**Generation of JSON Schema Draft 04**

ISO 20022 Technical Support Group

Secretariat: SIX-group.com

**Financial services — Universal financial industry message scheme —**

**Generation of JSON Schema Draft 04**

Working Draft

2024-06-12

**Warning for WDs and CDs**

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Contents

[Foreword iv](#_Toc164072500)

[Introduction v](#_Toc164072501)

[1 Scope 7](#_Toc164072502)

[2 Normative references 7](#_Toc164072503)

[3 Terms and definitions 7](#_Toc164072504)

[4 Background 7](#_Toc164072505)

[5 ISO 20022 transformation rules for MessageSet 8](#_Toc164072506)

[5.1 Preconditions 8](#_Toc164072507)

[5.2 Other rules for the specific syntax 8](#_Toc164072508)

[6 MessageInstances 8](#_Toc164072509)

[6.1 Completeness 8](#_Toc164072510)

[6.2 Other rules for the specific syntax 8](#_Toc164072511)

[7 Method 8](#_Toc164072512)

[8 Syntax Generation Rules of Logical Classes 9](#_Toc164072513)

[8.1 MessageSet 9](#_Toc164072514)

[8.2 MessageDefinitionIdentifier 9](#_Toc164072515)

[8.3 MessageDefinition 9](#_Toc164072516)

[8.3.1 File name and encoding. 9](#_Toc164072517)

[8.3.2 File contents 10](#_Toc164072518)

[8.3.2.1 MessageDefinition root schema 10](#_Toc164072519)

[8.3.2.2 MessageDefinition subschema 10](#_Toc164072520)

[8.4 MessageBuildingBlock 12](#_Toc164072521)

[8.5 MessageComponentType 13](#_Toc164072522)

[8.5.1 ExternalSchema 13](#_Toc164072523)

[8.5.2 MessageComponent 14](#_Toc164072524)

[8.5.2.1 Example MessageComponent SupplementaryData1 14](#_Toc164072525)

[8.5.3 ChoiceComponent 15](#_Toc164072526)

[8.5.3.1 Example 15](#_Toc164072527)

[8.6 MessageElement 16](#_Toc164072528)

[8.6.1 MessageElement is not an array 16](#_Toc164072529)

[8.6.1.1 MessageElement is typed by a MessageComponentType 16](#_Toc164072530)

[8.6.1.2 MessageElement is typed by a DataType 16](#_Toc164072531)

[8.6.2 MessageElement is an array 16](#_Toc164072532)

[8.6.3 MessageAttribute 17](#_Toc164072533)

[8.6.4 MessageAssociationEnd where isComposite is true 17](#_Toc164072534)

[8.6.5 MessageAssociationEnd where isComposite is false 17](#_Toc164072535)

[8.7 Other Considerations 17](#_Toc164072536)

[9 Syntax Generation of User-defined Data Types 18](#_Toc164072537)

[9.1 General 18](#_Toc164072538)

[9.2 Boolean 18](#_Toc164072539)

[9.3 Indicator 18](#_Toc164072540)

[9.3.1.1.1 Example 18](#_Toc164072541)

[9.4 Binary 18](#_Toc164072542)

[9.4.1.1.1 Example 18](#_Toc164072543)

[9.5 String 18](#_Toc164072544)

[9.6 Text 19](#_Toc164072545)

[9.6.1.1.1 Example 19](#_Toc164072546)

[9.7 CodeSet 19](#_Toc164072547)

[9.7.1 CodeSet without ExternalCodeSet semantic markup 19](#_Toc164072548)

[9.7.1.1.1 Example 19](#_Toc164072549)

[9.7.2 CodeSet with ExternalCodeSet semantic markup 20](#_Toc164072550)

[9.8 IdentifierSet 20](#_Toc164072551)

[9.9 Decimal 20](#_Toc164072552)

[9.10 Quantity 20](#_Toc164072553)

[9.10.1.1.1 Example 20](#_Toc164072554)

[9.11 Rate 1](#_Toc164072555)

[9.11.1.1.1 Example 1](#_Toc164072556)

[9.12 Amount 1](#_Toc164072557)

[9.12.1 CurrencyIdentifierSet is not empty 1](#_Toc164072558)

[9.12.2 CurrencyIdentifierSet is empty 2](#_Toc164072559)

[10 Syntax Generation of User-defined Data Types for Date & Time 2](#_Toc164072560)

[10.1 Duration 2](#_Toc164072561)

[DateTime 2](#_Toc164072562)

[10.2 Date 3](#_Toc164072563)

[10.3 Time 3](#_Toc164072564)

[10.4 YearMonth 3](#_Toc164072565)

[10.5 MonthDay 3](#_Toc164072566)

[10.6 Year 3](#_Toc164072567)

[10.7 Month 3](#_Toc164072568)

[10.8 Day 3](#_Toc164072569)

[11 Other Types 3](#_Toc164072570)

[11.1.1.1 DataType UUIDv4Identifier 3](#_Toc164072571)

[11.1.1.1.1 Example 3](#_Toc164072572)

[Annex A Converting ISO 20022 XML into JSON (informative) 5](#_Toc164072573)

[A.1 Introduction 5](#_Toc164072574)

[A.1.1 Conversion Procedure 5](#_Toc164072575)

[A.2 Example 6](#_Toc164072576)

[A.2.1 ISO 20022 XML message 6](#_Toc164072577)

[11.1.1.2 ISO 20022 JSON message 6](#_Toc164072578)

[Annex B Tabular Summary - Working Draft XSD generation (Informative) 8](#_Toc164072579)

[Annex C Tabular Summary (Informative) 12](#_Toc164072580)

[Annex D Further Work (Informative) 16](#_Toc164072581)

Foreword

ISO 20022 is an international standard developed under the ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). A list of all parts in the ISO 20022 series can be found on the ISO website.

This document was prepared by the ISO 20022 Technical Support Group (TSG). It is not part of the international standard but serves as supplement to users of ISO 20022.

Any feedback or questions on this document should be directed to the TSG via the ISO 20022 Registration Authority at https://iso20022.org/.

Introduction

This document specifies the generation of JSON Schema Draft 04 syntax message schemes, from repositories using the metamodel defined in ISO 20022-1:2013. It is based on the TSG’s API JSON whitepaper of 2018.

<https://www.iso20022.org/sites/default/files/documents/D7/ISO20022_API_JSON_Whitepaper_Final_20180129.pdf>

This working draft version is based on a template defined in a new part 9 for Syntax generation rules and requirements, under development in the revision of ISO 20022.

It is intended to serve as a baseline for comparison with future documents using the same template,  
for the update to JSON Schema Draft 2020-12, and potential improvements to the ISO 20022 metamodel.

Financial services — Universal financial industry message scheme —   
Generation of JSON Schema Draft 04

# Scope

This supplement to ISO 20022 is prepared to complement the ISO 20022 Metamodel as specified in ISO 20022-1, with the specific syntax generation rules for:

* JSON Schema Draft 04

# Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 20022‑1:2013,   
Financial services — Universal financial industry message scheme — Part 1: Metamodel

[JSON Schema: core definitions and terminology (json-schema.org/draft-04/draft-zyp-json-schema-04)](https://json-schema.org/draft-04/draft-zyp-json-schema-04)

# Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 20022-1 and draft-zyp-json-schema-04 apply.

# Background

In 2018, the ISO 20022 Technical Support Group published “[ISO 20022 and JSON: An Implementation Best Practices Whitepaper](https://www.iso20022.org/sites/default/files/documents/D7/ISO20022_API_JSON_Whitepaper_Final_20180129.pdf)” for discussion and further development. It served as the basis for the ISO Technical Specification [TS 23029:2020 “Web-service-based application programming interface (WAPI) in financial services”](https://www.iso.org/obp/ui/en/#iso:std:iso:ts:23029:ed-1:v1:en). Their target syntax was JSON Schema Draft 04.

Since 2021, JSON Schema Draft 2020-12 and the OpenAPI Specification v3.1.0 [have been aligned](https://www.openapis.org/blog/2021/02/18/openapi-specification-3-1-released). This document captures the specification as it was, to serve as a baseline for comparison of JSON Schema Draft 04 with later updates.

# ISO 20022 transformation rules for MessageSet

## Preconditions

“The MessageSet used as input for the transformation is a valid instance of the MessageSet meta-class.”

## Other rules for the specific syntax

<<Add necessary sub-clauses>>

# MessageInstances

## Completeness

“The list of transformation rules described in this subclause is complete. Therefore, no other transformation rules are applicable and no other information may be added to the ***JSON Schema*** outside of what is allowed by the transformation rules given below.

The ***JSON Schema*** is a representation of the MessageDefinition.”

## Other rules for the specific syntax

<<Add necessary sub-clauses>>

# Method

<<Fill in explanation about general method of transformation and definition of components of the transformation rules as specified in the following sections >>

# Syntax Generation Rules of Logical Classes

## MessageSet

Each MessageSet is transformed into an artefact of MIME type application/zip, containing JSON Schema for all its MessageDefinition. It may also contain associated documents such as Message Definition Reports and images of the sequence diagrams relevant to this MessageSet.

## MessageDefinitionIdentifier

The message definition identifier is dot concatenated in the sequence Business Area, Functionality, Flavour and Version, using the ISO10646 FULL STOP as the separator character.

It is used to distinguish the default and target namespace of each schema.

Example

acmt.002.001.07

## MessageDefinition

### File name and encoding.

Each MessageDefinition is transformed into a file whose name comprises the generated form of the MessageDefinitionIdentifier, then a hyphen-minus character (-, U+002D), followed by the name of the MessageDefinition, suffixed by “.schema.json” to indicate the file type is a JSON Schema.

Example file name

acmt.001.001.08-AccountOpeningInstructionV08.schema.json

The file comprises ISO 10646 (Unicode) characters. The preferred encoding is UTF-8. Applications systems may freely convert between UTF-8 and other encodings. Names, definitions, and other textual values are copied as is from the model, with the expectation that Unicode normalisation has taken place during modelling.

### File contents

The file contains a single root schema that defines the structure of the MessageDefinition. Within the definitions section it also includes subschema for the MessageDefinition, MessageComponentTypes and DataTypes. These subschemas are referenced and reused within the root schema and other subschemas as necessary.

#### MessageDefinition Root Schema

The MessageDefinition’s root schema is a JSON object comprising the following properties:

1. "$schema": a fixed string value of "http://json-schema.org/draft-04/schema#" indicating that JSON Schema draft-04 version is to be used.
2. "id": a string value used to uniquely identify the specific schema, formed by concatenating:
   1. The base URN: "urn:iso:std:iso:20022:tech:json:" ,
   2. The MessageDefinitionIdentifier (for example: acmt.002.001.07) ,
   3. And the number sign (#, U+0023). Requires further discussion.
   4. Example: "urn:iso:std:iso:20022:tech:json:acmt.002.001.07#"

The "id" serves as the base URI for resolving relative references within the schema.

1. "description": a string value describing the source of this schema. Text TBC.
2. "type": keyword set to the value "object".
3. "additionalProperties": keyword set to the Boolean value **false**, prohibiting any additional properties not strictly defined within the schema.
4. "properties": keyword with an object comprising the following property:
   1. The rootElement property name of the MessageDefinition object (e.g., "Document") with an object value that references ("$ref") the corresponding MessageDefinition subschema concatenated with the number sign (#, U+0023):
      1. Example: "Document": {"$ref": "AccountDetailsConfirmationV07#"}
5. "definitions": keyword with an object comprising subschemas for the MessageDefinition, its MessageComponentTypes and its DataTypes.

#### MessageDefinition Subschemas

The MessageDefinition’s subschema is a JSON object comprising of the following properties:

1. Name property of the MessageDefinition, with an object comprising the following properties:
   1. "id": a string value used to uniquely identify the explicit path of the specific subschema, formed by concatenating:
      1. The path to the subschema, located within the "definitions" object (e.g., "#/definitions/"),
      2. And the name property of the MessageDefinition.
      3. Example: "#/definitions/AccountDetailsConfirmationV07"
   2. "description": a string value of the definition property of the MessageDefinition.
   3. "type" keyword set to the value "object".
   4. "additionalProperties": keyword set to the Boolean value **false**, prohibiting any additional properties not strictly defined within the subschema.
   5. "required" keyword with an array listing the names of all   
      MessageBuildingBlocks whose cardinality property’s minOccurs property is greater than 0. If no such MessageBuildingBlocks exist, then the "required" keyword is omitted.
   6. "properties" keyword with an object comprising objects for each MessageBuildingBlock, each object value comprising the following properties:
      1. Name property of the MessageBuildingBlock with object value comprising:
         1. "$ref" with string value used to reference the MessageBuildingBlock’s referenced MessageComponentType, formed by concatenating:
            1. The location of the subschema, identified with the string "#/definitions/",
            2. And the name property of the MessageBuildingBlock’s referenced complexType or simpleType.

Example: "$ref":"#/definitions/MessageIdentification1"

* + - 1. "description" a string value of the definition property of the MessageDefinition.

### Example

{

"$schema": "http://json-schema.org/draft-04/schema#",

"id": "urn:iso:std:iso:20022:tech:json:acmt.002.001.07#",

"description": "Scope**\n**The AccountDetailsConfirmation message is sent ...",

"type": "object",

"additionalProperties": **false**,

"properties": {

"Document": {

"$ref": "AccountDetailsConfirmationV07#"

}

},

"definitions": {

"AccountDetailsConfirmationV07": {

"id": "AccountDetailsConfirmationV07",

"description": "Scope**\r\n**The AccountDetailsConfirmation message…",

"type": "object",

"required": [

"MessageIdentification",

"ConfirmationDetails"

],

"properties": {

"MessageIdentification": {

"$ref": "#/definitions/MessageIdentification1",

"description": "Reference that uniquely identifies the message…

},

"ConfirmationDetails": {

"$ref": "#/definitions/AccountManagementConfirmation5",

"description": "Information about the request or instruction…"

},

// ... (Further properties representing all MessageBuildingBlocks)

}

}

// ... (Further subschema for MessageComponentTypes and DataTypes)

}

}

## MessageBuildingBlock

The subschema for the MessageBuildingBlock is a JSON object with following members:

1. Pair named for its name property, with object value comprising:
   1. Pair named "id" with string value of its name property.
   2. Pair named "description" with string value of its definition property.
   3. If its cardinality permits a maximum of one occurrence then a   
      Pair named "$ref" with string value of the name property of its complexType or simpleType.
   4. If its cardinality permits more than one occurrence then a
      1. Pair named "type" with string value of “object”.
      2. Pair named "items" with object value comprising a
         1. Pair named "$ref" with string value of the name property of its complexType or simpleType.
      3. Pair named "maxItems" with number value of its cardinality property’s maxOccurs property if not unbounded.

### Example

Properties representing MessageBuildingBlocks:

"AccountParties": {

"id": "AccountParties",

"description": "Confirmation of information related to parties…",

"$ref": "#/definitions/AccountParties15"

},

"SavingsInvestmentPlan": {

"id": "SavingsInvestmentPlan",

"description": "Confirmation of the information related to a savings…",

"type": "array",

"items": {

"$ref": "#/definitions/InvestmentPlan17"

},

"maxItems": 50

}

## MessageComponentType

There are three concrete subclasses of abstract MessageComponentTypes, which generate different styles of subschema, in the definitions property of the root schema.

### ExternalSchema

The subschema for an ExternalSchema is a JSON object with following members:

1. Pair named for its name property, with object value comprising:
   1. Pair named “id” with string value of its name property.
   2. Pair named “description” with string value of its definition property.
      1. If its processContent property has value "SKIP" then,  
         pair named "type" with string value "object".
      2. If its processContent property has value "LAX" then,  
         pair named "type" with string value "object".
      3. If its processContent property has value "STRICT" then,  
         pair named "type" with string value "object" and,
         1. Pair named "$ref" with string value of URI to external schema.

#### Examples

Property representing MessageComponentType where processContent propery has value "LAX":

"SupplementaryDataEnvelope1": {

"id": "SupplementaryDataEnvelope1",

"description": "Technical component that contains the validated supplementary data … ",

"type": "object"

}

Property representing MessageComponentType where processContent propery has value "STRICT":

"SupplementaryDataEnvelope1": {

"id": "SupplementaryDataEnvelope1",

"description": "Technical component that contains the validated supplementary data … ",

"type": "object",

"$ref": "https://example.com/schemas/address.json"

}

### MessageComponent

Each MessageComponent’s subschema is a JSON object with following members:

1. Pair named for its name property, with object value comprising:
   1. Pair named “id” with string value of its name property.
   2. Pair named “description” with string value of its definition property.
   3. Pair named “type” with string value of “object”.
   4. Pair named “additionalProperties” with Boolean value of **false**.
   5. If there are any MessageElements whose cardinality property’s minOccurs property is greater than 0, then a  
      Pair named “required” with array value comprising an array of string of the name of each   
      MessageElement whose cardinality property’s minOccurs property is greater than 0.
   6. Pair named “properties” with object value comprising a member for each MessageElement:
      1. Pair named for its name property with object value comprising:
         1. Pair named “$ref” with string value of the name property of its referenced complexType or simpleType.
         2. Pair named “description” with string value of its definition property.

#### Example

Property representing MessageComponent SupplementaryData1:

"SupplementaryData1": {

"description": "Additional information …",

"type": "object",

"additionalProperties": **false**,

"required": [

"Envelope"

],

"properties": {

"PlaceAndName": {

"$ref": "#/components/schemas/Max350Text",

"description": "Unambiguous reference to the location where the supplementary data must be inserted in the message instance.**\r\n**In the case of XML, this is expressed by a valid XPath."

},

"Envelope": {

"$ref": "#/components/schemas/SupplementaryDataEnvelope1",

"description": "Technical element wrapping the supplementary data."

}

}

### ChoiceComponent

A ChoiceComponent is transformed into a JSON object with following characteristics:

* MessageComponent.Name is the name of the JSON object.
* ~~JSON value pair with Name "type" and Value “object”~~
* ~~JSON value pair with Name "additionalProperties" and Boolean value of~~ **~~false~~**~~.~~
* ~~JSON Object "properties" containing 1 or more MessageElements (see below on how MessageElements is transformed)~~
* JSON value pair with Name "oneOf" and array containing all the elements of the ChoiceComponent ~~whereby each element is a JSON value pair with Name "required" and Value an array containing the MessageElement.Name.~~

#### Example

Property representing ChoiceComponent with referenced property:

"CategoryPurpose1Choice": {

"oneOf": [

{

"type": "object",

"properties": {

"Cd": {

"type": "string"

}

},

"required": [

"Cd"

],

"additionalProperties": **false**

},

{

"type": "object",

"properties": {

"Prtry": {

"$ref": "#/definitions/Max35Text"

}

},

"required": [

"Prtry"

],

"additionalProperties": **false**

}

]

},

"Max35Text": {

"type": "string",

"maxLength": 35

}

## MessageElement

### MessageElement is not an array

#### MessageElement is typed by a MessageComponentType

A MessageElement is transformed into a JSON object (if the MessageElement is not an array) or a JSON array (if the MessageElement is an array), with following characteristics:

* MessageElement.Name is the name of the JSON object or the JSON array.
* JSON value pair with Name "type" and Value “object”
* JSON value pair with Name "additionalProperties" and Boolean value of **false**.
* JSON Object "properties" with following content:

o MessageElement Type is the JSON value pair with name "$ref" and as Value the concatenation of "#/definitions/” with MessageComponentType.Name

##### Example

Property representing "other\_account\_selection\_data" typed by "InvestmentAccount64":

"other\_account\_selection\_data": {

"type": "object",

"additionalProperties": **false**,

"$ref": "#/definitions/InvestmentAccount64"

}

#### MessageElement is typed by a DataType

A MessageElement is transformed into a JSON object (if the MessageElement is not an array) or a JSON array (if the MessageElement is an array), with following characteristics:

* MessageElement.Name is the name of the JSON object or the JSON array.
* JSON value pair with Name "type" and Value “string”
* JSON value pair with name "$ref" and as Value the concatenation of "#/definitions/” with Datatype.Name

##### Example

Property representing "additional\_information" types by "Max350Text":

"additional\_information": {

"type": "string",

"$ref": "#/definitions/Max350Text"

}

### MessageElement is an array

A MessageElement is transformed into a JSON array with following characteristics:

* MessageElement.Name is the name of the JSON object.
* JSON value pair with Name "type" and Value "array"
* Optional[[1]](#footnote-2) JSON value pair with name "minItems" containing as Value the minimal number of occurrences
* Optional[[2]](#footnote-3) JSON value pair with name "maxItems" containing as Value the maximum number of occurrences
* JSON Object "items" with following content:

o MessageElement Type is the JSON value pair with name "$ref" and as Value the con

catenation of "#/definitions/” with Datatype.Name

#### Example

Below example shows the element “cash\_settlement” may occur max 8 times:

"cash\_settlement": {

"type": "array",

"maxItems": 8,

"items": {

"$ref": "#/definitions/CashSettlement1"

}

}

### MessageAttribute

<< Fill in the content of syntax generation rules >>

### MessageAssociationEnd where isComposite is true

<< Fill in the content of syntax generation rules >>

### MessageAssociationEnd where isComposite is false

<< Fill in the content of syntax generation rules >>

## Other Considerations

<< Fill in the content of syntax generation rules >>

# Syntax Generation of User-defined Data Types

## General

<< Fill in explanation about DataTypes and their XSD-based Properties>>

## Boolean

DataType “boolean” is transformed into JSON simple type “boolean”.

### Example

{

"boolean": {

"type": "boolean"

}

}

## Indicator

DataType Indicator is transformed into a JSON object with following characteristics:

* Indicator.Name is the name of the JSON object.
* JSON value pair with Name "type" and Value “boolean” **Example**

### Example

Below example shows datatype YesNoIndicator.

"YesNoIndicator": {

"type": "boolean"

}

## Binary

DataType Binary is transformed into a JSON object with following characteristics:

* Binary.Name is the name of the JSON object.
* JSON value pair with Name "type" and value "string"
* If Property minLength is not emply, JSON value pair with Name "minLength" and Value the content of Property minLength.
* If Property maxLength is not emply, JSON value pair with Name "maxLength" and Value the content of Property maxLength.

### Example

Below example shows a binary datatype of minimum 1 character and maximum 102400 characters.

"Max100KBinary": {

"type": "string",

"maxLength": 102400,

"minLength": 1

}

## String

DataType “string” is transformed into JSON simple type “string”.

### Example

{

"string": {

"type": "string"

}

}

## Text

DataType Text is transformed into a JSON object with following characteristics:

* Text.Name is the name of the JSON object.
* JSON value pair with Name "type" and value "string"
* If Property minLength is not emply, JSON value pair with Name "minLength" and Value the content of Property minLength.
* If Property maxLength is not emply, JSON value pair with Name "maxLength" and Value the content of Property maxLength.
* If Property length is not emply, both JSON value pair with Name "minLength" and Value the content of Property length and JSON value pair with Name "maxLength" and Value the content of Property length.
* If Property Pattern is not emply, JSON value pair with Name "pattern" and Value the content of Property Pattern.

##### Example

Below example shows a datatype that validates the correct syntax for a BIC.

"AnyBICIdentifier": {

"type": "string",

"pattern": "^[A-Z]{6,6}[A-Z2-9][A-NP-Z0-9]([A-Z0-9]{3,3}){0,1}$"

}

## CodeSet

### CodeSet without ExternalCodeSet semantic markup

DataType CodeSet is transformed into a JSON object with following characteristics:

* CodeSet.Name is the name of the JSON object.
* If CodeSet.property is not empty: JSON value pair with Name "pattern" and Value the content of datatype CodeSet’s Property pattern
* JSON value pair with Name "type" and Value “string”
* JSON value pair with Name "enum" and Value the array containing all the CodeSetLiteral values of the CodeSet.

##### Example

Below example shows the codeSet “SettlementMethod1Code” containing codes INDA, INGA, COVE and CLRG:

"SettlementMethod1Code": {

"type": "string",

"enum": [

"INDA",

"INGA",

"COVE",

"CLRG"

]

}

### CodeSet with ExternalCodeSet semantic markup

<< Fill in the content of syntax generation rules >>

## IdentifierSet

DataType Identifier is transformed into a JSON object with following characteristics:

* Identifier.Name is the name of the JSON object.
* JSON value pair with Name "type" and Value “string”
* If Property Pattern is not empty, JSON value pair with Name "pattern" and Value the content of Property Pattern. **Example**

Below example shows a datatype of representation Identifier.

"IBAN2007Identifier": {

"type": "string",

"pattern": "^[A-Z]{2,2}[0-9]{2,2}[a-zA-Z0-9]{1,30}$"

}

## Decimal

Datatype “decimal” is transformed into JSON object with the following characteristics:

* JSON value pair with Name "type" and Value "string"
* JSON value pair with Name "type" and Value "^-?(**\\**d+)?**\\**.?**\\**d\*$"

##### Example

{

"type": "number",

"multipleOf": 0.000000000000000001

}

## Quantity

DataType Quantity[[3]](#footnote-4) is transformed into a JSON object with following characteristics:

* Quantity.Name is the name of the JSON object.
* JSON value pair with Name "type" and Value “string”
* If Quantity.totalDigits is not empty: JSON value pair with Name "maxLength" and Value the value of datatype Quantity’s Property totalDigits.

Below example shows a datatype of representation Quantity.

"DecimalNumber": {

"type": "string",

"maxLength": 19

}

## Rate

DataType Rate[[4]](#footnote-5) is transformed into a JSON object with following characteristics:

* Rate.Name is the name of the JSON object.
* JSON value pair with Name "type" and Value “string”
* If Rate.totalDigits is not empty: JSON value pair with Name "maxLength" and Value the value of datatype Rate’s Property totalDigits.

A Rate has a “baseValue” property, but this is not represented in the JSON schema.

##### Example

Below example shows a datatype of representation Rate.

"BaseOneRate": {

"type": "string",

"maxLength": 12

}

## Amount

### CurrencyIdentifierSet is not empty

DataType Amount is transformed into two JSON objects.

A JSON object with following characteristics:

* Amount.Name is the name of the JSON object.
* JSON value pair with Name "type" and Value "object"
* JSON Object "properties" with following content:
  + JSON value pair with Name “$” and value a JSON object with following values:
    - JSON value pair with Name "type" and Value “string” ”
    - If Amount. totalDigits is not empty: JSON value pair with Name "maxLength" and Value the value of datatype Amount’s Property totalDigits.
  + JSON value pair with the Name of the Property CurrencyIdentifierSet.Name and value a JSON object with following value:
    - JSON value pair with name "$ref" and as Value the concatenation of

"#/definitions/” with CurrencyIdentifierSet.Type A

JSON object with following characteristics:

* CurrencyIdentifier.Type is the name of the JSON object.
* JSON value pair with Name "type" and Value “string”
* If Property Pattern of CurrencyIdentifierSet’s Type is not empty: JSON value pair with Name "pattern" and Value the content of that Property.

###### Example

Below example shows element “$” to contain an amount of maximum 19 characters and “currency” to contain a currency code constrained by a pattern.

"ActiveCurrencyAndAmount": {

"type": "object",

"properties": {

"$": {

"type": "string",

"maxLength": 19

},

"currency": {

"$ref": "#/definitions/ActiveCurrencyCode"

}

}

}

"ActiveCurrencyCode": {

"type": "string",

"pattern": "^[A-Z]{3,3}$"

}

### CurrencyIdentifierSet is empty

DataType Amount is transformed into a JSON object with following characteristics:

* Amount.Name is the name of the JSON object.
* JSON value pair with Name "type" and value “string”
* If Amount.totalDigits is not empty: JSON value pair with Name "maxLength" and Value the content of datatype Amount’s Property totalDigits.

Below example shows an amount of maximum 18 characters.

"ImpliedCurrencyAndAmount": {

"type": "string",

"maxLength": 1\*

}

# Syntax Generation of User-defined Data Types for Date & Time

## Duration

<< Fill in the content of syntax generation rules >>

## DateTime

DataType DateTime is transformed into a JSON object with following properties:

1. DateTime.Name is the name of the JSON object.
2. JSON value pair with Name "type" and value "string"
3. JSON value pair with Name "pattern" and value "^(?:[1-9]**\\**d{3}-(?:(?:0[1-9]|1[0-2])-(?:0[1-9]|1**\\**d|2[0-8])|(?:0[13-9]|1[0-2])-(?:29|30)|(?:0[13578]|1[02])-31)|(?:[1-9]**\\**d{2}(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)T(?:[01]**\\**d|2[0-3]):[0-5]**\\**d:[0-5]**\\**d(?:**\\**.**\\**d+)?(?:Z|[+-](?:[01]**\\**d|2[0-3]):?[0-5]**\\**d)?$"

### Example

Datatype representation of ISODateTime.

"ISODateTime": {

"type": "string",

"pattern": "^(?:[1-9]**\\**d{3}-(?:(?:0[1-9]|1[0-2])-(?:0[1-9]|1**\\**d|2[0-8])|(?:0[13-9]|1[0-2])-(?:29|30)|(?:0[13578]|1[02])-31)|(?:[1-9]**\\**d{2}(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)T(?:[01]**\\**d|2[0-3]):[0-5]**\\**d:[0-5]**\\**d(?:**\\**.**\\**d+)?(?:Z|[+-](?:[01]**\\**d|2[0-3]):?[0-5]**\\**d)?$"

}

### NormalisedDateTime

For NormalisedDateTimes the transformation is into a JSON object with the following properties:

1. NormalisedDateTime is the name of the JSON object.
2. JSON value pair with Name "type" and value "string"
3. JSON value pair with Name "pattern" and value "^(?:[1-9]**\\**d{3}-(?:(?:0[1-9]|1[0-2])-(?:0[1-9]|1**\\**d|2[0-8])|(?:0[13-9]|1[0-2])-(?:29|30)|(?:0[13578]|1[02])-31)|(?:[1-9]**\\**d{2}(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)T(?:[01]**\\**d|2[0-3]):[0-5]**\\**d:[0-5]**\\**d(?:**\\**.**\\**d+)?Z$"

#### Example

DateType representation of ISONormalisedDateTime:

"ISONormalisedDateTime": {

"type": "string",

"pattern": "^(?:[1-9]**\\**d{3}-(?:(?:0[1-9]|1[0-2])-(?:0[1-9]|1**\\**d|2[0-8])|(?:0[13-9]|1[0-2])-(?:29|30)|(?:0[13578]|1[02])-31)|(?:[1-9]**\\**d{2}(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)T(?:[01]**\\**d|2[0-3]):[0-5]**\\**d:[0-5]**\\**d(?:**\\**.**\\**d+)?Z$"

}

## Date

DataType Date is transformed into a JSON object with the following properties:

1. ISODate is the name of the JSON object.
2. JSON value pair with Name "type" and value "string"
3. JSON value pair with Name "pattern" and value "^[1-9]**\\**d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]**\\**d|3[01])$"

### Example

DateType representation of ISODate:

"ISODate": {

"type": "string",

"pattern": "^[1-9]**\\**d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]**\\**d|3[01])$"

}

## Time

DataType Time is transformed into a JSON object with the following properties:

1. ISODate is the name of the JSON object.
2. JSON value pair with Name "type" and value "string"
3. JSON value pair with Name "pattern" and value "^(?:[01]**\\**d|2[0-3]):[0-5]**\\**d:(?:[0-5]**\\**d|60)(?:**\\**.[0-9]+)?(?:Z|[+-][01]**\\**d:?[0-5]**\\**d)?$"

### Example

DataType representation of ISOTime:

"ISOTime": {

"type": "string",

"pattern": "^(?:[01]**\\**d|2[0-3]):[0-5]**\\**d:(?:[0-5]**\\**d|60)(?:**\\**.[0-9]+)?(?:Z|[+-][01]**\\**d:?[0-5]**\\**d)?$"

}

## YearMonth

DataType YearMonth is transformed into a JSON object with the following properties:

1. ISOYearMonth is the name of the JSON object.
2. JSON value pair with Name "type" and value "string"
3. JSON value pair with Name "pattern" and value "^[1-9]**\\**d{3}-(0[1-9]|1[0-2])$"

### Example

DataType representation of ISOYearMonth:

"ISOYearMonth": {

"type": "string",

"pattern": "^[1-9]**\\**d{3}-(0[1-9]|1[0-2])(Z|[+-](?:2[0-3]|[01][0-9]):[0-5][0-9])?$"

}

## MonthDay

DataType MonthDay is transformed into a JSON object with the following properties:

1. MonthDay is the name of the JSON object.
2. JSON value pair with Name "type" and value "string"
3. JSON value pair with Name "pattern" and value "^--(0[1-9]|1[0-2])-(0[1-9]|[12]**\\**d|3[01])(Z|[+-](?:2[0-3]|[01][0-9]):[0-5][0-9])?$"

### Example

"gMonthDay": {

"type": "string",

"pattern": "^--(0[1-9]|1[0-2])-(0[1-9]|[12]**\\**d|3[01])(Z|[+-](?:2[0-3]|[01][0-9]):[0-5][0-9])?$"

}

## Year

<< Fill in the content of syntax generation rules >>

## Month

<< Fill in the content of syntax generation rules >>

## Day

<< Fill in the content of syntax generation rules >>

# Other Types

#### DataType UUIDv4Identifier

DataType UUIDv4Identifier is transformed into a JSON object with following characteristics:

* Name is “UUIDv4Identifier”.
* JSON value pair with Name "type" and Value “string”
* JSON value pair with Name "pattern" and Value "^[a-f0-9]{8}-[a-f0-9]{4}-4[a-f0-9]{3}-[89ab][a-f0-9]{3}-[a-f0-9]{12}$"

##### Example

Below example shows a datatype of representation Identifier.

"UUIDv4Identifier": {

"type": "string",

"pattern": "^[a-f0-9]{8}-[a-f0-9]{4}-4[a-f0-9]{3}-[89ab][a-f0-9]{3}-[a-f0-9]{12}$"

},

**Annex A**

1. Converting ISO 20022 XML into JSON   
   (informative)  
   1. Introduction

To check that the generated ISO 20022 JSON schemas are valid, we need to test them. The best way to test them is to convert ISO 20022 XML messages into JSON, and that validate that JSON against the JSON schemas.

While XML and JSON are structurally similar, there are some complications in converting ISO 20022 XML into JSON:

* You need to convert the abbreviated ISO 20022 XML names into unabbreviated snake\_case names.
  + There is no algorithmic way to do this, so instead we used name mapping tables creates during the JSON schema generation process.
* You need to know the type of element/attribute values, and you need to know whether an element is repeatable or not. A non-repeatable XML element is converted to a JSON property with a data type or object value, while a repeatable XML element is converted to a JSON property with an array value.
  + The only way to get this information is to use the XML Schema. In particular, if you validate the XML against the XML Schemas, you can use the XML parser’s PSVI (postSchema validation infoset) API to give you access to the XML Schema information that is required.
    1. Conversion Procedure

Using the name mapping tables and the PSVI API, the conversion process for each parsed file is as follows:

1. Walk the XML tree top-down from the root element, and populate properties of a JSON object
2. The “xmlns” value is converted to a JSON “@xmlns” property
   1. This assumes that the default namespace is the message’s ISO 20022 namespace.
3. The “Document” root element of the ISO 20022 XML message is ignored. Process its single child element as described below.
4. If an element has a simple type with no attributes, the name is mapped to produce a JSON property with a data type value.
   1. XML Booleans become JSON Booleans.
   2. All other data type values become strings – JSON numbers are not used, because JSON parsers do not have a standard decimal precision for numbers.
5. For an element with attributes or child elements, the name is mapped to produce a JSON property with an **object** value if the “maxOccurs” of the element is <= 1.
6. For an element with attributes or child elements, the name is mapped to produce a JSON property with an **array** value if the “maxOccurs” of the element is >= 2.
7. For an element with a simple type and one or more attributes, the simple type value becomes the value of a property called “$” within the element’s JSON object.
8. For each attribute within an element, the name is mapped and prefixed with “@” to produce a JSON property with a data type value.
9. For each child element within an element, the name is mapped to produce a JSON property with a data type value, object value or array value as appropriate.
10. Where the same child element occurs twice or more in a parent element, the child element is converted to a single JSON property with an array value.

* 1. Example
     1. ISO 20022 XML message

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<Document xmlns=**"urn:iso:std:iso:20022:tech:xsd:tsmt.002.001.04"** xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"**>

<ActvtyRpt>

<RptId>

<Id>**ARPMessage25**</Id>

<CreDtTm>**2009-09-09T11:38:00**</CreDtTm>

</RptId>

<RltdMsgRef>

<Id>**ARRMessage24**</Id>

<CreDtTm>**2009-09-09T11:37:00**</CreDtTm>

</RltdMsgRef>

<Rpt>

<TxId>**01190799181-6940-48**</TxId>

<RptdNtty>

<BIC>**ADIABE22**</BIC>

</RptdNtty>

<RptdItm>

<DtTm>**2009-09-06T08:52:00**</DtTm>

<Actvty>

<MsgNm>**tsmt.020.001.02**</MsgNm>

</Actvty>

<Initr>

<BIC>**ADIABE22**</BIC>

</Initr>

</RptdItm>

<RptdItm>

<DtTm>**2009-09-06T08:54:00**</DtTm>

<Actvty>

<MsgNm>**tsmt.011.001.02**</MsgNm>

</Actvty>

<Initr>

<BIC>**SWHQBEBB**</BIC>

</Initr>

</RptdItm>

</Rpt>

</ActvtyRpt>

</Document>

#### ISO 20022 JSON message

{

"activity\_report" : {

"related\_message\_reference" : {

"creation\_date\_time" : "2009-09-09T11:37:00",

"identification" : "ARRMessage24"

},

"report" : [ {

"transaction\_identification" : "01190799181-6940-48",

"reported\_entity" : [ {

"bic" : "ADIABE22"

} ],

"reported\_item" : [ {

"activity" : {

"message\_name" : "tsmt.011.001.02"

},

"initiator" : {

"bic" : "SWHQBEBB"

},

"date\_time" : "2009-09-06T08:54:00"

}, {

"activity" : {

"message\_name" : "tsmt.020.001.02" },

"initiator" : {

"bic" : "ADIABE22"

},

"date\_time" : "2009-09-06T08:52:00"

} ]

} ],

"report\_identification" : {

"creation\_date\_time" : "2009-09-09T11:38:00",

"identification" : "ARPMessage25"

}

},

"@xmlns" : "urn:iso:std:iso:20022:tech:xsd:tsmt.002.001.04"

}

Annex B Tabular Summary - Working Draft XSD generation  
(Informative)

This tabular summary of XSD generation in the working draft is provided to help align the JSON and XML.

Braces {} in the XSD Generation column indicate replacement by the named property or described item.

All items may have facet restrictions and annotations.

| ISO 20022  Repository Concept | XSD Generation |
| --- | --- |
| MessageDefinitionIdentifier | {businessArea}.{messageFunctionality}.{flavour}.{version} |
| MessageDefinition | <?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  xmlns= "urn:iso:std:iso:20022:tech:xsd:*{MessageDefinitionIdentifier}*"  targetNamespace= "urn:iso:std:iso:20022:tech:xsd:*{MessageDefinitionIdentifier}*"  elementFormDefault="qualified">  <xs:element name="*{abbreviated name}*" type="*{name}*"/>  <xs:element name="*{rootElement}*" type="*{rootElement}*"/>  <xs:complexType name="*{rootElement}*">  <xs:sequence>  <xs:element name="*{abbreviated name}*" "type="*{name)}*"/>  </xs:sequence>  </xs:complexType>  <xs:complexType name="*{name}*">  *{ each MessageBuildingBlock }*  </xs:complexType>  *{ each referenced DataType and MessageComponentType }*  </xs:schema> |
| MessageBuildingBlock | <xs:element name="{name}" type="{type}"   minOccurs="{minimumOccurrence}"   maxOccurs="{maximumOccurrence}"/> |
| MessageAttribute | <xs:element name="{name}" type="{type}"   minOccurs="{minimumOccurrence}"   maxOccurs="{maximumOccurrence}"/> |
| MessageAssociationEnd  [isComposite is true] | <xs:element name="{name}" type="{type}"   minOccurs="{minimumOccurrence}"   maxOccurs="{maximumOccurrence}"/> |
| MessageAssociationEnd  [isComposite is false] | <xs:element name="{name}" type="{type}\_ref"   minOccurs="{minimumOccurrence}"   maxOccurs="{maximumOccurrence}"/> |
| *MessageComponentType when referenced by Message Association End where isComposite is false* | <xs:complexType name="{name}">  *{ depends on type }*  <xs:attribute name="id" type="xs:ID"/>  </xs:complexType>  <xs:complexType name="{name}\_ref">  <xs:attribute name="ref" type="xs:IDREF" use="required"/>  </xs:complexType> |
| MessageComponent | <xs:complexType name="{name}">  <xs:sequence>  *{ each MessageElement }*  </xs:sequence>  </xs:complexType> |
| ChoiceComponent | <xs:complexType name="{name}">  <xs:choice>  *{ each MessageElement }*  </xs:choice>  </xs:complexType> |
| ExternalSchema | <xs:complexType name="{name}">  <xs:sequence>  <xs:any   processContents="{ ProcessContents }"   namespace="{ Namespace }"/>  </xs:sequence>  </xs:complexType> |

| ISO 20022  Repository Concept | XSD Generation |
| --- | --- |
| Amount CurrencyIdentifierSet is empty | <xs:simpleType name="{name}">  <xs:restriction base="xs:decimal">  </xs:restriction>  </xs:simpleType> |
| Amount CurrencyIdentifierSet is not empty | <xs:complexType name="{name}">  <xs:simpleContent>  <xs:extension base="{name}\_Quantity">  <xs:attribute name="ccy" use="required" type="ActiveCurrencyCode"/>  </xs:extension>  </xs:simpleContent> </xs:complexType>  <xs:simpleType name="{name}\_Quantity">  <xs:restriction base="xs:decimal">  </xs:restriction> </xs:simpleType> |
| Scalar types | <xs:simpleType name="{scalar type name}">  *{ annotations }*  <xs:restriction base="xs:{schema type name}">  *{ Facet restrictions }*  </xs:restriction>  </xs:simpleType> |
| Code Set with semantic markup ExternalCodeSet | <xs:simpleType name="{code set name}">  *{ annotations }*  <xs:restriction base="xs:string">  *{ Facet restrictions - no enumerations}*  </xs:restriction>  </xs:simpleType> |
| Code Set with no codes | <xs:simpleType name="{code set name}">  *{ annotations }*  <xs:restriction base="xs:string">  <xs:enumeration value=""/>  </xs:restriction>  </xs:simpleType> |
| Code Set with codes | <xs:simpleType name="{code set name}">  *{ annotations }*  <xs:restriction base="xs:string">  { for each code }  <xs:enumeration value="{ codeName else name }"/>  *{ Facet restrictions}*  </xs:restriction>  </xs:simpleType> |
| Annotations | <xs:annotation>  <xs:documentation>{description}</xs:documentation>  <xs:appinfo>  <constraint lang="{escape(expressionLanguage)}"   >{expression}</constraint>  <!-- Indicator -->  <meaningWhenFalse>{meaningWhenFalse}</meaningWhenFalse>  <meaningWhenTrue>{meaningWhenTrue}</meaningWhenTrue>  <!-- Quantity -->  <unitCode>{unitCode}</unitCode>  <!-- Rate -->  <baseUnitCode>{baseUnitCode}</baseUnitCode>  <baseValue>{baseValue}</baseValue>  <!-- Identifier Set -->  <identificationScheme  >{identificationScheme}</identificationScheme>  <!-- Code Set -->  <identificationScheme  >{identificationScheme}</identificationScheme>  <derivation>{derivation/name}</derivation>  <!-- Code -->  <name>{name}</name>  </xs:appinfo>  </xs:annotation> |
| Facet Restrictions | <xs:enumeration value="{codeName else name}">  <xs:pattern value="{ pattern }"/>  <xs:length value="{ length }"/>  <xs:minLength value="{ minLength }"/>  <xs:maxLegnth value="{ maxLength }"/>  <xs:minInclusive value="{ minInclusive }"/>  <xs:maxInclusive value="{ minInclusive }"/>  <xs:minExclusive value="{ minExclusive }"/>  <xs:maxExclusive value="{ maxExclusive }"/>  <xs:totalDigits value="{ totalDigits }"/>  <xs:fractionDigits value="{ fractionDigits }"/> |

Annex C Tabular Summary   
(Informative)

Empasis in the JSON Schema generation indicates replacement by the named property or described item.

All items have annotations.

| ISO 20022  Repository Concept | JSON Schema Generation |
| --- | --- |
| MessageDefinitionIdentifier | {businessArea}.{messageFunctionality}.{flavour}.{version} |
| MessageDefinition | {  "$schema": "http://json-schema.org/draft-04/schema#",  "id":  "urn:iso:std:iso:20022:tech:json:***MessageDefinitionIdentifier***#"  "description": "***A description of this schema.***",  "type": "object",  "additionalProperties": false,  "properties": {  "Document": {  "$ref": "#***name***"  }  },  "definitions": {  "***name***": {  "id": "***name***"  "description": "***definition***"  "required": [***List of required MessageBuildingBlocks***  ***Minimum cardinality is greater than 1***],  "properties": {  ***Properties representing MessageBuildingBlocks.***  }  },  ***Further subschema for MessageComponentTypes and DataTypes.***  }  } |
| MessageBuildingBlock | ***If maximum cardinality is one, then***  "***name***": {  "id":"#***name***",  "description": "***definition***",  "$ref": "#***name of its simpleType or complexType***"  },  ***If maximum cardinality is greater than one, then***  "Intermediaries": {  "id":"#***name***",  "description": "***definition***",  "type": "array",  "items": {  "$ref": "#***name of its simpleType or complexType***"  },  ***If maximum cardinality is not unbounded, then***  "maxItems":***cardinality.maximum***  }, |
| ExternalSchema | "***name***" : {  "id":"#***name***",  "description": "***definition***",  "type":"object"  }, |
| MessageComponent | <xs:complexType name="{name}">  <xs:sequence>  *{ each MessageElement }*  </xs:sequence>  </xs:complexType> |
| ChoiceComponent | <xs:complexType name="{name}">  <xs:choice>  *{ each MessageElement }*  </xs:choice>  </xs:complexType> |
| MessageAttribute | <xs:element name="{name}" type="{type}"   minOccurs="{minimumOccurrence}"   maxOccurs="{maximumOccurrence}"/> |
| MessageAssociationEnd  [isComposite is true] | <xs:element name="{name}" type="{type}"   minOccurs="{minimumOccurrence}"   maxOccurs="{maximumOccurrence}"/> |
| MessageAssociationEnd  [isComposite is false] | <xs:element name="{name}" type="{type}\_ref"   minOccurs="{minimumOccurrence}"   maxOccurs="{maximumOccurrence}"/> |
| *MessageComponentType when referenced by Message Association End where isComposite is false* | <xs:complexType name="{name}">  *{ depends on type }*  <xs:attribute name="id" type="xs:ID"/>  </xs:complexType>  <xs:complexType name="{name}\_ref">  <xs:attribute name="ref" type="xs:IDREF" use="required"/>  </xs:complexType> |

| ISO 20022  Repository Concept | XSD Generation |
| --- | --- |
| Amount CurrencyIdentifierSet is empty | <xs:simpleType name="{name}">  <xs:restriction base="xs:decimal">  </xs:restriction>  </xs:simpleType> |
| Amount CurrencyIdentifierSet is not empty | <xs:complexType name="{name}">  <xs:simpleContent>  <xs:extension base="{name}\_Quantity">  <xs:attribute name="ccy" use="required" type="ActiveCurrencyCode"/>  </xs:extension>  </xs:simpleContent> </xs:complexType>  <xs:simpleType name="{name}\_Quantity">  <xs:restriction base="xs:decimal">  </xs:restriction> </xs:simpleType> |
| Scalar types | <xs:simpleType name="{scalar type name}">  *{ annotations }*  <xs:restriction base="xs:{schema type name}">  *{ Facet restrictions }*  </xs:restriction>  </xs:simpleType> |
| Code Set with semantic markup ExternalCodeSet | <xs:simpleType name="{code set name}">  *{ annotations }*  <xs:restriction base="xs:string">  *{ Facet restrictions - no enumerations}*  </xs:restriction>  </xs:simpleType> |
| Code Set with no codes | <xs:simpleType name="{code set name}">  *{ annotations }*  <xs:restriction base="xs:string">  <xs:enumeration value=""/>  </xs:restriction>  </xs:simpleType> |
| Code Set with codes | <xs:simpleType name="{code set name}">  *{ annotations }*  <xs:restriction base="xs:string">  { for each code }  <xs:enumeration value="{ codeName else name }"/>  *{ Facet restrictions}*  </xs:restriction>  </xs:simpleType> |
| Annotations | <xs:annotation>  <xs:documentation>{description}</xs:documentation>  <xs:appinfo>  <constraint lang="{escape(expressionLanguage)}"   >{expression}</constraint>  <!-- Indicator -->  <meaningWhenFalse>{meaningWhenFalse}</meaningWhenFalse>  <meaningWhenTrue>{meaningWhenTrue}</meaningWhenTrue>  <!-- Quantity -->  <unitCode>{unitCode}</unitCode>  <!-- Rate -->  <baseUnitCode>{baseUnitCode}</baseUnitCode>  <baseValue>{baseValue}</baseValue>  <!-- Identifier Set -->  <identificationScheme  >{identificationScheme}</identificationScheme>  <!-- Code Set -->  <identificationScheme  >{identificationScheme}</identificationScheme>  <derivation>{derivation/name}</derivation>  <!-- Code -->  <name>{name}</name>  </xs:appinfo>  </xs:annotation> |
| Facet Restrictions | <xs:enumeration value="{codeName else name}">  <xs:pattern value="{ pattern }"/>  <xs:length value="{ length }"/>  <xs:minLength value="{ minLength }"/>  <xs:maxLegnth value="{ maxLength }"/>  <xs:minInclusive value="{ minInclusive }"/>  <xs:maxInclusive value="{ minInclusive }"/>  <xs:minExclusive value="{ minExclusive }"/>  <xs:maxExclusive value="{ maxExclusive }"/>  <xs:totalDigits value="{ totalDigits }"/>  <xs:fractionDigits value="{ fractionDigits }"/> |

Annex D Further Work   
(Informative)

The next steps are to work through the remainder of the document, from 8.5.3 ChoiceComponent onwards:

* clarifying the existing text to indicate kind of the pair's value (text, numeric, etc.)
* Replacing the screen short examples with editable code with the "code" style applied.
* Replacing the $ref JSON pointers with plain types names, as the URI's local fragment identifies the embedded schema resource. I've been advised this is simpler and faster.
* 8.6.5 MessageAssociationEnd where isComposite is false  
  Decide representation of internal cross references.   
  E.g. Some identified text { "@id":"a123", "@value":"some text"}  
  Some reference to the text { "reference" : { "@ref":"a123" } }
* Data type generation - most are straight forward.
  + Should we specify RegExs for date times, or a custom vocabulary or format in draft 4 ?
* In future edition, Text might get a "@language" tag, but not strings.  
  e.g. { "@id":"a123", "@value":"some text", "@language":"en"}
* CodeSets:
  + Although CodeSet could be represented by simple enum, it doesn't support documentation per code.  
    Instead the suggestion is to use the anyOf construct which completes on first match,   
    which is faster than oneOf which has to check every subschema.

For example:

"ExternalAccountIdentification1Code": {

"id": "ExternalAccountIdentification1Code",

"description": "Specifies the external account identification ...",

"type": "string",

"minLength": 1,

"maxLength": 4,

"anyOf": [

{

"enum": ["AIIN"],

"title": "IssuerIdentificationNumber",

"description": "Issuer Identification Number (IIN) - identifies a card issuing institution ..."

}

//... add more options as needed

]

}

Then there are the Annex C Open Issues, and the fully worked example in Annex B of the source:  
ISO 20022 and JSON: An Implementation Best Practices Whitepaper

1. Default value is zero in which case minItems may be omitted. [↑](#footnote-ref-2)
2. The default is that the array is unbounded in which case maxItems may be omitted. [↑](#footnote-ref-3)
3. Technically, a Quantity can have a “unit” property, but at present none of the Quantity definitions in the ISO 20022 repository use that attribute. [↑](#footnote-ref-4)
4. Technically, a Rate can have a “baseUnitCode” property, but none of the Rate definitions in the ISO 20022 repository use that attribute [↑](#footnote-ref-5)