JSON Binding Rules

Open API for FSP Interoperability Specification

JSON Binding Rules

Open API for FSP Interoperability Specification

Table of Contents

JSON Binding Rules

Open API for FSP Interoperability Specification

JSON Binding Rules

Open API for FSP Interoperability Specification

Table of Listings

1 Preface

This section contains information about how to use this document.

1.1 Conventions Used in This Document

The following conventions are used in this document to identify the specified types of information

Type of Information	Convention	Example
Elements of the API, such at resources	Boldface	**/authorization**
Variables	Italics witin angle brackets	*<ID>*
Glossary terms	Italics on first occurence; defined in *Glossary*	The purpose of the API is to enable interoperable financial transactions between a *Payer* (a payer of electronic funds in a payment transaction) located in one *FSP* (an entity that provides a digital financial service to an end user) and a *Payee* (a recipient of electronic funds in a payment transaction) located in another FSP.
Library documents	Italics	User information should, in general, not be used by API deployments; the security measures detailed in *API Signature* and *API Encryption* should be used instead.

1.2 Document Version Information

Version	Date	Change Description
1.0	2018-03-21	Initial version

2 Introduction

The purpose of this document is to express the data model used by the Open API for FSP Interoperability (hereafter cited as "the API") in the form of JSON Schema binding rules, along with the validation rules for their corresponding instances.

This document adds to and builds on the information provided in O*pen API for FSP Interoperability Specification.* The contents of the Specification are listed in Section 2.1.

The types used in the PDP API fall primarily into three categories:

- Basic data types and Formats used
- Element types
- Complex types

The various types used in *API Definition*, *Data Model* and the *Open API Specification*, as well as the JSON transformation rules to which their instances must adhere, are identified in the following sections.

2.1 Open API for FSP Interoperability Specification

The Open API for FSP Interoperability Specification includes the following documents.

2.1.1 General Documents

- *Glossary*

2.1.2 Logical Documents

- *Logical Data Model*
- *Generic Transaction Patterns*
- *Use Cases*

2.1.3 Asynchronous REST Binding Documents

- *API Definition*
- *JSON Binding Rules*
- *Scheme Rules*

2.1.4 Data Integrity, Confidentiality, and Non-Repudiation

- *PKI Best Practices*
- *Signature*
- *Encryption*

3 Keywords and Usage

The *keywords* used in the JSON Schemas and rules are derived from *JSON Schema Specification*[1]. The types of keywords used are identified in Sections 3.1, 3.2 and 3.3. As discussed in detail later, some of these keywords specify validation parameters whereas others are more descriptive, such as Metadata. The description that follows specifies details such as whether a field MUST[2] be present in the definition and whether a certain field is associated with a particular data type.

[1] The link for this is: http://json-schema.org/documentation.html

[2] MUST, MAY, OPTIONAL in this document are to be interpreted as described in [RFC2119]

3.1 Validation Keywords

This section[3] provides descriptions of the keywords used for validation in *API Definition*. Validation keywords in a schema impose requirements for successful validation of an instance.

3.1.1 maxLength

The value of this keyword MUST be a non-negative integer. A string instance is valid against this keyword if its length is less than, or equal to, the value of this keyword. The length of a string instance is defined as the number of its characters as defined by RFC 7159[RFC7159].

3.1.2 minLength

The value of this keyword MUST be a non-negative integer. A string instance is valid against this keyword if its length is greater than, or equal to, the value of this keyword. The length of a string instance is defined as the number of its characters as defined by RFC 7159[RFC7159]. Omitting this keyword has the same behavior as assigning it a value of **0**.

3.1.3 pattern

The value of this keyword MUST be a string. This string SHOULD be a valid regular expression, according to the ECMA 262 regular expression dialect. A string instance is considered valid if the regular expression matches the instance successfully. Recall: regular expressions are not implicitly anchored.

3.1.4 items

The value of *items* MUST be either a valid JSON Schema or an array of valid JSON Schemas. This keyword determines how child instances validate for arrays; it does not directly validate the immediate instance itself. If *items* is a schema, validation succeeds if all elements in the array successfully validate against that schema. If *items* is an array of schemas, validation succeeds if each element of the instance validates against the schema at the same position, if such a schema exists. Omitting this keyword has the same behavior as specifying an empty schema.

3.1.5 maxItems

The value of this keyword MUST be a non-negative integer. An array instance is valid against *maxItems* if its size is less than, or equal to, the value of this keyword.

3.1.6 minItems

The value of this keyword MUST be a non-negative integer. An array instance is valid against *minItems* if its size is greater than, or equal to, the value of this keyword. Omitting this keyword has the same behavior as a value of **0**.

3.1.7 required

The value of this keyword MUST be an array. Elements of this array (if there are any) MUST be strings and MUST be unique. An object instance is valid against this keyword if every item in the array is the name of a property in the instance. Omitting this keyword results in the same behavior as does having the array be empty.

[3] Most of the items in this section and the next one, "Metadata Keywords" are taken from: http://json-schema.org/latest/json-schema-validation.html. Some changes are made based on Open API limitations or constraints. Also, only relevant keywords are referenced.

3.1.8 properties

The value of *properties* MUST be an object. Each value of this object MUST be a valid JSON Schema. This keyword determines how child instances validate for objects; it does not directly validate the immediate instance itself. Validation succeeds if, for each name that appears in both the instance and as a name within this keyword's value, the child instance for that name successfully validates against the corresponding schema. Omitting this keyword results in the same behavior as does having an empty object.

3.1.9 enum

The value of this keyword MUST be an array. This array SHOULD have at least one element. Elements in the array SHOULD be unique. An instance validates successfully against this keyword if its value is equal to one of the elements in this keyword's array value. Elements in the array might be of any value, including null.

3.1.10 type

The value of this keyword MUST be either a string or an array. If it is an array, elements of the array MUST be strings and MUST be unique. String values MUST be one of the six primitive types (null, boolean, object, array, number, or string), or integer which matches any number with a zero-fractional part. An instance validates if and only if the instance is in any of the sets listed for this keyword.

This specification uses string type for all basic types and element types, but enforces restrictions using regular expressions as *patterns*. Complex types are of object type and contain properties that are either element or object types in turn. Array types are used to specify lists, which are currently only used as part of complex types.

3.2 Metadata Keywords

This section provides descriptions of the fields used in the JSON definitions of the types used. The description specifies whether a field MUST be present in the definition and specifies whether a certain field is associated with a primary data type. Validation keywords in a schema impose requirements for successful validation of an instance.

3.2.1 definitions

This keyword's value MUST be an object. Each member value of this object MUST be a valid JSON Schema. This keyword plays no role in validation. Its role is to provide a standardized location for schema authors to incorporate JSON Schemas into a more general schema.

3.2.2 "title" and "description"

The value of both keywords MUST be a string. Both keywords can be used to provide a user interface with information about the data produced by this user interface. A title will preferably be short, whereas a description will provide explanation about the purpose of the instance described in this schema.

3.3 Instance and $ref

Two keywords, **Instance** and **$ref** are used in either the JSON Schema definitions or the transformation rules in this document, which are described in Sections 3.3.1 and 3.3.2. **Instance** is not used in the Open API Specification; this term is used in this document to describe validation and transformation rules. **$ref** contains a URI value as a reference to other types; it is used in the Specification.

3.3.1 Instance

JSON Schema interprets documents according to a data model. A JSON value interpreted according to this data model is called an *instance*[4]. An instance has one of six primitive types, and a range of possible values depending on the type:

null: A JSON **null** production

boolean: A **true** or **false** value, from the JSON **true** or **false** productions

object: An unordered set of properties mapping a string to an instance, from the JSON **object** production

array: An ordered list of instances, from the JSON **array** production

number: An arbitrary-precision, base-10 decimal number value, from the JSON **number** production

string: A string of Unicode code points, from the JSON **string** production

Whitespace and formatting concerns are outside the scope of the JSON Schema. Since an object cannot have two properties with the same key, behavior for a JSON document that tries to define two properties (the **member** production) with the same key (the **string** production) in a single object is undefined.

3.3.2 Schema references with $ref keyword

The **$ref**[5] keyword is used to reference a schema and provides the ability to validate recursive structures through self-reference. An object schema with a **$ref** property MUST be interpreted as a **"$ref"** reference. The value of the **$ref** property MUST be a URI Reference. Resolved against the current URI base, it identifies the URI of a schema to use. All other properties in a **"$ref"** object MUST be ignored.

The URI is not a network locator, only an identifier. A schema need not be downloadable from the address if it is a network-addressable URL, and implementations SHOULD NOT assume they should perform a network operation when they encounter a network-addressable URI. A schema MUST NOT be run into an infinite loop against a schema. For example, if two schemas "#alice" and "#bob" both have an "allOf" property that refers to the other, a naive validator might get stuck in an infinite recursive loop trying to validate the instance. Schemas SHOULD NOT make use of infinite recursive nesting like this; the behavior is undefined.

It is used with the syntax **“$ref”** and is mapped to an existing definition. From the syntax, the value part of *$ref*, **#/definitions/**, indicates that the type being referenced is from the Definitions section of the Open API Specification (Typically, an Open API Specification has sections named Paths, Definitions, Responses and Parameters.). An example for this can be found in Listing 26, where the types for properties **authentication** and **authenticationValue** are provided by using references to Authenticationtype and AuthenticationValue types, respectively.

[4] The description for “Instance” keyword is taken from: http://json-schema.org/latest/json-schema-core.html#rfc.section.4.2

[5] Meaning and usage of $ref as specified here: http://json-schema.org/latest/json-schema-core.html#rfc.section.8

3.4 JSON Definitions and Examples

JSON definitions and examples are provided after most sections defining the transformation rules, where relevant. These are provided in JSON form, taken from the JSON version of the Open API Specification. The Regular Expressions in the examples may have minor differences (sometimes having an additional '\' symbol) when compared to the ones in rules and descriptions because the regular expressions in the examples are taken from the JSON version whereas the rules and descriptions are from the standard Open API (Swagger) Specification. They are provided in the relevant section as a numbered Listing. For example, Listing 1 provides the JSON representation of the definition of data type Amount.

For each of the data types, a description of the JSON Schema from the Open API Specification and (where relevant) an example of that type are provided. Following the Schema description are transformation rules that apply to an instance of that particular type.

4 Element and Basic Data Types

This section contains the definitions of and transformation rules for the basic formats and *element* types used by the API as specified in *API Definition* and *API Data Model*. These definitions are basic in the context of the API specification, but *not* the Open API Technical Specification. Often, these basic data types are derived from the basic types supported by Open API Specification standards, such as string type.

4.1 Data Type Amount

This section provides the JSON Schema definition for the data type Amount. Listing 1 provides a JSON Schema for the Amount type.

- JSON value pair with Name "**title**" and Value "**Amount**"
- JSON value pair with Name **"type"** and Value "**string**"
- JSON value pair with Name "**pattern**" and Value "**^([0]|([1-9][0-9]{0,17}))([.][0-9]{0,3[1-9])?$**"
- JSON value pair with Name "**description**" and Value **"The API data type Amount is a JSON String in a canonical format that is restricted by a regular expression for interoperability reasons. This pattern does not allow any trailing zeroes at all, but allows an amount without a minor currency unit. It also only allows four digits in the minor currency unit; a negative value is not allowed. Using more than 18 digits in the major currency unit is not allowed.**"

```
"Amount":{
    "title":"Amount",
    "type":"string",
    "pattern":"^([0]|([1-9][0-9]{0,17}))([.][0-9]{0,3}[1-9])?$",
    "maxLength":32,
    "description":"The API data type Amount is a JSON String in a canonical format that is restricted by a regular
    expression for interoperability reasons."
}
```

Listing 1 – JSON Schema for Data type Amount

The transformation rules for an instance of Amount data type are as follows:

- A given Instance of Amount type MUST be of String Type.
- The instance MUST be a match for the regular expression **^([0]|([1-9][0-9]{0,17}))([.][0-9]{0,3}[1-9])?$**.
- The length of this instance is restricted by the regular expression above as 23, with 18 digits in the major currency unit and four digits in the minor currency unit.

Valid example values for Amount type: **124.45**, **5, 5.5, 4.4444, 0.5, 0, 181818181818181818**

4.2 Data Type BinaryString

This section provides the JSON Schema definition for the data type BinaryString. Listing 2 provides a JSON Schema for the BinaryString type.

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"BinaryString"**
- JSON value pair with Name **"pattern"** and Value "**^[A-Za-z0-9-_]+[=]{0,2}$**"
- JSON value pair with Name **"description"** and Value the content of Property **description**

```
"BinaryString":{
    "title":"BinaryString",
    "type":"string",
    "pattern":"^[A-Za-z0-9-_]+[=]{0,2}$",
    "description":"The API data type BinaryString is a JSON String. The string is the base64url encoding of a
    string of raw bytes, where padding (character '=') is added at the end of the data if needed to ensure that
    the string is a multiple of 4 characters. The length restriction indicates the allowed number of characters."
}
```

Listing 2 – JSON Schema for Data type BinaryString

Section 4.2.1 gives an example for BinaryString type.

The transformation rules for an instance of BinaryString data type are as follows:

- A given Instance of BinaryString type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[A-Za-z0-9-_]+[=]{0,2}$**.

An example value for BinaryString type is
AYIBgQAAAAAAASwNGxldmVsb25lLmRmc3AxLm1lci45T2RTOF81MDdqUUZERmZlakgyOVc4bXFmNEpLMHlGTFGCAUBQU0svMS4wCk5vbmNlOiB1SXlweUYzY3pYSXBFdzVVc05TYWh3CkVuY3J5cHRpb246IG5vbmUKUGF5bWVudC1JZDogMTMyMzZhM2ItOGZhOC00MTYzLTg0NDctNGMzZWQzZGE5OGE3CgpDb250ZW50LUxlbmd0aDogMTM1CkNvbnRlbnQtVHlwZTogYXBwbGljYXRpb24vanNvbgpTZW5kZXItSWRlbnRpZmllcjogOTI4MDYzOTEKCiJ7XCJmZWVcIjowLFwidHJhbnNmZXJDb2RlXCI6XCJpbnZvaWNlXCIsXCJkZWJpdE5hbWVcIjpcImFsaWNlIGNvb3BlclwiLFwiY3JlZGl0TmFtZVwiOlwibWVyIGNoYW50XCIsXCJkZWJpdElkZW50aWZpZXJcIjpcIjkyODA2MzkxXCJ9IgA.

4.2.1 BinaryString Type IlpPacket

This section provides the JSON Schema definition for the BinaryString type IlpPacket. Listing 3 provides a JSON Schema for the IlpPacket type. The transformation rules for an instance of IlpPacket are the same as those of Data Type BinaryString.

- JSON value pair with Name **“title”** and Value **“IlpPacket”**
- JSON value pair with Name **“type”** and Value **“string”**
- JSON value pair with Name **“pattern”** and Value **“^[A-Za-z0-9-_]+[=]{0,2}$”**
- JSON value pair with Name **“minLength”** and Value **1**
- JSON value pair with Name **“pattern”** and Value **32768**
- JSON value pair with Name **“description”** and Value **“Information for recipient (transport layer information).”**

```
"IlpPacket":{
    "title":"IlpPacket",
    "type":"string",
    "pattern":"^[A-Za-z0-9-_]+[=]{0,2}$",
    “minLength”:1
    “maxLength”:32768
    "description":"Information for recipient (transport layer information)."
}
```

Listing 3 – JSON Schema for BinaryString type IlpPacket

4.3 Data Type BinaryString32

This section provides the JSON Schema definition for the data type BinaryString32. Listing 4 provides a JSON Schema for the BinaryString32 type.

- JSON value pair with Name **“type”** and Value **“string”**
- JSON value pair with Name **“title”** and Value **“BinaryString32”**
- JSON value pair with Name **“pattern”** and Value “**^[A-Za-z0-9-_]{43}$**”
- JSON value pair with Name **“description”** and Value the content of Property **description**

```
"BinaryString32":{
    "title":"BinaryString32",
    "type":"string",
    "pattern":"^[A-Za-z0-9-_]{43}$",
    "description":"The API data type BinaryString32 is a fixed size version of the API data type BinaryString,
    where the raw underlying data is always of 32 bytes. The data type BinaryString32 should not use a padding
    character as the size of the underlying data is fixed."
}
```

Listing 4 – JSON Schema for Data type BinaryString32

Section 4.3.1 gives an example for BinaryString32 type. Another example in the API of BinaryString32 type is IlpFulfilment. The transformation rules for an instance of Amount data type are as follows:

- A given Instance of BinaryString type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[A-Za-z0-9-_]{43}$**.

An example value for BinaryString32 type is:

**AYIBgQAAAAAAAASwNGxldmVsb25lLmRmc3AxLm1lci45T2RTOF81MDdqUUZERmZlakgyOVc4bXFmNEpLMHlGTFGC
AUBQU0svMS4wCk5vbmNlOiB1SXlweUYzY3pYSXBFdzVVc05TYWh3CkVuY3J5cHRpb246IG5vbmUKUGF5bWVudC1JZ
DogMTMyMzZhM2ItOGZhOC00MTYzLTg0NDctNGMzZWQzZGE5OGE3CgpDb250ZW50LUxlbmd0aDogMTM1CkNvbn
RlbnQtVHlwZTogYXBwbGljYXRpb24vanNvbgpTZW5kZXItSWRlbnRpZmllcjogOTI4MDYzOTEKCiJ7XCJmZWVcIjowLFwi
dHJhbnNmZXJDb2RlXCI6XCJpbnZvaWNlXCIsXCJkZWJpdE5hbWVcIjpcImFsaWNlIGNvb3BlclwiLFwiY3JlZGl0TmFtZVwi
OlwibWVyIGNoYW50XCIsXCJkZWJpdElkZW50aWZpZXJcIjpcIjkyODA2MzkxXCJ9IgA**.

4.3.1 BinaryString32 Type IlpCondition

This section provides the JSON Schema definition for the BinaryString32 type IlpCondition. Listing 5 provides a JSON Schema for the IlpCondition type. The transformation rules for an instance of IlpCondition are the same as those of Data Type BinaryString32.

- JSON value pair with Name **"title"** and Value **"IlpCondition"**
- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"pattern"** and Value **"^[A-Za-z0-9-_]{43}$"**
- JSON value pair with Name **"maxLength"** and Value **48**
- JSON value pair with Name **"description"** and Value **"Condition that must be attached to the transfer by the Payer."**

```
"IlpCondition":{
    "title":"IlpCondition",
    "type":"string",
    "pattern":"^[A-Za-z0-9-_]{43}$",
    "maxLength":48,
    "description":"Condition that must be attached to the transfer by the Payer."
}
```

Listing 5 – JSON Schema for BinaryString32 type IlpCondition

4.4 Data Type BopCode

This section provides the JSON Schema definition for the data type BopCode. Listing 6 provides a JSON Schema for the BopCode type.

- JSON value pair with Name **"type"** and Value "**string**"
- JSON value pair with Name "**title"** and Value **"BalanceOfPayments"**

- JSON value pair with Name "**pattern"** and Value "**^[1-9]\d{2}$**"
- JSON value pair with Name "**description"** and Value "**The API data type BopCode is a JSON String of 3 characters, consisting of digits only. Negative numbers are not allowed. A leading zero is not allowed. https://www.imf.org/external/np/sta/bopcode/."**

```
"BalanceOfPayments":{
    "title":"BalanceOfPayments",
    "type":"string",
    "pattern":"^[1-9]\\d{2}$",
    "description":"(BopCode) The API data type BopCode is a JSON String of 3 characters, consisting of digits only. Negative numbers are not allowed. A leading zero is not allowed. https://www.imf.org/external/np/sta/bopcode/"
}
```

Listing 6 – JSON Schema for Data type BopCode

The transformation rules for an instance of BopCode data type are as follows:

- A given Instance of BopCode type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[1-9]\d{2}$**.

An example value for BopCode type is **124**.

4.5 Data Type Enum

This section provides the JSON Schema definition for the data type Enum. These are generic characteristics of an Enum type, alternately known as CodeSet. Listing 8 provides an example JSON Schema for the data type Enumeration (CodeSet).

- CodeSet.Name is the name of the JSON object.
- JSON value pair with Name "**title"** and Value "**CodeSet.Name"**
- 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
- If Property description is not empty, JSON value pair with Name "**description"** and Value the content of Property description

An example for Enum/CodeSet type – "AmountType" can found in Section 4.5.1. Listing 7 lists the other Enum types defined and used in the API.

AuthenticationType, AuthorizationResponse, BulkTransferState, Currency, PartyIdentifier, PartyIdType, PartySubIdOrType , PersonalIdentifierType, TransactionInitiator, TransactionInitiatorType, TransactionRequestState, TransactionScenario, TransactionState, TransferState.

Listing 7 – List of Enum types specified and used in the API

The transformation rules for an instance of Enum data type are as follows:

- A given Instance of Enum type MUST be of String Type.
- The instance MUST be one of the values specified in the CodeSetLiteral values.

4.5.1 Enumeration AmountType

This section provides the JSON Schema definition for the Enum type AmountType. Listing 8 provides a JSON Schema for the AmountType type.

- CodeSet.Name **"AmountType"**
- JSON value pair with Name "**title"** and Value **"AmountType"**
- JSON value pair with Name **"type"** and Value "**string**"

- JSON value pair with Name **description** and Value ***"Below are the allowed values for the enumeration AmountType***

  - SEND The amount the Payer would like to send, i.e. the amount that should be withdrawn from the Payer account including any fees.

  - RECEIVE The amount the Payer would like the Payee to receive, i.e. the amount that should be sent to the receiver exclusive fees."
- JSON value pair with Name **enum** and Value the array containing the values:
  - → **SEND**
  - → **RECEIVE**

```
"AmountType":{
    "title":"AmountType",
    "type":"string",
    "enum":[
        "SEND",
        "RECEIVE"
    ],
    "description":"Below are the allowed values for the enumeration AmountType - SEND The amount the
    Payer would like to send, i.e. the amount that should be withdrawn from the Payer account including any
    fees. - RECEIVE The amount the Payer would like the Payee to receive, i.e. the amount that should be sent to
    the receiver exclusive fees."
}
```

Listing 8 – JSON Schema for Enumeration Type AmountType

The transformation rules for an instance of AmountType data type are as follows (same as those of Data Type Enum, but listing the rules here to demonstrate a valid set of literal values):

- A given Instance of AmountType type MUST be of "string" type.
- The instance MUST be one of the values defined in the set: {**"SEND", "RECEIVE"**}.

An example value for AmountType type is **"SEND"**.

4.6 Data Type Date

This section provides the JSON Schema definition for the data type Date. Listing 9 provides a JSON Schema for the Date type.

- JSON value pair with Name **"type"** and Value "**string"**
- JSON value pair with Name **"title"** and Value **"Date"**
- 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(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)$**"
- JSON value pair with Name **"description"** and Value **"The API data type Date is a JSON String in a lexical format that is restricted by a regular expression for interoperability reasons. This format is according to ISO 8601 containing a date only. A more readable version of the format is "yyyy-MM-dd", e.g. "1982-05-23" or "1987-08-05."**

```
"Date":{
    "title":"Date",
    "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(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)$",
    "description":"The API data type Date is a JSON string in a lexical format that is restricted by a regular
    expression for interoperability reasons. This format is according to ISO 8601 containing a date only. A more
    readable version of the format is "yyyy-MM-dd", e.g. "1982-05-23" or "1987-08-05"."
}
```

Listing 9 – JSON Schema for Data type Date

The transformation rules for an instance of Date data type are as follows:

- A given Instance of AmountType type MUST be of string type.
- The instance MUST be a match for the regular expression **^(?:[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(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)$**.

An example value for AmountType type is **1971-12-25**.

4.6.1 Date Type DateOfBirth

This section provides the JSON Schema definition for the Date type DateOfBirth. Listing 10 provides a JSON Schema for the DateOfBirth type. The transformation rules for an instance of DateOfBirth are the same as those of Data Type Date.

- JSON value pair with Name "**title**" and Value "**DateOfBirth (type Date)**"
- JSON value pair with Name **"type"** and Value "**string**"
- JSON value pair with Name "**description**" and Value "**Date of Birth for the Party**"
- 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(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)$**"

```
"DateOfBirth":{
    "title":"DateOfBirth (type Date)",
    "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(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)-02-29)$",
    "description":"Date of Birth for the Party."
}
```

Listing 10 – JSON Schema for Date type DateOfBirth

4.7 Data Type DateTime

The JSON Schema definition for this section provides the JSON Schema definition for the data type DateTime. Listing 11 provides a JSON Schema for the DateTime type.

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"DateTime**"
- 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(?: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{3}))(?:Z|[+-][01]\d:[0-5]\d)$"

- JSON value pair with Name **"description"** and Value **"The API data type DateTime is a JSON String in a lexical format that is restricted by a regular expression for interoperability reasons. This format is according to ISO 8601, expressed in a combined date, time and time zone format. A more readable version of the format is "yyyy-MM-ddTHH:mm:ss.SSS[-HH:MM]", e.g. "2016-05-24T08:38:08.699-04:00" or "2016-05-24T08:38:08.699Z" (where Z indicates Zulu time zone, same as UTC).**"

```
"DateTime":{
    "title":"DateTime",
    "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(?: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{3}))(?:Z|[+-][01]\\d:[0-5]\\d)$",
    "description":"The API data type DateTime is a JSON String in a lexical format that is restricted by a regular
    expression for interoperability reasons. This format is according to ISO 8601, expressed in a combined date,
    time and time zone format. A more readable version of the format is "yyyy-MM-ddTHH:mm:ss.SSS[-
    HH:MM]", e.g. \"2016-05-24T08:38:08.699-04:00\" or \"2016-05-24T08:38:08.699Z\" (where Z indicates
    Zulu time zone, same as UTC)."
}
```

Listing 11 – JSON Schema for Data type DateTime

The transformation rules for an instance of DateTime data type are as follows:

- A given Instance of type AmountType MUST be of String Type.
- The instance MUST be a match for the regular expression **^(?:[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(?: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{3}))(?:Z|[+-][01]\d:[0-5]\d)$**.

An example value for DateTime type is **2016-05-24T08:38:08.699-04:00**.

4.8 Data Type ErrorCode

This section provides the JSON Schema definition for the data type ErrorCode. Listing 12 provides a JSON Schema for the ErrorCode type.

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"ErrorCode"**
- JSON value pair with Name **"pattern"** and Value **"^[1-9]\d{3}$"**
- JSON value pair with Name **"description"** and Value **"The API data type ErrorCode is a JSON String of 4 characters, consisting of digits only. Negative numbers are not allowed. A leading zero is not allowed. Specific error number in the form <C><E><SS> where <C> is a one-digit category <E> is a one-digit error within the category <SS> is a scheme defined two-digit sub-error within the error. Please refer to x.x for the list of the possible category/error codes."**

```
"ErrorCode":{
    "title":"ErrorCode",
    "type":"string",
    "pattern":"^[1-9]\\d{3}$",
    "description":"The API data type ErrorCode is a JSON String of 4 characters, consisting of digits only.
    Negative numbers are not allowed. A leading zero is not allowed. Specific error number in the form
    <C><E><SS> where <C> is a one-digit category <E> is a one-digit error within the category <SS> is a scheme
    defined two-digit sub-error within the error. Please refer to x.x for the list of the possible category/error
    codes"
}
```

Listing 12 – JSON Schema for Data type ErrorCode

The transformation rules for an instance of ErrorCode data type are as follows:

- A given Instance of ErrorCode type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[1-9]\d{3}$**.

An example value for ErrorCode type is **5100**.

4.9 Data Type Integer

This section provides the JSON Schema definition for the data type Integer. Listing 13 provides a JSON Schema for the Integer type.

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"Integer"**
- JSON value pair with Name **"pattern"** and Value **"^[1-9]\d*$"**
- JSON value pair with Name **"description"** and Value **"The API data type Integer is a JSON String consisting of digits only. Negative numbers and leading zeroes are not allowed. The data type is always limited by a number of digits."**

```
"Integer":{
    "title":"Integer",
    "type":"string",
    "pattern":"^[1-9]\\d*$",
    "description":"The API data type Integer is a JSON String consisting of digits only. Negative numbers and
    leading zeroes are not allowed. The data type is always limited by a number of digits."
}
```

Listing 13 – JSON Schema for Data type Integer

The transformation rules for an instance of Integer data type are as follows:

- A given Instance of Integer type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[1-9]\d*$**.

An example value for Integer type is **12345**.

4.10 Data Type Latitude

This section provides the JSON Schema definition for the data type Latitude. Listing 14 provides a JSON Schema for the Latitude type.

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"Latitude"**
- JSON value pair with Name **"pattern"** and Value "**^(\+|-)?(?:90(?:(?:\.0{1,6})?)|(?:[0-9]|[1-8][0-9])(?:(?:\.[0-**

9]{1,6})?))$"

- JSON value pair with Name **"description"** and Value **"The API data type Latitude is a JSON String in a lexical format that is restricted by a regular expression for interoperability reasons."**

```
"Latitude":{
    "title":"Latitude",
    "type":"string",
    "pattern":"^(\\+|-)?(?:90(?:(?:\\.0{1,6})?)|(?:[0-9]|[1-8][0-9])(?:(?:\\.[0-9]{1,6})?))$",
    "description":"The API data type Latitude is a JSON String in a lexical format that is restricted by a regular expression for interoperability reasons."
}
```

Listing 14 – JSON Schema for Data type Latitude

The transformation rules for an instance of Latitude data type are as follows:

- A given Instance of Latitude type MUST be of String Type.
- The instance MUST be a match for the regular expression **^(\+|-)?(?:90(?:(?:\.0{1,6})?)|(?:[0-9]|[1-8][0-9])(?:(?:\.[0-9]{1,6})?))$**.

An example value for Latitude type is **+45.4215**.

4.11 Data Type Longitude

This section provides the JSON Schema definition for the data type Longitude. Listing 15 provides a JSON Schema for the Longitude type.

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **title** and Value **"Longitude"**
- If Property pattern is not empty, JSON value pair with Name **"pattern"** and Value **"^(\+|-)?(?:180(?:(?:\.0{1,6})?)|(?:[0-9]|[1-9][0-9]|1[0-7][0-9])(?:(?:\.[0-9]{1,6})?))$".**
- JSON value pair with Name **"description"** and Value **"The API data type Longitude is a JSON String in a lexical format that is restricted by a regular expression for interoperability reasons."**

```
"Longitude":{
    "title":"Longitude",
    "type":"string",
    "pattern":"^(\\+|-)?(?:180(?:(?:\\.0{1,6})?)|(?:[0-9]|[1-9][0-9]|1[0-7][0-9])(?:(?:\\.[0-9]{1,6})?))$",
    "description":"The API data type Longitude is a JSON String in a lexical format that is restricted by a regular expression for interoperability reasons."
}
```

Listing 15 – JSON Schema for Data type Longitude

The transformation rules for an instance of Longitude data type are as follows:

- A given Instance of Longitude type MUST be of String Type.
- The instance MUST be a match for the regular expression **^(\+|-)?(?:180(?:(?:\.0{1,6})?)|(?:[0-9]|[1-9][0-9]|1[0-7][0-9])(?:(?:\.[0-9]{1,6})?))$**.

An example value for Longitude type is **+75.6972**.

4.12 Data Type MerchantClassificationCode

This section provides the JSON Schema definition for the data type MerchantClassificationCode. Listing 16 provides a JSON Schema for the MerchantClassificationCode type.

- JSON value pair with **Name "type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"MerchantClassificationCode"**
- JSON value pair with Name **"pattern"** and Value "**^[\d]{1,4}$**".
- JSON value pair with Name **"description"** and Value **"A limited set of pre-defined numbers. This list would be a limited set of numbers identifying a set of popular merchant types like School Fees, Pubs and Restaurants, Groceries, etc."**

```
"MerchantClassificationCode":{
    "title":"MerchantClassificationCode",
    "type":"string",
    "pattern":"^[\\d]{1,4}$",
    "description":"A limited set of pre-defined numbers. This list would be a limited set of numbers identifying a
set of popular merchant types like School Fees, Pubs and Restaurants, Groceries, etc."
}
```

Listing 16 – JSON Schema for Data type MerchantClassificationCode

The transformation rules for an instance of MerchantClassificationCode data type are as follows:

- A given Instance of MerchantClassificationCode type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[\d]{1,4}$**.

An example value for MerchantClassificationCode type is **99**.

4.13 Data Type Name

This section provides the JSON Schema definition for the data type Name. Listing 17 provides the JSON Schema for a Name type. Section 4.13.1 contains an example of Name type – "First Name". Other Name types used in the API are "Middle Name" and "Last Name".

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"Name"**
- JSON value pair with Name **"minLength"** and Value the content of Property **minLength**
- JSON value pair with Name **"maxLength"** and Value the content of Property **maxLength**
- JSON value pair with Name **"pattern"** and Value "**^(?!\\s*$)[\\w .,'-]+$**".
- JSON value pair with Name **"description"** and Value **"The API data type Name is a JSON String, restricted by a regular expression to avoid characters which are generally not used in a name. The restriction will not allow a string consisting of whitespace only, all Unicode characters should be allowed, as well as the characters ".", "'" (apostrophe), "-", "," and " " (space). Note - In some programming languages, Unicode support needs to be specifically enabled. As an example, if Java is used the flag UNICODE_CHARACTER_CLASS needs to be enabled to allow Unicode characters."**

```
"Name":{
    "title":"Name",
    "type":"string",
    "pattern":"^(?!\\s*$)[\\w .,'-]+$",
    "description":"The API data type Name is a JSON String, restricted by a regular expression to avoid
characters which are generally not used in a name. The restriction will not allow a string consisting of
whitespace only, all Unicode characters should be allowed, as well as the characters ".", "'" (apostrophe), "-
", "," and " " (space). Note - In some programming languages, Unicode support needs to be specifically
enabled. As an example, if Java is used the flag UNICODE_CHARACTER_CLASS needs to be enabled to allow
Unicode characters."
}
```

Listing 17 – JSON Schema for Data type Name

The transformation rules for an instance of Name data type are as follows:

- A given Instance of Name type MUST be of String Type.
- The instance MUST be a match for the regular expression **^(?!\\s*$)[\\w .,'-]+$**.

An example value for Name type is **Bob**.

4.13.1 Name Type FirstName

This section provides the JSON Schema definition for the Name type FirstName. Listing 18 provides a JSON Schema for the FirstName type. The transformation rules for an instance of FirstName are the same as those of Data Type .

- JSON value pair with Name **"title"** and Value **"FirstName"**
- JSON value pair with Name **"type"** and Value "**string"**
- JSON value pair with Name **"pattern"** and Value **"^(?!\s*$)[\w .,'-]+$"**
- JSON value pair with Name **"maxLength"** and Value **128**
- JSON value pair with Name **"minLength"** and Value **1**
- JSON value pair with Name **"description"** and Value **"First name of the Party (Name type)."**

```
"FirstName":{
    "title":"FirstName",
    "type":"string",
    "minLength":1,
    "maxLength":128,
    "pattern":"^(?!\\s*$)[\\w .,'-]+$",
    "description":"First name of the Party (Name Type)."
}
```

Listing 18 – JSON Schema for Name type FirstName

4.14 Data Type OtpValue

This section provides the JSON Schema definition for the data type OtpValue. Listing 19 provides a JSON Schema for the OtpValue type.

- JSON value pair with Name **"type"** and Value "**string**"
- JSON value pair with Name "**title"** and Value **"OtpValue"**
- JSON value pair with Name "**pattern"** and Value **"^\d{3,10}$"**
- JSON value pair with Name "**description"** and Value "**The API data type OtpValue is a JSON String of 3 to 10 characters, consisting of digits only. Negative numbers are not allowed. One or more leading zeros are allowed.**"

```
"OtpValue":{
    "title":"OtpValue",
    "type":"string",
    "pattern":"^\\d{3,10}$",
    "description":"The API data type OtpValue is a JSON String of 3 to 10 characters, consisting of digits only.
    Negative numbers are not allowed. One or more leading zeros are allowed."
}
```

Listing 19 – JSON Schema for Data type OtpValue

The transformation rules for an instance of OtpValue data type are as follows:

- A given Instance of OtpValue type MUST be of String Type.
- The instance MUST be a match for the regular expression **^\d{3,10}$**.

An example value for OtpValue type is **987345**.

4.15 Data Type String

This section provides the JSON Schema definition for the data type String. Listing 21 provides an example JSON Schema for a String type.

- **String.Name** is the name of the JSON object.
- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"String.Name"**
- JSON value pair with Name **"minLength"** and Value the content of Property **"minLength"**
- JSON value pair with Name **"maxLength"** and Value the content of Property **"maxLength"**
- If Property pattern is not empty, JSON value pair with Name **pattern** and Value the content of Property **pattern**.
- JSON value pair with Name **"description"** and Value the content of Property **description**.

Below, in section 4.15.1, is an example for String type, ErrorDescription. Listing 20 has other String types specified and used in the API.

AuthenticationValue, ExtensionKey, ExtensionValue, FspId, Note, PartyName, QRCODE, RefundReason, TransactionSubScenario.

Listing 20 – String types specified and used in the API

The transformation rules for an instance of String data type are as follows:

- A given Instance of String type MUST be of String Type.
- The length of this instance MUST not be greater than the **maxLength** specified.
- The length of this instance MUST not be less than the **minLength** specified.
- The instance MUST be a match for the regular expression specified by a **pattern** property if one is specified.

An example value for String type is **Financial Services for the Poor**.

4.15.1 String Type ErrorDescription

This section provides the JSON Schema definition for the String type ErrorDescription. Listing 21 provides a JSON Schema for the ErrorDescription type.

- JSON value pair with Name "**title"** and Value "**ErrorDescription"**
- JSON value pair with Name **"type"** and Value "**ErrorDescription"**
- JSON value pair with Name **"description"** and Value "**Error description string"**
- JSON value pair with Name **"minLength"** and Value **1**
- JSON value pair with Name **"maxLength"** and Value **128**

```
"ErrorDescription":{
    "title":"ErrorDescription",
    "type":"string",
    "minLength":1,
    "maxLength":128,
    "description":"Error description string."
}
```

Listing 21 – JSON Schema for Data type ErrorDescription

The transformation rules for an instance of ErrorDescription data type are as follows (same as those of Data Type String, but listing the rules here to demonstrate a valid set of values for length properties):

- A given Instance of AmountType type MUST be of String Type.
- The length of this instance MUST not be greater than 128.
- The length of this instance MUST not be less than 1.

An example value for ErrorDescription type is **This is an error description**.

4.16 Data Type TokenCode

This section provides the JSON Schema definition for the data type TokenCode. Listing 22 provides a JSON Schema for the TokenCode type.

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and Value **"TokenCode"**
- JSON value pair with Name **"pattern"** and Value **"^[0-9a-zA-Z]{4,32}$"**
- JSON value pair with Name **"description"** and Value **"The API data type TokenCode is a JSON String between 4 and 32 characters, consisting of digits or characters from a to z (case insensitive)."**

```
"TokenCode":{
    "title":"TokenCode",
    "type":"string",
    "pattern":"^[0-9a-zA-Z]{4,32}$",
    "description":"The API data type TokenCode is a JSON String between 4 and 32 characters, consisting of
    digits or characters from a to z (case insensitive)."
}
```

Listing 22 – JSON Schema for Data type TokenCode

The transformation rules for an instance of TokenCode data type are as follows:

- A given Instance of TokenCode type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[0-9a-zA-Z]{4,32}$**.

An example value for TokenCode type is **Test-Code**.

4.16.1 TokenCode Type Code

This section provides the JSON Schema definition for the TokenCode type Code. Listing 23 provides a JSON Schema for the Code type. The transformation rules for an instance of Code are the same as those of Data Type TokenCode.

- JSON value pair with Name "**title"** and Value **"Code"**
- JSON value pair with Name **"type"** and Value **"String"**
- JSON value pair with Name "**pattern"** and Value **"^[0-9a-zA-Z]{4,32}$"**
- JSON value pair with Name "**description"** and Value **"Any code/token returned by the Payee FSP (TokenCode type)."**

```
"Code":{
    "title":"Code",
    "type":"string",
    "pattern":"^[0-9a-zA-Z]{4,32}$",
    "description":"Any code/token returned by the Payee FSP (TokenCode Type)."
}
```

Listing 23 – JSON Schema for TokenCode type Code

4.17 Data Type UndefinedEnum

This section provides the JSON Schema definition for the data type UndefinedEnum. Listing 24 provides the JSON Schema for the data type UndefinedEnum (Enumeration).

- JSON value pair with Name **"title"** and Value "**UndefinedEnum**"
- JSON value pair with Name **"type"** and Value **string**"
- JSON value pair with Name **"pattern"** and Value **"^[A-Z_]{1,32}$"**
- If Property description is not empty, JSON value pair with Name "**description"** and Value "**The API data type UndefinedEnum is a JSON String consisting of 1 to 32 uppercase characters including "_" (underscore).**"

```
"UndefinedEnum":{
    "title":"UndefinedEnum",
    "type":"string",
    "pattern":"^[A-Z_]{1,32}$",
    "description":"The API data type UndefinedEnum is a JSON String consisting of 1 to 32 uppercase characters
    including "_" (underscore)."
}
```

Listing 24 – JSON Schema for Data type UndefinedEnum

The transformation rules for an instance of UndefinedEnum data type are as follows:

- A given Instance of UndefinedEnum type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[A-Z_]{1,32}$**.

An example value for UndefinedEnum type depends on the list of values specified.

4.18 Data Type UUID

This section provides the JSON Schema definition for the data type UUID. Listing 25 provides a JSON Schema for CorrelationId which is of UUID type. Since CorrelationID is an element type in the *API Definition*, it is being used interchangeably with UUID in the Open API Specification version.

- JSON value pair with Name **"type"** and Value **"string"**
- JSON value pair with Name **"title"** and **"Value CorrelationId"**
- JSON value pair with Name **"pattern"** and Value **"^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$"**
- JSON value pair with Name **"description"** and Value **"Identifier that correlates all messages of the same sequence. The API data type UUID (Universally Unique Identifier) is a JSON String in canonical format, conforming to RFC 4122, that is restricted by a regular expression for interoperability reasons. An example of a UUID is "b51ec534-ee48-4575-b6a9-ead2955b8069". An UUID is always 36 characters long, 32 hexadecimal symbols and 4 dashes ("-")."**

```
"CorrelationId":{
    "title":"CorrelationId",
    "type":"string",
    "pattern":"^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$",
    "description":"Identifier that correlates all messages of the same sequence. The API data type UUID
    (Universally Unique Identifier) is a JSON String in canonical format, conforming to RFC 4122, that is restricted
    by a regular expression for interoperability reasons. An example of a UUID is: "b51ec534-ee48-4575-b6a9-
    ead2955b8069". An UUID is always 36 characters long, 32 hexadecimal symbols and 4 dashes ("-")."
}
```

Listing 25 – JSON Schema for Data type UUID

The transformation rules for an instance of UUID data type are as follows:

- A given Instance of UUID type MUST be of String Type.
- The instance MUST be a match for the regular expression **^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$**.

An example value for UUID type is **b51ec534-ee48-4575-b6a9-ead2955b8069**.

5 Complex Types

This section contains definitions of and transformation characteristics for complex types that are used by the API. Along with the complex types defined in the *API Definition* and *Data Model* documents, there are other complex types defined in the PDP Open API Specification based on the objects present in several requests and responses in the **body** section. These are discussed in Section 5.2.

5.1 Complex Type Definition, Transformation

This section provides the JSON Schema definition for a complex type. Listing 26 provides an example JSON Schema for a complex type, AuthenticationInfo. Transformation rules specific to a complex type are listed in Section 5.1.1.

- JSON value pair with Name **"type"** and Value **"object"**
- JSON value pair with Name **"title"** and Value **"complextype.Name"**
- If Property **description** is not empty, JSON value pair with Name "**description"** and Value the content of Property **description**.
- An array with Name **"required"** and Value the list of properties that MUST be present in an instance of complex type.
- A JSON object **properties** with the following content:
    - A list of key, value pairs with Name **key.Name** and Value of element, complex or Array type
        - JSON value pair with Name **$ref** and as Value the concatenation of **#/definitions/** with **key.Type** type.

An example for a complex type is provided under Section 5.1.1.

5.1.1 Complex Type AuthenticationInfo

This section provides the JSON Schema definition for the complex type AuthenticationInfo can be expressed as follows. Listing 26 provides the JSON Schema for AuthenticationInfo.

- JSON value pair with Name **"type"** and Value **"object"**
- JSON value pair with Name **"title"** and Value **"AuthenticationInfo"**
- JSON value pair with Name **"description"** and Value "**complex type AuthenticationInfo"**
- An array with Name **"required"** and as Value a list with elements "**authentication"** and "**authenticationValue**"
- A JSON object **properties** with the following content:
    - A JSON object **authentication** with the following content:
        - JSON value pair with Name "**$ref"** and as Value the concatenation of **#/definitions/** with Authenticationtype type.
    - A JSON object **authenticationValue** with the following content:
        - JSON value pair with Name **"$ref"** and as Value the concatenation of **#/definitions/** with AuthenticationValue type.

```
"AuthenticationInfo":{
    "title":"AuthenticationInfo",
    "type":"object",
    "description":"complex type AuthenticationInfo",
    "properties":{
        "authentication":{
            "$ref":"#/definitions/AuthenticationType",
            "description":"The type of authentication."
        },
        "authenticationValue":{
            "$ref":"#/definitions/AuthenticationValue",
            "description":"The authentication value."
        }
    },
    "required":[
        "authentication",
        "authenticationValue"
    ]
}
```

Listing 26 – JSON Schema for complex type AuthenticationInfo

The transformation rules for an instance of AuthenticationInfo complex type are as follows:

- A given Instance of AuthenticationInfo type MUST be of Object type.
- The instance MUST contain a property with name **authentication**.
- The JSON object titled **authentication** MUST be an instance of AuthenticationType type, provided in the definitions.
- The instance MUST contain a property with name **authenticationValue**.
- The JSON object titled **authenticationValue** MUST be an instance of AuthenticationValue type, provided in the definitions.

An example instance for AuthenticationInfo complex type is given in Listing 27.

```
"authenticationInfo":{
    "authentication":"OTP",
    "authenticationValue":"1234"
}
```

Listing 27 – Example instance of AuthenticationInfo complex type

5.1.2 Complex Types in the API

The examples for complex type from the API appear in Listing 28. The list includes complex types defined in *API Definition* and *Data Model* and does not contain the complex types defined only in the Open API version of the specification which capture the objects in Requests and Reponses.

ErrorInformation, Extension, ExtensionList, GeoCode, IndividualQuote, IndividualQuoteResult, IndividualTransfer, IndividualTransferResult, Money, Party, PartyComplexName, PartyIdInfo, PartyPersonalInfo, PartyResult, Refund, Transaction, TransactionType.

Listing 28 – Complex type Examples

5.2 Types of objects in Requests or Responses

This section contains the definitions and transformation characteristics of the complex types that are used in the API Specification of the PDP API to capture objects in Requests and Responses. These have the same characteristics as the complex data types discussed in the Complex Type section.

5.2.1 Complex Type AuthorizationsIDPutResponse

This section provides the JSON Schema definition for the complex type AuthorizationsIDPutResponse. Listing 29 provides a JSON Schema for complex type AuthorizationsIDPutResponse.

- JSON value pair with Name **"type"** and Value **"object"**
- JSON value pair with Name **"title"** and Value **"AuthorizationsIDPutResponse**.
- JSON value pair with Name **"description"** and Value **"PUT /authorizations/{ID} object**.
- A JSON object **"properties"** with the following content:
    - If Property "**authenticationInfo"** is present, a JSON object **"authenticationInfo"** with the following content:
        - JSON value pair with Name "**$ref"** and as Value the definition of AuthenticationInfo type, located under **definitions** as indicated by **#/definitions/**
    - A JSON object **responseType** with the following content:
        - JSON value pair with Name "**$ref"** and as Value the definition of AuthorizationResponse type, located under **definitions** as indicated by "**#/definitions/**".
- An array **required** and as Value a list with single element **"responseType"**

```
"AuthorizationsIDPutResponse":{
    "title":"AuthorizationsIDPutResponse",
    "type":"object",
    "description":"PUT/authorizations/{ID}object",
    "properties":{
        "authenticationInfo":{
            "$ref":"#/definitions/AuthenticationInfo",
            "description":"OTPorQRCodeifentered,otherwiseempty."
        },
        "responseType":{
            "$ref":"#/definitions/AuthorizationResponse",
            "description":"Enumcontainingresponseinformation;ifthecustomerenteredtheauthentication
            value,rejectedthetransaction,orrequestedaresendofheauthenticationvalue."
        }
    },
    "required":[
        "responseType"
    ]
}
```

Listing 29 – JSON Schema for complex type AuthorizationsIDPutResponse

The transformation rules for an instance of AuthorizationsIDPutResponse complex type are as follows:

- A given Instance of AuthorizationsIDPutResponse type MUST be of object type.
- The instance MUST contain a property with name **"authenticationInfo"**.
- The JSON object titled **"authenticationInfo"** MUST be an instance of AuthenticationInfo type, provided in the definitions.
- The instance MUST contain a property with name **"responseType"**.

- The JSON object titled **"responseType"** MUST be an instance of AuthorizationReponse type, provided in the definitions.

An example instance for AuthorizationsIDPutResponse complex type is given in Listing 30.

```
{
    "authenticationInfo":{
        "authentication":"OTP",
        "authenticationValue":"1234"
    },
    "responseType":"ENTERED"
}
```

Listing 30 – Example instance of AuthorizationsIDPutResponse complex type

5.2.2 Other Complex types in Requests and Responses

Other complex type examples from the API used in Requests or Responses can be found in Listing 31.

BulkQuotesPostRequest, BulkQuotesIDPutResponse, BulkTransfersPostRequest, BulkTransfersIDPutResponse, ErrorInformationObject, ErrorInformationResponse, ParticipantsTypeIDSubIDPostRequest, ParticipantsTypeIDPutResponse, ParticipantsIDPutResponse, ParticipantsPostRequest, QuotesPostRequest, QuotesIDPutResponse, TransactionRequestsIDPutResponse, TransactionsIDPutResponse, TransfersPostRequest, TransactionRequestsPostRequest, TransfersIDPutResponse.

Listing 31 – Complex type Examples for Objects in Requests and Responses in the API