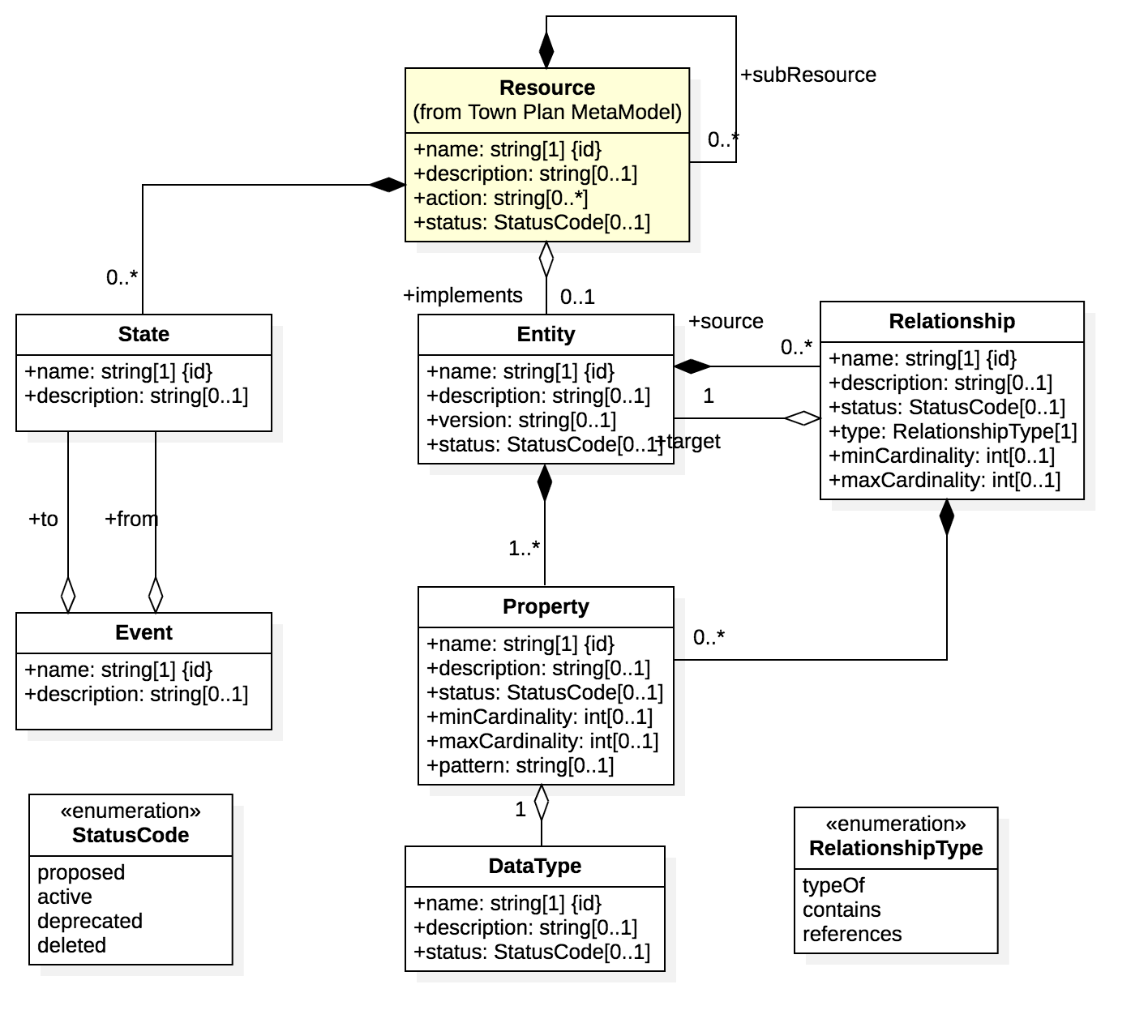
{  
 "schema": {  
 "model": "Town Plan MetaModel",  
 "type": "object",  
 "properties": {},  
 "definitions": {  
 "Domain": {  
 "description": "A buisness domain within a governing organisation (eg Agriculture).",  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 },  
 "description": {  
 "description": "full description of the metadata element.",  
 "type": "string"  
 },  
 "repositoryURL": {  
 "description": "The authoritative location of the repository containing version controlled domain information (eg https://github.com/edi3/edi3-regulatory)",  
 "type": "string"  
 },  
 "Resource": {  
 "items": {  
 "$ref": "#/definitions/Resource"  
 },  
 "type": "array"  
 },  
 "subDomain": {  
 "items": {  
 "$ref": "#/definitions/Domain"  
 },  
 "type": "array"  
 }  
 },  
 "required": [  
 "name"  
 ]  
 },  
 "IdentityScheme": {  
 "description": "A globally unique identity scheme for the related resource ",  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 },  
 "description": {  
 "description": "full description of the metadata element.",  
 "type": "string"  
 },  
 "scopes": {  
 "items": {  
 "description": "missing description",  
 "type": "string"  
 },  
 "type": "array",  
 "minItems": 1  
 },  
 "issuer": {  
 "allOf": [  
 {  
 "$ref": "#/definitions/OrganisationIds"  
 },  
 {  
 "type": "object"  
 }  
 ]  
 }  
 },  
 "required": [  
 "name",  
 "scopes",  
 "issuer"  
 ]  
 },  
 "OrganisationIds": {  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "missing description",  
 "type": "string"  
 }  
 },  
 "required": [  
 "name"  
 ]  
 },  
 "Organisation": {  
 "description": "An organisation (eg UN/CEFACT) that is the governing authroity over several domains.",  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "missing description",  
 "type": "string"  
 },  
 "description": {  
 "description": "missing description",  
 "type": "string"  
 },  
 "dnsDomain": {  
 "description": "the DNS domain name of the organisation (eg unece.org)",  
 "type": "string"  
 },  
 "Domain": {  
 "items": {  
 "$ref": "#/definitions/Domain"  
 },  
 "type": "array"  
 }  
 },  
 "required": [  
 "name",  
 "dnsDomain"  
 ]  
 },  
 "Resource": {  
 "description": "An information resource owned by the domain that has a defined information model, state lifecycle, and globally unique public identifier (eg consignments). Resource names are always plural nouns",  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 },  
 "description": {  
 "description": "full description of the metadata element.",  
 "type": "string"  
 },  
 "action": {  
 "items": {  
 "description": "A list of allowed actions on this resource (create, read, update, delete). ",  
 "type": "string"  
 },  
 "type": "array"  
 },  
 "status": {  
 "description": "Status of the element from a library management perspective. \n using COSS lifecycle values (raw, draft, stable, deprecated, deleted).",  
 "type": "string",  
 "enum": [  
 "proposed",  
 "active",  
 "deprecated",  
 "deleted"  
 ]  
 },  
 "subResource": {  
 "items": {  
 "$ref": "#/definitions/Resource"  
 },  
 "type": "array"  
 },  
 "State": {  
 "items": {  
 "$ref": "#/definitions/State"  
 },  
 "type": "array"  
 },  
 "IdentityScheme": {  
 "items": {  
 "allOf": [  
 {  
 "$ref": "#/definitions/IdentitySchemeIds"  
 },  
 {  
 "type": "object"  
 }  
 ]  
 },  
 "type": "array"  
 },  
 "implements": {  
 "allOf": [  
 {  
 "$ref": "#/definitions/EntityIds"  
 },  
 {  
 "type": "object"  
 }  
 ]  
 },  
 "Event": {  
 "items": {  
 "$ref": "#/definitions/Event"  
 },  
 "type": "array"  
 }  
 },  
 "required": [  
 "name"  
 ]  
 },  
 "IdentitySchemeIds": {  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 }  
 },  
 "required": [  
 "name"  
 ]  
 },  
 "EntityIds": {  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 }  
 },  
 "required": [  
 "name"  
 ]  
 },  
 "State": {  
 "description": "",  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 },  
 "description": {  
 "description": "full description of the metadata element.",  
 "type": "string"  
 }  
 },  
 "required": [  
 "name"  
 ]  
 }  
 }  
 },  
}

## Sample

coming soon

# Domain Model Interchange Specification

## Logical Model



domain model

The domain model details the data structures and state lifecycles of the resources managed by a domain. It is essentially a collection of simple data models and state charts that are defined withing the domain namespace.

A data model is constrained to include only

* entities (eg “handling\_instruction”) with name, description, version, and status. An entitity has one or more properties.
* properties (eg “minimumStorageHumidity”) with name, descriptiopn, status, min/max cardinality, and pattern (eg a regex expression defining the string format). A property has exactly one data type.
* data types (eg “quantity”) with name, description, status, and optionally a reference to a code list scheme that controls the allowed value domain.
* relationships (eg “consignee\_party”) with name, description, status, min/max cardinality, source/target entity and type. The relationship type is one of:
  + typeOf (eg like a UML generalisation)
  + contains (eg like a UML composition)
  + references (eg like a UML aggregation)

A State chart is constrained to include only

* states (eg “approved”, “paid”, “disputed”, etc) with a name and description of each state
* events ( eg “paymentReceived”, “invoiceCreated”) that also have a name and description and define a transition between two states.
* a parent resource (eg “invoice”) that the state lifecycle describes.

## JSON Schema

[domain schema](domain-schema.json)

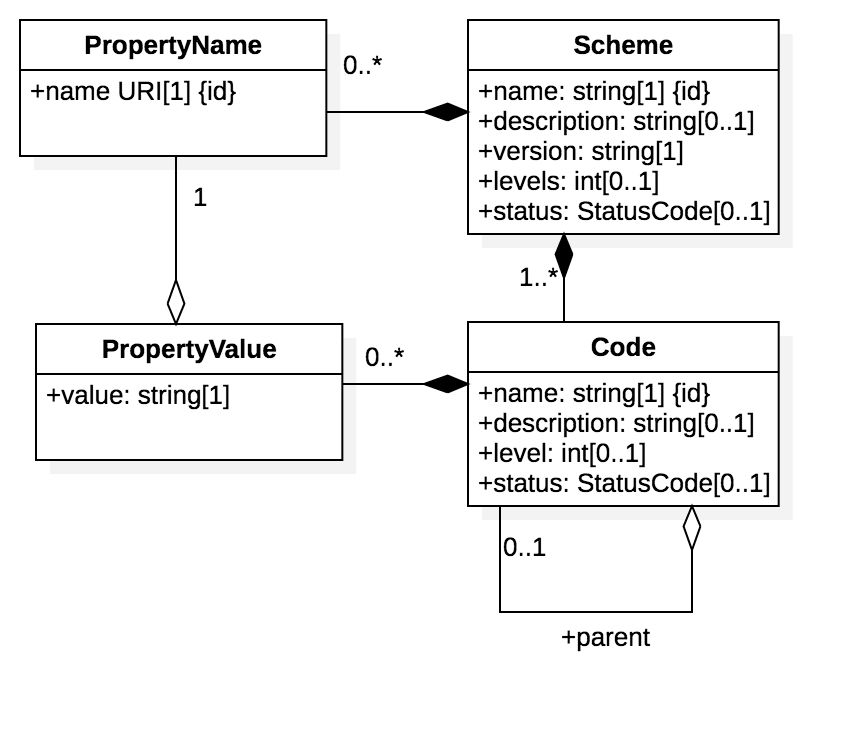
{  
 "$id": "https://edi3.org/specs/edi3-model-interchange/develop/domain-schema.json",  
 "$schema": "http://json-schema.org/draft-07/schema#",  
 "description": "The json schema for domain model interchange format",  
 "title": "DICF",  
 "type": "object",  
 "required": ["resources", "dataTypes"],  
 "properties": {  
 "resources": {  
 "type": "array",  
 "items": { "$ref": "#/definitions/Resource" },  
 "minItems": 1  
 },  
 "dataTypes": {  
 "type": "array",  
 "items": { "$ref": "#/definitions/DataType" },  
 "minItems": 1  
 }  
 },  
 "definitions": {  
 "general": {  
 "name": {  
 "type": "string",  
 "description": "The general name property."  
 },  
 "desc": {  
 "type": "string",  
 "description": "The general description property."  
 },  
 "statusCode": {  
 "type": "string",  
 "enum": ["active", "proposed", "deleted", "deprecated"],  
 "description": "The general property."  
 }  
 },  
 "Resource": {  
 "type": "object",  
 "description": "The resource object",  
 "required": [ "name", "entities" ],  
 "properties": {  
 "name" : {"$ref": "#/definitions/general/name"},  
 "description": {"$ref": "#/definitions/general/desc"},  
 "entities": {  
 "type": "array",  
 "items": { "$ref": "#/definitions/Entity" },  
 "minItems": 1  
 }  
 }  
 },  
 "Entity": {  
 "type": "object",  
 "description": "The entity object",  
 "required": [ "name", "properties" ],  
 "properties": {  
 "name" : {"$ref": "#/definitions/general/name"},  
 "description": {"$ref": "#/definitions/general/desc"},  
 "version": {  
 "type": "string",  
 "description": "The version property."  
 },  
 "status": {"$ref": "#/definitions/general/statusCode"},  
 "properties": {  
 "type": "array",  
 "items": { "$ref": "#/definitions/Property" }  
 },  
 "relationships": {  
 "type": "array",  
 "items": { "$ref": "#/definitions/Relationship" }  
 },  
 "states": {  
 "type": "array",  
 "items": { "$ref": "#/definitions/State" }  
 },  
 "events": {  
 "type": "array",  
 "items": { "$ref": "#/definitions/Event" }  
 }  
 }  
 },  
 "Property": {  
 "type": "object",  
 "description": "The property object",  
 "required": [ "dataType", "name" ],  
 "properties": {  
 "name" : {"$ref": "#/definitions/general/name"},  
 "description": {"$ref": "#/definitions/general/desc"},  
 "status": {"$ref": "#/definitions/general/statusCode"},  
 "minCardinality": {  
 "type": "integer"  
 },  
 "maxCardinality": {  
 "type": "integer"  
 },  
 "pattern": {  
 "type": "string"  
 }  
 }  
 },  
 "Relationship": {  
 "type": "object",  
 "description": "The relationship object",  
 "required": ["name", "type", "target"],  
 "properties": {  
 "name" : {"$ref": "#/definitions/general/name"},  
 "description": {"$ref": "#/definitions/general/desc"},  
 "status": {"$ref": "#/definitions/general/statusCode"},  
 "type": {  
 "type": "string",  
 "enum": ["typeOf", "contains", "references"],  
 "description": "typeOf - generalisation, contains - composition, references - aggregation"  
 },  
 "minCardinality": {  
 "type": "integer"  
 },  
 "maxCardinality": {  
 "type": "integer"  
 },  
 "target": {  
 "type": "object",  
 "required": ["name"],  
 "properties": {  
 "name" : {"$ref": "#/definitions/general/name"},  
 "resource" : {  
 "type": "string"  
 }  
 }  
 }  
 }  
 },  
 "State": {  
 "type": "object",  
 "description": "The state object",  
 "required": ["name"],  
 "properties": {  
 "name": {  
 "$ref": "#/definitions/general/name"  
 },  
 "description": {  
 "$ref": "#/definitions/general/desc"  
 }  
 }  
 },  
 "Event": {  
 "type": "object",  
 "description": "The event object",  
 "required": ["name"],  
 "properties": {  
 "name": {  
 "$ref": "#/definitions/general/name"  
 },  
 "description": {  
 "$ref": "#/definitions/general/desc"  
 },  
 "from": {  
 "$ref": "#/definitions/State"  
 },  
 "to": {  
 "$ref": "#/definitions/State"  
 }  
 }  
 },  
 "DataType": {  
 "type": "object",  
 "description": "The dataType object",  
 "required": ["name"],  
 "properties": {  
 "name": {  
 "$ref": "#/definitions/general/name"  
 },  
 "description": {  
 "$ref": "#/definitions/general/desc"  
 },  
 "status": {"$ref": "#/definitions/general/statusCode"}  
 }  
 }  
 }  
}

## Sample

{  
 "name": "Handling\_Instructions",  
 "description": "Handling information of an instructive nature.",  
 "properties": [  
 {  
 "name": "description",  
 "description": "A textual description of these handling instructions.",  
 "dataType": "Text",  
 "minCardinality": 0,  
 "maxCardinality": 1  
 },  
 {  
 "name": "descriptionCode",  
 "description": "A code specifying a description of these handling instructions.",  
 "dataType": "Code",  
 "minCardinality": 0,  
 "maxCardinality": 1  
 },  
 {  
 "name": "maximumStackabilityQuantity",  
 "description": "The maximum number of units which can be stacked on top of each other according to these handling instructions.",  
 "dataType": "Quantity",  
 "minCardinality": 0,  
 "maxCardinality": 1  
 },  
 {  
 "name": "maximumStackabilityWeight",  
 "description": "The maximum stackability weight applicable to these handling instructions.",  
 "dataType": "Measure",  
 "minCardinality": 0,  
 "maxCardinality": 1  
 },  
 {  
 "name": "maximumStorageHumidity",  
 "description": "The measure of the maximum storage humidity applicable to these handling instructions.",  
 "dataType": "Measure",  
 "minCardinality": 0,  
 "maxCardinality": 1  
 },  
 {  
 "name": "minimumStorageHumidity",  
 "description": "The measure of the minimum storage humidity applicable to these handling instructions.",  
 "dataType": "Measure",  
 "minCardinality": 0,  
 "maxCardinality": 1  
 }  
 ],  
 "relationships": [  
 {  
 "name": "storageInstructedTemperature",  
 "description": "The instructed temperature for storage applicable to these handling instructions.",  
 "type": "references",  
 "minCardinality": 0,  
 "maxCardinality": 1,  
 "target": {  
 "name": "Instructed\_Temperature"  
 }  
 },  
 {  
 "name": "transportSettingTemperature",  
 "description": "A transport related temperature setting applicable to these handling instructions.",  
 "type": "references",  
 "minCardinality": 0,  
 "maxCardinality": 1,  
 "target": {  
 "name": "TransportSetting\_Temperature"  
 }  
 }  
 ]  
 }

# CodeList Interchange Specification

## Logical Model



codes model

The code list model defines all the allowed values and additional properties of a single standard code list (eg UNECE Rec20 Units of measure). It is a flxible model that is designed to accomodate various additonal properties of different codelists. For example

* UNLOCODES have additional properties like “country”, “sub-division”, “geolocation”, “function”
* Units of Measure have additional properties like “conversionFactor”, “category”, “symbol”

A code list is constrained to include:

* A scheme (eg “unece.un.org/rec20”) with name, description, version, levels (for hierarchical codes) and status.
* One codelist scheme contains one or more codes (eg “KG”) with name, description, level and status (eg proposed, active, deprecated)
* One code list scheme also contains zero or more additonal property names (eg “Symbol”)
* Each code contains zero or more additional property values (eg “m/s2”) one for each property name defined by the scheme.
* each code may reference a parent - this is unly applicable for hierarchical code lists like such as ISIC.

This simple but flexible model supports the publishing and maintainence of any flat or hierarchical code list with any num,ber of additional properties per code.

## JSON Schema

[codes schema](codes-schema.json)

{  
 "schema": {  
 "model": "Codes MetaModel",  
 "type": "object",  
 "properties": {},  
 "definitions": {  
 "Code": {  
 "description": "",  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 },  
 "description": {  
 "description": "full description of the metadata element.",  
 "type": "string"  
 },  
 "level": {  
 "description": "missing description",  
 "type": "string"  
 },  
 "status": {  
 "description": "Status of the element from a library management perspective. \n using COSS lifecycle values (raw, draft, stable, deprecated, deleted).",  
 "type": "string",  
 "enum": [  
 "proposed",  
 "active",  
 "deprecated",  
 "deleted"  
 ]  
 },  
 "parent": {  
 "allOf": [  
 {  
 "$ref": "#/definitions/CodeIds"  
 },  
 {  
 "type": "object"  
 }  
 ]  
 },  
 "PropertyValue": {  
 "items": {  
 "$ref": "#/definitions/PropertyValue"  
 },  
 "type": "array"  
 }  
 },  
 "required": [  
 "name"  
 ]  
 },  
 "CodeIds": {  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 }  
 },  
 "required": [  
 "name"  
 ]  
 },  
 "PropertyName": {  
 "description": "",  
 "type": "object",  
 "properties": {  
 "name URI[1]": {  
 "description": "missing description",  
 "type": "string"  
 }  
 }  
 },  
 "PropertyValue": {  
 "description": "",  
 "type": "object",  
 "properties": {  
 "value": {  
 "description": "missing description",  
 "type": "string"  
 },  
 "PropertyName": {  
 "allOf": [  
 {  
 "$ref": "#/definitions/PropertyNameIds"  
 },  
 {  
 "type": "object"  
 }  
 ]  
 }  
 },  
 "required": [  
 "value",  
 "PropertyName"  
 ]  
 },  
 "PropertyNameIds": {  
 "type": "object",  
 "properties": {  
 "name URI[1]": {  
 "description": "missing description",  
 "type": "string"  
 }  
 }  
 },  
 "Scheme": {  
 "description": "",  
 "type": "object",  
 "properties": {  
 "name": {  
 "description": "The name of the library element - ie the name of the domain, resource, entity, property, code, or any other specialised metamodel model element.",  
 "type": "string"  
 },  
 "description": {  
 "description": "full description of the metadata element.",  
 "type": "string"  
 },  
 "version": {  
 "description": "missing description",  
 "type": "string"  
 },  
 "levels": {  
 "description": "missing description",  
 "type": "string"  
 },  
 "status": {  
 "description": "Status of the element from a library management perspective. \n using COSS lifecycle values (raw, draft, stable, deprecated, deleted).",  
 "type": "string",  
 "enum": [  
 "proposed",  
 "active",  
 "deprecated",  
 "deleted"  
 ]  
 },  
 "PropertyName": {  
 "items": {  
 "$ref": "#/definitions/PropertyName"  
 },  
 "type": "array"  
 },  
 "Code": {  
 "items": {  
 "$ref": "#/definitions/Code"  
 },  
 "type": "array",  
 "minItems": 1  
 }  
 },  
 "required": [  
 "name",  
 "version",  
 "Code"  
 ]  
 }  
 }  
 }  
}

## Sample

coming soon