**Generation of JSON Schema Draft 2020-12 for ISO 20022:2013**

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

**Trial Recommendation**

**2025-03-21**

**Warning**

This document is not an ISO International Standard. It is subject to change without notice and may not be referred to as an International Standard. It is a Recommendation of the independent ISO 20022 Technical Support Group distributed for review and comment. Recipients are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Feedback on this document including errata and suggestions for improvement are welcome by contacting:

[ISO 20022 Technical Support Group](https://www.iso20022.org/technical-support-group-tsg)

Secretariat: SIX-group.com

Contents

[Foreword 3](#_Toc193447668)

[Introduction 4](#_Toc193447669)

[1 Scope 5](#_Toc193447670)

[2 Normative references 5](#_Toc193447671)

[3 Terms and definitions 5](#_Toc193447672)

[4 Background 5](#_Toc193447673)

[5 ISO 20022 transformation rules for MessageSet 6](#_Toc193447674)

[5.1 Preconditions 6](#_Toc193447675)

[6 MessageInstances 6](#_Toc193447676)

[6.1 Encoding 6](#_Toc193447677)

[6.2 Completeness 6](#_Toc193447678)

[7 Method 6](#_Toc193447679)

[8 Syntax Generation Rules of Logical Classes 8](#_Toc193447680)

[8.1 MessageSet 8](#_Toc193447681)

[8.2 MessageDefinitionIdentifier 8](#_Toc193447682)

[8.3 MessageDefinition 8](#_Toc193447683)

[8.3.1 File name and encoding. 8](#_Toc193447684)

[8.3.2 File contents 9](#_Toc193447685)

[8.3.3 MessageDefinition Root Schema 9](#_Toc193447686)

[8.3.4 rootElement subschema 10](#_Toc193447687)

[8.3.5 MessageDefinition subschema 11](#_Toc193447688)

[8.4 MessageBuildingBlock 12](#_Toc193447689)

[8.5 MessageComponentType 13](#_Toc193447690)

[8.5.1 ExternalSchema 13](#_Toc193447691)

[8.5.2 MessageComponent 14](#_Toc193447692)

[8.5.3 ChoiceComponent 15](#_Toc193447693)

[8.6 MessageElement 16](#_Toc193447694)

[9 Syntax Generation of User-defined Data Types 18](#_Toc193447695)

[9.1 Boolean 19](#_Toc193447696)

[9.2 Indicator 19](#_Toc193447697)

[9.3 Binary 20](#_Toc193447698)

[9.4 String & Text 21](#_Toc193447699)

[9.5 CodeSet 22](#_Toc193447700)

[9.5.1 CodeSet without ExternalCodeSet semantic markup 22](#_Toc193447701)

[9.5.2 CodeSet with ExternalCodeSet semantic markup 23](#_Toc193447702)

[9.6 IdentifierSet 23](#_Toc193447703)

[10 Syntax Generation of User-defined Data Types for Numbers 24](#_Toc193447704)

[10.1 Pattern for Decimal 24](#_Toc193447705)

[10.2 Decimal 25](#_Toc193447706)

[10.3 Quantity 26](#_Toc193447707)

[10.4 Rate 26](#_Toc193447708)

[10.5 Amount 27](#_Toc193447709)

[10.5.1 CurrencyIdentifierSet is not empty 27](#_Toc193447710)

[10.5.2 CurrencyIdentifierSet is empty 28](#_Toc193447711)

[11 Syntax Generation of User-defined Data Types for Date & Time 29](#_Toc193447712)

[11.1 Duration 29](#_Toc193447713)

[11.2 DateTime 30](#_Toc193447714)

[11.2.1 NormalisedDateTime 30](#_Toc193447715)

[11.3 Date 31](#_Toc193447716)

[11.4 Time 31](#_Toc193447717)

[11.5 YearMonth 32](#_Toc193447718)

[11.6 MonthDay 32](#_Toc193447719)

[11.7 Year 33](#_Toc193447720)

[11.8 Month 33](#_Toc193447721)

[11.9 Day 33](#_Toc193447722)

[Annex A Converting ISO 20022 XML into JSON (Informative) 34](#_Toc193447723)

[A.1 Introduction 34](#_Toc193447724)

[A.2 Example 35](#_Toc193447725)

[A.2.1 ISO 20022 XML message 35](#_Toc193447726)

[Annex B Tabular Summary of JSON Schema generation (Informative) 37](#_Toc193447727)

[Annex C Tabular Summary – 20022-4:2013 XSD generation (Informative) 43](#_Toc193447728)

[Annex D Design Decisions (Informative) 46](#_Toc193447729)

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 2020-12 syntax message schemes, from repositories using the metamodel defined in ISO 20022-1:2013. It is based on the ISO 20022 Technical Support Groups’s API JSON whitepaper of 2018.

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

This version is based on a draft 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.

This document can also be used as a basis for the generation of JSON schemas to be included within ISO 20022-based API resources. As the repository is hierarchically defined, resources can be based on lower-level constructs such as the MessageComponentTypes without requiring the usage of an entire MessageDefinition.

Financial services — Universal financial industry message scheme —   
JSON Schema Draft 2020-12 Generation

# 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 2020-12

# 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/2020-12/draft-bhutton-json-schema-01)](https://json-schema.org/draft/2020-12/draft-bhutton-json-schema-01)

[XML\_Tag\_algorithm.pdf](https://www.iso20022.org/sites/default/files/documents/D7/XML_Tag_algorithm.pdf) (<https://www.iso20022.org/sites/default/files/documents/D7/XML_Tag_algorithm.pdf>)

# Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 20022-1 and  [draft-bhutton-json-schema-01](https://json-schema.org/draft/2020-12/draft-bhutton-json-schema-01.html) 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 has used Draft 2020-12 as the basis for syntax generation to leverage this alignment as well as the financial industries general move towards this standard.

# ISO 20022 transformation rules for MessageSet

## Preconditions

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

# MessageInstances

## Encoding

For universality, all JSON instances derived from the generation guidelines defined here shall be encoded using UTF-8.

RFC 8259 states “JSON text exchanged between systems that are not part of a closed ecosystem MUST be encoded using UTF-8”.

## 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.

# Method

A MessageDefinition is composed of a limited number of distinct modelling patterns. By defining the transformation rules from those patterns to JSON Schema, any MessageDefinition can be transformed into its corresponding JSON Schema.

The following clauses specify the transformation of repository concepts from the logical level to the physical level. Logical classes are specified instances of logical level metaclasses, or user-defined DataTypes, each with their own set of transformation rules. User defined DataTypes are grouped into logical, textual, numeric, and temporal types.

Names of MessageElements (MessageAttributes and MessageAssociationEnds) are derived from their MessageDefinition names according to the algorithm posted on the ISO 20022 website, whereby the algorithm is a normative part of this Recommendation. In practice, the derivation has been performed at design time so that the abbreviation is stored in the repository.

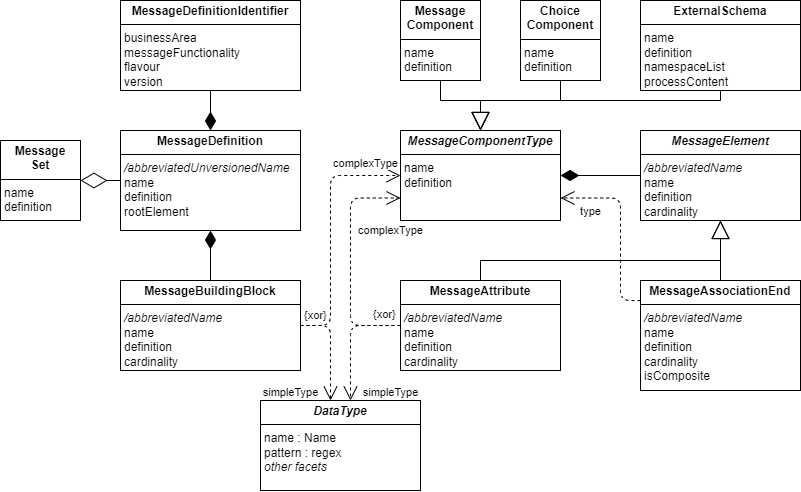


Figure 1 Outline of the ISO 20022:2013 Metamodel for MessageDefinition

# 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 ISO 10646 FULL STOP (U+002E) as the separator character.

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

Example

acmt.002.001.08

## 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.002.001.08-AccountDetailsConfirmationV08.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 transformed into a JSON object comprising the following properties:

1. "$schema": a fixed string value of "https://json-schema.org/draft/2020-12/schema" indicating that JSON Schema draft 2020-12 version is to be used.
2. "$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.002.001.08) ,

Example: "urn:iso:std:iso:20022:tech:json:acmt.002.001.08"

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, including
   1. the date of the repository (in place of the generation date);
   2. the generation software (in place of the ISO 20022 RA issued release number);
   3. documentation text (optional).
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. "required": keyword with an array containing the rootElement property name of the MessageDefinition object (e.g., "Document").
5. "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 ("$ref") the definition of the corresponding rootElement subschema:
      1. Example: "Document": {"$ref": "#\_Document"}
6. "$defs": keyword with an object comprising subschemas for the MessageDefinition, its root element, its MessageComponentTypes and its DataTypes.

Example of root schema, with root element property referencing a root element subschema.

{

"$schema": "https://json-schema.org/draft/2020-12/schema",

"$id": "urn:iso:std:iso:20022:tech:json:acmt.002.001.08",

"description": "Generated from ISO 20022 repository of 2024-07-29 by …",

"type": "object",

"additionalProperties": false,

"required": ["Document"],

"properties": {

"Document": {

"$ref": "#\_Document"

}

},

"$defs": {

…

}

}

### rootElement subschema

A subschema is created for the root element of the MessageDefinition, comprising:

1. name of the root element property prefixed with an underscore, with an object comprising :
   1. "type" keyword set to the value "object".
   2. "additionalProperties": keyword set to the Boolean value **false**, prohibiting any additional properties not strictly defined within the subschema.
   3. "required" keyword with an array listing the unversioned abbreviated name of the MessageDefinition.  
      e.g. "AcctOpngInstr"
   4. "properties" keyword with an object comprising :
      1. unversioned abbreviated name of the MessageDefinition with object comprising:
         1. "$ref" property with string value used to reference the subschema of the MessageDefinition, formed by concatenating:
            1. The location of subschema "#",
            2. And the name property of the MessageDefinition.

Example: "$ref":"#AccountDetailsConfirmationV08"

* 1. "$anchor" keyword as the name of the root element property prefixed with an underscore.

Example of subschema for a rootElement

"\_Document":{

"type": "object",

"additionalProperties": false,

"required": ["AcctDtlsConf"],

"description":"Root element for AccountDetailsConfirmationV08.",

"properties": {

"AcctDtlsConf": {

"$ref": "#AccountDetailsConfirmationV08"

}

},

"$anchor":"\_Document"

}

### MessageDefinition subschema

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

1. Name property of the MessageDefinition, with an object comprising the following properties:
   1. "type" keyword set to the value "object".
   2. "description": a string value of the definition property of the MessageDefinition.
   3. "additionalProperties": keyword set to the Boolean value **false**, prohibiting any additional properties not strictly defined within the subschema.
   4. "required" keyword with an array listing the names of each MessageBuildingBlock whose cardinality’s minimum is set to one or more. If no such MessageBuildingBlocks exist, then the "required" keyword is omitted.
   5. "properties" keyword with an object comprising a name value pair for each MessageBuildingBlock.
   6. "$anchor" keyword as per the name property of the MessageDefinition.

Example of subschema for a MessageDefinition

"AccountDetailsConfirmationV08": {

"type": "object",

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

"additionalProperties": false,

"required": [

"MsgId",

"ConfDtls"

],

"properties": {

"MsgId": {

"$ref": "#MessageIdentification1",

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

},

"ConfDtls": {

"$ref": "#AccountManagementConfirmation5",

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

},

"RltdRef": {

"$ref": "#AdditionalReference13" ,

"description": "RelatedReference\nReference to a linked message…"

},

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

},

"$anchor": "AccountDetailsConfirmationV08"

}

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

## MessageBuildingBlock

Each MessageBuildingBlock is transformed into a name value pair comprising the following properties:

1. the abbreviated name of the MessageElement, with an object comprising the following properties:
   1. "description": a string value concatenating
      1. the name property of the MessageElement,
      2. a space and escaped newline " \n",
      3. the definition property of the MessageElement.
2. If its cardinality permits a maximum of one occurrence, then
   1. if it is a MessageAssociationEnd with false isComposite, then "type":"string"
   2. otherwise "$ref": has string value which references their subschema by their $anchor, by concatenating the octothorpe # with the name property of the referenced complexType or simpleType.  
      Example: "$ref": "#InvestmentPlan17"
3. Else if its cardinality permits more than one occurrence, then:  
    "anyOf": an array of two subschema objects for
   1. a single item, as above;
   2. or an array of the same items, so a subschema object comprising properties
      1. "type": with string value of "array",
      2. "items": with subschema object as for a single item, as above.
      3. If cardinality’s maximum is set to one or more, then "maxItems": with number value of the cardinality property’s maximum.
      4. "minItems": with number value of the larger of 1 or the cardinality’s minimum, to prevent empty arrays.

Example of properties representing MessageBuildingBlocks:

"AcctPties": {

"description": "AccountParties \nConfirmation of information related to …",

"$ref": "#AccountParties17"

},

"SvgsInvstmtPlan": {

"anyOf": [

{"$ref": "#InvestmentPlan17"},

{

"type": "array",

"minItems":1,

"maxItems":50,

"items":{"$ref": "#InvestmentPlan17"}

}

],

"description": "SavingsInvestmentPlan\nConfirmation of the information …"

},

## MessageComponentType

There are three concrete subclasses of the abstract MessageComponentType, which generate different styles of subschema in the “$defs” property of the root schema.

### ExternalSchema

The subschema for an ExternalSchema is an object comprising the following properties:

1. Name property of the MessageComponentType, with an object comprising the following properties:
   1. "$anchor": string value of its name property.
   2. "description": string value of its definition property .

Example: ExternalSchema where processContent property has value "LAX":

"SupplementaryDataEnvelope1": {

"$anchor": "SupplementaryDataEnvelope1",

"description": "Technical component that contains the validated … "

}

### MessageComponent

The subschema for a MessageComponent is a JSON object comprising the following properties:

1. Name property of the MessageComponent, with an object comprising the following properties:
   1. "description": string value of its definition property.
   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 subschema.
   4. "minProperties" keword set to number value 1, if there is no MessageElement with a minimum cardinality of one or more, otherwise the "minProperties" keword is omitted.
   5. "required" keyword with an array listing the names of each MessageElement whose cardinality’s minimum is set to one or more. If no such MessageElement exists, then the "required" keyword is omitted.
   6. "properties" keyword with an object comprising a name value pair for each MessageElement.
   7. "$anchor": string value of its name property.

Example: Property representing MessageComponent SupplementaryData1:

"SupplementaryData1": {

"description": "Additional information where Envelope is Mandatory and PlaceAndName is Optional…",

"type": "object",

"additionalProperties": **false**,

"required": [

"Envlp"

],

"properties": {

"PlcAndNm": {

"$ref": "#Max350Text",

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

},

"Envlp": {

"$ref": "#SupplementaryDataEnvelope1",

"description": "Envelope \nTechnical element wrapping the supplementary data."

},

"$anchor": "SupplementaryData1"

}

### ChoiceComponent

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

1. Name property of the ChoiceComponent, with an object comprising the following properties:
   1. "description": string value of its definition property.
   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 subschema.
   4. "minProperties" set to number value 1.
   5. "maxProperties" set to number value 1.
   6. "properties" keyword with an object comprising a name value pair for each MessageElement.
   7. "$anchor": string value of its name property.

Note: The keywords minProperties and maxProperties test that there is exactly one property, and then the relevant property subschema is applied. This is expected to be more efficient than the keyword "oneOf" , which could validate every subschema before counting to test that only one subschema is valid.

Example Property representing ChoiceComponent with referenced property:

"BlockedStatusReason2Choice": {

"$anchor": "BlockedStatusReason2Choice",

"description": "Choice of formats for a blocked status reason.",

"type": "object",

"additionalProperties": false,

"minProperties": 1,

"maxProperties": 1,

"properties": {

"NoSpcfdRsn": {

"$ref": "#NoReasonCode"

},

"Rsn": {

"items": {

"$ref": "#BlockedStatusReason2"

},

"type": "array"

}

}

},

## MessageElement

There are two concrete subclasses of the abstract MessageElement – MessageAttribute and MessageAssociationEnd.

Each MessageElement is transformed into a name value pair comprising the following properties:

1. the abbreviated name of the MessageElement, with an object comprising the following properties:
   1. "description": a string value concatenating
      1. the name property of the MessageElement,
      2. a space and escaped newline " \n",
      3. the definition property of the MessageElement.
2. If its cardinality permits a maximum of one occurrence, a single item, then
   1. if it is a MessageAssociationEnd with false isComposite, then "type":"string"
   2. otherwise "$ref": has string value which references their subschema by their $anchor, by concatenating the octothorpe # with the name property of the referenced complexType or simpleType.  
      Example: "$ref": "#InvestmentPlan17"
3. Else if its cardinality permits more than one occurrence, then:  
    "anyOf": an array of two subschema objects for
   1. a single item, as above;
   2. or an array of the same items, so a subschema object comprising properties
      1. "type": with string value of "array",
      2. "items": with subschema object as for a single item, as above.
      3. If cardinality’s maximum is set to one or more, then "maxItems": with number value of the cardinality property’s maximum.
      4. "minItems": with number value of the larger of 1 or the cardinality’s minimum, to prevent empty arrays.

Example of element representing "AdditionalInformation" typed by "Max350Text":

"AddtlInf": {

"$ref": "#Max350Text",

"description": "AdditionalInformation \n…"

}

Example of element "CashSettlement" which may occur up to 8 times:

"CshSttlm": {

"anyOf": [

{"$ref": "#CashSettlement1"},

{"type": "array",

"minItems":1,

"maxItems":8,

"items":{"$ref": "#CashSettlement1"}

}

],

"description": "CashSettlement\nCash settlement standing instruction associated to transactions on the account."

},

Example of further MessageElements

"AcctOwnr": {

"$ref": "#PartyIdentification125Choice",

"description": "AccountOwner \nParty that legally owns the account."

},

"AcctSvcr": {

"type": "string",

"description": "AccountServicer \nParty that manages the account"

},

"AcctOwnrOthrId": {

"anyOf": [

{

"$ref": "#GenericIdentification82"

},

{

"type": "array",

"minItems": 1,

"items": {

"$ref": "#GenericIdentification82"

}

}

],

"description": "AccountOwnerOtherIdentification\nAlternative identification, …"

},

# Syntax Generation of User-defined Data Types

A user-defined data type is an instance of a DataType metaclass that has properties restricting its range of accepted values. Each user defined data type is generated as a named subschema with properties

1. "$anchor": string value of its name property.
2. "description": string value of its definition property, preceded by the name and value of its other data type properties, if applicable.

Generation of additional properties specific to each user defined data type is specified below.

**Table 1 DataType Properties**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **pattern** | **length** | **minLength** | **maxLength** | **minInclusive** | **maxInclusive** | **minExclusive** | **maxExclusive** | **totalDigits** | **fractionDigits** | **other** |
| Boolean | x |  |  |  |  |  |  |  |  |  |  |
| Indicator | x |  |  |  |  |  |  |  |  |  |  |
| Binary | x | x | x | x |  |  |  |  |  |  |  |
| String | x | x | x | x |  |  |  |  |  |  |  |
| Text | x | x | x | x |  |  |  |  |  |  |  |
| CodeSet | x | x | x | x |  |  |  |  |  |  | identificationScheme |
| IdentifierSet | x | x | x | x |  |  |  |  |  |  | identificationScheme |
| Decimal | x |  |  |  | x | x | x | x | x | x |  |
| Rate | x |  |  |  | x | x | x | x | x | x | base |
| Quantity | x |  |  |  | x | x | x | x | x | x | baseValue baseUnitCode |
| Amount | x |  |  |  | x | x | x | x | x | x | currencyIdentifierSet |
| Duration | x |  |  |  | x | x | x | x |  |  |  |
| DateTime | x |  |  |  | x | x | x | x |  |  |  |
| Date | x |  |  |  | x | x | x | x |  |  |  |
| Time | x |  |  |  | x | x | x | x |  |  |  |
| Year | x |  |  |  | x | x | x | x |  |  |  |
| Month | x |  |  |  | x | x | x | x |  |  |  |
| Day | x |  |  |  | x | x | x | x |  |  |  |
| YearMonth | x |  |  |  | x | x | x | x |  |  |  |
| MonthDay | x |  |  |  | x | x | x | x |  |  |  |

## Boolean

DataType “boolean” is transformed into JSON object with following characteristics:

1. Name property of the Boolean with an object comprising the following properties:
   1. "type" keyword set to value "string"
   2. "enum" keyword with array of strings "1", "0", "true" and "false".

Example

"Boolean": {

"type": "string",

"enum": [ "1", "0", "true", "false" ],

"description": "W3C XML Schema Built-in datatype \"Boolean\"."

"$anchor": "Boolean"

}

## Indicator

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

1. Name property of the Indicator with an object comprising the following properties:
   1. "type" keyword set to value "string"
   2. "enum" keyword with array of strings "1", "0", "true" and "false".

Example of datatype BatchBookingIndicator.

"BatchBookingIndicator":{

"type": "string",

"enum": [ "1", "0", "true", "false" ],

"description": "true=Identifies that a batch entry for the sum of the amounts of all transactions in the batch or message is requested.\nfalse=Identifies that a single entry for each of the transactions in the batch or message is requested.\n\nIdentifies whether the sending party requests a single debit or credit entry per individual transaction or a batch entry for the sum of the amounts of all transactions. ",

"$anchor": "BatchBookingIndicator"

},

## Binary

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

1. Name property of the binary with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. If Property minLength is not empty, then "minLength" with value equal to the content of Property minLength.
   3. If Property maxLength is not empty, then "maxLength" with value equal to the content of Property maxLength.
   4. If Property length is not empty, then "minLength" with value equal to the content of Property Length and "maxLength" with value equal to the content of Property Length.

Example of a binary datatype of minimum 1 character and maximum 102400 characters.

"Max100KBinary": {

"type": "string",

"minLength": 1,

"maxLength": 102400,

"description": "Binary data of 100K maximum.",

"$anchor": "Max100KBinary"

}

## String & Text

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

1. Name property of the string with an object comprising the following properties
   1. "type" keyword set to the value "string"
   2. If Property minLength is not empty, then "minLength" with value equal to the content of Property minLength.
   3. If Property maxLength is not empty, then "maxLength" with value equal to the content of Property maxLength.
   4. If Property length is not empty, then "minLength" with value equal to the content of Property Length and "maxLength" with value equal to the content of Property Length.
   5. If Property Pattern is not empty, "pattern" and value equal to the content of Property Pattern.

Example of a datatype that restricts the characters and length by use of a regular expression pattern.

"Max6AlphaText" : {

"type":"string",

"pattern": "^[a-zA-Z]{1,6}$",

"description": "Specifies an alpha string between 1 and 6 characters.",

"$anchor": "Max6AlphaText"

},

## CodeSet

### CodeSet without ExternalCodeSet semantic markup

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

1. Name property of the CodeSet with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. If CodeSet.Pattern is not empty: "pattern" and value equal to the content of datatype CodeSet’s Property pattern
   3. If there are any Codes in the CodeSet, then "enum" and array containing all the Code values of the CodeSet,
   4. else include specified facets:
      1. If Pattern is not empty: "pattern" and value equal to the content of datatype Identifier’s Property pattern.
      2. If Property minLength is not empty, then "minLength" with value equal to the content of Property minLength.
      3. If Property maxLength is not empty, then "maxLength" with value equal to the content of Property maxLength.
      4. If Property length is not empty, then "minLength" and "maxLength" with value equal to the content of property length.

Example

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

"SettlementMethod1Code": {

"type": "string",

"description": "identificationScheme=\n\nSpecifies the method used to settle the credit transfer instruction.",

"enum":["INDA","INGA","COVE","CLRG"],

"$anchor": "SettlementMethod1Code"

},

### CodeSet with ExternalCodeSet semantic markup

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

1. Name property of the CodeSet with an object comprising the following properties:
   1. JSON value pair with Name "type" and Value "string"
      1. If Pattern is not empty: "pattern" and value equal to the content of datatype Identifier’s Property pattern.
      2. If Property minLength is not empty, then "minLength" with value equal to the content of Property minLength.
      3. If Property maxLength is not empty, then "maxLength" with value equal to the content of Property maxLength.
      4. If Property length is not empty, then "minLength" and "maxLength" with value equal to the content of property length.

Example codeSet “ExternalClearingSystemIdentification1Code” which implicitly refers to the external codeSet of the same name.

"ExternalClearingSystemIdentification1Code": {

"type": "string",

"minLength": 1,

"maxLength": 5,

"description": "Specifies the clearing system identification code,…",

"$anchor": "ExternalClearingSystemIdentification1Code"

}

## IdentifierSet

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

1. Name property of the Identifier with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. If Pattern is not empty: "pattern" and value equal to the content of datatype Identifier’s Property pattern.
   3. If Property minLength is not empty, then "minLength" with value equal to the content of Property minLength.
   4. If Property maxLength is not empty, then "maxLength" with value equal to the content of Property maxLength.
   5. If Property length is not empty, then "minLength" and "maxLength" with value equal to the content of property length.

Example 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}$",

"description": "identificationScheme=RFC4122; UUIDv4\n\nUniversally Unique IDentifier (UUID) version 4, as described in IETC RFC 4122 \"Universally Unique IDentifier (UUID) URN Namespace\".",

"$anchor": "UUIDv4Identifier"

},

# Syntax Generation of User-defined Data Types for Numbers

## Pattern for Decimal

As strings have to be used to represent decimal values, a pattern has to be defined to restrict the allowed values. XML Schema has two main facets to constrain a decimal: totalDigits and fractionDigits; these have to find their way into the pattern. The pattern for a decimal follows the lexical representation of a decimal in XML schema.

Assumptions:

* If defined, totalDigits >= fractionDigits =>0
* If defined, totalDigits > 0

Given the following definitions:

"IntegerExpression" = (0|[1-9][0-9]{0,"totalDigits-1"})([.]0\*)?

"IntegerExpression2" = ((0|[1-9][0-9]{0,"totalDigits-1"})([.]0\*)?)|[.][0]+

"FractionExpression" = [.][0-9]{1,"fractionDigits"}0\*

"FractionExpression2" = [.][0-9]{1,"totalDigits"}0\*

"RealExpression" = (?=[1-9.][0-9.]{1,"totalDigits"}0\*$)([1-9][0-9]\*)?[.][0-9]{0,"fractionDigits"}0\*

"SignExpression" is set to:

1. "[+]?" if the minInclusive or minExclusive property is greater than or equal to 0
2. "-" if the maxInclusive is less than zero or maxExclusive property is less than or equal to 0
3. "[+-]?" otherwise

If any of totalDigits or fractionDigits is not defined, it (including any addition/substraction) shall be replaced with the empty string. For instance, if total digits is not defined:

{0,"totalDigits"} becomes {0,}

{0,"totalDigits-1"} becomes {0,}

The pattern for a decimal for a JSON object shall be (using the previous definitions):

1. if totalDigits>0
   1. if fractionDigits > 0

pattern = ^"SignExpression"0\*(("IntegerExpression")|("FractionExpression")|("RealExpression"))$

* 1. else if fractionDigits = 0

pattern = ^"SignExpression"0\*("IntegerExpression2")$

* 1. else // fractionDigits is not defined

pattern = ^"SignExpression"0\*(("IntegerExpression")|("FractionExpression2")|("RealExpression"))$

1. else // totalDigits is not defined
   1. if fractionDigits > 0

pattern = ^"SignExpression"([0-9]+([.][0-9]{0,"fractionDigits"}0\*)?|0\*[.][0-9]{1,"fractionDigits"}0\*)$

* 1. else if fractionDigits = 0

pattern = ^"SignExpression"0\*("IntegerExpression2")$

* 1. else // fractionDigits is not defined

pattern = ^"SignExpression"(([0-9]+[.]?)|(0\*[.][0-9]+)|((?=[0-9.]{2,})[0-9]\*[.][0-9]\*))$

## Decimal

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

1. Name property of the Decimal with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value of the pattern property, or else the default pattern.

Example of a decimal type with Decimal.totalDigits = 18 and Decimal.fractionDigits=17

"DecimalNumber" : {

"type": "string",

"pattern":"^[+-]?0\*(((0|[1-9][0-9]{0,17})([.]0\*)?)|([.][0-9]{1,17}0\*)|((?=[1-9.][0-9.]{1,18}0\*$)([1-9][0-9]\*)?[.][0-9]{0,17}0\*))$",

"description": "Number of objects represented as a decimal number, for example 0.75 or 45.6.",

"$anchor": "DecimalNumber"

}

## Quantity

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

1. Name property of the Quantity with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value of the pattern property, or else the default pattern.

Example of a Quantity data type with minInclusive="1" totalDigits="5" fractionDigits="0".

"Max5PositiveNumber" : {

"type": "string",

"pattern":"^[+]?0\*(((0|[1-9][0-9]{0,4})([.]0\*)?)|[.][0]+)$",

"description": "unitCode=\n\nNumber of objects represented as a positive integer.",

"$anchor": "Max5PositiveNumber"

},

Example of a Quantity data type with minInclusive="0" maxInclusive="9.9" totalDigits="2" fractionDigits="1"

"Max2Fraction1NonNegativeNumber" : {

"type": "string",

"pattern":"^[+]?0\*(((0|[1-9][0-9]{0,1})([.]0\*)?)|([.][0-9]{1,1}0\*)|((?=[1-9.][0-9.]{1,2}0\*$)([1-9][0-9]\*)?[.][0-9]{0,1}0\*))$",

"description": "unitCode=\n\nNumber of objects represented as a non negative decimal number, for example 1.1 or 8.0.",

"$anchor": "Max2Fraction1NonNegativeNumber"

},

## Rate

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

1. Name property of the Rate with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value of the pattern property, or else the default pattern.
   3. "description": the definition, along with a string representation of the baseValue property.

Example of a Rate datatype with minInclusive="0" maxInclusive="100" totalDigits="11" fractionDigits="10" baseValue="100.0".

"PercentageBoundedRate" : {

"type": "string",

"pattern":"^[+]?0\*(((0|[1-9][0-9]{0,10})([.]0\*)?)|([.][0-9]{1,10}0\*)|((?=[1-9.][0-9.]{1,11}0\*$)([1-9][0-9]\*)?[.][0-9]{0,10}0\*))$",

"description": "baseValue=100.0\n\nRate expressed as a percentage, ie, in hundredths, eg, 0.7 is 7/10 of a percent, and 7.0 is 7%.",

"$anchor": "PercentageBoundedRate"

},

## Amount

### CurrencyIdentifierSet is not empty

DataType Amount is transformed into two JSON objects, the first of which has the following characteristics:

1. Name property of the Rate with an object comprising the following properties:
2. "type" keyword set to the value "object".
3. "properties" keyword with an object comprising two objects:
   1. Object named "amt" comprising the following properties:
      1. "type" keyword set to the value "string"
      2. "pattern" keyword set to the value of the pattern property, or else the default pattern.
   2. Object named "Ccy" comprising the following properties:
      1. "$ref" with string value used to reference the Amount’s referenced CurrencyIdentifierSet, formed by concatenating:
         1. The location of the CurrencyIdentifierSet, identified with the string "#",
         2. The type property of the CurrencyIdentifierSet.
            1. Example: "$ref": "#ActiveCurrencyCode"

The second JSON object has the following characteristics:

1. Type property of the CurrencyIdentifierSet with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. If Property Pattern of CurrencyIdentifierSet’s Type is not empty: "pattern" with the value of the content of that Property.

Example of ActiveCurrencyAndAmount object with the CurrencyIdentifierSet, ActiveCurrencyCode.

"ActiveCurrencyAndAmount" : {

"type":"object",

"required":["amt","Ccy"],

"properties":{

"amt": {

"type": "string",

"pattern":"^[+]?0\*(((0|[1-9][0-9]{0,17})([.]0\*)?)|([.][0-9]{1,5}0\*)|((?=[1-9.][0-9.]{1,18}0\*$)([1-9][0-9]\*)?[.][0-9]{0,5}0\*))$"

},

"Ccy":{

"$ref":"#ActiveCurrencyCode"

}

},

"additionalProperties": false,

"description": "A number of monetary units specified in an active currency where the unit of currency is explicit and compliant with ISO 4217.\n\ncurrencyIdentifierSet=",

"$anchor": "ActiveCurrencyAndAmount"

},

"ActiveCurrencyCode": {

"type": "string",

"description": "identificationScheme=\n\nA code allocated to a currency by a Maintenance Agency under an international identification scheme as described in the latest edition of the international standard ISO 4217 \"Codes for the representation of currencies and funds\".",

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

"$anchor": "ActiveCurrencyCode"

},

### CurrencyIdentifierSet is empty

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

1. Name property of the Amount with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value of the pattern property, or else the default pattern.

Example of an amount with totalDigits = 18 and fractionDigits = 5:

"ImpliedCurrencyAndAmount" : {

"type": "string",

"pattern":"^[+]?0\*(((0|[1-9][0-9]{0,17})([.]0\*)?)|([.][0-9]{1,5}0\*)|((?=[1-9.][0-9.]{1,18}0\*$)([1-9][0-9]\*)?[.][0-9]{0,5}0\*))$",

"description": "Number of monetary units specified in a currency where the unit of currency is implied by the context and compliant with ISO 4217. The decimal separator is a dot.\nNote: a zero amount is considered a positive amount.",

"$anchor": "ImpliedCurrencyAndAmount"

},

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

Note: Patterns have been used throughout this section instead of in-built “format” keywords for two reasons: 1) Patterns in JSON Schema 2020-12 are only optionally validated and 2) the specific date-time format that should be used for validation is RFC 3339 which is a subset of ISO 8601, and doesn’t support some of the use cases that are implemented today using ISO 20022 – specifically end of day reporting using the value “24:00:00”. This wouldn’t preclude the usage of format keywords for duration or date but for consistency pattern keywords have been used throughout.

## Duration

DataType Duration is transformed into a JSON object with the following characteristics:

1. Name property of the Duration with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^-?P((([0-9]+Y([0-9]+M)?([0-9]+D)?|([0-9]+M)([0-9]+D)?|([0-9]+D))(T(([0-9]+H)([0-9]+M)?([0-9]+(**[.]**[0-9]+)?S)?|([0-9]+M)([0-9]+(**[.]**[0-9]+)?S)?|([0-9]+(**[.]**[0-9]+)?S)))?)|(T(([0-9]+H)([0-9]+M)?([0-9]+(**[.]**[0-9]+)?S)?|([0-9]+M)([0-9]+(**[.]**[0-9]+)?S)?|([0-9]+(**[.]**[0-9]+)?S))))$"

Example

"Duration": {

"type": "string",

"description": "W3C XML Schema Built-in datatype \"duration\"",

"pattern": "^-?P((([0-9]+Y([0-9]+M)?([0-9]+D)?|([0-9]+M)([0-9]+D)?|([0-9]+D))(T(([0-9]+H)([0-9]+M)?([0-9]+(**[.]**[0-9]+)?S)?|([0-9]+M)([0-9]+(**[.]**[0-9]+)?S)?|([0-9]+(**[.]**[0-9]+)?S)))?)|(T(([0-9]+H)([0-9]+M)?([0-9]+(**[.]**[0-9]+)?S)?|([0-9]+M)([0-9]+(**[.]**[0-9]+)?S)?|([0-9]+(**[.]**[0-9]+)?S))))$",

"$anchor": "Duration"

}

## DateTime

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

1. Name property of the DateTime with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword with value set to "pattern": "^-?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])T(([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](.[0-9]+)?|(24:00:00(**[.]**0+)?))(Z|(**[+]**|-)((0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

Example representation of datatype ISODateTime.

"ISODateTime": {

"type": "string",

"pattern": "^-?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])T(([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](.[0-9]+)?|(24:00:00(**[.]**0+)?))(Z|(**[+]**|-)((0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

"description": "A particular point in the progression of time defined …",

"$anchor": "ISODateTime"

}

### NormalisedDateTime

NormalisedDateTimes are transformed into JSON objects with the following properties:

1. Name of the NormalisedDateTime with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^-?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])T(([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](**[.]**[0-9]+)?|(24:00:00(**[.]**0+)?))Z$"

Example of representation of the datatype ISONormalisedDateTime:

"ISONormalisedDateTime": {

"type": "string",

"pattern": "^-?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])T(([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](**[.]**[0-9]+)?|(24:00:00(**[.]**0+)?))Z$",

"description": "A particular point in the progression of time defined …",

"$anchor": "ISODateTime"

}

## Date

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

1. Name property of the Date with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^-?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

Example representation of the datatype ISODate:

"ISODate": {

"type": "string",

"pattern": "^-?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$",

"description": "A particular point in the progression of time defined …",

"$anchor": "ISODate"

}

## Time

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

1. Name property of the Time with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^(([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](**[.]**[0-9]+)?|(24:00:00(**[.]**0+)?))(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

Example representation of the datatype ISOTime:

"ISOTime": {

"type": "string",

"pattern": "^(([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](**[.]**[0-9]+)?|(24:00:00(**[.]**0+)?))(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$",

"description": "A particular point in the progression of time defined …",

"$anchor": "ISOTime"

}

## YearMonth

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

1. Name property of the YearMonth with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^-?([1-9][0-9]{3,})-(0[1-9]|1[0-2])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

Example representation of the datatype ISOYearMonth:

"ISOYearMonth": {

"type": "string",

"pattern": "^-?([1-9][0-9]{3,})-(0[1-9]|1[0-2])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$",

"description": "Month within a particular calendar year…",

"$anchor": "ISOYearMonth"

}

## MonthDay

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

1. Name property of the MonthDay with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^--(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

Example representation of the datatype ISOMonthDay:

"ISOMonthDay": {

"type": "string",

"pattern": "^--(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$",

"description": "W3C XML Schema Built-in datatype \"gMonthDay\".",

"$anchor": "ISODateTime"

}

## Year

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

1. Name property of the Year with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^-?([1-9][0-9]{3,}|0[0-9]{3})(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

Example representation of the datatype ISOYear:

"ISOYear": {

"type": "string",

"pattern": "^-?([1-9][0-9]{3,}|0[0-9]{3})(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$",

"description": "Year represented by YYYY (ISO 8601).",

"$anchor": "ISODateTime"

}

## Month

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

1. Name property of the Month with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^--(0[1-9]|1[0-2])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

Example representation of the datatype ISOMonth:

"ISOMonth": {

"type": "string",

"pattern": "^--(0[1-9]|1[0-2])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$",

"description": "Month represented by MM (ISO 8601).",

"$anchor": "ISOMonth"

}

## Day

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

1. Name property of the Day with an object comprising the following properties:
   1. "type" keyword set to the value "string"
   2. "pattern" keyword set to the value "^---(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$"

Example representation of the datatype ISODay:

"ISODay": {

"type": "string",

"pattern": "^---(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$",

"description": "W3C XML Schema Built-in datatype \"gDay\".",

"$anchor": "ISODay"

}

Annex A   
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 bespoke features of ISO 20022 XML to be converted into JSON.

* As JSON Schema doesn't support namespaces nor prefixes, so prefixes are stripped.
* If an element has a Ccy attribute, convert into an object with “amt” and “Ccy” properties.
* If an element contains text, copy its value.
* If an element contains other elements, convert into an object   
  with a property for each group of elements with the same name
  + If the group contains just one value, convert  the value,  
    if the group contains several values, convert into an array of converted values.

* 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>

Example conversion to ISO 20022 JSON message

{

"Document": {

"ActvtyRpt": {

"RptId": {

"Id": "ARPMessage25",

"CreDtTm": "2009-09-09T11:38:00"

},

"RltdMsgRef": {

"Id": "ARRMessage24",

"CreDtTm": "2009-09-09T11:37:00"

},

"Rpt": {

"TxId": "01190799181-6940-48",

"RptdNtty": {

"BIC": "ADIABE22"

},

"RptdItm": [

{

"DtTm": "2009-09-06T08:52:00",

"Actvty": {

"MsgNm": "tsmt.020.001.02"

},

"Initr": {

"BIC": "ADIABE22"

}

},

{

"DtTm": "2009-09-06T08:54:00",

"Actvty": {

"MsgNm": "tsmt.011.001.02"

},

"Initr": {

"BIC": "SWHQBEBB"

}

}

]

}

}

}

}

Annex B Tabular Summary of JSON Schema generation  
(Informative)

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

| ISO 20022  Repository Concept | JSON Schema Generation |
| --- | --- |
| MessageDefinitionIdentifier | {businessArea}.{messageFunctionality}.{flavour}.{version} |
| MessageDefinition | {  "$schema": "https://json-schema.org/draft/2020-12/schema",  "$id":  "urn:iso:std:iso:20022:tech:json:***MessageDefinitionIdentifier***#"  "description": "***A description of this schema.***",  "type": "object",  "additionalProperties": false,  "required": ["***rootElement***"],  "properties": {  "***rootElement***": {  "$ref": "#\_***rootElement***"  }  },  "$defs": {  "\_***rootElement***": {  "type": "object",  "additionalProperties": false,  "minProperties": 1,  "required": [**"*unversioned abbreviated name***"],  "properties": {  "***unversioned abbreviated name***": {  "$ref": "#***name***"  }  },  "$anchor": "\_***rootElement***"  },  "***name***": {  "description": "***definition***",  "type": "object",  "additionalProperties": false,  "minProperties": 1,  "required": [***List of required MessageBuildingBlocks***  ***Minimum cardinality is greater than 1***],  "properties": {  ***Properties representing MessageBuildingBlocks.***  },  "$anchor": "***name***"  },  ***Further subschema for each MessageComponentType and DataType.***  }  } |
| MessageBuildingBlock  *MessageElement*  - MessageAttribute  - MessageAssociationEnd  [isComposite is true] | ***If maximum cardinality is one, then***  "***abbreviated name***": {  "description": "***name*** \n***definition***",  "$ref": "#***name of its simpleType or complexType***"  },  ***If maximum cardinality is greater than one, then***  "***abbreviated name***": {  "description": "***name*** \n***definition***",  "anyOf": [  {"$ref": "#***name of its simpleType or complexType***"},  {"type": "array",  ***If maximum cardinality is set then***  "maxItems":***cardinality.maximum*** ,  ***If minimum cardinality is set then***  "minItems":***maximum of 1 and*** ***cardinality.minimum*** ,  "items":  {"$ref": "#***name of its simpleType or complexType***"},  }  ]  }, |
| *MessageElement*  - MessageAssociationEnd  [isComposite is false] | ***If maximum cardinality is one, then***  "***abbreviated name***": {  "description": "***name*** \n***definition***",  "type": "string"  },  ***If maximum cardinality is greater than one, then***  "***abbreviated name***": {  "description": "***name*** \n***definition***",  "anyOf": [  {"type":"string"},  {"type": "array",  ***If maximum cardinality is set then***  "maxItems":***cardinality.maximum*** ,  ***If minimum cardinality is set then***  "minItems":***maximum of 1 and*** ***cardinality.minimum*** ,  "items":  {"type":"string"},  }  ]  }, |
|  |  |
| ExternalSchema | "***name***" : {  "$anchor": "***name***",  "description": "***definition***",  }, |
| MessageComponent | "***name***": {  "$anchor": "***name***",  "description": "***definition***",  "type": "object",  "additionalProperties": false,  "minProperties": 1,  "required": [  "***abbreviated names of MessageElements with minimum cardinality of one***"  ],  "properties": {  ***Properties representing MessageElements***    }  } |
| ChoiceComponent | "***name***": {  "$anchor": "***name***",  "description": "***definition***",  "type": "object",  "additionalProperties": false,  "minProperties": 1,  "maxProperties": 1,  "properties": {  ***Properties representing MessageElements***  }  } |

| ISO 20022  Repository Concept | JSON Schema Generation |
| --- | --- |
| Amount CurrencyIdentifierSet is empty | "***name***": {  "$anchor": "***name***",  "description": "***definition***",  "type": "string",  "pattern": ***numeric pattern***  } |
| Amount CurrencyIdentifierSet is not empty | "***name***": {  "$anchor": "***name***",  "description": "***definition***",  "type": "object",  "required":["Amt","ccy"],  "properties": {  "amt": {  "type": "string",  "pattern": ***numeric pattern*** },  "ccy": {  "$ref": "#***CurrencyIdentifierSet***"  }  }  } |
| Code Set with codes | "***name***": {  "$anchor": "***name***",  "description": "***definition***",  "type": "string",  "enum": [  ***For each Code:***  **"*code*",**  ]  } |
| Code Set with no codes, including ExternalCodeSet | "***name***": {  "$anchor": "***name***",  "description": "***definition***",  "type": "string",  "minLength": ***minLength or length***,  "maxLength": ***maxLength or length***,  "pattern": "***pattern***"  } |
| Other Scalar types | "***name***": {  "$anchor": "***name***",  "description": "***definition***",  "type": "string",  ***and string facets such as***  "minLength": ***minLength or length***,  "maxLength": ***maxLength or length***,  "pattern": "***pattern or default pattern for scalar type***"  } |

|  |  |
| --- | --- |
| DataType | Default pattern |
| Duration | ^-?P((([0-9]+Y([0-9]+M)?([0-9]+D)?|([0-9]+M)([0-9]+D)?|([0-9]+D))(T(([0-9]+H)([0-9]+M)?([0-9]+([.][0-9]+)?S)?|([0-9]+M)([0-9]+([.][0-9]+)?S)?|([0-9]+([.][0-9]+)?S)))?)|(T(([0-9]+H)([0-9]+M)?([0-9]+([.][0-9]+)?S)?|([0-9]+M)([0-9]+([.][0-9]+)?S)?|([0-9]+([.][0-9]+)?S))))$ |
| DateTime | ^-?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])T(([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](.[0-9]+)?|(24:00:00([.]0+)?))(Z|([+]|-)((0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$ |
| Date | ^-?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$ |
| Time | ^(([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9]([.][0-9]+)?|(24:00:00([.]0+)?))(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$ |
| Year | ^-?([1-9][0-9]{3,}|0[0-9]{3})(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$ |
| Month | ^--(0[1-9]|1[0-2])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$ |
| Day | ^---(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$ |
| YearMonth | ^-?([1-9][0-9]{3,})-(0[1-9]|1[0-2])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$ |
| MonthDay | ^--(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])(Z|([+-](0[0-9]|1[0-3]):[0-5][0-9]|14:00))?$ |
| Decimal types  Decimal, Quantity, Rate, Amount | Assumptions:   * If defined, totalDigits >= fractionDigits =>0 * If defined, totalDigits > 0   Otherwise error.  Given the following definitions:  "SignExpression" is set to:   1. "[+]?" if the minInclusive or minExclusive property is greater than or equal to 0 2. "-" if the maxInclusive is less than zero or maxExclusive property is less than or equal to 0 3. "[+-]?" otherwise   The pattern for a decimal for a JSON object shall be (using the previous definitions):  if totalDigits>0  if fractionDigits > 0  pattern = ^"SignExpression"0\*(((0|[1-9][0-9]{0,"totalDigits-1"})([.]0\*)?)|([.][0-9]{1,"fractionDigits"}0\*)|((?=[1-9.][0-9.]{1,"totalDigits"}0\*$)([1-9][0-9]\*)?[.][0-9]{0,"fractionDigits"}0\*))$  else if fractionDigits = 0  pattern = ^"SignExpression"0\*(((0|[1-9][0-9]{0,"totalDigits-1"})([.]0\*)?)|[.][0]+)$  else // fractionDigits is not defined  pattern = ^"SignExpression"0\*(((0|[1-9][0-9]{0,"totalDigits-1"})([.]0\*)?)|([.][0-9]{1,"totalDigits"}0\*)|((?=[1-9.][0-9.]{1,"totalDigits"}0\*$)([1-9][0-9]\*)?[.][0-9]{0,}0\*))$  else // totalDigits is not defined  if fractionDigits > 0  pattern = ^"SignExpression"([0-9]+([.][0-9]{0,"fractionDigits"}0\*)?|0\*[.][0-9]{1,"fractionDigits"}0\*)$  else if fractionDigits = 0  pattern = ^"SignExpression"0\*(((0|[1-9][0-9]{0,})([.]0\*)?)|[.][0]+)$  else // fractionDigits is not defined  pattern = ^"SignExpression"(([0-9]+[.]?)|(0\*[.][0-9]+)|((?=[0-9.]{2,})[0-9]\*[.][0-9]\*))$ |

Annex C Tabular Summary – 20022-4:2013 XSD generation  
(Informative)

* This tabular summary of 20022-4:2013 XSD generation 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="*{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}"/> |
| 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> |
| 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="xs:IDREF{type}\_ref"   minOccurs="{minimumOccurrence}"   maxOccurs="{maximumOccurrence}"/> |

| 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}\_SimpleType ">  <xs:attribute name="ccy" use="required" type="ActiveCurrencyCode"/>  </xs:extension>  </xs:simpleContent> </xs:complexType>  <xs:simpleType name="{name}\_SimpleType">  <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 no codes, including ExternalCodeSet | <xs:simpleType name="{code set name}">  *{ annotations }*  <xs:restriction base="xs:string">  *{ Facet restrictions - no enumerations}*  </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> |
| 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 Design Decisions  
(Informative)

1. Abbreviated Names
   1. In order to maintain compatibility with XML Schema generated using ISO 20022-4:2013, the names of Message Elements and Message Building Blocks are abbreviated. Their representation as unabbreviated names was considered, along with various conventions for snake and camel case.
2. No empty leaf nodes
   1. Every property contains at least one information item.
   2. Absent items mean the information has not been supplied, whether or not the sender has the information. That there is no value must be explicitly modelled if required.
   3. This means there is no "null" value in the modelled part of the JSON instance, with the possible exception of ExternalSchema.
   4. Arrays must have a minimum of one item.
   5. Objects must have aminimum of one property.
3. Numbers as strings
   1. Number type are represented as strings because there was a preference for validating the total digits and fraction digits in the schema.
   2. Leading and trailing zeros count towards the validation of total and fraction digits.
   3. The negative sign is prohibited if the inclusive minimum is zero or more.
4. Booleans as strings
   1. Booleans are represented as strings as with numbers being string, this enables schemaless round trip conversion with XML.
5. Format keyword
   1. The Format keyword in JSON Schema Draft 2020-12 is an annotation keyword and does not validate the contents. Default regular expressions are provided instead for temporal types.
6. Code Sets without Codes
   1. In order to maintain compatibility with XML Schema generated using ISO 20022-4:2013, code sets without codes are generated as subschema without an enum, but with pattern and lengths.
7. Cardinality
   1. Where maximum cardinality is one, the MessageElement has a single item.
   2. Where maximum cardinality is greater than one, the MessageElement has an array of items, or a single item.
   3. Where minimum cardinality is one, in a MessageComponent, the MessageElement is required.
   4. Where minimum cardinality is zero, in a MessageComponent, the MessageElement is not required.
   5. In a ChoiceComponent, the selected MessageElement must have a value of the specified type.
8. Message Components
   1. Message Component require at least one Message Element. The minProperties is set to one.
9. Choice Components
   1. Choice Component model a choice of exactly one Message Element. Instead of oneOf nor anyOf, the minProperties and maxProperties are specified so that only one property subschema is tested.
   2. Choice Component with Message Elements having minimum cardinality of 0
      1. XML Schema generated using ISO 20022-4:2013 permits the omission of XML elements representing Message Elements in a Choice Component. If more than one has minimum cardinality of 0, then because they are omitted, it cannot be determined which Message Element has no value.
      2. JSON can represent the presence of a Message Element without a value, as a name value pair whose value is null. Instead, this spec requires a single item if the maximum cardinality is one, or an array of items if the maximum cardinality is greater than one. If there is no value, the element typed by the choice component is absent.