FINANCIAL™ DATA EXCHANGE

API Style Guide

Version 6.0
December 2023

Legal Notice

Financial Data Exchange, LLC (FDX) is a standards body and adopts this API Style Guide for general use among industry stakeholders and specific use for published FDX APIs. Many of the terms, however, are subject to additional interpretations under prevailing laws, industry norms, and/or governmental regulations. While referencing certain laws that may be applicable, readers, users, members, or any other parties should seek legal advice of counsel relating to their particular practices and applicable laws in the jurisdictions where they do business. See FDX's complete Legal Disclaimer located at http://www.financialdataexchange.org for other applicable disclaimers.

Revision History

Document Version	Notes	Date
6.0	Corrections and updates for FDX API v6.0	**December 2023**
5.3	New format, minor edits	**June 2023**

Contents

Purpose

Analogous to a user interface style guide, this document provides conventions and standards for the FDX API to present a consistent and predictable experience for API implementers. An API that follows a comprehensive style guide can increase API usability and consistency and thereby decrease the learning curve.

Primary goals are:

- Consistency with FDX conventions for releases 4.6 – 6.0
- Adherence to REST conventions
- Adherence to the HTTP 1.1 application protocol, primarily IETF RFC 7230 (“HTTP: Message Syntax and Routing”) and IETF RFC 7231 (“HTTP: Semantics and Content”)
- Internal self-consistency
- Support of direct code generation (software development kits, etc.) for deterministic FDX provider and consumer implementations for a wide set of target programming languages and platforms

Style Guide Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" when used in ALL CAPS in this document are to be interpreted as described in IETF RFC 2119 (“Key words for use in RFCs to Indicate Requirement Levels”).

Examples are presented in `{ "font" : " monospace" }`.

Sample resource IDs are presented with strings of letters, digits and hyphen characters such as `3389dx-djkd3-kxnv0` for a `{consentId}.` This is for illustration and clarity only and does not imply actual resource IDs follow any specific format.

For brevity, ellipses (...) are used to denote the root of the API or other API resource context when the full URL is not necessary or is overly verbose. For example, `.../consents/{consentId}` is an abbreviation for `https://fdxhost.example.com/consents/{consentId}`

OpenAPI

The FDX API will be described with OpenAPI 3.1.x or later versions of the OpenAPI Specification (OAS).

The FDX API document will only be published in standard YAML format. A number of industry tools are available to convert this to JSON Schema format.

Resource Identifiers

URI path parameters MUST be defined using OAS path parameters with curly braces.

Example:

`/.../consents/{consentId}`

Resource identifiers MUST be string values. Resource identifiers MUST NOT use numeric format or be direct encodings of numeric identifiers. Resource identifiers MUST NOT be based solely on encodings of numerical incrementing numbers.

FDX APIs MUST NOT constrain the runtime format or meaning of such resource IDs (i.e. MUST NOT specify that resource IDs are UUIDs or GUIDs), only that such IDs are always unique in their container context.

REST Standards

The FDX API MUST follow the accepted definitions of RESTful (Representational State Transfer) APIs over HTTP:

- Operations in the API act on or exchange representations of resources and application state.
- URIs are the addresses of resources, otherwise known as the API's domain objects. Thus, URIs are inherently nouns and do not contain verbs.
- The HTTP methods define the verbs for the API: POST to create resources, GET to fetch resource representations, PUT to update resources, PATCH to update resources incrementally, and DELETE to delete resources. POST may also be used for operations such as actions that do not fit well into the other HTTP verb constraints.
- The client and the server are decoupled, allowing the server to define new resources and new operations independently of the client, allowing both to change independently.

In general, REST does not require meaningful URLs for resources and clients should not depend on specific URLs or URL templates. However, the FDX REST APIs MUST use clean, legible, self-descriptive URLs to facilitate application development and debugging, especially in cases where client applications persist URLs.

Syntax

Names

All names MUST use only US ASCII characters 'a' - 'z', 'A' - 'Z', '0' - '9', '-', or '_'. All names MUST start with a letter, 'a' - 'z', 'A' - 'Z'.

All names MUST end with a letter 'a' - 'z', 'A' - 'Z' or digit '0' - '9'.

Path elements MUST use kebab-case format, such as `transaction-images`, `payment-networks`. Schema names MUST use PascalCase format, such as `ConsentRecord`, `ConsentRequest`.

Property names in schemas MUST use camelCase, such as `dataClusterName` (i.e. start with lowercase), EXCEPT for names defined by other industry standards. For example, `JwtProfile.client_id` in `fdxapi.consent.yaml`.

Property names SHOULD NOT repeat the containing schema name as a component of the property name. For example, in the `Transaction` schema , use the property names `memo` and `amount` and not `transactionMemo` and `transactionAmount`. However, resource identifier properties, such as `Transaction.transactionId`, SHOULD use at least part of the schema name in the property name to differentiate the resource id from other identifiers.

Specification extensions MUST use kebab-case. Extensions defined by FDX MUST begin with `x-fdx-`. Extensions SHOULD be described by a published Semoasa format (Specification Extension Metadata for OAS Annotations). The extension definition SHOULD be at a URL derived by inserting the extension name into a common URL template.

Names SHOULD NOT use empty words like “information” or “data”. (All API messages are implicitly information or data, so these words do not convey any additional, useful information.)

Names SHOULD NOT use abbreviations, acronyms, or initialisms unless such words are widely understood and not ambiguous in the FDX context, such as “ATM”. When a name includes an abbreviation, acronym, or initialism, that name element SHOULD be treated as a word and case changes applied to it as per the type of name. For example, a property for an ATM fee would be `atmFee` and a schema for an ATM fee would be `AtmFee`.

Indentation

YAML source MUST use uniform indentation of 2 characters.

Array elements MUST be indented. For example, indent the `allOf:` and `enum:` arrays as follows:

```
InsuranceAccount:
  description: An insurance account
  allOf:
    - $ref: '#/components/schemas/Account'
    - type: object
      properties:
        accountCategory:
          type: string
          description: Category of account
          enum:
            - DEPOSIT_ACCOUNT
```

```
              - INVESTMENT_ACCOUNT
```

Instead of this

```
InsuranceAccount:
  description: An insurance account
  allOf:
  - $ref: '#/components/schemas/Account'
  - type: object
    properties:
      accountCategory:
        type: string
        description: Category of account
        enum:
        - DEPOSIT_ACCOUNT
        - INVESTMENT_ACCOUNT
```

Examples

All `example` and `examples` objects in the YAML-formatted OpenAPI document SHOULD use YAML format. Avoid embedding JSON formatted data. The YAML flow mapping `{}` or empty array `[]` MAY be used in examples to denote an empty JSON object or array, respectively. Example objects SHOULD use `examples`, per OpenAPI Specification Version 3.1.0.

> **Deprecated:** The `example` property has been deprecated in favor of the JSON Schema `examples` keyword. Use of `example` is discouraged, and later versions of this specification may remove it.

Parameter Names

Request and response header parameters SHOULD be defined following HTTP conventions, using hybrid Pascal-Kebab-Case format such as `Content-Type`, `If-Match`, and `Last-Modified`. Since HTTP header names are case-insensitive in HTTP traffic, intermediaries, and servers, the actual case used at runtime MAY vary, but the FDX API SHOULD document headers with hybrid Pascal-Kebab-Case format.

Query and path parameter names MUST use camelCase format, such as `consentId`, `dataClusterName`.

Enum Values

Enumeration values SHOULD use ALL_CAPS_SNAKE_CASE (aka SCREAMING_SNAKE_CASE), such as DEPOSIT_ACCOUNT and LOAN_ACCOUNT. Enumeration values MAY use non-ALL_CAPS_SNAKE_CASE only if the enumeration encodes specific values defined by an

external, non-FDX standard. For example, an enumeration of IANA content types may use values such as `application/json` or `image/png`.

> Note: Through version v4.6, FDX API has been inconsistent in how enum values are named. New enumeration types for future versions of FDX MUST use ALL_CAPS_SNAKE_CASE unless one of the exemptions applies. New enumeration values added to existing enumeration lists MUST follow each list's existing case style.

Enumeration values SHOULD be listed in alphabetical order. The values MAY instead be listed in order of a meaningful context such as "length of time" in `RecurringPaymentFrequency` which begins `WEEKLY`, `BIWEEKLY`, `TWICEMONTHLY`, `MONTHLY`.

Operations

Each operation MUST have a unique `operationId`. The `operationId` MUST use camelCase format and MUST contain a verb that conveys the primary action described by the service. `POST` operations whose primary purpose is to create a resource SHOULD begin with `create`. `GET` operations SHOULD begin with `get` or `search` . `PUT` operations SHOULD begin with `update`. `DELETE` operations SHOULD begin with `delete`.

All operationIds MUST be unique across the entire FDX API, and MUST be added to the `OperationId` type defined in `fdxapi.meta.yaml`.

Every operation SHOULD have exactly one tag in its `tags` property. All tags MUST be defined in the top-level `tags` section with a `name` and `description`.

Tag names SHOULD be plural English noun phrases in Title Case. Tag description values SHOULD be descriptive sentence case phrases.

Schema Objects

The FDX API schema objects MUST use the JSON schema specification version defined for OAS 3.1.0, namely JSON Schema 2020-12.

Each schema object defined in the `#/components/schemas` section MUST have a `title`.

Each schema object defined in the `#/components/schemas` section MUST have a `description`.

Each schema object defined in the `#/components/schemas` section MUST have a `type`. Each schema object with defined properties MUST be `type: object`. Each array schema MUST have `type: array` and define the `items`. Schema objects or properties which are nullable MAY instead be defined as `type: [ object, 'null' ]` or `type: [ <primitive type>, 'null' ]`.

All schema objects defined in `#/components/schemas` MUST be listed in alphabetical order in each of two sequences:

- First sequence: schema objects of `type: object` or `type: array`

- Second sequence, those of primitive types: `boolean`, `integer`, `number` or `string`.

Schema Properties

Every property in a schema SHOULD have a `description`. The description MAY be omitted if the property is defined with a schema reference object and its description is appropriate for the property.

Every property in a schema MUST have a `type`.

String properties SHOULD have `minLength` and `maxLength`.

String properties SHOULD have a valid regular expression `pattern` if the value can be described with a pattern. `minLength` and `maxLength` SHOULD be included even if the regular expression pattern imposes length constraints.

Properties which convey a discrete set of coded values MUST NOT use integer enumerations. Instead, such properties MUST use string enumerations. Integer enumerations are allowed for properties which have an underlying, intrinsic numeric value but only allow a discrete set of values.

> Explained:
>
> For example, rather than using integer codes 0, 1, 2, 3 to denote an account type, use strings such as SAV, DDA, MM, LOAN.
>
> The exception refers to properties which describe inherently numeric values. For example, consider a property such as CD term - if FDX were to only allow terms of 1, 3, 6, 9, 12, 18, 24, 30, and 36 months (hypothetically), that could be described with an integer enum.

Properties with `type: boolean` MUST NOT use names that start with the word `is`. For example, use `completed` instead of `isCompleted` .

Names of boolean properties SHOULD use past-particle form, such as `enabled` or `signed`.

Properties which convey a binary true/false value MUST use the type `boolean` and MUST NOT use other string or numeric values such as `"true"`/`"false"`, `"yes"`/`"no"`, `"on"`/`"off"`, or `0`/`1`.

Date values MUST use `format: date` and require IETF RFC 3339 YYYY-MM-DD dates only ("Date and Time on the Internet"). This SHOULD be done via defined `DateString` type with `$ref: './fdxapi.components.yaml#/components/schemas/DateString'` .

Timestamps and date/time values MUST use `format: date-time` as per JSON Schema. This SHOULD be done via defined `Timestamp` type with `$ref: './fdxapi.components.yaml#/components/schemas/Timestamp'` . The API must accept RFC 3339 `YYYY-MM-DDThh:mm:ss.sssZ` formatted UTC and `YYYY-MM-DDThh:mm:ss.sss[-+]hh:mm` (time zone offsets from UTC) formatted date-time. The API MUST always return date-time values in `YYYY-MM-DDThh:mm:ss.sssZ` formatted UTC.

Structure

The API source MUST present OAS elements in the following order, if used:

- `openapi:`
- `info:`
- `servers:`
- `tags:`
- `paths:`
- `components:`
    - `parameters:`
    - `headers:`
    - `schemas:`
    - `responses:`

Specification Extensions MAY be inserted before or after any of these elements in the document as allowed by OAS. Specification Extensions MUST NOT be inserted before the `openapi:` element.

Paths

Path variables that are common to multiple operations SHOULD be defined in the `parameters` child element of the path, not repeated in each operation.

URIs of container resources (resources that contain nested resources) SHOULD be plural nouns. Examples: `…/accounts` , `…/consents` .

URIs of resources MUST NOT use path parameters whose values are personally identifying information (PII) or sensitive data, including but not limited to: account numbers, tax id numbers, email addresses.

Path parameters MUST NOT be values which require URL or UTF encoding.

Paths MUST NOT be overloaded. That is, controller routing in FDX implementations MUST be deterministic from the URL paths. For example, the following hypothetical structure is not allowed because the second path element is overloaded -- `rules` vs `{consentId}` vs. `{consentRuleId}:`

```
paths:
  /consents/rules:
    get: ...
  /consents/{consentId}/dataClusters:
    put:
  /consents/{consentRuleId}/dataClusters:
    get:
```

Paths MUST be listed in alphabetical order. Paths MUST be listed as unquoted strings. (I.e. use `/consents/{consentId}` rather than `'/consents/{consentId}'`.)

HTTP Application Protocol

The FDX API MUST use the HTTP protocol as it is defined and described by various IETF RFCs, such as RFC 7230 and RFC 7231.

The API MUST use the HTTP verbs as per the HTTP specification. Namely:

- `PUT`, `GET`, and `DELETE` operations must be idempotent.
- `GET` operations must be safe.
- `PUT` operations MUST have the semantics of a full replacement of the resource.

Headers and other elements used in the API MUST follow the syntax and semantics as defined by HTTP, such as `Accept`, `Content-Type`, `Location`, `Last-Modified`.

HTTP Response Codes

Operations must use HTTP response codes as per HTTP. Each request MUST return the correct HTTP status codes.

- 2XX codes represent successful operations.
- 4XX codes are used for client errors - the client should invoke the operation differently, such as providing correct request body or query parameters.
- 5XX codes represent failures in the service that the client does not have control over (such as service not available, database error).

The table below shows which response codes APIs should return and under what circumstances. Note that the infrastructure (such as an API Gateway or middleware) may return some of these response codes before hitting the resource endpoint (such as 405, 406).

In particular, the API MUST NOT return a 2XX response with an error or similar property when an error occurs that is defined by a valid HTTP response code.

The following table is for reference from https://httpstatuses.com/ and is not comprehensive.

Status Code	GET	PUT	POST	PATCH	DELETE	Description
200	X	X	X	X		OK – operation succeeded without error. For PUT and POST and PATCH , 200 indicates the operation succeeded. Note: Successful DELETE operations return 204 No Content (not 200 OK)

Status Code	GET	PUT	POST	PATCH	DELETE	Description
201		X	X			Created – a new resource was created. Most common with POST but may be returned from PUT operations which create resources (rare).
202			X			Accepted – the request was accepted and processing has started, but the operation has not completed. Used for long-running actions such as business process activities, "batch" jobs.
204		X	X	X	X	No Content – the normal response when deleting a resource. There is no response body. Very rarely used for PUT and POST and PATCH if the operation does not produce a body.
303	X	X	X	X	X	See Other – the requested resource is at a different URL, as noted in the Location header. The client should retry the operation.
304	X		X			Not Modified – the resource has not been modified since the client last obtained the entity tag passed in a If-None-Match header. Also possible for POST when using GET as POST pattern.
400	X	X	X	X	X	Bad Request – the request was not well formed. For example, a query parameter is syntactically incorrect, an application/json body did not contain valid JSON, or a field has the wrong type, or the body is missing a required field. For non-syntax errors, more specific 4xx codes should be used as described below if appropriate.
401	X	X	X	X	X	Unauthorized – The request did not include proper/valid authentication credentials.

Status Code	GET	PUT	POST	PATCH	DELETE	Description
403	X	X	X	X	X	Forbidden – the user/agent does not have authorization to perform the requested operation; the user lacks the correct entitlement, or business rules mean the user is not allowed to perform the action.
404	X	X	X	X	X	Not Found – A resource identified in the request URI does not exist.
405	X	X	X	X	X	Method Not Allowed – an HTTP method is not supported by the API.
406	X	X	X	X	X	Not Acceptable – the service cannot return a representation that matches the Accept header
408	X	X	X	X	X	Request Timeout – An operation took too long to complete.
409		X	X	X		Conflict – The request cannot be completed because of a conflict. Most commonly used if a request tries to create a resource that already exists or attempts to put a resource into an inconsistent or unallowed state.
412		X	X	X		Precondition Failed – The If-Match or other precondition is not satisfied. For Conditional operations, if used.
422	X	X	X	X	X	Unprocessable Entity - the syntax of the request (body, parameters, headers) was valid, but the content is not semantically valid.
428		X	X	X		Precondition Required – The required If-Match request header is missing
5xx	X	X	X	X	X	Server side error. 5xx errors MUST NOT be returned in response to user input errors.

Data Representations

Request and Response Bodies

For operations which consume or produce `application/json` request/response bodies, the corresponding body JSON schema SHOULD be an object schema. Bodies SHOULD NOT be `type: array`, as such schemas are harder to evolve as new requirements arise during evolution of the API. (Arrays MAY be used for properties within the request/response body schemas.) The request/response body MUST be a defined schema object included by `$ref:`.

Request body schemas MUST include either `additionalProperties: false` or `unevaluatedProperties: false` so that schema validation and security tools can block API consumers (or attackers) from inadvertently or intentionally sending unexpected data.

Where applicable, properties which are contextually related to other resources SHOULD use the FDX `HateoasLink` schema.

Responses on collection resources SHOULD extend the FDX `PaginatedArray` or `SynchronizableArray` schemas.

Where applicable, the FDX API MUST use existing FDX schemas and MUST NOT create new versions of those schemas. The FDX API MAY extend the existing FDX schemas through schema composition.

Use of `oneOf`, `anyOf`

OAS 3 introduced the `oneOf` and `anyOf` JSON Schema constructs. These constructs may cause difficulty for several classes of client SDKs and code generation.

FDX releases 4.6 – 6.0 do not have any uses of `anyOf`. The Core API uses `oneOf` in several areas of `account`, `holding`, and `transaction` types.

Beginning with FDX release v6.0, all uses of `oneOf:` MUST have a defined discriminator object.

Error Response Representation

API operations which return `4xx` or `5XX` errors MUST use the FDX Error schema:

```
Error:
  title: Error
  description: >-
    An error entity which can be used at the API level for error
    responses or at the account level to indicate a problem specific
    to a particular account
  type: object
  properties:
```

```
    code:
      type: string
      description: >-
       Long term persistent identifier which can be used to trace
       error condition back to log information
    message:
      type: string
      description: >-
        End user displayable information which might help the
        customer diagnose an error
    debugMessage:
      type: string
      description: >-
        Message used to debug the root cause of the error.
        Contents should not be used in consumer's business logic.
        Can change at any time and should only be used for consumer
        to communicate with the data provider about an issue.
        Provider can include an error GUID in message for their use.
```

Future Considerations

Response Content application/problem+json

FDX is considering use of the `application/problem+json` standard defined by IETF RFC 7807, or possibly return that format if the `Accept` header includes `application/problem+json`.