FINANCIAL™ DATA EXCHANGE

API Specification

Version 6.3
November 2024

Legal Notice

Financial Data Exchange, LLC (FDX) is a standards body and adopts this API Specification Document for general use among industry stakeholders. 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.

FDX Traffic Light Protocol (TLP) Green:

Recipients may share TLP GREEN information with peers, trusted government and critical infrastructure partner organizations, and service providers with whom they have a contractual relationship, who have a need-to-know but not via publicly accessible channels. This document is a product of the FDX API and Data Structures Working Group.

Revision History

Document Version	Notes	Date
6.3	Adds a Success Responses table, a partial success response example, and updates the description of error code 703 per RFC 0310. Adds a Payment Flow Options section per RFC 0329. Updates the Pagination section to describe the use of pageKey and deprecation of Offset per RFC 0295.	**November 2024**
6.2	Updated Event Notifications Framework to specify revoked consents and sample code.	**September 2024**
6.1	Updates to links, format, grammar, code samples, consent-scope alignment, "programId" updated to "rewardProgramId". Updates to PDF-Embedded Tax Data and QR Codes sections. Includes RFC 0288 and 0299.	**June 2024**
6.0	Updated sections 2, 6.1.8, and 6.1.9.	**December 2023**
5.3.3	Update to change table for v5.3.3 from RFC 0283	**November 2023**
5.3.2	Update to change table for v5.3.2 from RFC 0277	**August 2023**
5.3.1	Updated GUID example in Interaction Tracking. Added appendix with attribution statement.	**June 2023**
5.3	Updates to Error Codes, Message Transport and Security (adoption of FAPI)	**June 2023**

Document Version	Notes	Date
5.2	Addition of clarification of key words, overview of Payment Initiation Party Maintenance.	**December 2022**
5.1	Combined version of Common, Core, Tax, and Money Movement API Specifications. YAML, request, and return parameters have been moved to the specification view.	**May 2022**

Contents

About This Document

This document provides overview information and key concepts for developers implementing the FDX API. It provides general information for all implementers as well as specific information for Core Banking, Tax, Money Movement, and other portions of the FDX API.

This is a companion document along with the FDX API YAML files, which can be accessed from your FDX developer portal. Other FDX API documentation should be referenced as well, such as the FDX API Security Model document for security protocols and our Taxonomy document that defines key industry terms.

Change Log / Release Notes: FDX API Specification

Please refer to the appendix for a complete change log history of previous versions.

Version	Date	Originator/ Contributor	Reason for Change	Ratified
6.3	November 2024	Neeraj Verma Cameron Jones Clyde Cutting	RFC 0274 - Payroll Paystub Data	Yes
		Viktors Garkavijs Clyde Cutting	RFC 0287 - Japanese Multi-Language Support	
		Bryant Sheehy Elvin Prizmic Emmett Bankston	RFC 0293 - Adding Authorized Signer to the Account Holder Relationship Entity Enums	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Francesca Loiodice Satyam Kumar Katie Volz Clyde Cutting Dinesh Katyal	RFC 0295 - Clarifying Updates for Request Parameters	
		Francesca Loiodice Satyam Kumar Clyde Cutting	RFC 0298 - Adding detail to ambiguous field descriptions	
		Kalun Ho Clyde Cutting	RFC 0301 - Add Preauthorized Payment Type to DepositTransactionType	
		Mehjabin Kapasi Clyde Cutting	RFC 0302 - Additional Elements for Student Loan Debt Data	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Niraj Thacker Sean Fitzpatrick Angel Fagundo Clyde Cutting	RFC 0303 - Enhancements to Investment Account/Contribution to include 401k Data	
		Francesca Loiodice Clyde Cutting	RFC 0309 - Clarifying Updates for Accounts Endpoint	
		Francesca Loiodice Satyam Kumar Clyde Cutting Chris Nguyen	RFC 0310 - Clarifying Updates for Error Messaging	
		Bruce Wilcox Clyde Cutting	RFC 0312 - Add IRS Form 1065 Schedule K-3, Partner's Share of Income, Deductions, Credits, etc.-International	
		Cristian Popescu Clyde Cutting	RFC 0313 - Update TelephoneNumber with mobile and primary flags	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Jake Wainwright Umer Khan Grace Hui Shen Martin Crawford	RFC 0321 - Add Institution and Transit Number (RTN) for Canadian Accounts	
		Martin Crawford	RFC 0322 - Add FHSA (First Home Savings Account) for Canadian Accounts	
		Bruce Wilcox Clyde Cutting	RFC 0325 - Uniform Tax Data Provider Server Information	
		Francesca Loiodice Satyam Kumar Olajuwon Ogunsanwo Cristian Popescu Clyde Cutting	RFC 0326 - Add fields to the Transactions Endpoint	
		Francesca Loiodice Satyam Kumar Dinesh Katyal	RFC 0329 - Add "once" to Recurring-Payments in Money Movement YAML	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Mat Belanger Dinesh Katyal Clyde Cutting	RFC 0334 - Errata to Correct DCR Scopes	

1. Introduction

The Financial Data Exchange API (FDX API) enables Data Recipients and Data Access Platforms to obtain end user's financial data from a Data Provider via a secure tokenized access, instead of utilizing screen scraping or credentials-based access. This provides greater security as the end user authenticates directly with the data provider, without any need to share credentials with third parties.

The FDX API uses tokens granted by the Data Provider as a result of end user authorization. Such tokens are generated by industry-accepted secure methods such as those based in OAuth and OpenID Connect. A full treatment of such methods may be found in the FDX API Security Model document. The tokens are used via the FDX API to gain access to the end user's financial data.

> **Note**: The FDX API uses the `oneOf` keyword in the specification to indicate that the data is valid against one of the specified schemas, per the OpenAPI specification. Some code generation tools interpret that keyword incorrectly. To work around that, implementers may consider replacing the `oneOf` keyword with `allOf` solely for an automated code generation step.

1.1 Definitions

Please refer to the Taxonomy of Permissioned Data Sharing document for definitions of key terms such as Consumer, Data Provider, and Data Recipient.

1.2 Key Words

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 Editor BCP 14 (with IETF RFC 2119 “Key words for use in RFCs to Indicate Requirement Levels” and IETF RFC 8174 “Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words“).

2. Message Transport and Security

Implementers of the FDX API MUST use the FDX API Security Profile as specified in the *FDX API Security Model* document for secure interactions. The FDX API contains data elements that are generally accepted as sensitive data. To protect such data, techniques specified in the *FDX API Security Model* document ‘Part 3: Best Practices for Sharing Sensitive Data’ MAY be followed, and if data encryption is implemented, it MUST follow ‘Part 4.1: End to End Encryption’ of the *FDX API Security Model* document.

3. Service Delivery Expectations

FDX API server responses to requests MUST start within an expected duration as dictated by the use case. Specific requirements are detailed in the *Certification Requirements for Data Providers* document.

The server MAY use HTTP 100 continue or 200 chunked encoding responses to extend the response time for large data sets. Server responses SHOULD NOT last longer than 120 seconds in order to prevent long running transactions.

4. Message Syntax

The FDX API only supports JSON syntax options, with the exception of image file responses. If data encryption is implemented, it MUST follow 'Part 4: End to End Encryption' of the *FDX API Security Model* document.

5. Residual Data

Residual data is defined as data that is no longer being used, for example if an account has been closed. Data Access Platforms SHOULD delete residual data from their systems within 180 days.

6. Protocol

The FDX API client requests data using HTTP GET, POST and PUT methods. The request SHOULD include an appropriate Request-URI. Requests MUST include an OAuth token in the authorization header. The following is a sample of the headers provided in a typical request.

```
GET /accounts HTTP/1.1
Host: example.com
Authorization: Bearer w0mcJylzCn-AfvuGdqkty2-KP48=
```

The FDX API server MUST use HTTP status codes to indicate the success or failure of a request. Response code details specific to FDX API follow. The HTTP response body MUST contain an `Error` entity for status codes other than 2XX or 409 (Duplicate Request error).

6.1 Headers

6.1.1 Transport Security

All FDX API communication MUST be secured from network sniffing with SSL/TLS. Using TLS will secure the entire request and response including any headers. It is RECOMMENDED that both the FDX API client and server use certificates. Additionally, FDX API server responses SHOULD include Cache-Control headers to prevent any caching or storing of the response:

```
Cache-Control: no-cache, no-store
```

Refer to the *FDX API Security Model* document for more information.

6.1.2 Request Authorization

The FDX API client MUST NOT identify a User to the FDX API server. Instead, the User's financial institution Login is implied via an OAuth token. The data returned by any FDX API request is limited to what the User could see using his/her Login and further limited by the scope of the OAuth token.

The FDX API client MUST use the Authorization request header with a Bearer or MAC token. Bearer tokens are RECOMMENDED although the server MAY issue MAC tokens as an alternative if the client supports it. How to obtain this token is detailed in the *FDX API Security Model* document.

```
Authorization: Bearer w0mcJylzCn-AfvuGdqkty2-KP48=
```

6.1.3 Content Negotiation

The FDX API clients and servers MUST use standard HTTP headers to negotiate transport options.

The FDX API client MUST use the Accept request header to ask for its preferred syntax. The server MUST respond with one of the requested syntaxes or with a 406 status code.

```
Accept: application/json
```

The FDX API client MUST use the Accept-Charset request header to ask for its preferred character set. The server MUST respond with the body encoded in one of the requested character sets or with a 406 status code.

```
Accept-Charset: UTF-8
```

The FDX API server MUST use the Content-Type response header to inform the client of the response syntax and charset.

```
Content-Type: application/json; charset=UTF-8
```

The FDX API client MUST use the Accept-Encoding request header to ask for its preferred compression encoding. The server MUST either respond with the body compressed with one of the requested compressions, or with the body not compressed.

```
Accept-Encoding: compress, gzip
```

The FDX API server MUST use the Content-Encoding response header to inform the client of the response encoding.

```
Content-Encoding: gzip
```

For queries, the FDX API client MAY use the If-Modified-Since request header to ask for a data response only if the data has been modified since the given date. If the server supports this header and the data has not been modified, a 304 HTTP response code MUST be returned to the client.

```
If-Modified-Since: Wed, 12 Sep 2012 06:00:00 GMT
```

6.1.4 Server Environment

The FDX API server MUST return a Date header with every response.

```
Date: Tue, 11 Sep 2012 19:43:31 GMT
```

6.1.5 Host

The Host request header field MUST specify the Internet host and port number of the resource being requested. A Host header without any trailing port information implies the default port for the service requested (e.g. "80" for an HTTP URL).

```
Host: example.com
```

6.1.6 Client Identity

The FDX API client MUST supply a User-Agent header with every request. This header SHOULD NOT be used to change the content of the response. This header is designed to only collect statistics on the products using the FDX API data service. The first token is the Data Aggregator and Data AccessPoint version. The second token is the product and product version.

```
User-Agent: Example/1.2.3 Crawler/4.3.1
```

6.1.7 Customer's Last Login Time

The FDX API client MAY optionally supply the last time the customer logged into the Data Access Platform product if this data is available.

```
CustomerLastLoggedTime: Tue, 11 Sep 2012 19:43:31 GMT
```

6.1.8 Customer's IP Address

The FDX API client MAY optionally supply the customer's IP address if this data is available or applicable, as defined by FAPI Profile 1.0 Part 1: Baseline, 6.2.2 Client provisions.

```
x-fapi-customer-ip-address: 2001:DB8::1893:25c8:1946
```

Or

```
x-fapi-customer-ip-address: 198.51.100.119
```

> **Note**: This attribute was formerly documented as `CustomerIPAdress` through version 5.3 of the FDX API. That name is now deprecated.

6.1.9 Interaction Tracking

The FDX API client MUST use the `x-fapi-interaction-id` request header to inform the server of an interaction tracing identifier, as defined by FAPI Profile 1.0 Part 1: Baseline, 6.2.2 Client provisions. An `x-fapi-interaction-id` MUST be unique to an FDX API client request as a UUID defined by IETF RFC 4122. The `x-fapi-interaction-id` allows support personnel to trace a full path of interactions through multiple sub-systems. The FDX API server MUST include the value of this header and the client identifier in any logs.

```
x-fapi-interaction-id: c770aef3-6784-41f7-8e0e-ff5f97bddb3a
```

The FDX API server MUST use the `x-fapi-interaction-id` response header to inform the client of the request's `x-fapi-interaction-id`. The echoed `x-fapi-interaction-id` value MUST be the same as the corresponding client request header value.

```
x-fapi-interaction-id: c770aef3-6784-41f7-8e0e-ff5f97bddb3a
```

> **Note**: This attribute was formerly documented as `FDX-InteractionId`, `API-InteractionId`, and `InteractionId` through version 4.5 of the FDX API. Those names are now deprecated.

6.1.10 Financial Institution Identification

If the FDX API service is provided by a service bureau which uses the same end point for multiple institutions, the FDX API client MUST provide a header that identifies the desired

financial institution. The service bureau defines this value. For example, it is often the financial institution's routing number (RTN).

```
FinancialId: 123456789
```

6.1.11 Real-time vs Batch Traffic Identifier

The FDX client SHOULD use the `FDX-API-Actor-Type` header to inform the server the source of the traffic. If the customer is present and has requested the operation, the client SHOULD send `FDX-API-Actor-Type: USER`. If the customer is not present and a batch job has triggered the operation, the client SHOULD send `FDX-API-Actor-Type: BATCH`.

The FDX server MAY use this information to route the request to the appropriate infrastructure and/or prioritize the real-time request over the batch request if congestion is observed.

6.2 Success Responses

The HTTP success response codes used by FDX, as defined and documented in IETF RFC 9110: HTTP Semantics, section “15.3. Successful 2xx”:

HTTP Status Code	Description
200	The 200 (OK) status code indicates that the request has succeeded. The content sent in a 200 response depends on the request method. The intended meaning of the response content can be summarized as: GET - the target resource POST - the status of, or results obtained from, the action PUT, DELETE - the status of the action
201	The 201 (Created) status code indicates that the request has been fulfilled and has resulted in one or more new resources being created. The 201 response content typically contains or links to the resource(s) created.
204	The 204 (No Content) status code indicates that the server has successfully fulfilled the request and that there is no additional content to send in the response.
206	The 206 (Partial Content) status code indicates that the server is successfully fulfilling a range request for the target resource by transferring one or more parts of the selected representation. At least one but not all elements were returned. For example, in response to /accounts request at least one but not all of the accounts for which the customer granted access were returned.

6.2.1 Partial Success Response Examples

There are two dimensions to partial success: **breadth** and **depth**. Each supports different scenarios and API providers have the option to implement them where they see appropriate.

Breadth-based Partial Success: From the number of records that should be returned, only some are returned. The ones that are returned are complete.

Example: *There are 100 Accounts records that should have been returned but one of the two systems of record the API provider draws from is down. The API provider only returns 50 accounts records but each record returned has all the data in each record.*

Depth-based Partial Success: The record that was requested is returned but some of the data in the record is missing. This could apply to a collection as well where all of the records are returned, but some of the data within each record is missing.

Example: *Each Account record contains 20 properties gathered from two different systems of record: ten properties each. One of the systems of record is down so each record contains only 10 properties.*

To implement a 206 partial response in addition to a 200 success, the following is an example for the /accounts endpoint:

```
  /accounts:
  # ...
      responses:
        '200':
          description: >-
            Array of accounts (DepositAccount, LoanAccount,
LineOfCreditAccount, InvestmentAccount,
            InsuranceAccount, AnnuityAccount, CommercialAccount,
or DigitalWallet)
          headers:
            x-fapi-interaction-id:
              $ref:
'./fdxapi.components.yaml#/components/headers/x-fapi-
interaction-id'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Accounts'
  # ...
        '206':
          description: >-
            Partial success array of accounts (DepositAccount,
LoanAccount, LineOfCreditAccount, InvestmentAccount,
```

```
            InsuranceAccount, AnnuityAccount, CommercialAccount,
or DigitalWallet)
          headers:
            x-fapi-interaction-id:
              $ref:
'./fdxapi.components.yaml#/components/headers/x-fapi-
interaction-id'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Accounts'
```

6.3 Errors

When FDX API servers are unable to fulfill a request, they SHOULD send an Error entity as the response payload along with an appropriate HTTP Status Code. Error messages SHOULD contain just enough information for an end user to understand what went wrong without compromising security.

Error Code	Error Message	HTTP Status Code	Description
401	Invalid Input	400	Input sent by client does not satisfy API specification
403	Forbidden, inadequate authorization	403	Forbidden due to inadequate authorization which cannot be corrected due to insufficient rights to the resource
409	Conflict with current state of target resource	409	Request parameters conflict with current state of target resource
500	Internal server error	500	Catch-all exception where request was not processed due to an internal outage/issue. Consider other more specific errors before using this error
501	Subsystem unavailable	500	A system required to process the request was not available. Request was not processed
503	Scheduled Maintenance	503	System is down for maintenance Retry-After HTTP header may be used to communicate estimated time of recovery.
601	Customer not found	404	Customer with id not found
602	Customer not authorized	403	Authenticated customer does not have the authorization to perform this action
603	Authentication failed	401	Request lacks valid authentication credentials for the target resource.
701	Account not found	404	Account with id not found
702	Invalid start or end date	400	Start or end date value is not in the ISO 8601 format

Error Code	Error Message	HTTP Status Code	Description
703	Invalid date range	400	If the start date is later than the end date or later than today's date, or if the date range is beyond what the system supports
704	Account type not supported	422	Request made for investment, loans, taxes, statements and other functions that are currently not supported. Error also covers certain account types that are not supported
705	Account is Closed	409	Operation is not supported by the closed account
800	Payee not found	404	Payee with provided ID was not found
801	Payee cannot be modified or deleted	400	Payee cannot be modified or deleted due to pending or active payments
802	Payment not found	404	A payment with provided ID was not found
803	Due date too soon	400	The due date is too soon to schedule a payment
804	Payment rejected	400	Payment could not be scheduled
805	Payment cannot be modified or cancelled	400	Payment cannot be modified or cancelled at this time. Likely due to the state that it is in.
806	Recurring payment not found	404	A recurring payment with provided ID was not found
807	Recurring payment rejected	400	Recurring payment could not be scheduled
808	Recurring payment cannot be modified or cancelled	400	Recurring payment cannot be modified or cancelled at this time. Likely due to the state that it is in.
901	Source account not found	404	Source account with id was not found
902	Source account closed	404	Source account is closed
903	Source account not authorized for transfer	401	Source account is not authorized for transfer
904	Destination account not found	404	Destination account with id was not found
905	Destination account closed	404	Destination account is closed
906	Destination account not authorized for transfer	401	Destination account is not authorized for transfer
907	Invalid amount	400	Amount for transfer is not in valid range, i.e. is not greater than 0

Error Code	Error Message	HTTP Status Code	Description
908	Duplicate transfer request	409	Duplicate transfer with id was requested
909	Transfer not available due to end of day processing	503	Transfer not available due to end of day processing
910	Insufficient funds	400	Source account does not have sufficient funds
911	Transaction limit exceeded	400	Transaction will cause a limit to be exceeded. Limit can be due to amount requested for transfer, amount requested for transfer over a certain period or number of transactions requested over a certain period
950	Transfer not found	404	Transfer with id was not found
1000	Unable to retrieve key from JWKS endpoint	500	The JWKS endpoint was either configured incorrectly, the JWKS endpoint returned an unexpected response or did not return a key that can be used for encryption
1100	Update ID not found	404	Update ID was not found or is too old. A resync is required.
1101	Reward program Id not found	404	rewardProgramId not found for getRewardProgram API
1102	Categories not found for the reward program	404	rewardProgramId not found for getRewardProgramCategories API
1103	Image Id not found for Transaction	404	No image link found for the getAccountTransactionImages API
1104	Statement id not found for Account	404	No image link found for the getAccountStatement API
1106	FdxVersion not supported or not implemented	501	Error when FdxVersion in Header is not one of those implemented at backend
1107	Data not found for request parameters	404	When API optional query parameters (filters) are supplied and the query result for the given search criteria is empty. The validation for path parameters is done separately and specific error code is given. This error code is limited only to query parameters.
1108	Statements not found for given Account	404	When account is present with no statements in it.
1200	Tax Form not Found	404	Tax Form for provided Tax Form ID was not found
1201	Tax Form Type not Supported	400	Tax Form type is not supported
1202	Tax Year not Supported	400	Tax Year is not supported

Error Code	Error Message	HTTP Status Code	Description
1203	Content Type not Supported	406	Content type is not supported
1204	Account ID is Required	400	Account ID is required for searching or validating authorization
1205	Tax Forms not yet been made available	409	Tax Forms not yet been made available for this tax year
1206	Method Not Allowed	405	Method Not Allowed
1207	Too Many Requests	429	Too Many Requests
1300	Statement is Processing	400	Statement is processing and is not yet available

7. Pagination

The FDX API supports both cursor-based and offset-based (using next record count) pagination by providing an opaque cursor in the response that can either provide the next cursor value or as a numeric string give the next record offset in the collection of records being paginated. This leverages the de facto pagination query parameters in the request:

- `limit` an optional field to specify the maximum number of elements that the consumer wishes to receive in a single call. Providers SHOULD implement reasonable defaults and maximum. Providers MAY use time-based limits where the result set might be less than the limit and each page could contain a different number of elements. The provider MUST NOT return more records than specified by `limit`.
- `pageKey` an optional field to specify an opaque or offset cursor used to retrieve the next or prior page of the requested data. It could be a numeric string or an opaque string. If it is a numeric string it SHOULD use a 0-based index.

The response MUST be a "mixin" with `PaginatedArray`. The response MUST contain a `links` property of type `PageMetadataLinks` and a `page` property of type `PageMetadata`. The `PageMetadata` fields returned are as follows:

- `page.nextPageKey` is an opaque or offset cursor used to retrieve the next page of results, if any (in v6.3, `nextPageKey` replaced the now deprecated field `nextOffset`)
- `page.previousPageKey` is an opaque or offset cursor used to retrieve the previous page of results, if any (in v6.3 `previousPageKey` replaced the now deprecated field `prevOffset`)
- `page.totalElements` contains the number of total elements matching the query

- `links.next.href` represents the URL used to retrieve the next page, if any. This URL SHOULD contain the original parameters of the search query such as `startDate` and `endDate` if applicable
- `links.prev.href` represents the URL used to retrieve the previous page, if any. This URL SHOULD contain the original parameters of the search query such as `startDate` and `endDate` if applicable

The client SHOULD look for `page.nextPageKey` or `links.next.href` to determine if additional API calls are required to retrieve the data requested. The client SHOULD continue to make API calls until next page entities are no longer returned in the response.

7.1 Sample implementation

```
GET /accounts?limit=10
```

7.1.1 Initial Response

```
{
  "page":{
    "nextPageKey": "nextPageKey@10",
    "totalElements": 100
  },
  "links":{
    "next": "href":"/accounts?pageKey=nextPageKey@10&limit=10"}
  },
  "accounts":[
    {"accountId": "1", "nickname": "Account 1"},
    {"accountId": "2", "nickname": "Account 2"},
    {"accountId": "3", "nickname": "Account 3"},
    {"accountId": "4", "nickname": "Account 4"},
    {"accountId": "5", "nickname": "Account 5"},
    {"accountId": "6", "nickname": "Account 6"},
    {"accountId": "7", "nickname": "Account 7"},
    {"accountId": "8", "nickname": "Account 8"},
    {"accountId": "9", "nickname": "Account 9"},
    {"accountId": "10", "nickname": "Account 10"}
  ]
}
```

7.1.2 Final Response

```
{
```

```
  "page":{
    "prevPageKey": "prevPageKey@81",
    "totalElements": 100
  },
  "links":{
    "prev": {"href":"/accounts?pageKey=prevPageKey@81&limit=10"}
  },
  "accounts":[
    {"accountId": "91", "nickname": "Account 91"},
    {"accountId": "92", "nickname": "Account 92"},
    {"accountId": "93", "nickname": "Account 93"},
    {"accountId": "94", "nickname": "Account 94"},
    {"accountId": "95", "nickname": "Account 95"},
    {"accountId": "96", "nickname": "Account 96"},
    {"accountId": "97", "nickname": "Account 97"},
    {"accountId": "98", "nickname": "Account 98"},
    {"accountId": "99", "nickname": "Account 99"},
    {"accountId": "100", "nickname": "Account 100"},
  ]
}
```

Since this is the last page, `page.nextPageKey` and `links.next.href` are not returned in the response. This indicates to the client that no further pages are available.

The response attribute `links.prev` identifies the previous page url and `pageKey` parameter.

8. Versioning

APIs SHOULD be designed for change in order to expose new behavior or data.

Non-breaking (backward compatible) service changes are denoted by minor versions (e.g. v1 moves to v1.1) while breaking changes (not backward compatible) are denoted by a major version (e.g. v1 moves to v2). When minor versions of services are released, they are backward-compatible so that the services are released *in place of* the existing service. When major versions of services are released, there are multiple versions of the same service available at the same time.

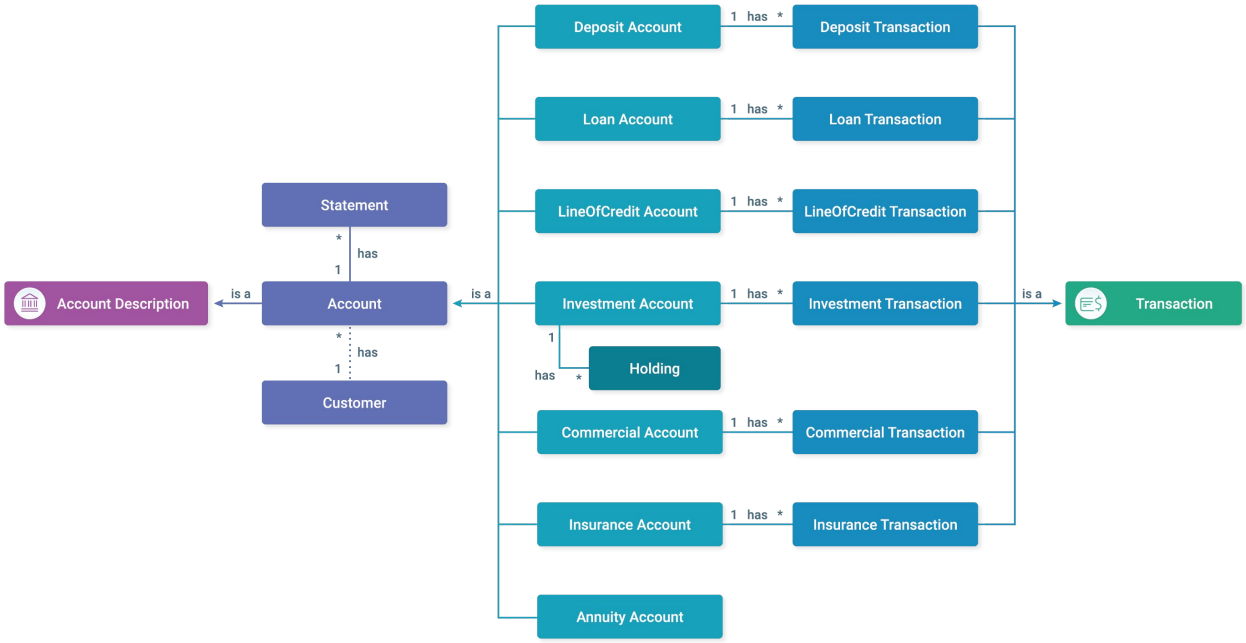
The reason for releasing a new service version is due to a change in a major piece of service functionality. Thus consumers of the previous version should be motivated to adopt the new version. It is understood that rapid movement to the new service will not always occur.

Therefore, it is RECOMMENDED that no more than three approved major versions of a service exist at any one time. When a fourth major version is released into production, the oldest

version SHOULD be considered deprecated and scheduled for retirement within a period not to exceed 12 months.

9. Logical Data Model

The FDX API supports multiple financial domains. The following image shows the account and transactions data model.



9.1 Entity Identifier

The User entity is not expressed in FDX API messages.

The Login entity has an identifier unique to its owning Institution. The Login identifier is usually the username part of a username / password login. The Login surrogate identifier is the OAuth token obtained from the Financial Institution.

The Account entity has an identifier that is unique to the owning Institution.

The Transaction entity has an identifier that is unique to the owning Account and is usually unique to the owning Institution.

The entity identifier (or surrogate identifier) is REQUIRED when transmitting the entities and is used to relate the entities.

The FDX API identifier properties defined referencing shared `Identifier` type have a maximum of 256 characters. (IBAN account identifiers are 31 characters, ACH has 9 digits for routing, 17 digits for account number, and SHA-256 generates an almost-unique 256-bit (32-byte) signature for a given text.)

9.2 Surrogate Identifier

OAuth creates a surrogate identifier for a Login. An FDX API server MUST NOT expose the financial institution's principal identifier of the Login. To limit the exposure of personally identifiable information, the other identifiers transmitted by the FDX API server SHOULD be surrogate identifiers. Surrogates MUST provide the same uniqueness constraints on the entity relationships as described above. Any surrogate identifiers MUST be long-term persistent.

If the Data Provider's account identifier is considered confidential, a surrogate identifier SHOULD be used (AccountId SHOULD NOT equal AccountNumber).

10. Rewards Programs Overview

The /reward-programs endpoint returns reward program and membership information for the currently authenticated user.

10.1 Rewards

10.1.1 Endpoint

GET /reward-programs

10.1.2 Sample Response

The following is an example response of reward program data for an authenticated user.


```
{
  "programName": "Marriott Bonvoy",
  "rewardProgramId": "4FRCCQvGW0GZEMtsOQWlkQ",
  "programUrl": "https://www.marriott.com/loyalty.mi",
  "memberships": [
    {
      "accountIds": [
        "af0f8e58-9649-4c29-bab2-0295d522cd6f",
        "e75e31eb-bf04-4d87-9f20-4554f63a639e"
      ],
      "businessOrConsumer": "CONSUMER",
      "customerId": "kBA5C3d7cBK9DuRngsQRwt6Ydo80bjYDR7n",
      "memberId": "5ee28848b4f242a6b7a41e0daa03a824",
      "memberNumber": "1783949940",
      "memberTier": "Gold",
      "balances": [
        {
          "name": "Points",
          "type": "POINTS",
```

```
          "balance": "900",
          "accruedYtd": "1000",
          "redeemedYtd": "200",
          "qualifying": false
        },
        {
          "name": "Promotional",
          "type": "POINTS",
          "balance": 900,
          "accruedYtd": 1000,
          "redeemedYtd": 200,
          "qualifying": false
        }
      ]
    }
  ]
}
```

10.1.3 Accounts linked to a single reward program

Customers may hold multiple credit cards with a single institution where each credit card accrues rewards for the same reward program. For example, a customer could have an AMEX Marriott Bonvoy card and an AMEX Marriott Bonvoy Business card, both linked to their personal Marriott Bonvoy account. This specific scenario is popular with Marriott Bonvoy members because they can receive 15 bonus elite nights for each card (30 elite nights total).

In this scenario a RewardProgram response would contain two account ids.

```
"accountIds": [
  "af0f8e58-9649-4c29-bab2-0295d522cd6f",
  "e75e31eb-bf04-4d87-9f20-4554f63a639e"
]
```

10.1.4 Balances

Each RewardProgram response contains an array of balances. If a reward program accrues points and cash back, both can be tracked separately like the example below.

```
"balances": [
  {
    "name": "Points",
    "type": "POINTS",
    "balance": "900",
```

```
    "accruedYtd": "1000",
    "redeemedYtd": "200",
    "qualifying": false
  },
  {
    "name": "Bonus Cash",
    "type": "CASHBACK",
    "balance": 100.0,
    "accruedYtd": 500.0,
    "redeemedYtd": 400.0,
    "qualifying": false
  }
]
```

Balance Fields

Field	Description
name	String used to label balances. Examples: SkyMiles, Bonvoy Points, Avios, Stars, ThankYou Rewards
type	The general type of the balance indicated by one of these values: POINTS, MILES, CASHBACK
balance	The total balance that is available to be redeemed
accruedYtd	The year to date accrued balance
redeemedYtd	The year to date balance redeemed
qualifying	Default: false. A value of true means the balance is used for qualifying events like advancement to a tier in tiered reward programs

10.1.5 Qualifying Balances

Balances with `qualifying` set to true indicate that those balances are used by the reward program to determine a member's progress toward the next status in a tiered reward program. Qualifying balances are not redeemable balances.

Using Delta Airlines as an example, Delta tracks multiple balances.

1. SkyMiles: A balance that's redeemable for awards on Delta Airlines

2. MQMs: The number of Medallion Qualifying Miles a member needs to be considered for the next level of status.

3. MQSs: The number of Medallion Qualifying Segments a member needs to fly to be considered for the next level of status.

```
"balances": [
  {
    "name": "SkyMiles",
```

```
    "type": "MILES",
    "balance": "6900",
    "accruedYtd": "1000",
    "redeemedYtd": "200",
    "qualifying": false
  },
  {
    "name": "MQMs",
    "type": "POINTS",
    "balance": 0,
    "accruedYtd": 1000,
    "redeemedYtd": 0,
    "qualifying": true
  },
  {
    "name": "MQSs",
    "type": "POINTS",
    "balance": 0,
    "accruedYtd": 3,
    "redeemedYtd": 0,
    "qualifying": true
  }
]
```

10.1.6 Customer

The customerId field associates a customer from the `/customers` endpoint.

```
"customerId": "kBA5C3d7cBK9DuRngsQRwtDR7n"
```

10.1.7 Membership

A customer's participation in a reward program is found these three fields:

```
  "memberId": "5ee28848b4f242a6b7a41e0daa03a824",
  "memberNumber": "1783949940",
  "memberTier": "Gold",
```

Field	Description
memberId	The long term persistent id of the reward program member
memberNumber	The member's real reward program membership number
memberTier	The customer's tier if the reward program is tiered

10.1.8 Reward Program

Details related to a reward program are found in these three fields:

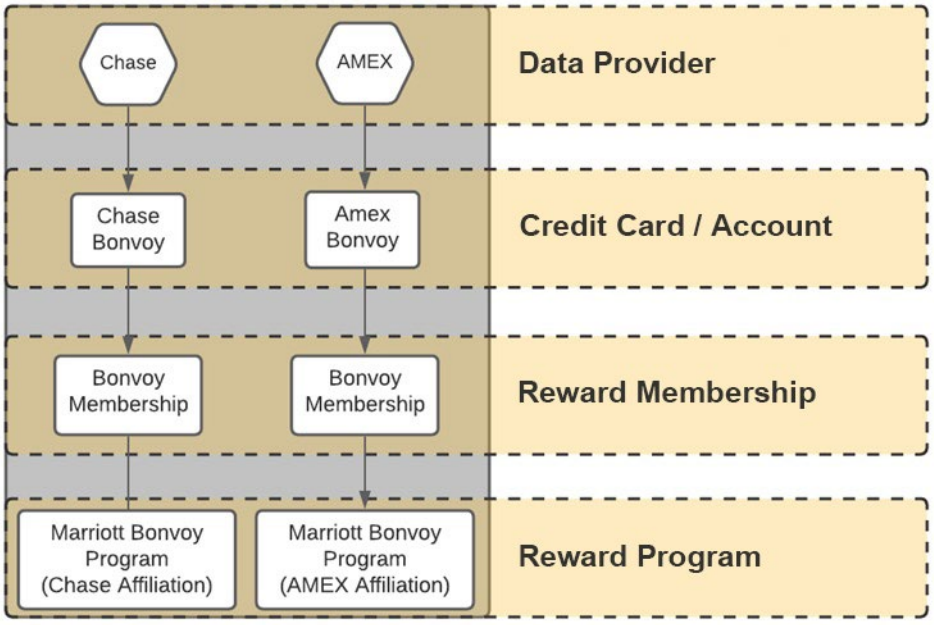
```
"rewardProgramId": "4FRCCQvGW0GZEMtsOQWlkQ",
"programName": "Marriott Bonvoy",
"programUrl": "https://www.marriott.com/loyalty.mi",
```

Field	Description
rewardProgramId	The long term persistent id of the reward program
programName	The reward program's name. Example: Delta Skymiles, Asia Miles, Citi ThankYou Rewards
programUrl	The URL of the reward program's website

10.2 Reward Program Affiliations

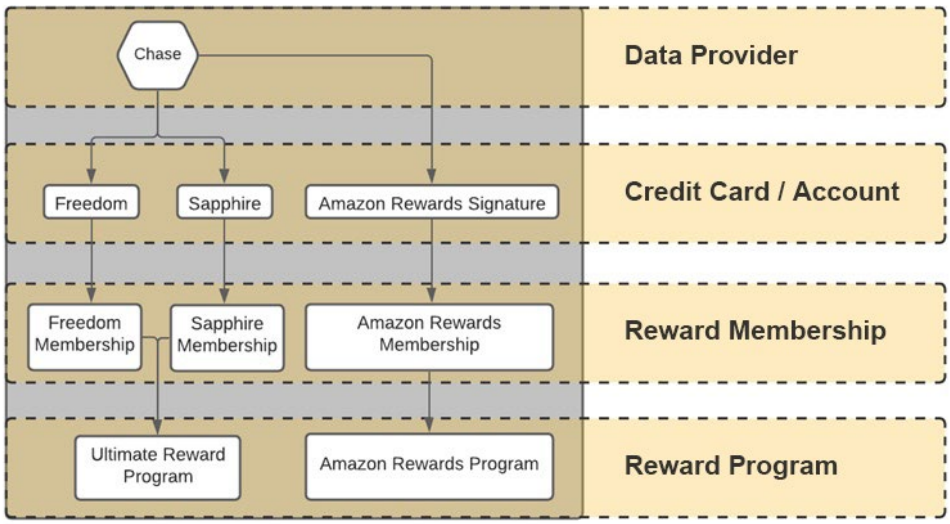
10.2.1 Example 1

The following diagram shows that a single reward program may partner with multiple financial institutions. In these cases, even though the reward program is the same, each data provider will send their own unique `rewardProgramId`.



10.2.2 Example 2

The following diagram shows how a data provider can also be the reward program owner. In this case Chase is the data provider and the owner of the Chase Ultimate Rewards program.



10.3 Multiple reward programs and memberships

The authenticated user may participate in multiple rewards programs associated with multiple credit cards at a single institution. For example, if a user had a Starbucks Rewards Visa and a Marriott Bonvoy Visa with Chase Bank, a call to `/reward-programs` would return two reward programs like this:


```
{
  "rewardPrograms": [
    {
      "programName": "Marriott Bonvoy",
      "rewardProgramId": "4FRCCQvGW0GZEMtsOQWlkQ",
      "programUrl": "https://www.marriott.com/loyalty.mi",
      "memberships": [
        {
          "accountIds": [
              "af0f8e58-9649-4c29-bab2-0295d522cd6f",
              "e75e31eb-bf04-4d87-9f20-4554f63a639e"
          ],
          "businessOrConsumer": "CONSUMER",
          "customerId": "kBA5C3d7cBK9DuRngsQRwt6Ydo80bjYDR7n",
          "memberId": "5ee28848b4f242a6b7a41e0daa03a824",
          "memberNumber": "1783949940",
```

```
          "memberTier": "Gold",
          "balances": [
            {
              "name": "Points",
              "type": "POINTS",
              "balance": "900",
              "accruedYtd": "1000",
              "redeemedYtd": "200",
              "qualifying": false
            },
            {
              "name": "Promotional",
              "type": "POINTS",
              "balance": 900,
              "accrued": 1000,
              "redeemed": 200,
              "qualifying": false
            }
          ]
        }
      ]
    },
    {
      "programName": "Starbucks Rewards",
      "rewardProgramId": "iqOtPUEYb0Go6SCL8As4fQ",
      "programUrl": "https://www.starbucks.com/rewards",
      "memberships": [
        {
          "accountIds": [
              "89cf3262-ff38-4f6a-afbc-aafc50cac751"
          ],
          "businessOrConsumer": "CONSUMER",
          "customerId": "kBA5C3d7cBK9DuRngsQRwt6Ydo80bjYDR7n",
          "memberId": "95c1aeacd85e4783950a9c2d6e76efa9",
          "memberNumber": "7417973194",
          "balances": [
            {
              "name": "Stars",
              "type": "POINTS",
              "balance": 900,
              "accrued": 1000,
              "redeemed": 200,
```

```
              "qualifying": false
            }
          ]
        }
      ]
    }
  ]
}
```

10.4 Reward Program Categories

Some reward programs calculate points based on a group of like transaction categories. The composition of these categories varies between providers, so multiple programs that have a dining category will vary in what transactions they recognize as dining purchases.

If a reward program has partnered with multiple financial institutions, categories and their ids will be unique to each financial institution.

10.4.1 Endpoint

```
GET /reward-programs/{rewardProgramId}/categories
```

10.4.2 Sample Response

```
{
  "categories": [
    {
      "categoryName": "Airline tickets when purchased directly
with the airline",
      "categoryId": "10001",
      "multiplier": 1,
      "description": "Only purchases for airline tickets made
directly with the airline will qualify. Other air travel-related
purchases will not qualify; for example, in-flight purchases,
the purchasing of airline miles or points, non-ticket purchases
made within the airport, and airline tickets purchased through
travel agencies, discount travel sites, vacation clubs, tour
operators, or tickets booked as part of a travel package offered
by non-airline merchants."
    },
    {
      "categoryName": "Car rental agencies",
      "categoryId": "10002",
      "multiplier": 2,
      "description": "Merchants in the car rental agencies
category rent vehicles for the purpose of driver and passenger
```

```
transportation. Car rentals not booked directly with the car
rental agency will not qualify; for example, car rentals
purchased through travel agencies, discount travel sites,
vacation clubs, tour operators, or as part of a package offered
by merchants such as cruise lines and railways. Merchants that
provide RV rentals or vehicle rentals for the purpose of hauling
are not included in this category."
    },
    {
      "categoryName": "Department stores",
      "categoryId": "10003",
      "multiplier": 2,
      "description": "Merchants in the department store category
generally sell a line of apparel, home furnishings, furniture,
electronics, cosmetics, housewares, and major household
appliances. These stores have departments that usually have
their own separate check-out counters. Purchases made on a
merchant's website are included. Supercenters, discount stores,
or specialty stores (i.e. stores that sell primarily one line of
products such as shoe stores, pet stores, electronics stores,
clothing stores) are not included in this category. Department
Store gift card purchases outside of the specific Department
Store are excluded and will not earn bonus rewards."
    },
    {
      "categoryName": "Dining/Restaurants",
      "categoryId": "10004",
      "multiplier": 5,
      "description": "Restaurant category merchants' primary
business is sit-down or eat-in dining, including fast food
restaurants and fine dining establishments. Merchants that sell
food and drinks located within facilities such as sports
stadiums, hotels and casinos, theme parks, grocery and
department stores will not be included in this category unless
the merchant has set up such purchases to be classified in a
restaurant category. Purchases made at bakeries, caterers, or
gift card merchants are not included in the category. Delivery
service merchants will be included if they classify as a
restaurant merchant."
    },
    {
      "categoryName": "Drugstores",
      "categoryId": "10005",
      "multiplier": 2,
      "description": "Merchants in the drugstores category
specialize in selling prescription drugs and over-the-counter
medicines, supplements and various health-related items. These
```

```
merchants may also sell cosmetics, toiletries, greeting cards,
and various household items such as cleaning supplies and
packaged foods and drinks.  Some merchants that sell a wide
variety of goods including these items, and which may contain an
onsite pharmacy, for example, warehouse clubs, discount stores,
or grocery stores, are not included in this category."
    },
    {
      "categoryName": "Fitness club and gym memberships",
      "categoryId": "10006",
      "multiplier": 2,
      "description": "Merchants in the fitness club and gym
memberships category include health clubs, exercise, or athletic
facilities requiring membership and offering access to services
related to physical fitness, such as fitness clubs, fitness
centers, fitness studios, gyms, aerobics, cardio fitness and
other services such as yoga and CrossFit(TM) training.
Merchants that specialize in offering personalized or
therapeutic services such as massage therapy, dietary and weight
management counseling and personal training are not included in
this category.  In addition, some merchants that sell a wide
variety of general goods, which may include fitness or athletic
apparel, sporting goods, dietary food, health food or similar
supplements are not included in this category. Also, certain
lodging, hotel, motel, resort and central reservation services
offering access to third party facilities that include fitness
clubs or gyms are not included in this category unless they
classify as a fitness club or gym membership merchant."
    },
    {
      "categoryName": "Gas stations",
      "categoryId": "10007",
      "multiplier": 5,
      "description": "Merchants in the gas stations category
sell automotive gasoline that can be paid for either at the pump
or inside the station, and may or may not sell other goods or
services at their location. Merchants that do not specialize in
selling automotive gasoline are not included in this category;
for example, truck stops, boat marinas, oil and propane
distributors, and home heating companies."
    },
    {
      "categoryName": "Grocery stores",
      "categoryId": "10008",
      "multiplier": 5,
      "description": "Merchants in the grocery stores category
include supermarkets, merchants that offer a full service
```

```
grocery line of merchandise including a deli and bakery as well
as smaller grocery stores. Some merchants that sell grocery
items are not included in this category; for example, larger
stores that sell a wide variety of goods and groceries, such as
warehouse clubs, discount stores and some smaller merchants such
as drugstores, and merchants that specialize in only a few
grocery items. Purchases made at gas stations from merchants who
also operate grocery stores are not included in this category.
Delivery service merchants will be included if they classify as
a grocery store merchant."
    },
    {
      "categoryName": "Home improvement stores",
      "categoryId": "10009",
      "multiplier": 2,
      "description": "Merchants in the home improvement stores
category specialize in selling a variety of home improvement
supplies, from larger home improvement stores to smaller
hardware stores. Merchants that sell a wide variety of general
goods which may include home improvement supplies, for example,
warehouse clubs, discount stores, or grocery stores, are not
included in this category. Also, merchants that specialize in
home furnishings, garden and landscaping supplies are not
included."
    },
    {
      "categoryName": "Hotel accommodations when purchased
directly with the hotel",
      "categoryId": "10010",
      "multiplier": 1,
      "description": "Merchants in the hotel accommodations
category include hotels and motels, and many smaller
establishments like bed & breakfasts, and inns. Hotel
accommodations not booked directly with the hotel will not
qualify; for example, hotel accommodations purchased through
travel agencies, discount travel sites, vacation clubs, tour
operators, or as part of a package offered by merchants such as
cruise lines and railways. Points will not be earned until
charges are posted to your account. Purchases other than your
room bill on the hotel premises will not be included in the
category unless they classify as a hotel merchant; for example,
restaurant or entertainment purchases. In addition, the
purchasing of hotel points will not qualify in this category."
    }
  ]
}
```

10.4.3 Example

Suppose a customer makes a purchase at a local hardware store using their Chase Ultimate Rewards Visa and this is the transaction the data provider sends.


```
{
  "accountId": "10001",
  "transactionId": "20001",
  "postedTimestamp": "2017-11-05T13:15:30.751Z",
  "description": "Ace Hardware",
  "debitCreditMemo": "DEBIT",
  "amount": 3.74,
  "reward": {
    "accrued": 7.48,
    "adjusted": 0,
    "categoryID": "10009"
  }
}
```


The `reward` associated with this transaction shows a `categoryId` of 10009.


```
  "reward": {
    "accrued": 7.48,
    "adjusted": 0,
    "categoryID": "10009"
  }
```


Matching this categoryId to the list of categories for the reward program, the data recipient finds that the reward category was `Home improvement stores` and it uses a 2x multiplier to calculate rewards accrued.


```
    {
      "categoryName": "Home improvement stores",
      "categoryId": "10009",
      "multiplier": 2,
      "description": "Merchants in the home improvement stores
category specialize in selling a variety of home improvement
supplies, from larger home improvement stores to smaller
hardware stores. Merchants that sell a wide variety of general
goods which may include home improvement supplies, for example,
warehouse clubs, discount stores, or grocery stores, are not
included in this category. Also, merchants that specialize in
home furnishings, garden and landscaping supplies are not
included."
    }
```

11. Bill Payment Overview

This section provides the API specifications related to money movement and bill payment.

The Money Movement API defines Bill Pay services using the Payee, Payment and Recurring Payment entities with their corresponding /payees, /payments and /recurring-payments endpoints. This API can be used to create, update, delete and search for payees, payments and recurring payments. A payment can be created to schedule a one-time transfer to a payee and a recurring payment can be created to repeat payments on a schedule. A recurring payment will spawn the upcoming payment. The spawned payment object behaves like a payment created via the API. The upcoming payment can be modified and cancelled, allowing the customer to adjust the upcoming-payment of a recurrence.

11.1 Managing Payees

A payee is used to represent the receiver of the funds. The Money Movement API supports Merchants. Merchants are represented by name, address and phone number. A merchant account identifier can be used to represent the payer's account with the payee and can be provided to allow the payee to allocate funds accordingly.

11.1.1 Payee Lifecycle

A payee can be **PENDING**, **ACTIVE**, **REJECTED** or in the **DELETED** status. A **PENDING** payee has been registered, but is not yet ready for payments. An **ACTIVE** payee is ready to receive payments. A **REJECTED** payee was found to be invalid and cannot receive payments. A **DELETED** payee has been soft-deleted and can be hard-deleted sometime in the future.

Creating a Payee

A payee can be created using the createPayee operation (POST /payees) of the FDX API. The server will respond with either the created payee with the payeeId field populated or an existing payee. The HTTP code 201 vs 200 determines if the payee was created or an existing payee was found. 201 indicating creation and 200 indicating that an existing payee was found. Once the payee has been created, the payeeId can be used to reference and query the entity in future calls.

The server is responsible for assigning a payeeId for a newly created payee. If a server fails to acknowledge a createPayee request, the client can try again with the same information. The duplicate detection logic will catch if the payee had successfully been created in a prior call.

Duplicate Detection

A server can choose to implement duplicate detection by ensuring that only unique payees are created given various payee fields. If a duplicate payee is requested to be created, the server is expected to return a HTTP 200 response with the details of the existing payee.

Deleting a Payee

A payee can be deleted using the deletePayee operation (DELETE /payees/{payeeId}). The payee is expected to be soft-deleted and can be hard-deleted sometimes in the future. The soft-delete allows other clients to discover the deleted payee using synchronization before it is hard-deleted and made permanently inaccessible via the FDX API.

Merchant Account Identifier

When creating or modifying a payee an array of merchantAccountIds can be provided. This array represents the customer's accounts with the merchant. When paying a merchant for a specific account, the merchant's account identifier can be provided when a payment is submitted. The stored merchantAccountIds on a payee are for reference only.

11.1.2 Syncing Payees

The list of payees can be replicated by the client. The client is expected to use the searchForPayees operation (GET /payees) to query for all payees. Following the initial import, the client is expected to search for payees that have changed by providing the updatedSince parameter in the searchForPayees operation.

The client is expected to poll the GET /payees endpoint to search for new changes. The client will obtain the state at the point of query. The payee could have updated multiple times between two queries, but the client will only be aware of the final change.

See section 11.6 Synchronization for more information.

Updating a Payee

The updatePayee operation (PATCH /payees/{payeeId}) can be used to modify an existing payee. The payee type must match the existing payee. (For example, if the payee is a merchant, the merchant key with the associated MerchantForUpdate entity must be provided. If an incorrect type is provided, the server will return an Invalid Input error.) Given that updatePayee is a PATCH call, only the fields provided for the merchant must be updated. Fields not provided will not be changed and all changes should be made atomically, all-at-once or none-at-all. The server will return the complete Payee entity with all of the fields of the payee, including those not updated. The client can locally update the payee once the server has acknowledged the request and responded with the full payee entity.

The returned payee might have unexpected field changes as it is possible for multiple clients to update the same payee. Updates will be applied in the order that they are received. The syncing process will make the clients aware of the final state of the payee.

Deleting and Expiring Payees

Once a payee has moved to the **DELETED** state, the client is expected to delete the payee. The payee will be soft-deleted and in the **DELETED** status for a period of time. Once the payee is hard-deleted, it will no longer appear in the searchForPayees result set.

Some organizations might choose to automatically delete inactive payees. In this case, the payee will not move into the **DELETED** status. Instead, the expiresTimestamp field will be populated. Once the expiration timestamp passes, the entity is expected to be deleted. The entity might not be soft-deleted and might be hard-deleted directly. Thus the entity might not appear as a deleted entity in the searchForPayees operation. The client must delete the payee after the expiresTimestamp date passes.

As a payee is used within the payments system, the expiresTimestamp date will update to reflect a new expiration timestamp based on the server's policy. The client should use the getPayee operation to query the entity to ensure it's been removed by the server by observing the Payee not found error before removing it locally. The expiration timestamp could have been updated between the last sync and the last recorded expiration timestamp.

11.2 Scheduling a Payment

A payment represents a scheduled one-time transfer of funds to a payee. A payment defines the account to source funds, the payee, amount, merchant account number and due date. The server will assign a payment ID and manage the state of the payment. If the payment was spawned from a recurring payment, the payment will link back to the recurring payment via the recurringPaymentId field.

11.2.1 Payment Lifecycle

A payment can be in the **SCHEDULED**, **PROCESSING**, **PROCESSED**, **NOFUNDS**, **FAILED** or **CANCELLED** state. A newly created payment will be in the **SCHEDULED** state until the scheduled time occurs, the payment is cancelled or a failure occurs. The payment will move into the **PROCESSING** state while the payment is being attempted. When the payment has been successfully sent, the payment will move to the **PROCESSED** status. If the payment was attempted but not enough funds were available, the payment will move to the **NOFUNDS** status. If for any reason, the payment could not be executed, it will move to the **FAILED** status.

11.2.2 Creating a Payment

A payment can be created using the schedulePayment operation. The server will acknowledge the request to schedule a payment by responding with a HTTP 2xx or a 4xx response code. The client should relay the response to the end-user handling duplicate detection, successful creation and client errors. If the request fails at the network layer or if the server responds with a 5xx, the request should be considered unacknowledged. The client can retry unacknowledged payments allowing the server's duplicate detection to catch if the payment had been previously processed or the client can synchronize the scheduled payments with the server to discover if the scheduled payment was successfully created.

Duplicate Payment Detection

The server can implement duplicate detection using the combination of payee, amount, due date and merchant account id. When a client attempts to create a duplicate payment, the server can indicate that it is a duplicate by the HTTP response code returned. A 201 is returned if a

new scheduled payment has been created. If a duplicate scheduled payment is found, the server will return a 200 HTTP response code along with the existing payment in the body of the response.

11.2.3 Syncing Payments

The list of payments can be replicated by the client. The client is expected to use the searchForPayments operation (GET /payments) to query for all payments. Following the initial import, the client is expected to search for payments that have changed by providing the updatedSince parameter in the searchForPayments operation.

The client is expected to poll the GET /payments endpoint to search for new changes. The client will obtain the state at the point of query. The payment could have updated multiple times between two queries, but the client will only be aware of the final change.

See 11.6 Synchronization for more information.

11.2.4 Updating Payments

The updatePayment operation (PATCH /payments/{paymentId}) can be used to modify an existing payment. The payment's status can prevent the payment from being modified. Payment providers will notify the client by returning a "Payment cannot be modified or deleted" error. The updatePayment is a PATCH call. Only the fields provided must be updated. Fields not provided will not be changed and all changes should be made atomically, all-at-once or none-at-all. The server will return the complete Payment entity with all of the fields of the payment. Including those not updated. The client can locally merge in changes once the server has acknowledged the request and responded with the full payment entity.

The returned payment might have unexpected field changes as it is possible for multiple clients to update the same payment. Updates will be applied in the order that they are received. The syncing process will make the clients aware of the final state of the payment.

11.2.5 Cancelling Payments

Payments can be cancelled using the cancelPayment operation (DELETE /payments/{paymentId}). The payment's status may prevent the payment from being cancelled. Payment providers will notify the client by returning a "Payment cannot be modified or cancelled" error.

11.3 Scheduling a Recurring Payment

A recurring payment represents a schedule of transfers to a payee. A recurring payment defines all of the fields of a payment (source funds, the payee, amount, merchant account number and due date) as well as fields that define the schedule, frequency and duration. The server will assign a recurring payment ID and manage the state of the recurring payment. The recurring payment will spawn a payment object for the next. This spawned payment will link back to the payment via the recurringPaymentId field. Payments spawned by a recurring payment can be

retrieved using the getPaymentsForRecurringPayment (GET /recurring-payments/{recurringPaymentId}/payments) operation.

11.3.1 Recurring Payment Lifecycle

A recurring payment shares the lifecycle statuses of a Payment. When a recurring payment is created, it will be in the **SCHEDULED** state, returning the timestamp of the next scheduled payment. If a recurring payment's duration has passed, the recurring payment will be in the **PROCESSED** state, returning the timestamp of the last processed payment. If a payment is cancelled, the payment will be in the **CANCELLED** state, returning the timestamp when the recurrent payment was cancelled.

Payments Spawned from a Recurring Payment

An active recurring payment will at any given time have at least one spawned payment in the **SCHEDULED** state. This spawned payment behaves the same as a payment created via the schedulePayment operation (POST /payments). If the upcoming payment is cancelled or modified, it will affect just that payment. The recurrence will spawn payments for other occurrences that will remain unaffected. So a single payment from a recurring payment can be cancelled or modified without affecting other payments.

Payments spawned from a recurring payment will link back to the recurrence via the recurringPaymentId field. Payments linked to a recurring payment can also be retrieved via the getPaymentsForRecurringPayment operation (GET /recurring-payments/{recurringPaymentId}/payments).

11.3.2 Creating a Recurring Payment

A recurring payment can be created using the scheduleRecurringPayment operation (POST /recurring-payments). The server will acknowledge the request to schedule a recurring payment by responding with a HTTP 2xx or a 4xx response code. The client should relay the response to the end-user handling duplicate detection, successful creation and client errors. If the request fails at the network layer or if the server responds with a 5xx, the request should be considered unacknowledged. The client can retry unacknowledged recurring payments allowing the server's duplicate detection to catch if the payment had been previously processed or the client can synchronize the scheduled recurring payments with the server to discover if the scheduled recurring payment was successfully created.

Duplicate Detection

The server can implement duplicate detection using the combination of payee, amount, due date, merchant account id and recurrence fields. When a client attempts to create a duplicate recurring payment, the server can indicate that it is a duplicate by the HTTP response code returned. A 201 is returned if a new recurring payment has been created. If a duplicate recurring payment is found, the server will return a 200 HTTP response code along with the existing recurring payment in the body of the response.

11.3.3 Syncing Recurring Payments

The list of recurring payments can be replicated by the client. The client is expected to use the searchForRecurringPayments operation (GET /recurring-payments) to query for all recurring payments. Following the initial import, the client is expected to search for recurring payments that have changed by providing the updatedSince parameter in the searchForRecurringPayments operation.

The client is expected to poll the GET /recurring-payments endpoint to search for new changes. The client will obtain the state at the point of query. The recurring payment could have updated multiple times between two queries, but the client will only be aware of the final change.

See 11.6 Synchronization for more information.

11.3.4 Updating Recurring Payments

The updateRecurringPayment operation (PATCH /recurring-payments/{recurringPaymentId}) can be used to modify an existing recurring payment. The recurring payment's status can prevent the payment from being modified. Payment providers will notify the client by returning a "Recurring payment cannot be modified or deleted" error. The updateRecurringPayment is a PATCH call. Only the fields provided must be updated. Fields not provided will not be changed and all changes should be made atomically (all-at-once or none-at-all). The server will return the complete RecurringPayment entity with all of the fields of the recurring payment, including those not updated. The client can locally merge in changes once the server has acknowledged the request and responded with the full recurring payment entity.

The returned recurring payment might have unexpected field changes as it is possible for multiple clients to update the same payment. Updates will be applied in the order that they are received. The syncing process will make the clients aware of the final state of the recurring payment.

11.3.5 Cancelling Recurring Payments

Payments can be cancelled using the cancelRecurringPayment operation (DELETE /recurring-payments/{recurringPaymentId}). The recurring payment's status may prevent the recurring payment from being cancelled. Payment providers will notify the client by returning a "Recurring payment cannot be modified or deleted" error.

Cancelling a recurring payment may result in the cancellation of spawned payments that have not yet been processed. The client should synchronize payments after cancelling a recurring payment. The getPaymentsForRecurringPayment operation (GET /recurring-payments/{recurringPaymentId}/payments) can be used to retrieve the payments associated with the recurring payment.

11.4 Account Participation

Accounts will indicate their ability to participate in bill payments via their billPayStatus field. The status value can be **NOT_AVAILABLE**, **AVAILABLE**, **PENDING** and **ACTIVE**. Accounts that

cannot be used as source of funds for a bill payment will be in the **NOT_AVAILABLE** status. Accounts that can have the bill pay capabilities enabled are in the **AVAILABLE** status. Accounts that have been requested to be enabled, but have not yet been are in the **PENDING** status. Accounts that are ready for bill payments are in the **ACTIVE** status. Activating accounts for bill payments is outside of the scope of the FDX API.

11.5 Handling Duplicate Requests

Some operations are not idempotent and require the use of the `idempotency-key` header. This header is used to ensure that the server does not process the same request more than once. Idempotency keys are scoped to the client to avoid collisions.

The server will keep a record of the requests it has processed and the result of the operation. If an idempotency-key that has already been processed is encountered, the server will not re-execute the operation but will return the original result. If the result contains an entity, the server will return the most recent version of the entity.

In the unlikely scenario that the most recent version of the entity is hard-deleted, an HTTP Code 404 response with corresponding error will be returned.

Providing the most recent version of the entity saves the client a second call to search for changes.

Client should implement a dead-letter queue for payment requests that fail acknowledgement multiple times and communicate the condition to the end-user. The client should synchronize with the server to ensure that it has the most recent updates.

Servers are expected to remember idempotency keys long enough for clients to recover from network outages, system outages and other disasters. Worst case scenarios for when services and desktop applications might be out of communication should be considered.

11.5.1 Duplicate Payment Request Use Cases

Use Case 1a: Acknowledged successful payment request

1. Given a client that schedules a payment.
2. The server will return HTTP 201 Created along with the newly created payment.

Use Case 1b: Failed to acknowledge a succeeded payment request with change

1. Given a client that attempted to schedule a payment for the 1st of the month that succeeds but failed to receive acknowledgment.
2. When a customer modifies the created unacknowledged payment on the FI's website for the 5th of the month,
3. And the client retries the payment request with the original idempotency-key,
4. Then the server will return HTTP Code 201 Created, along with the payment reflecting the 5th of the month.

The payment was scheduled in step 1 and was not re-attempted in step 4.

Use Case 1c: Failure to receive original payment request

1. Given a client attempting to schedule a payment that is not received by the server.
2. When the client retries the payment request with the original idempotency-key,
3. Then the server will return HTTP Code 201 Created, along with the newly created payment

The payment was scheduled in step 3.

Use Case 2a: Acknowledged failed payment request

1. Given a client attempts to schedule a payment that is rejected
2. Then the server will return HTTP Code 400 along with the error response

Use Case 2b: Failure to acknowledge a failed payment request

1. Given a client attempting to schedule a payment that is rejected and is unacknowledged.
2. When the reason for the rejection is corrected,
3. And the client retries the payment request with the original idempotency-key,
4. Then the server will return HTTP Code 400 along with the original error response.

The payment was rejected in step 1 and was not re-attempted in step 4.

Use Case 2c: Failure to receive original payment request

1. Given a client attempting to schedule a payment that is not received by the server that would have failed.
2. When the reason for the rejection is corrected,
3. And the client retries the payment request with the original idempotency-key,
4. Then the server will return HTTP Code 201 Created, along with the newly created payment

The payment was scheduled in step 4. Unlike Use Case 2b, the payment request ends up being successful.

11.6 Synchronization

A synchronizable array is used to return a set of data that can be queried for changes. The endpoint that returns a synchronizable array will accept the updatedSince query parameter. The returned entity will inherit from the SynchronizableArray entity.

11.6.1 Seeding Data

The client will seed their database by searching for entities from an endpoint that supports synchronization. As of FDX API V5.0, the Payees, Payments, Recurring Payments and Transfers entities support synchronization. The server will return the dataset (using pagination as necessary) and return a nextUpdateId that can be used to query for changes. The client is expected to store the dataset along with the nextUpdateId.

11.6.2 Keeping Data in Sync

When the client is ready to query for updates, the client will use the updatedSince query parameter to pass the stored nextUpdateId back to the endpoint used to retrieve the original data set. The server will return created or updated entities using the nextUpdateId as reference. The client is expected to replace the local entities with their updated version. The server will return a new nextUpdateId that the client will store. The new nextUpdateId will be used for the next call to query for updates.

The resulting synchronization call might require pagination if the number of updated entities exceeds what the server supports. The client is expected to use pagination to retrieve all of the updated entities.

Expired NextUpdateId

A nextUpdateId can expire. Typically due to the amount of time between queries. The server will indicate as such by returning the nextUpdateId not found error. The client is expected to reseed the data to recover from this scenario.

If the time between queries is longer than the amount of time an object remains soft-deleted before it is hard-deleted, the nextUpdateId should be considered expired to allow for the client to discover deleted objects via a reseed.

A nextUpdateId is expected to remain valid unless necessary for purge, periodic cleanup and/or other data retention policies. If update Ids are not valid for a sufficiently long period, the client will need to reseed resulting in significant resource utilization.

11.7 Payment Initiation Party Maintenance

In addition to bill payment *payees* that serve as merchants, the FDX API provides an interface for management of payment initiators. A payment initiation party is the individual or business with which the customer needs to communicate in order to receive or send a payment.

This portion of the API enables maintenance of payment initiation parties and their IDs, payment methods, accounts, and other necessary information. Note that this portion of the API does not

handle payment instructions or execution, it is meant to provide identifying information for the parties and accounts involved.

A payment initiation party is the party identified at the beginning of a payment process, and can be either a *payee* or a *payer*.

- If the business purpose is to request money, then the payment initiation party that receives the request is the payer and will send the payment according to the request.
- When the business purpose is to send money, the payment initiation party is the payee that receives the payment.

The interface enables management of a person or an organization across payment rails, such as Interac, Zelle, TCH, or Wire for both domestic and international payment. It provides the following benefits:

- A broader solution of payment initiation party management that will be rail agnostic. Users may set up a list of payment initiation parties and register them separately with the products offered by various rails.
- The payment initiation party will store the static payment attributes. Thus, each time a payment initiation party is involved in the money movement process, it can reuse the particular static payment attributes configured under the profile.
- The capabilities will support multiple payment initiation parties to rail configurations. It provides end-user flexibility on low-cost alternatives.
- The payment initiation party will cover both individuals and organizations, supporting Retail and Commercial.
- Being able to switch between different payment rails without significant changes due to standardization.

11.8 Payment Flow Options

The Money Movement YAML offers flexibility to data providers on how they wish to schedule and return one time and recurring payments.

In /payments, the Data Provider may return all one time or next occurring payments. Payments populated here can be future dated. This endpoint is intended to return the actual payment instance and provide details on if the payment goes through.

In /recurring-payments, the Data Provider may return the schedule associated with the payment. A schedule may be one time or recurring. This endpoint returns information around the next upcoming instance of the payment (or the sole instance if it is a one-time scheduled payment) and the schedule information (recurring payment status, duration, and frequency).

A single payment can be scheduled in either the /payments endpoint or in the /recurring-payments endpoint by setting `frequency: "ONETIME"`.

A Data Provider can consider using /payments to return information on if a payment was processed successfully and /recurring-payments to return payment schedules. If a Data Provider is interested in returning only a schedule of upcoming payments for a bank switch or personal finance management use case, they have the option to implement just /recurring-payments.

To get a list of full scheduled payments, recipients should make calls to both /accounts to retrieve bills and /recurring-payments to retrieve payments.

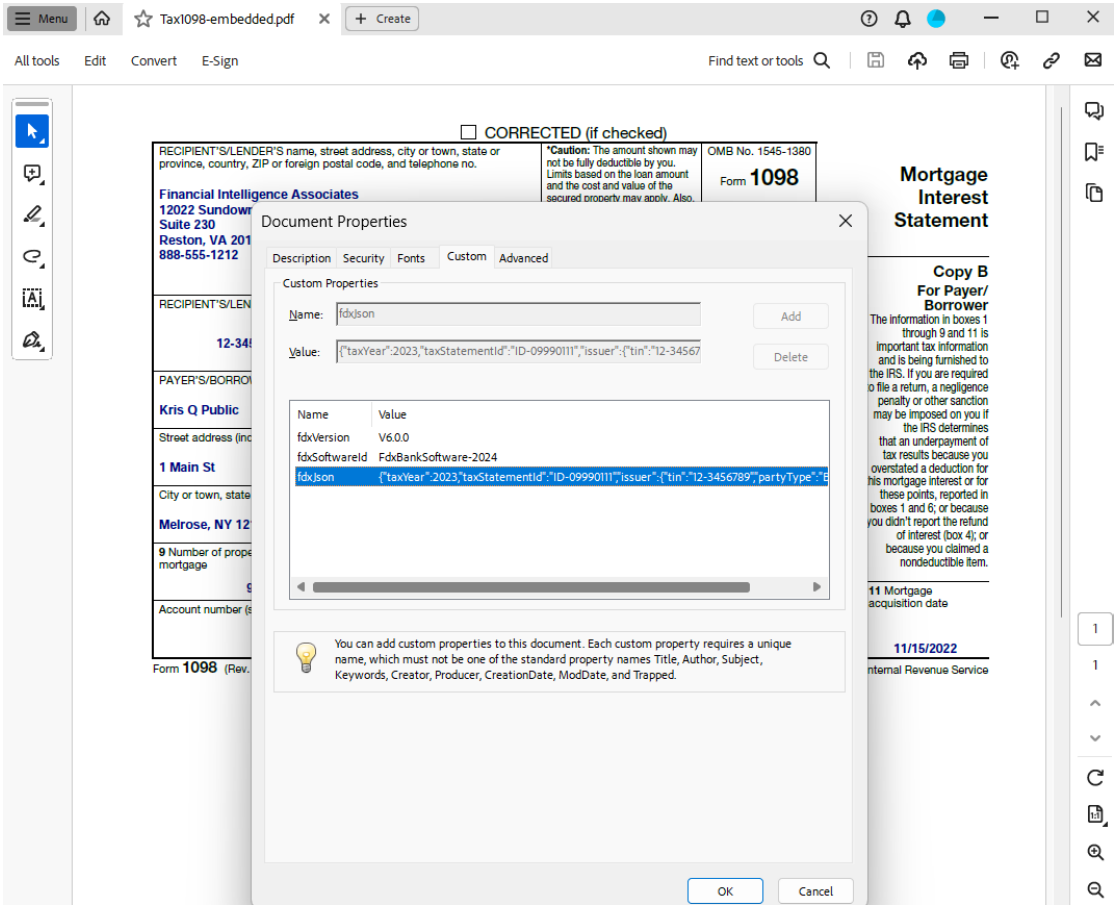
12. PDF-Embedded Tax Data

You can use the FDX US Tax entities to include and transmit tax data embedded within any PDF tax document. See the *FDX Tax Document PDF Embedded Data_v2024* document from FDX, published with release v6.1 in May 2024.

That document describes how to modify a tax form PDF file to attach custom attributes directly containing its data using the FDX Tax schema definitions. For more information see the FDX web page to which embedded data PDF recipients are directed, https://financialdataexchange.org/tax-pdf.

12.1 Example of PDF Embedded Tax Data

The following example shows custom properties for form 1098 from the *FDX Tax Document PDF Embedded Data_v2024* document.



13. QR Codes for Tax Data

You can use the FDX US Tax entities to encapsulate and print tax data as QR Codes on any tax document. See the *FDX Tax Document QR Code Specification_v2024* document from FDX, published with release v6.1 in June 2024.

That document contains the details of the image frame which FDX has defined to surround tax data QR codes. For more information see the FDX web page to which QR code recipients are directed, https://financialdataexchange.org/us-tax.

13.1 Example of QR code with Tax Data

The following is a QR code example showing an image frame in horizontal format. A vertical format is also available.

14. Consent API Operations & Data Structures

14.1 Consent API Use Cases and Exclusions

The FDX Consent API enables two primary use cases:

1. It provides API operations and data structures to support the Consent Request user flow defined by FDX User Experience Guidelines.
2. It provides API operations and data structures to support *Consent Synchronization* — the process by which multiple parties can ensure they have a consistent record of a user's permission for financial data-sharing.

The FDX Consent API does **not** provide (or even define) enforcement mechanisms for ensuring that data is exchanged in concordance with the user's intents. Furthermore, the FDX Consent API does **not** define the application or business logic internal to Data Provider and Data Recipient systems and services that align Consent capabilities with data availability or API capabilities.

14.1.1 Consent Grant vs `ConsentGrant`

In an effort to avoid ambiguity, throughout this document, **Consent**, **User Consent**, and **Consent Grant** are used colloquially to refer to the *record* of the permission granted by an end user. `ConsentGrant` is the specific data entity defined as part of the FDX API specification. A record of user consent is exchanged using the `ConsentGrant` data entity.

14.1.2 A Three-Party Model of Consent

A Consent Grant in the FDX framework represents permission to exchange an end user's financial data applicable to three specific parties: the **End User** ("EU"), the **Data Provider** ("DP"), and the **Data Recipient** ("DR"). It is a *singular* representation of this permission; in other words, between any unique set of three parties only one record of consent can be active at a given time.

Consent represents the following to each of the three parties:

- **Data Recipient** should interpret a Consent Grant as permission to access End User's financial data. The Data Recipient's use of this data is generally governed by terms of service with the End User (these terms of service are **not** incorporated into the FDX model of Consent).
- **Data Provider** should interpret a Consent Grant as permission to provide access to the Data Recipient for the End User's financial data. A Data Provider generally makes no assertions about Data Recipient's use of the data.
- **End User** should interpret a Consent Grant as their record that they have given permission for data access. End User should not be expected to disambiguate access vs. use of their financial data.

14.1.3 Considerations for Intermediaries

In the North American ecosystem, the DR's interface(s) for requesting user consent is frequently serviced by an intermediary Data Access Platform ("DAP"). In this case, the DAP occupies the role of DR in the interaction. Throughout this proposal "DR" should be interpreted as a *role* in the protocol and data exchange, rather than as a specific category of business entity.

The API operations defined in this proposal are restricted to this three-party model. Data Access Platforms are not excluded from the interaction: they may act on behalf of other entities to perform the Data Recipient or Data Provider role.

A four-party model of consent is **not** defined by this proposal. FDX plans to expand considerations for intermediaries in future enhancements to the Consent API.

14.2 Consent Data Lifecycle

Consent is requested and granted as follows in the three-party model; this user journey through presentation and collection of permission must be performed in accordance with the FDX User Experience Guidelines:

- Before the consent journey begins, EU must have independent relationships with each DR and DP;
- Before the consent journey begins, DR must be pre-registered with the DP:
    - DR must establish the superset of what may be requested from DP, and
    - DR must establish identity with DP (both in terms of client authentication and in terms of a bilateral business agreement);
- EU must initiate their consent journey from within DR's user experience, per UX Guidelines;
- While DR determines the scope of data access it seeks to access from DP, it must disclose its intent to the EU;
- EU must *actively* authorize Data Provider to enable Data Recipient's access to End User's data.

14.2.1 Simplified Consent Journey

1. DR presents its Consent Request to the EU, disclosing the scope of desired data access.
2. If accepted by the EU, DR passes its request using the `ConsentRequest` entity within an authorization request to the DP while synchronously redirecting EU to DP.
3. DP authenticates the EU, then "replays" the Consent Request to the EU. The EU may authorize the requested access, "granting consent" for the data exchange.
4. If authorized, Data Provider provides the `ConsentGrant` to the DR.

5. Later, Consent may expire, be revoked operationally or revoked by the EU.

14.3 Consent Data Entities & Types

14.3.1 Consent Request

The `ConsentRequest` is a data entity consisting of the following scopes:

1. **Time Scopes** — a collection of parameters asserting Duration Type, Duration Period of consent, and Lookback Period which describes a historical period for which data can be retrieved;
2. **Resource Types** — enumeration of the core Resource Types (ie: Accounts, or others such as a Customer Profile or a Document) for which the Data Clusters are requested;
3. **Data Clusters** — enumeration of the Clusters of data elements requested.

14.3.2 Consent Grant

The `ConsentGrant` is a data entity consisting of the following members:

1. **ID** — an identifier for the record, to permit operations such as retrieval; the `consentId` must be unique to the Consent Issuer (a service of the DP) and it must be a URL-safe string;
2. **Consent Status** — enumeration of the current status of the Consent Grant;
3. **Time Scopes** — a collection of parameters asserting Duration Type, Duration Period of consent, Expiration Time, and Lookback Period which describes a historical period for

which data can be retrieved, and timestamps asserting the Creation and Expiration time of the Consent Grant;

4. **Parties** — a collection of parameters identifying the Parties (including the legal entity operating branded products or services) in the data exchange.
    - Descriptive information is collected during DR registration at DP, and populated during issuance by Data Provider from its registry;
5. **Resources** — identifiers of the core Resources (ie: Accounts, or others such as a Customer Profile or a Document) for which a set of Data Clusters are permissioned;
6. **Data Clusters** — enumeration of the Clusters of data elements permissioned by this Consent Grant.

14.4 API Operations

This section specifies a protocol for the DR to make consent requests and for the DP to respond to those requests conveying the user's intent to grant their permission for data sharing. Broadly speaking, the defined operations enable:

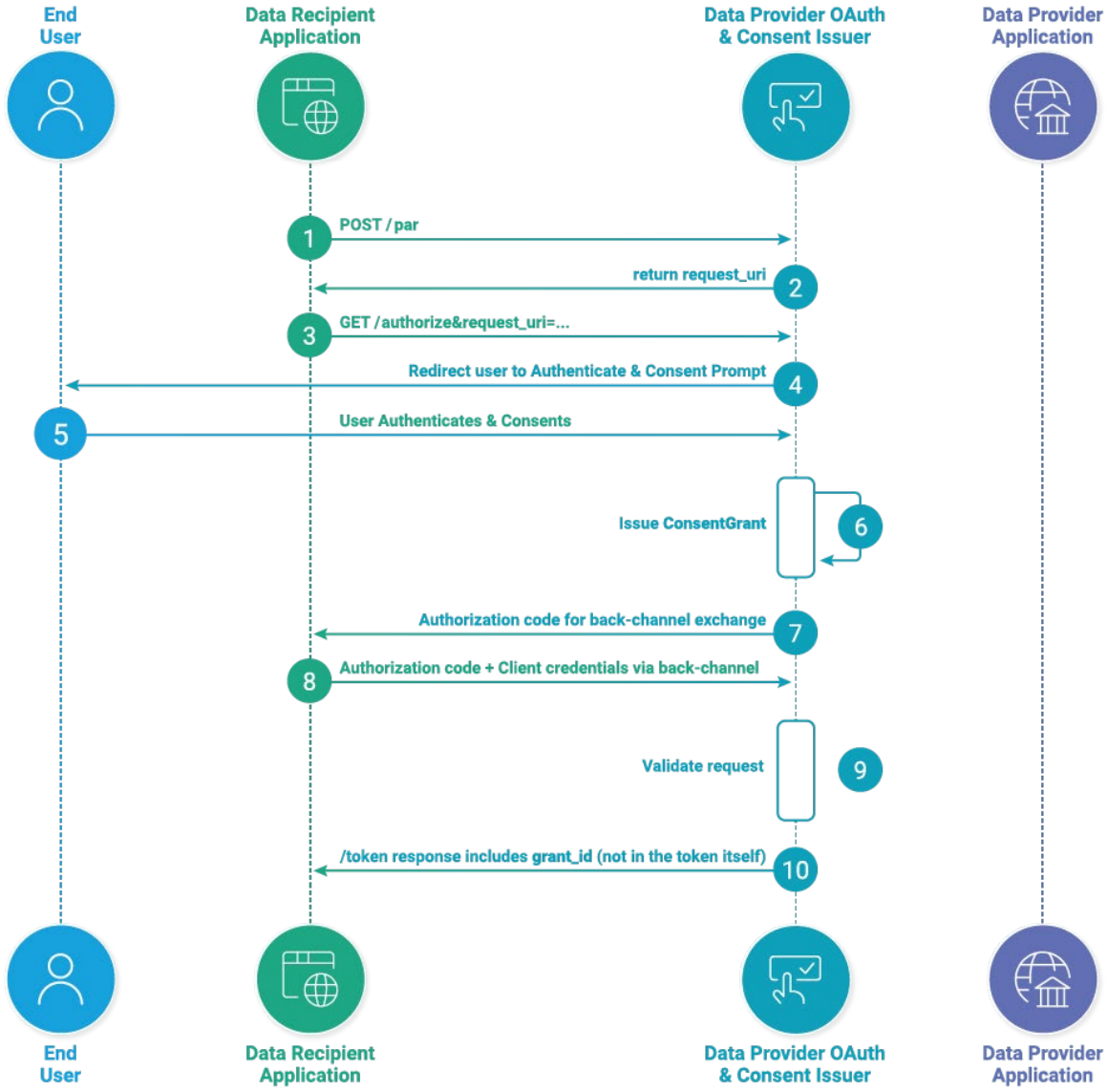
- Requesting → Granting Consent
- Querying and/or retrieving the current record of Consent
- Revoking Consent

The Consent API protocol as defined here relies on the presence of user and client authentication protocols and security profiles, including some in draft form, such as OAuth 2.0 Rich Authorization Requests and Pushed Authorization Requests. Where specific extensions to external or FDX standards are intended, they are explicitly identified as such. Where silent on a particular mechanism of authentication or authorization, this specification intends to enable parties freedom to implement in a manner they see fit.

14.4.1 Requesting Consent with POST /par

DR requests creation of the consent by calling DP's `POST /par` endpoint. As specified by OAuth 2.0 Pushed Authorization Requests, this operation has 4 stages. This protocol tightly couples consent creation with an OAuth 2.0 Authorization Code grant flow IETF RFC 6749 and RAR & PAR draft specifications IETF memos OAuth 2.0 Rich Authorization Requests and OAuth 2.0 Pushed Authorization Requests, respectively. In the event of discrepancies, RAR and PAR specifications prevail.

Granting Consent Flow



Step 1 — DR initiates a POST request to DP's `POST /par` endpoint using the Pushed Authorization Request (PAR) method

DR must be authenticated with methods allowed by the FDX Security Profile for OAuth client authentication. The `authorization_details` request parameter MUST contain a JSON-formatted object with two members in compliance with the RAR format specified by RAR memo, section 2:

1. `type` parameter with value `fdx_v1.0`, and
2. `consentRequest` parameter containing a valid `ConsentRequest` entity; For example:

```
{
  "authorization_details": [
    {
      "type": "fdx_v1.0",
      "consentRequest": {
        "durationType": "ONE_TIME",
        "lookbackPeriod": 60,
        "resources": [
          {
            "resourceType": "ACCOUNT",
            "dataClusters": [
              "ACCOUNT_DETAILED",
              "TRANSACTIONS",
              "STATEMENTS"
            ]
          }
        ]
      }
    }
  ]
}
```

Step 2 — On success, DP responds with a `201 Created` HTTP response code and JSON response

Endpoint behavior and responses defined in detail by PAR memo, section 2.

Step 3 — DR uses the returned `request_uri` to build its subsequent request to `GET /authorize`

Endpoint behavior and responses defined in detail by PAR memo, section 4.

Steps 4, 5 — User authentication, consent and authorization

Defined in detail by FDX UX Guidelines. DP requires input and explicit authorization from authenticated EU to issue the `ConsentGrant`.

Step 6 — If successful, DP issues the `ConsentGrant` including a unique identifier for the record, `consentId`

Steps 7, 8, 9 — If successful, DP returns a `302 Found` HTTP response code, and an authorization `code` for DR to build its request to `POST /token`

Endpoint behavior and responses defined in detail by IETF RFC 6749, section 5.

Step 10 — Successful token response will include the `grant_id` parameter containing `consentId` in addition to token response contents

Error Handling

Errors for `GET /authorize` MUST follow OAuth 2.0 standards: In all cases, DP responds with `302 Found` HTTP code and informs the client of the error by adding error parameters to the query component of the redirection URI using the `application/x-www-form-urlencoded` format.

In instances of mismatch between scopes requested by the DR in the `ConsentRequest` and the pre-registered scopes (e.g.: invalid Data Clusters, Resource Types, or Duration), service MUST respond with `invalid_scope` error code.

For example:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=invalid_scope
```

Other errors (eg: invalid client, failed user authentication, or access denied) MUST be handled per IETF RFC 6749, section 5.2.

14.4.2 Retrieving Consent with GET /consents/{consentId}

Request the record of Consent Grant. Request originates from the DR (or another authorized service acting on behalf of EU, DP, DR).

Step 1 — The client initiates the operation by calling the `GET /consents/{consentId}`. The `consentId` was delivered in successful `POST /token` response.

The Data Recipient must be authenticated with methods allowed by the FDX security profile

Step 2 — DP's Consent Issuer responds with the `ConsentGrant` in the JSON response body.

Data Providers MUST handle two categories of errors for `GET /consents/{consentId}`: unauthorized requests and invalid requests.

If a client attempts to retrieve a `ConsentGrant` with invalid credentials, DP MUST respond with a `401 Unauthorized` HTTP code, further error description is not required or recommended.

If a known client attempts to retrieve a `ConsentGrant` they are either not permitted to access or that doesn't exist, DP MUST respond with a `404 Not Found` in both cases, further error description is not required or recommended.

14.4.3 Revoking Consent with PUT /consents/{consentId}/revocation

Revoking Consent is classified as a write operation on the `ConsentGrant` data object. Revocation is complete and cannot be reversed; once revoked, access to all resources

and data clusters by data recipient is no longer permitted. To reestablish access, a new `ConsentGrant` must be created.

Consent revocation is modeled as a status change from `ACTIVE` to `REVOKED`.

Step 1 — Requestor initiates a PUT request to DP's `PUT /consents/{consentId}/revocation` endpoint

DR must be authenticated with methods allowed by the FDX Security Profile for OAuth client authentication. The `consentId` was originally delivered in successful POST /token response. The request body consists of the reason for the revocation and the Party initiating the revocation.

Consent can be revoked by user action, such as in the following examples:

- *User "disconnects" an FI (Data Provider) from an application (Data Recipient), or*
- *User ends their relationship with a given application (Data Recipient)*

```
{
  "reason": "USER_ACTION",
  "initiator": "INDIVIDUAL"
}
```

Or consent can be revoked via a business rule at one or more parties that invalidates the Consent, such as in the following example:

- *An FI (Data Provider) and application (Data Recipient) discontinue their data-sharing relationship*

```
{
  "reason": "BUSINESS_RULE",
  "initiator": "DATA_RECIPIENT"
}
```

Step 2 — DP's Consent Issuer responds with 204 (no response body)

`204` SHOULD be used to indicate that the revocation request has been received and will be processed, but may not be immediately revoked.

- If revocation is asynchronous, DP MAY follow up with a notification to signal a completed revocation.

Error Handling

Errors for `PUT consents/{consentId}/revocation` MUST follow FDX API standards, also see section 6.2 Errors:

- `400 Bad Request` SHOULD be used with an improperly formatted request
- `401 Unauthorized` SHOULD be used if the error results from inadequate authentication
- `403 Forbidden` SHOULD be used if the error results from inadequate authorization which cannot be corrected due to insufficient rights to the resource
- `404 Not Found` SHOULD be used if the specified Consent Grant is not applicable to the validly authenticated client
- `409 Conflict` SHOULD be used if a request is made to revoke an already revoked or expired Consent Grant

14.4.4 Retrieving Consent Revocation Record with GET /consents/{consentId}/revocation

A requestor or other permissioned party may wish to display the history of a Consent Grant to the End User, following revocation. This record of this change is available to Data Recipient or Data Access Platform requestor with necessary permission to retrieve the Consent Grant.

Requestor initiates a GET request to DP's **`GET /consents/{consentId}/revocation`** endpoint.

DP's Consent Issuer responds with an array of timestamped records of all revocations:

```
{
  "revocations": [
    {
      "status": "REVOKED",
      "reason": "BUSINESS_RULE",
      "initiator": "DATA_PROVIDER",
      "updatedTime": "2023-12-14T15:45:16.987Z"
    }
  ]
}
```

Retrieve Revoked Consent Grant from `GET /consents/{consentId}`

A requestor or other permissioned party may request the record of Consent Grant.

- Request is authorized with user's access token in accordance with *FDX API Security Model* and *FDX Control Considerations* documents.

DP's Consent Issuer responds with the Consent Grant with `status: REVOKED`:


```
{
  "id":"9585694d3ae58863",
  "status":"REVOKED",
  "parties":[
    {
      "name": "Seedling App",
      "type": "DATA_RECIPIENT",
      "homeUri": "https://www.seedling.com",
      "logoUri": "https://www.seedling.com/assets/seedling-
logo.png",
      "registry": "FDX",
      "registeredEntityName": "Oak Tree Holdings, Inc",
      "registeredEntityId": "5493001052I34KDC1O18"
    },
    {
      "name": "Midwest Primary Bank, NA",
      "type": "DATA_PROVIDER",
      "homeUri": "https://www.midwest.com",
      "logoUri": "https://www.midwest.com/81d88112572c.jpg",
      "registry": "GLEIF",
      "registeredEntityName": "Midwest Primary Bank, NA",
      "registeredEntityId": "549300ATG070THRDJ595"
    }
  ],
  "createdTime":"2021-07-03T21:28:10.375Z",
  "updatedTime":"2021-12-14T15:45:16.987Z",
  "durationType":"TIME_BASED",
  "lookbackPeriod":"60",
  "resources":[
    {
      "resourceType":"ACCOUNT",
      "resourceId":"b14e1e714693bc00",
      "dataClusters":[
        "ACCOUNT_DETAILED",
        "TRANSACTIONS",
        "STATEMENTS"
      ]
    }
  ],
  "links": [
```

```
    {
      "href": ".../consents/9585694d3ae58863/revocation",
      "action": "GET",
      "rel": "revocation"
    }
  ]
}
```

14.4.5 Future API Operations

A number of additional Consent API operations have been considered and will be included in future versions of the specification. These include (non-exhaustive) operations for:

- Consent Grant discovery
- Consent Modification
- Additional mechanisms to enable Consent Portability

14.5 ConsentGrant ↔ OAuth Scope Alignment

While the User Consent flow defines a fine-grained approach to authorization within the FDX ecosystem, scope alignment seeks to “close the gap” in implementing FDX Consent Framework via OAuth. It complements Granting of Consent, Recipient Registration, and recommended Security Profile specified by FDX. Scope alignment offers a unified path across:

- *Recipient Registration* — defining the total universe of data-sharing available between a DP and DR
- *Consent Request and Grant* — defining the specific data clusters and resource types desired for a particular user’s connectivity
- *Authorization Token Format & Lifecycle* — defining the scope of authorization permissioned by the user for data exchange via API operation

14.5.1 FDX JWT Profile

A JWT-compliant OAuth `access_token` may be used to transport some details of the `ConsentGrant`: the unique identifier (`consentId`) and FDX data clusters. Together, these "FDX scopes" may be declared as optional members of the Authorization Claim, aka `scope` parameter. No changes to the token issuance flows, grant types, or other OAuth API operations are intended.

Data Cluster Scopes

FDX scopes are constructed as follows:

- Scopes will prepend an `fdx` prefix to encourage interoperability with other standards and avoid potential scope name collisions.
- Scopes will append an operation identifier to explicitly declare what actions are authorized.

Example: FDX prefix + separator + *Data Cluster name* + separator + *Operation verb*

The **Data Cluster name** is written in lower case and will include only alphanumeric characters without spaces. For example, the scope to allow reading transactions from a bank account would be: `fdx:transactions:read`. (All existing Data Cluster-Operation pairs supported by FDX are noted in the table below.)

The data cluster scopes included in an `access_token` should be identical or narrower subset of scopes authorized by the `ConsentGrant` if the DP applies principles of least-privilege to its API authorization.

FDX ConsentGrant Identifier (the Consent Id) Claim

Authorization Service may optionally convey to the resource server FDX `consentId` within the `access_token` included in a Private Claim claim named `fdxConsentId`. The `fdxConsentId` claim contains a single string member; this member is the identifier of the current user's active `ConsentGrant`.

Example Decoded access token using FDX JWT Profile

```
{
  "iss": "https://authorization-server.example.com/",
  "sub": "5ba552d67",
  "aud": "https://rs.example.com/",
  "exp": 1639528912,
  "iat": 1618354090,
  "jti" : "dbe39bf3a3ba4238a513f51d6e1691c4",
  "client_id": "s6BhdRkqt3",
```

```
  "scope": "fdx:transactions:read fdx:statements:read openid
offline_access",
  "fdxConsentId": "28920138920832"
}
```

14.5.2 Use in Token Introspection

These data elements may optionally be used in the token introspection response, extending RFC 7662: OAuth 2.0 Token Introspection, for use in implementations that utilize opaque access tokens.

```
{
  "active": true,
  "iss": "https://authorization-server.example.com/",
  "sub": "5ba552d67",
  "aud": "https://rs.example.com/",
  "exp": 1639528912,
  "iat": 1618354090,
  "jti" : "dbe39bf3a3ba4238a513f51d6e1691c4",
  "client_id": "s6BhdRkqt3",
  "scope": "fdx:transactions:read fdx:statements:read openid
offline_access",
  "fdxConsentId": "28920138920832"
}
```

14.5.3 Data Cluster ↔ Scope Mapping

Data Cluster	*API Operation*	*FDX Scope*	*Description*
ACCOUNT_BASIC	read	`fdx:accountbasic:read`	Authorizes retrieval of account display name, masked account number, type, description

Data Cluster	*API Operation*	*FDX Scope*	*Description*
ACCOUNT_DETAILED	read	`fdx:accountdetailed:read`	Authorizes retrieval of account information. Includes data from "Basic" + account balances, credit limits, due dates, interest rates, rewards balances, recurrences
BILLS	read	`fdx:bills:read`	Authorizes retrieval of bill payments data
CUSTOMER_CONTACT	read	`fdx:customercontact:read`	Authorizes retrieval of name, email, address, phone on file with this institution. Applies to both user and other account holders
CUSTOMER_PERSONAL	read	`fdx:customerpersonal:read`	Authorizes retrieval of name, email, address, phone, DoB, tax ID, SSN on file with this institution
IMAGES	read	`fdx:images:read`	Authorizes retrieval of check and receipt images, which may include PII such as name, full account and routing number
INVESTMENTS	read	`fdx:investments:read`	Authorizes retrieval of details about individual underlying investment positions

Data Cluster	*API Operation*	*FDX Scope*	*Description*
NOTIFICATIONS (SUBSCRIBE)	write	`fdx:notifications:subscribe`	Authorizes creation of a new notification subscription
NOTIFICATIONS (PUBLISH)	write	`fdx:notifications:publish`	Authorizes publishing of a new notification
PAYMENT_SUPPORT	read	`fdx:paymentsupport:read`	Authorizes retrieval of full account number and payment network routing number
REWARDS	read	`fdx:rewards:read`	Authorizes retrieval of loyalty rewards programs, transactions and balances
STATEMENTS	read	`fdx:statements:read`	Authorizes retrieval of periodic PDF statement showing personal information, account and transaction details. May contain PII such as name, address
TAX	read	`fdx:tax:read`	Authorizes retrieval of information reporting tax forms and data
TRANSACTIONS	read	`fdx:transactions:read`	Authorizes retrieval of historical and current transactions, transaction types, amounts, dates and descriptions

15. Event Notification Framework

An Event Notification Framework has been established in the FDX API as of version 5.1. It is described as the *Event Publication and Subscription API* in the *fdxapi.event-notifications.yaml* file.

The framework will notify subscribed entities of an event. This enables the subscriber to trigger further actions if needed, for example if a customer has revoked consent, has new transactions or an FI has a planned outage. See the second sample payload below for a revoked consent.

This notification process has numerous benefits including reduction of unnecessary calls, reduction of network traffic on the Data Provider side, and provides the most current information.

The Notification Framework should be utilized via the following methods:

- Clients must be identified by a client registration process to include Party Type and Party Name.
- Refer to the *FDX API Security Model* document for securing endpoints.
- During authentication use the following scopes:
  - Scope - `fdx:notifications:subscribe`
  - Scope - `fdx:notifications:publish`
- Party identifier information is included for informational purpose and it is not validated against the FDX registry.
- Notification Subscription
  - The subscription is a specific client request (POST method) to start receiving notifications from the publishing entity.
  - A `201: Created` success response is sent back to the requesting entity if the subscription is successful.
  - The notification type and category need to be set in each subscription and must follow the list of supported types and categories as per the API specification.
  - Only one notification subscription can be requested at a time.
  - An effective date by subscriber for each notification type is optional.
  - The callback URL should specify the API endpoint for the publisher to send the notification.
- Publish notification
  - Publish notification reacts to an event (e.g. consent change, fraud etc.) and makes an outbound API call to all previously subscribed entities.
  - The notification should have identifiable attributes that can be matched to the subscription request to determine if a notification needs to be published. E.g.: If a customer updates consent for an account on the Data Provider side, this should trigger an internal lookup to see if there are subscribers who have requests for this data by matching the Party information that was tied to the consent record. (It is assumed that Consent captured by Data Providers will have metadata that includes customer and party information).
  - Subscribers are expected to acknowledge receipt of the notification with the appropriate `204: No Content` success response.

- Resiliency guidelines and best practice recommendation should be implemented for up to three retries on the `/notifications` endpoint.
- The notification is not intended to trigger a specific business process unless the subscriber decides to do so and therefore the success response should not wait for completion of any business processes that may be triggered.
- `notificationPayload` will be defined by use case.
- Notification payloads shall not contain any sensitive data. For detailed information refer to *Part 3: Best Practices for Sharing Sensitive Data* of the *FDX API Security Model* document.
- The header information on the HTTP request will follow the approved FDX specification as defined in the *fdxapi.event-notifications.yaml*.

15.1 Event Notifications Implementation for Consent Updates

Payload Schema

The event notification framework is rich and extensible to support many scenarios. For consent update notifications, it is only necessary to satisfy the following sub-schema:

```
    Notification:
      properties:
        notificationId:
          type: string
          description: Id of notification
        type:
          $ref: '#/components/schemas/NotificationType'
          description: Type of notification
        sentOn:
          $ref:
'./fdxapi.components.yaml#/components/schemas/Timestamp'
          description: Time notification was sent
        category:
          $ref: '#/components/schemas/NotificationCategory'
          description: Category of notification
        publisher:
          $ref:
'./fdxapi.components.yaml#/components/schemas/Party'
          description: Publisher of notification
        subscriber:
          $ref:
'./fdxapi.components.yaml#/components/schemas/Party'
          description: Subscriber to this notification
```

```
        notificationPayload:
          $ref: '#/components/schemas/NotificationPayload'
          description: Notification-specific key-value paired
data
        url:
          $ref:
'./fdxapi.components.yaml#/components/schemas/HateoasLink'
          description: URL to retrieve further details related
to notification
      required:
        - notificationId
        - type
        - sentOn
        - category
        - publisher
        - notificationPayload

    NotificationPayload:
      properties:
        id:
          type: string
          description: ID for the origination entity related to
the notification
        idType:
          $ref: '#/components/schemas/NotificationPayloadIdType'
          description: Type of entity causing origination of the
notification with the given ID
        customFields:
          $ref:
'./fdxapi.components.yaml#/components/schemas/FiAttribute'
          description: Custom key-value pairs for a notification
```

As follows, for a revoked consent:

```
{
  "category": "CONSENT",
  "type": "CONSENT_REVOKED", // or "CONSENT_UPDATED"
  "notificationPayload": {
    "id": "9585694d3ae58863",
    "idType": "CONSENT",
    "customFields": {
      "name": "INITIATOR",
```

```
      "value": "INDIVIDUAL"
    }
  },
  "url": {
    "href": ".../consents/9585694d3ae58863/revocation",
    "action": "GET",
    "rel": "consent"
  }
}
```

Using the HATEOAS URL optionally contained in the response, the notification receiver can fetch the revoked (or updated) consent and take any relevant next steps. For example, the receiver can send an in-app notification confirming that access was changed.

Example Payloads

High-level scenario: *User has made a change to the accounts or data clusters shared*

Example: *User is using a fictional app with both PFM and money movement features called Wonder Wallet. The user has been sending funds with the app but no longer wants to; they just want to use it as a PFM. The user revokes access to* ACCOUNT_PAYMENTS *on their Data Provider's portal, and the Data Provider sends a notification to Wonder Wallet.*

```
{
  "notificationId": "req123456-GUID",
  "type": "CONSENT_UPDATED",
  "sentOn": "2024-03-12T15:47:42.375Z",
  "category": "CONSENT",
  "publisher": {
    "name": "MyBank Inc",
    "type": "DATA_PROVIDER"
  },
  "subscriber": {
    "name": "Wonder Wallet",
    "type": "DATA_RECIPIENT"
  },
  "notificationPayload": {
    "id": "9585694d3ae58863",
    "idType": "CONSENT",
    "customFields": {
      "name": "INITIATOR",
      "value": "INDIVIDUAL"
    }
  },
```

```
  "url": {
    "href":
"https://api.mybank.com/fdx/v6/consents/9585694d3ae58863"
    "action": "GET",
    "rel": "consent"
  }
}
```

High-level scenario: *Business rule has revoked consent*
Example: *The user decides to end their relationship with their data provider. As part of this process, the Data Provider needs to revoke consent to any Data Recipients for which the user had consented access. The Data Provider will send a notification to Wonder Wallet alerting them that consent was revoked.*

```
{
  "notificationId": "req123456-GUID",
  "type": "CONSENT_REVOKED",
  "sentOn": "2024-03-15T14:46:41.375Z",
  "category": "CONSENT",
  "publisher": {
    "name": "MyBank Inc",
    "type": "DATA_PROVIDER"
  },
  "subscriber": {
    "name": "Wonder Wallet",
    "type": "DATA_RECIPIENT"
  },
  "notificationPayload": {
    "id": "ConsentID-1",
    "idType": "CONSENT",
    "customFields": {
      "name": "INITIATOR",
      "value": "DATA_PROVIDER"
    }
  },
  "url": {
    "href": "https://api.mybank.com/fdx/v6/consents/ConsentID-
1/revocation"
    "action": "GET",
    "rel": "consent"
  }
}
```

Calling back for detail:

When consent changes, the observer will send a notification request to the receiver with a minimal payload containing a description of the change and the Consent ID. Upon receipt, the receiver MAY call back to the sender to fetch a full view of consent. For example, if the user changed which accounts were shared with a Data Recipient, the Data Recipient will receive a notification message containing CONSENT_UPDATED and the Consent ID.

Appendix - Attributions

This API specification and accompanying documentation contains the intellectual property of ACORD Corporation. Used with Permission. All rights reserved.

The X9 BTRS/BTR3 standard (formerly named BAI2) for corporate and treasury data elements is proposed to provide the basis of (or starting point for) representing summary, account, and transaction codes for the Corp/Treasury Entity, subject to any necessary modifications useful to the members of the FDX consortium and users of the API specification. See: https://x9.org/standards/btrs/.

Appendix - Change Log - Prior Releases

Version	Date	Originator/ Contributor	Reason for Change	Ratified
1.0	May 2015	Anoop Saxena	Initial Document	Yes
2.0	December 2017	Anil Mahalaha	Entities Updated: 12.7 Capability Entity -- new fields 12.12 Debt Security - new fields like call type 13.19 Holding - one change 12.27 LOC Account	Yes
			Entities Added: 12.49-12.57 Tax Form Data 12.58 Insurance Account 12.59 Insurance Transaction 12.60 Bill 12.61 Pension Source * 12.62 Annuity Account	
			Simple Types Added: * 13.53 Annuity Product * 13.54 Annuity Value Basis * 13.55 Annual Increase Type * 13.56 Period Certain Guarantee	
2.1	September 2018	Anil Mahalaha	Updated Section 8 - Logical Data Model	Yes
			Section 7: Removed security guidelines in favor of FS-ISAC Aggregation Services Working Group's document "Control Considerations for Consumer Financial Account Aggregation Services - Reference Security Architecture and Standard Specification" Version 2.1 -June 2018 for securing FDX API.	
			Section 7: Add Scopes (Tax and Insurance)	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			Section 8: Changed to allow Max 256 Char	
			Section 10: Changed Error 501 to return HTTP Status Code 500	
			Section 11: Added Cursor based Pagination	
			Section 12: Removed sample Request/Response in favor of accompanying OpenAPI Specification schema document ddav2.1_oas3.yaml * Added GET methods to all POST only resources * Changed Resource names to spinal-case * Added Resource /account/tax	
			Section 13: Added Versioning	
			Section 14: * Account Entity - Added accountNumberDisplay field * Accounts Entity - Added AnnuityAccount & InsuranceAccount * InsuranceAccount Entity - Added Array of Insurance Transactions * Investment Account Entity - Added Array of PensionSource * Added Tax Entity * Removed all references to XML	
3.0	May 2019	Anil Mahalaha		Yes
4.0	September 2019	Ravneet Singh	Major restructuring changes as outlined in the REST Best Practices RFC and OpenAPI and PDF Inconsistencies RFC. The API includes updates to comply with _REST best practices_ and is now fully REST compliant. This is largely a technical structural change to the specification that makes for easier and more consistent implementations. Resources were relabeled to comply with REST conventions and HTTP methods are now aligned with RESTful recommendations. Different content types are served from the same endpoint. _HATEOAS_ type links are used in tax document searches, pagination links, and transaction image results. The _GET_ query parameter is used to determine how many fields to return.	Yes

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Clyde Cutting Bruce Wilcox	Added complex types for tax and tax entities as described in the FDX 4.0 Tax Entities and FDX 4.0 Tax APIs RFCs. There has been a complete restructuring of how tax is handled within the specification. It is a breaking change, but necessary to resolve issues and to support the exchange of tax forms and form data between tax data suppliers and tax data access platforms. There is a complete replacement of the Tax Entities for FDX version 4.0. These entities include significant breaking changes to the Tax Entities previously published in FDX version 3.0 documentation. This change was necessary as those entities had significant issues that prevented use by tax software and tax preparation companies, such as missing data elements, incorrectly typed data elements, not extending the base Tax entity, etc. 58 new entities have been introduced, including 34 entities corresponding to various types of tax forms. New Tax APIs have been added to version 4.0 that replace the previously published single Tax API. These APIs are an entirely new suite of resource endpoints to support the exchange of tax forms and tax form data between tax data suppliers and tax data access platforms.	
		Anil Mahalaha	BOND has been added as a _securityType_. OTHER has been added as a _holdingType_. Updates to telephone number, address, and name handling as described in the following RFCs: • TelephoneNumber RFC • DeliveryAddress RFC • CustomerName RFC The length of _TelephoneNumber_ entity has been increased from 10 digits to 15 digits to support international subscriber telephone numbers which can have lengths greater than 10, per ITU-T recommendation E.164.	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			A new _base address entity_ is included in API 4.0. This simplifies increased the consistency of any part of the specification where an address is required. For example, _DeliveryAddress_ entity now extends and inherits all fields from address instead of merely duplicating some of the elements as previous. A new _IndividualName_ entity has now been added to the specification to simplify and provide consistency wherever 'human' names are required in the spec. For example, _CustomerName_ is now modified to use the new _IndividualName_ entity instead of defining each name element separately.	
		Jonathan Kassan	Update of Accounts and AccountDescriptor entities as described in Account Descriptor RFC (formerly Combining Account Entity & AccountDescriptor). The AccountDescriptor entity was missing currency, which was added. The displayName field on the AccountDescriptor and InsuranceAccount entities referenced a confusing concatenation ("Account identity to display to customer. This may be a masked account number or product name followed by masked number) which is now separated into two fields: productName and accountNumberDisplay. _AccountDescriptors_ have been modified to better distinguish the fields used to describe an account. New elements _productName_ , _nickName_ , and _accountNumberDisplay_ have been added to provide an easier, more consistent way to assist the end user in account selection. • _nickName_ is the user specified name for the account, used in user interfaces (UIs) to assist in account selection • _productName_ is the Marketed product name for this account, used in UIs to assist in account selection	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			• _accountNumberDisplay_ is typically the masked account number that is displayed on a bank's UI.	
	February 2020	Rich Dudley	Errata	
			OPTIONS (plural) has been changed to OPTION (singular) in _holdingType_.	
			MILITARYLOAN and INSTITUTIONALTRUST have been fixed in _AccountType_.	
			Indentation corrections were made for occurrences of "Accounts" and "AccountDetailsRequest" in the fdxapi4 YAML file.	
			API Data Structures Documentation edits: • "Th" has been corrected to "The" in section 6 Message Syntax". • Accounts DetailsRequest" (plural) has been changed to "AccountDetailsRequest" (singular).	
4.1	April 2020	Paul Allen Clyde Cutting Bruce Wilcox	Implemented the final IRS 2019 tax form changes for Tax1065K1, Tax1120SK1	
			Corrected the Tax1099Oid field from "description" to "oidDescription"	
			Updated the tax form entity descriptions to match the YAML file	
4.2	October 2020	Ravneet Singh	Core errata: RFC 0045: Updated AnnuityAccount to inherit from Account so they can be returned in results of /accounts API	Yes
		Bruce Wilcox Clyde Cutting Paul Allen	Core errata: RFCs 0069-0071: Corrected the naming of Holding.inv401kSource and TaxLot.positionType properties and deprecated original inv401kSurce and postionType properties. Added FEE and INTEREST enumeration values to DepositTransactionType, and a NONE enumeration value to PeriodCertainGuarantee	
		DevCon 4.0 and Tax Taskforce	RFC 0014: Removed Boolean and Number simple types and replaced all their references to types boolean and number. Removed unused types String2, String9,	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			SingleAccountDetailsRequest and AccountDetailsRequest	
			RFC 0023: Replaced references to once-used simple types String3, String10 and String15 with type: string and maxLength: 3, 10, 15	
		Bruce Wilcox Clyde Cutting Paul Allen	RFCs 0015, 0022, 0030: Split single combined FDX API Specification documentation and yaml into separate Core and Tax yaml (and combined yaml) and three documents: FDX_Specification, FDX_Core_API and FDX_US_Tax_API. Implemented generation of the latter two specification documents directly from the yaml to create consistency and remove manual maintenance steps. This included numerous updates to description attributes in yaml files so that generated documents retained completeness and consistency with prior versions	
			RFCs 0028-0029, 0031, 0038, 0041-0044, 0046-0050, 0060, 0062, 0064-0065: Implemented new IRS tax reporting information forms and statements Tax1042S, TaxW2C, Tax1099Nec, Tax5227K1, Tax1098Q, Tax1099Ls, Tax1099Sb, Tax3921, Tax3922, Tax1098Ma, Tax1099Qa, Tax5498Qa, BusinessIncomeStatement, FarmIncomeStatement, FarmRentalIncomeStatement, RentalIncomeStatement and RoyaltyIncomeStatement	
			RFCs 0036, 0057-0059: Implemented the IRS 2020 tax form changes for Tax1099Misc, Tax1095C, Tax1099Patr and Tax1120SK1	
			RFC 0024: Replaced all Tax entity references to Timestamp for dates with a new DateString simple type	
			RFC 0034: Added a generic TaxFormAttribute property to base Tax entity to permit universal backwards/forwards compatibility using name/value pairs	
			RFC 0056: Added an Error property to base Tax entity to permit provider communication of errors on tax forms	
			RFCs 0032, 0039, 0059: Matched to OFX schemas by adding ESPP properties to TaxW2, a studentTinCertification property to	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			Tax1098T, and a secondTinNotice property to Tax1099B, Tax1099Div, Tax1099G, Tax1099Int, Tax1099K, Tax1099Nec, Tax1099Oid and Tax1099Patr	
			Tax errata: RFCs 000035, 0037, 0055: Corrected and deprecated existing misnamed properties for spelling and consistency on Tax5498Esa, Tax1065K1, Tax1095B and Tax1095C, adding new properties with correct names	
4.5	December 2020	Ravneet Singh	RFC 0017: Add three new error codes 401, 500 and 704 to section 9.2	Yes
			RFC 0005: Account Number for payment networks, including Tokenized Account Numbers. New endpoint /accounts/{accountId}/payment-networks, new entities AccountPaymentNetworkList and AccountPaymentNetwork, new types PaymentNetworkIdentifierType and PaymentNetworkType	
			RFC 0004: Account Holder information, with new endpoints /accounts/{accountId}/contact, /customers and /customers/{customerId}; new entities AccountHolder, AccountContact, and CustomerToAccountRelationship; new type AccountHolderRelationship; and Account.contact and Customer.accounts properties	
4.6	May 2021	Ravneet Singh	RFC 0040: Adds the ability to execute merchant bill payments using the FDX API. The Money Movement API makes use of Payee, Payment and Recurring Payment entities with their corresponding /payees, /payments and /recurring-payments endpoints.	Yes
			RFC 0063: Define the end-to-end encryption process within the FDX API documentation as defined in RFC 0011 Message Encryption. The changes are additive to RFC 0011 and have not redefined any part of RFC 0011.	
		Cort Pahl	RFC 0106: Adds enums for new Investment, Annuity, and Insurance account types.	
		Gimus Young	RFC 0075: Adds enums for Deposit and Investment account types.	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Jeff Johnson Abbas Moosavi	RFC 0093: Adds the ability to associate rewards with an account since individual customers may have multiple accounts, such as personal or business. This allows for `AssociatedAccounts` tied to a balance which can indicate the specific customer level relationship. Adds new endpoints /reward-programs, /reward-programs/{rewardProgramId}, /reward-programs/{rewardProgramId}/categories	
		Clyde Cutting Bruce Wilcox Paul Allen	RFC 0110: Corrected taxFormId parameter to Identifier ref in /tax-forms, renamed two more elements for consistency, and adedd two missing elements on Tax1042S.	
		Bruce Wilcox Clyde Cutting	RFC 0118: Describes how to use the specification to print tax data QR Codes on tax documents.	
5.0	October 2021	Clyde Cutting	RFC 0033 DateString type was added in v4.2. This replaces 53 uses of Timestamp references with _DateString_ e.g., dateOfBirth will now be a _DateString_, not a Timestamp.	Yes
		Clyde Cutting	RFC 0074 v5.0 Deprecation Cleanup, TY 2020	
		Cort Pahl	RFC 0090 Payment is changed to Payout, with other enums being added and deprecated	
		Ravneet Singh	RFC 0101 Internal Transfers This capability allows the authorized user to transfer funds from her own account to another account within the same financial institution. It adds an ability to handle duplicate transactions, search for transactions, cancel transfers, and retrieve transfers. The standalone status retrieval endpoint is now deprecated in favor of _getTransfer_. Other changes were renamed _Transfer_ to _TransferForCreate_, defined a Transfer that inherits from _TransferForCreate_, created the Transfers entity, edited '/transfers', made	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			'IdempotencyKeyHeader' required, and removed the 'TransferStatusStatus' enum.	
		Roberto Irribarren Clyde Cutting	RFC 0104 Adds _CertificationMetrics_ and _/certification-metrics_ entities.	
		Cort Pahl	RFC 0105 Add New Policy Statuses for Insurance Products Adds a new attribute _policyStatus_ to _InsuranceAccount_ entity, along with the enumeration of policy statuses. This now allows the implementers to accurately represent the following insurance policy statuses via the FDX API: ACTIVE, DEATH_CLAIM_PAID, DEATH_CLAIM_PENDING, EXPIRED, GRACE_PERIOD, LAPSE_PENDING, TERMINATED, WAIVER.	
		Clyde Cutting	RFC 0109 FDX v4.5 implemented 3 RFCs (0069, 0070, and 0071) which left in place deprecated components. Removing them are breaking changes which we can now do in version 5.0.	
		Clyde Cutting	RFC 0111 Modifies the existing _links:_ properties for simpler and consistent implementation by providers across all API responses to make it an array everywhere, and embed the `rel` value inside link object. It results in consistent references to _HateoasLink_ and unambiguous context associations with API operations through new `rel` property, and a consistent model for future use of _HateoasLink_.	
		Roberto Irribarren	RFC 0112 Update Capability Endpoint for Certification Allows a Data Recipient to query a Data Provider to get both the FDX Data API Host and the URI. Currently, this has to be done manually so this new capability will speed implementation and promote interoperability by introducing endpoint discoverability in the spec.	
		David Biesack	RFC 0113 Updates schema documentation for Timestamp as string format date-time	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Clyde Cutting	and describes the detailed pattern as `YYYY-MM-DDThh:mm:ss.nnn[Z [+/-]hh:mm]`.	
		Clyde Cutting	RFC 0114 Fixes the startTime and endTime format restrictions in the /accounts API to make them consistent with other APIs.	
		Joseph Heenan	RFC 0119 Changes naming of request header in section 9.1.9 of FDX API Specification from `API-InteractionId` to the international standard, `x-fapi-interaction-id`.	
		Vivek Gandhi William Rowan Kyle Caldwell	RFC 0124 Fetch Data from Providers to Assess Fraud, and Return Fraud Signals Back to the Provider. Adds _SuspectedFraudIncident_ for fraud notification and _Party_ and _PartyType_ entities.	
		John Becaley Josef Corkery	RFC 0126 Proposal to Update TaxID Element Description Changes the description of _taxId_ element so it now allows country specific tax ids – SIN (Social Insurance Number) or NAS (Numero D'assurance Sociale) for Canada – in addition to SSN and TIN for the US.	
		Ravneet Singh	RFC 0136 Recurring Payment Lifecycle Creates a new status type _RecurringPaymentStatus_ to distinguish recurring payment statuses from payment statuses to better represent the recurring payment lifecycle. Also adds _scheduledTimestamp_ to both _Payment_ and _RecurringPayment_ entities to mark when the payment or the recurring payment was first created. This now allows better tracking of the payment and recurring payment lifecycles.	
		John Becaley	RFC 0141 Enhance the Values Associated with the Status Element Often an account is restricted for various reasons e.g., suspected fraud, deceased owner, internal issues, or legal issues, due to which certain operations may be prohibited. This proposal introduces a general status of RESTRICTED that can be returned when an account is blocked for requested operation.	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			No other information is returned in order to protect the user's privacy.	
		Clyde Cutting	RFC 0142 Tax 1099-B: Add Expired Options Flag	
		Bruce Wilcox Clyde Cutting	RFC 0143 Tax: New Cryptocurrency Tax Statement entity	
		John Becaley Josef Corkery	RFC 0144 Changes postalCode length to 16, adds _region_ property to replace _state_ and deprecates the _state_ property.	
		Clyde Cutting	RFC 0145 Fixes current schema validation errors in Core schema: Entity _AccountDescriptorList_ is defined, but not referenced anywhere Entity _AccountStatus_ is defined, but not referenced anywhere _AccountDescriptor_ entity inlines the _AccountStatus_ enums, should be a $ref	
		Clyde Cutting	RFC 0146 The v4.6 FDX YAML uses OpenAPI Spec v3.0 which was released 7-26-2017. This has been updated to OpenAPI Spec v3.1, released on 2-15-2021. Updating to v3.1 makes the schema valid and supported for tools that support OAS 3.1. FDX YAML will be backwards-compatible for tools which only support OpenAPI Spec v3.0.	
		Clyde Cutting	RFC 0147 Add FDX Tax1099ConsolidatedStatement	
		Camellia George Ben Simmons Lukasz Jaromin	RFC 0156 Consent API: Request, Issue, and Retrieval Operations and Data Structures	
		Josef Corkery	RFC 0157 Represent Card Art and Logo/issuer for an account Adds a link to the card art and logo to the LocAccount entity. Data Recipients highlighted this feature as highly beneficial as it gives them the ability to represent the	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			customers card as they would see in channels today. This information is currently scraped by the Data Recipients.	
		Josef Corkery	RFC 0158 Change Account relationship type list Adds _AUTHORIZED_USER_ to _AccountHolderRelationship_ entity. It is fairly common to have authorized users on a credit card. This change allows representation of such a customer relationship to the account	
		Shishir Bhat John Becaley	RFC 0159 Add Values to TransactionType ENUMs for LOAN Transactions Adds the following new enumerations to transactionType: LUMP_SUM_PAYMENT, SKIP_PAYMENT, DOUBLE_UP_PAYMENT, PAYOFF. This allows for a more accurate representation and analysis of lump sum payments (a single payment of money), skip payments (mortgage required payment deferral), double ups (additional payment to the requirement payment to reduce the principal), and payoff (a payment that satisfies the terms of loan and completely pays off debt).	
		Clyde Cutting	RFC 0161 _/availability_ schema and endpoint changes to support RFC 112's _/capability_ schema changes.	
		Clyde Cutting	RFC 0163 FDX IRS 2021 tax form changes: 1099-DIV, 1099-NEC, 1099-MISC	
		Bruce Wilcox Clyde Cutting	RFC 0165 FDX US Tax v5.0 synch with OFX Tax 2020.0: Add Tax1099B.securityDetails.correctedCostBasis	
		Clyde Cutting	RFC 0174 Many APIs use a number of common parameter definitions, re-defining them inline in the API. The OpenAPI Specification allows common definitions of parameters.	
		Bruce Wilcox Clyde Cutting	RFC 0175 US Tax - FDX-Enriched PDF with embedded data	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Clyde Cutting	RFC 0184 Switch Tax API request/response arrays to TaxDataList entity	
		Bruce Wilcox Clyde Cutting	RFC 0189 FDX 5.0: match OFX 2021 adding investment sale types to 1099-B schema	
		Clyde Cutting	RFC 0190 Corrections to the Rewards feature. Adds limit and offset queries to categories, _/ProgramID_ to RewardCategory, and typo corrections.	
		David Biesack	RFC 0191 OpenApi Spec errata - removed _Accept_ header.	
		API Authoring Task Force	RFC 0192 Removal of Codegen yaml	
		Clyde Cutting	RFC 0193 Renamed _operationId_ to _getPaymentsForRecurringPayments_.	
5.1	May 2022	Randy Brandt Susan Pandy Jason Kocherhans	RFC 0102 Data for Account Owner Verification - Certification Use Case Defines the data set that is used by Data Recipients to verify their end-user information against owner information and account information of the bank account claimed by end-user as belong to him or her.	Yes
		Randy Brandt Susan Pandy Jason Kocherhans	RFC 0103 Data for Account Linking (for Payments) -Certification Use Case Defines the data set that is used by Data Recipients to put end-user's payment account information on file within their product for the purposes of withdrawing funds from, or depositing funds into the specified payment account using a sperate payment processing company.	
		Ravneet Singh Julie Jackson	RFC 0148 Add `US_RTP` to `PaymentNetworkType` Enumeration	
		Jeff Schulte Lukasz Jaromin	RFC 0153 Recipient Registration Automation Establish Data Recipient Registration requirements and guidelines.	
		Dinesh Katyal	RFC 0181 Errata: Move `AccountCategory` from `InsuranceAccount` to `AccountDescriptor`	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Clyde Cutting		
		Ravneet Singh	RFC 0187 Add `debugMessage` to the `Error` entity	
		Anoop Saxena Anil Mahalaha Ravneet Singh Ray Voss Ben White Alwin M. Thomas Max Rozen	RFC 0188 Event Notifications Framework Creates a mechanism/framework to communicate between entities when a customer or entity event occurs. The framework notifies subscribed entities of the occurrence of the notification that will trigger further actions if needed (eg: customer has new transactions, FI has planned outage etc).	
		Yumiko Kato Clyde Cutting	RFC 0194 API Enhancements for Crypto Extend the FDX data model to address treatment of accounts, holdings and transactions for cryptocurrencies and other digital assets as held by retail consumers. New enum and transaction values are proposed to represent cryptocurrency transactions, especially those with tax implications.	
		Camellia George Ben Simmons	RFC 0195 Consent API: Revocation A new addition to the Consent API that allows end users to revoke a consent that has been previously granted, as defined in RFC 0156 (please see `POST /par` Step 10). The ability to revoke consent is a critical component of the consent lifecycle. If a data recipient and/or data provider allows end users to grant consent, it must also provide a way to revoke it. A standardized API supporting this flow will ensure FDX members can easily introduce this functionality to their end users.	
		Max Rozen Ben White	RFC 0198 Consent Event Notifications Consent Management enables End Users to view and revoke consent at multiple entities, including Data Providers, Data Access Platforms, and Data Recipients. As consent actions occur, three parties (DP, DR, DAP) need to stay in sync to ensure two outcomes: • End User Transparency: An end user's consent must be displayed accurately	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			across all parties in the user experience (e.g., consent edit at DP must reflect at DAP dashboard and DR's app; a revocation at a DAP must reflect at a DP's consent management dashboard). • Data Minimization: End User data can only be shared with consent, so when changes are made all parties must be notified such that data retrieval promptly reflects consumers choices.	
		Ravneet Singh	RFC 0202 Asynchronous Statements Statements retrieved via the FDX API might require additional processing that cannot be completed synchronously. For example redacting sensitive data. Asynchronous Statements modifies the FDX API such that statements can be requested and generated asynchronously.	
		Ravneet Singh	RFC 0204 Tax API Response Error Codes Defines error codes for tax retrieval operations.	
		Ben White Ray Voss Michael Cypher	RFC 0206 Recipient Registration With Delegation to an Ecosystem Registry Defines a mechanism by which Data Providers delegate Recipient registration to Recipient Registries. This mechanism could be Data Access Platforms allowing Data Providers to quickly onboard and update information on Data Recipients, both during and outside of the user experience.	
5.2	December 2022	Cristian Popescu Ravneet Singh	RFC 0208 Payment Initiation Party Maintenance	Yes
		Clyde Cutting	RFC 0217 Add IETF RFC 2119 keywords MUST, SHOULD, etc in API Specification document	
		Clyde Cutting	RFC 0229 Errata to Externalize Embedded Enums	
		Clyde Cutting	RFC 0231 Add Domicile to Account entity	
		Camellia George Chris Kennedy	RFC 0233 Extension of Transaction and Account Entities	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		David Biesack Camellia George Liza Moran	RFC 0234 Extension of Customer Entity	
		Ravneet Singh Dominiqu e Shells	RFC 0083 Investment Transaction Settlement Date	
		Josef Corkery	RFC 0160 Additional values in the AccountType Enum List to handle Canadian values	
		Clyde Cutting David Biesack Bruce Wilcox	RFC 0225 Supporting Tax business processes with HTTP status 412 error code 1205	
		Clyde Cutting Bruce Wilcox	RFC 0230 Tax API errata - Document use of IRS FIRE fields in Tax.attributes	
5.3	June 2023	Ravneet Singh	RFC 0149 Invalid Auth Token Error Code	Yes
		Ravneet Singh	RFC 0154 Make 'Customer not authorized' HTTP Status 403	
		Ravneet Singh	RFC 0207 Identify Batch vs Realtime Traffic	
		Ray Voss Wesley Dunningto n	RFC 0218 FDX Adoption of FAPI 1.0 Part 1: Baseline Please refer to FAPI 1.0 — Part 1: Baseline API Security Profile (final) The following considerations shall apply to all certified FDX data providers, certified data access platforms, and certified data recipients. The considerations refer to the FAPI specifications, and must be read in conjunction. The following document will only seek to clarify potential areas of concern. The use of key words is defined in RFC 2119 *Key Words for Use in RFCs to Indicate Requirement Levels*. RFC ft-bradner-key-words: Key words for use in RFCs to Indicate Requirement Levels	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Ravneet Singh Eli Abbe	RFC 0243 Add `currentAmortizationFactor` for Bonds (Mortgage Backed Securities) to Holding	
		Steven Gregory Elvin Prizmic	RFC 0244 Additional transaction types to be added to the FDX Specification for LoC Transactions	
		Chris Kennedy	RFC 0249 Account and Customer Details Extension	
		Ravneet Singh Jeff Johnson	RFC 0254 Update to Error Codes	
		Camellia George Lukasz Jaromin	RFC 0257 Consent Grant details ↔ OAuth Scopes alignment	
		Amy Fisher Nihar Nanavaty	RFC 0258 Description Enhancements for Digital Assets	
		Clyde Cutting	RFC 0260 V5.3 Cleanup from Authoring Taskforce	
5.3.1	June 2023	Stephen Weiss	DRAFT 0261 Errata: Commercial Accounts and Transactions descriptions missing in multiple spots	Yes
5.3.2	August 2023	Clyde Cutting Cristian Popescu	RFC 0277: v5.3.2, v5.2.3 Errata - example transferDate properties, ISO type descriptions, consent scope, phone country calling code, PaymentInitiationParty.name This errata release contains the following updates: • ISO Country Code and ISO Currency Code descriptions updated to match enumeration updates from RFC 0260 • Updates type definitions for scope in FdxTokenIntrospectionResponse and JwtProfile	Yes

Version	Date	Originator/ Contributor	Reason for Change	Ratified
			• Changes transferDate to transferTime in three places in fdxapi.money-movement.yaml • Changes example value country code from “US” to “+1” in nine occurrences of TelephoneNumber in fdxapi.money-movement.yaml • Modifies property name on PaymentInitiationParty to paymentInitiationPartyName	

5.3.3	November 2023	Clyde Cutting	RFC 0283 - Errata from Reference Implementation for API v6.0
			
			• Adds limit & offset request parameters to both:
			  ○ fdxapi.registry.yaml GET /recipients endpoint
			  ○ fdxapi.core.yaml GET /reward-programs endpoint
			• Updates DeliveryAddress type description in fdxapi.components.yaml to include all enum values
			• Clarifies wording of description for PaymentDeliveryAddress entity in fdxapi.money-movement.yaml
			• Adds enum values to descriptions of AccountPaymentNetwork properties identifierType and type
			• In fdxapi.components.yaml, changes the domicile reference from $ref: './fdxapi.components.yaml#/components/schemas/Domicile' to $ref: '#/components/schemas/Domicile'
				Yes

Version	Date	Originator/ Contributor	Reason for Change	Ratified
6.0	Dec 2023	Clyde Cutting	RFC 0232 Add discriminators to AccountDescriptor and Transaction (AccountCategory) and Security (SecurityCategory) **Note**: This contains significant breaking changes. Refer to the Migration Guide for details.	Yes
		Karen Boyer Jen Buchino	RFC 0236 Extend Fraud Notification Payload Data Structure **Note**: This contains a breaking change by adding a new enum value 'RISK' in NotificationType.	
		Jake Wainwright Clyde Cutting Nihar Nanavaty	RFC 0262 Asset-Transfer-Networks Endpoint (to pre-fill form for transfer initiation)	
		Florian Wahl	RFC 0266 Add US_FEDNOW, and RequestForPayments to Payments Network **Note**: This contains a breaking change by adding a new enum value 'US_FEDNOW' in PaymentNetworkType.	
		Florian Wahl	RFC 0267 Add PaymentNetworkTransferLimits to AccountPaymentNetwork	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Clyde Cutting	RFC 0268 Remove deprecations and obsolete markup for v6.0 **Note**: This change **REMOVES** several dozen deprecated elements and results in a breaking change. This includes **"state"** being removed from Address in fdxapi.components.yaml and replaces it with "**region"**.	
		Clyde Cutting	RFC 0269 Add definitions of FDX-API-Actor-Type header to API requests and x-fapi-interaction-id header to all API requests and responses **Note**: This contains a breaking change by adding required heading parameters to APIs.	
		Sugantha Nagarajan	RFC 0272 Required enums for Simple Types This creates a strategy for future RFCs and does not result in any direct changes to this version of the FDX API.	
		Neeraj Verma Justin Stolzenberg	RFC 0273 Payroll Data Structures for VOI/VOE This establishes a new fdxapi.payroll.yaml file in the FDX API for basic payroll verification of income and verification of employment.	
		Camellia George	RFC 0276 Endpoint Authorization per FDX Security Profile using SecuritySchemes	
		Clyde Cutting	RFC 0278 v6.0 consistency cleanup to match Style Guide **Note**: This changes an operationId which could be a breaking change for some implementations.	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Bruce Wilcox Paul Allen Clyde Cutting	RFC 0280 US Tax adding support for compound TaxStatements, common issuer and recipient entities **Note**: This contains structural breaking changes for implementers of fdxapi.tax.yaml.	
		Camellia George	RFC 0281 Adoption of X9 Standard	
		Clyde Cutting	RFC 0282 Updates for Consistent Property Naming for FDX API v6.0 **Note**: This renames several properties which results in breaking changes.	
		Clyde Cutting	RFC 0283 Errata from Reference Implementation for API v6.0	
		Clyde Cutting	RFC 0285 API Specification updates to match RFC 0276	
		Clyde Cutting Bruce Wilcox	DRAFT 0286 - Tax API 2023 Tax Year Errata plus RFC 0280 v6.0 corrections	
6.1	June 2024	Camellia George Clyde Cutting	RFC 0296 - Correct scopes for event-notifications and subscriptions	Yes
		Clyde Cutting Bruce Wilcox Paul Allen	RFC 0299 - Tax API and document updates for Spring 2024	

Version	Date	Originator/ Contributor	Reason for Change	Ratified
		Clyde Cutting	RFC 0300 - FDX API yaml updates to match FDX API Specification document	
		Neeraj Verma	RFC 0297 - Errata for Payroll VOE/VOI DataAsOf	
		Clyde Cutting	RFC 0288 - FDX API Specification V6.0 document errata	
6.2	September 2024	Katie Volz	RFC 0259 - Prepaid Account Types	Yes
		Clyde Cutting Doug Hall David Biesack	RFC 0319 - Include Digital Wallet Data Types in Account/Transaction Schemas	
		Clyde Cutting Francesca Loiodice	RFC 0314 - Errata to update Zimbabwe Currency Code (ZWL to ZWG) and non-material description updates	