FINANCIAL™
DATA EXCHANGE

API Specification

Version 5.1
May 2022

Legal Notice

This document is a product of the FDX API and Data Structures Working Group.

Revision History

Document Version	Notes	Date
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 - FDX API Specification

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

Version	Date	Originator	Reason for Change	Ratified
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	RFC 0153 Recipient Registration Automation Establish Data Recipient Registration requirements and guidelines.	

Version	Date	Originator	Reason for Change	Ratified
		Dinesh Katyal Clyde Cutting	RFC 0181 Errata: Move `AccountCategory` from `InsuranceAccount` to `AccountDescriptor`	
		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.	

Version	Date	Originator	Reason for Change	Ratified
		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 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.	

Version	Date	Originator	Reason for Change	Ratified
		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.	

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.

2. Message Transport and Security

The FDX API recommends the use of the FDX API Security Profile as specified in the *FDX API Security Model* document for secure interactions.

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 *Foundational 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.

4.1 End to End Encryption

4.1.1 Nested JSON Web Token (JWT) using RSA

Overview

This section provides implementation details on using RSA encryption algorithms for JSON Web Signatures and JSON Web Encryption.

Pre-conditions

Both the data provider and data recipients should generate RSA key pairs and the public keys should be available as signed certificates on a JSON Web Key set (JWKS) endpoint.

RSA: Creating Key Pairs Using OpenSSL

Creating a private RSA key using the command line

```
openssl genrsa -out private_key.pem 2048
```

Creating a public RSA key from a private key

```
openssl rsa -in private_key.pem -outform PEM -pubout -out
public_key.pem
```

RSA: Sample Keys Used in Examples

Data Provider’s Key Pair

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA40b3nY6KgCBekJVfj3MbbXEMvcDFz1ew97LQ2B8AuQi3Khtl
RpaSXr8VstL3PQ9WarGcWHKepor8dK+rgWAnpAM8UJdmW2jtvUyE+7WjmEZkW9E+
xPBAhRiLeGkkDdVJifaenqrHSLj222IKpw/16wRZ1HonV6OjXOjp5fF2og4OhL3S
2EVx22m071VJ0f6mDK4Gg5F4iJWfuEL1Hie5AYbGEVoP2VKMe2KKQIBypqawxBIl
qVJdPGkK+V8YxRYrcZMzep9XIsvEAb90XC4+a7e66pWlk7XfykZ5Bg9/dm+1aZPc
653Dq0SWR0dtd5qKVWqgPgcz1i4LTJnwn3Zi4QIDAQABAoIBAQCR1UOlcYUlWZ/U
HkTHAxEiVFvclglXVelTxwWC58HK+PCusA14Eb2x1eLSb9P89g55P6YWitWJ/7ym
EuJ1jYFiGEFnZP5kwRtrT+I0sQRb/S8AS29/Zrm/rVQw2yYrSiR5xilZnjNpUmXz
tyEmuXMObihF0mvULEfyofETfrCkojKDRe0AO5B1FCY4EHKcssuqdpux57+nB1rd
2CUv+6LuAb0IB0fB1KQP24d0NCuhbKUly9aiJMfLJoHmb6KEJfsySYpmzBxOSH71
V1GTjDzmrqRSbgAse7+XnlYRSAz091hpqQpj1EJge0l/1VDEJessxCbOLO1g5P55
fUgxZQbBAoGBAPy1ZGbafqAeQuHQxg1ehix3n2kLnwM4HBOhmNf+GvL0vHbIyAVY
VuuCiGSlyOPCJsjLH7d0/PMqMMtR1AgeS8LZS8PAAB6Yy8HRAZQhOnyx8aqKHEyc
i0+q2IqQXLld1R8QRdEy2KmvE08iQ+fQo112q4XCxIdkio3rbjbGFznFAoGBAOY8
x3Yh6Of92Rnn5muJ3WHn1gK9i4rS1KgJNFvzEaaBc26QQeEmP9kldx3KFOVUBk6S
ZlZzSLc9QggB05/om5w9wztS8f6kg1+jOa6V/SEKkosyHsueSUsRV8lNgbF9ODOf
45Vx0eWHa5YFrnYI0KgLYrgXDvrE8+An2boA1kJtAoGAIJBLQMm0+XMM0UZyzvQ4
O/CqNQIPWn3XeFwhcuvGkzogMvpKdA3fHXfzlWybh2XUU5mBG8XSdo8gPILt3KHy
x0fy8GWEXmz4DKCfIHRrsffIGV60qNafSQPN5YUWvbgup1MUfBGeQ7dQuKjEsVF8
S6XoElN3ua6mAAWvbRV3lrUCgYEAqtBA82XpE+UDYvLnwrT/6BlGb7YMhywv1ZMu
o2FMoQm9iDPCjLYB/KqNGy7IHfQe0cBP6KeTNU9fY+1nAmZivKId7C93loKbbSL+
MobYy+C6JEdFDbAblHQDezfjlrjeL37aLA/Lt8ymhyEj9DJKC8KWtRl2ZZoljRJD
uHnSfGkCgYAsYv6QDMPnJH1T2rGOm0X6roH3M4OTiXnQXBMOxbUzYacg4/1bBY1L
/uGXmq1mS19ntHHAaf/wTrqlP/hCh7cHBy3SCh3Pt/ktd+90ZXaU3au9mAVAX72j
xYzYot10UYj/autRMH/Vxt399DZ5reA/gaOBnK2pSHwwW0oAr1ERgA==
-----END RSA PRIVATE KEY-----
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA40b3nY6KgCBekJVfj3Mb
bXEMvcDFz1ew97LQ2B8AuQi3KhtlRpaSXr8VstL3PQ9WarGcWHKepor8dK+rgWAn
pAM8UJdmW2jtvUyE+7WjmEZkW9E+xPBAhRiLeGkkDdVJifaenqrHSLj222IKpw/1
6wRZ1HonV6OjXOjp5fF2og4OhL3S2EVx22m071VJ0f6mDK4Gg5F4iJWfuEL1Hie5
AYbGEVoP2VKMe2KKQIBypqawxBIlqVJdPGkK+V8YxRYrcZMzep9XIsvEAb90XC4+
a7e66pWlk7XfykZ5Bg9/dm+1aZPc653Dq0SWR0dtd5qKVWqgPgcz1i4LTJnwn3Zi
4QIDAQAB
-----END PUBLIC KEY-----
```

Data Recipient’s Key Pair

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAuRK7Tfo+7OBUcuI/Tk4Qnh0lamAnwMXK/Kcd5Sz1P9DJl4QE
4tY5Wv6IKMCbdYKX5muo8DjahR/ZtAv7UvPobprmmWU2201uLaLROmuhTJzWWFqE
JGhvtabwdr7/7JsXywQI1ZchzON4RUJaukqu0MsUnfVZEJAedYvUZBfY5Pa6GCl+
dgSV9vZ6nsFGcH/hmQU7qKrk4PrTNV4lc5ehO7Tkckn0BA9soDjBGfdxaYwxm9kU
jgkJMQYYIkm4gzPfLk7UlLhMRy9UWBoRvajGXz5vrif/0KIFKVQtjbt108rwC+/u
```

```
KVlmEHqybkanSG8T1tDL1BLKpKxwtZo0fsAURQIDAQABAoIBAQCyuNwYWWka5yem
KcZooAp8JjlTmLfK+Tck9V1xSxErJz0GDH+LbsTEkrh6YkW+HPcDlUP3d2/Ozws1
S7zQRqpW5U97Irru3L9hYrIacIW7rllvyTmCzzfRe/0LLzeGDd/UNXkyilghvCkQ
+RsUe7qF9xdZ0uzHieVgLkBUyJKzx8k40VfJE/2FdPZafdDMv1p+4CkDpxZUS7F6
MPFSuAumLJ3lSUey0B1+IpUZhZBaSjHVyBfUOWluOEtz99NjQKS/8I8lYetAJ+Mo
GkkXLW5sHRwkdhc050cuKp41BEOeByCtESH30vJLEvwbF4o2/vj6Z+v4NvpSZMmA
0/p02goBAoGBAPCn0kRjMKOxmYJVkZkQFHpVs6LH/J0SZY3K3bdJfiNmOYdYVyAn
NZQ9IvQ712AJTuVawkOhjBnTeMD5TUQvjV0WAVp9AIu8oV3vTcl/8oubZ9D4COt9
S+aJOTkwD8voH2MkQc7tNl7Qw31wcE+MGIKh3qigdgwifkhFzmaqpt1xAoGBAMTf
phXzMmkwKEW5v/ULLsOHS96x89izjebu8P8fWbuVgU0sK6n46Gfxb798hEd7gjU/
EzOSbNRIFyertRIiUUWcmeo6d9aqxbzfkc2GqEJ+Jym5QFSlJgxqxMtmTPnO5Bat
S58/Scl7TSqk5bE3j2yBR4rJEjur5V1SldByQYoVAoGATKpJ8/tdbWiQrNKxtX9H
5skSlxL6yNcpfwhXpaJGCuTwAswDxXx4Nyda0U+XB0Mv3SUSqhT22uthlqhVExnL
ARKXj8ouuFV5WsF3mG+oRw1U/19lCBA8c87Xaf6DqcPi6+SLCm7LWV1MSdPeE5lf
3Y3PrwyfTrJWZJPIczB+RCECgYBYoVbkCthnAoce3MDOUHp9DCvb1cExjaQUkv1r
3XFIQcY0N+5wVt5J7SehzSzAAZpc3kiGryTPbKT/9w1NXKW58QZZrHjG65qZrQy2
uiiFxsVaw0tyz+aRMH/oEeYVkE6e5uVki9lsG1ZiHFpLrfejoY/TqzHKK1jW6pcH
gGiBAQKBgAIeLlx91P4DLHuHBSmAWElqVVJf+nNQYBX7uAnftGlD+/PcyquPE/wL
eKgQzj6g0/kDM6Erex08hcw2bkc0ZULfhFFMt0NwGGhVGyt+eQOR1pkIJFLy72TX
WrsXAt3Nkhzw59ZyMT2hxpdEsxdeosoioJkDBBXaNcdeck4OLTNN
-----END RSA PRIVATE KEY-----
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuRK7Tfo+7OBUcuI/Tk4Q
nh0lamAnwMXK/Kcd5Sz1P9DJl4QE4tY5Wv6IKMCbdYKX5muo8DjahR/ZtAv7UvPo
bprmmWU2201uLaLROmuhTJzWWFqEJGhvtabwdr7/7JsXywQI1ZchzON4RUJaukqu
0MsUnfVZEJAedYvUZBfY5Pa6GCl+dgSV9vZ6nsFGcH/hmQU7qKrk4PrTNV4lc5eh
O7Tkckn0BA9soDjBGfdxaYwxm9kUjgkJMQYYIkm4gzPfLk7UlLhMRy9UWBoRvajG
Xz5vrif/0KIFKVQtjbt108rwC+/uKVlmEHqybkanSG8T1tDL1BLKpKxwtZo0fsAU
RQIDAQAB
-----END PUBLIC KEY-----
```

RSA: JSON Web Signature

This example shows how to create a JWS using RSASSA-PKCS1-v1_5 and SHA-256

Assemble the JWS Protected Header by adding all of the header elements that are to be included as part of the signature

```
{
  "alg": "RS256",
  "jku": "https://server.example.com/keys.jwks",
  "kid": "123"
}
```

JWS Payload

```
{
  "customerId": "abc-12345-xyz",
  "name": {
    "first": "Jane",
    "last": "Smith"
  },
  "email": [
    "jane_smith@email.com"
  ]
}
```

JWS Signature

To create the JWS signature, the protected header and payload are Base64URL encoded and then joined together by a period `.`

Encoded header and payload

BASE64URL(UTF8(JWS Protected Header))

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyIs
ImtpZCI6IjEyMyJ9
```

BASE64URL(JWS Payload)

```
eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJK
YW5lIiwibGFzdCI6
IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWwuY29tIl19
```

The encoded header and payload are then joined by a `.` with no spaces before or after

BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload)

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyIs
ImtpZCI6IjEyMyJ9.eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWU
iOnsiZmlyc3QiOiJ
KYW5lIiwibGFzdCI6IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWw
uY29tIl19
```

This is the BASE64URL encoded signature using

```
0TTIMtQCJCBWTmKmTexZj8b0qlZe73bCQtuKCq_GWvLDvpS70VMUq0ZCFL5kn3HF
-wspL3BhGtxFTBu3
aNkdbTslyBrRwN0NnPEdYRnibiKp2qjmVwttsSbJtfq3wDEd23ANU4_YEZYDhD3i
ykGTxzGif2JkRmlB
```

```
t0GQASGvPrDT2vcVfIza2JxKCEB6VCtJtFGQp27vH7Eb2MP7EqTkNK2q47p2euvK
fLZuOqdC3RA9eo_W
fpa4B1ro0evXLsr3ZTIfX-z4p-
_ja3NOB0NkZ3IM3g1b32dMPwq7xUnbGCidUMk5sYaKiog9ZDuYZQxe
_nHcgwEfagVBMYOt08HhVQ
```

RSA: JSON Web Encryption

Encryption

> **Note**: This example uses an RSA signed JWS. You may see signed payloads encrypted using other algorithms such as Elliptic Curve.

1. JWE Protected Header

```
{
  "alg": "RSA-OAEP-256",
  "cty": "JWT",
  "enc": "A256GCM"
}
```

2. Generate a 256-bit AES Content Encryption Key (Symmetric Key) Hex Byte Array

```
[b1 29 bc ff 2b 76 01 d3 a9 06 d7 0c b6 ac 7b 5a fe c5 49 0a 86
f6 f1 1a a1 5c
8a fa 20 39 18 22]
```

3. Generate the Initialization Vector Hex Byte Array

```
[b2 37 21 88 5c a4 21 9f 5a 6f f3 96]
```

BASE64URL

```
sjchiFykIZ9ab_OW
```

4. Encrypt the compact serialized form of the JWS using A256GCM with the content encryption key and initialization vector byte arrays as inputs. The two outputs of the process will be the ciphertext and authentication tag value

Ciphertext

```
lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJYH
7XwpvG6YoceqWYF7
vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSrgc80ZbYajZkaZZqfTgY2QY3pEGmz
pELnC65vQF2No6fH
Xdd3CewdoZHagyHFhBtqqoDRAw5q-Naj5O36GJXT--
l2D1yxBTN8tZs7FNZKqRmFngPUj17wyM27vjqC
8liKc3DJM3wzFNC98ixsqjlAsiMJJ7lT_tXW-nTY-xm-
fH6xUL6ZQvFQnBa2jnJIsQ_WYIkEOddBd_0X
NJMFf62l6g6IObrHSc1pEGsviIAFFvlOXzGCZO7WFFh580Af2cyOe7vTD8irUm02
qnxFGIMgOjLGbS87
1zJ_40_T3xfe_8e7nKgX4ti4PuwDmi5aBySYHGIv70xJq1ZQQ1MohxwJ_0jUkWcT
VWa-n-xDSESMGEH_
```

```
1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIkwB0xDIJOERB4
EjRoWVWgYLGSduZb
SpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jrm95ot48_Chs2G7A8JRXUl2sx9pbf
ayxktr0hBNmcVx3k
Poww-xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn9XLj8dD0aimhz6vPMOX4zAtdaWc0
3fe_neuU3B19A2UvTzymO4EnY4fA_dngBYzJPMDbC92G1oEc
```

Authentication Tag

```
4aT8aykEG7TAVrBfS27qaw
```

5. Encrypt the Content Encryption Key with Data Recipient’s public encryption key using RSA-OAEP-256

```
JdC7ydHPnn-lrrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLL
D4D0FtLm6f_S-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8
f0_D3-JNkkNpMLUYZAWSE8ZgRclwOHL1KD6zhBIF-
uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-CJcxYPo
JmjAeSdooahBrsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto
0kDQMDf_Tk2igBfslXYiTA
```

JSON Web Encryption Generation

A JWE consists of the following 5 elements from the previous encryption steps encoded in BASE64URL form .

1. BASE64URL(UTF8(JWE Protected Header))

```
eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NN
In0
```

2. BASE64URL(JWE Encrypted Content Encryption Key)

```
JdC7ydHPnn-lrrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLL
D4D0FtLm6f_S-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8
f0_D3-JNkkNpMLUYZAWSE8ZgRclwOHL1KD6zhBIF-
uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-CJcxYPo
JmjAeSdooahBrsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto
0kDQMDf_Tk2igBfslXYiTA
```

3. BASE64URL(JWE Initialization Vector)

```
sjchiFykIZ9ab_OW
```

4. BASE64URL(JWE Ciphertext)

```
lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJYH
7XwpvG6YoceqWYF7
vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSrgc80ZbYajZkaZZqfTgY2QY3pEGmz
pELnC65vQF2No6fH
Xdd3CewdoZHagyHFhBtqqoDRAw5q-Naj5O36GJXT--
l2D1yxBTN8tZs7FNZKqRmFngPUj17wyM27vjqC
8liKc3DJM3wzFNC98ixsqjlAsiMJJ7lT_tXW-nTY-xm-
fH6xUL6ZQvFQnBa2jnJIsQ_WYIkEOddBd_0X
NJMFf62l6g6IObrHSc1pEGsviIAFFvlOXzGCZO7WFFh580Af2cyOe7vTD8irUm02
qnxFGIMgOjLGbS87
1zJ_40_T3xfe_8e7nKgX4ti4PuwDmi5aBySYHGIv70xJq1ZQQ1MohxwJ_0jUkWcT
VWa-n-xDSESMGEH_
1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIkwB0xDIJOERB4
EjRoWVWgYLGSduZb
SpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jrm95ot48_Chs2G7A8JRXUl2sx9pbf
ayxktr0hBNmcVx3k
Poww-xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn9XLj8dD0aimhz6vPMOX4zAtdaWc0
3fe_neuU3B19A2UvTzymO4EnY4fA_dngBYzJPMDbC92G1oEc
```

5. BASE64URL(JWE Authentication Tag)

```
4aT8aykEG7TAVrBfS27qaw
```

JWE in JSON Serialized Form

```
{
  "protected":
"eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0N
NIn0",
  "encrypted_key":
"JdC7ydHPnn-lrrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLLD4D0FtLm6f_S-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8f0_D3-JNkkNpMLUYZAWSE8ZgRclwOHL1KD6zhBIF-
uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-
CJcxYPoJmjAeSdooahBrsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto0kDQMDf_Tk2igBfslXYiTA",
  "ciphertext":
"lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJY
H7XwpvG6YoceqWYF7vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSrgc80ZbYajZk
aZZqfTgY2QY3pEGmzpELnC65vQF2No6fHXdd3CewdoZHagyHFhBtqqoDRAw5q-
Naj5O36GJXT--
l2D1yxBTN8tZs7FNZKqRmFngPUj17wyM27vjqC8liKc3DJM3wzFNC98ixsqjlAsi
MJJ7lT_tXW-nTY-xm-
fH6xUL6ZQvFQnBa2jnJIsQ_WYIkEOddBd_0XNJMFf62l6g6IObrHSc1pEGsviIAF
FvlOXzGCZO7WFFh580Af2cyOe7vTD8irUm02qnxFGIMgOjLGbS871zJ_40_T3xfe
_8e7nKgX4ti4PuwDmi5aBySYHGIv70xJq1ZQQ1MohxwJ_0jUkWcTVWa-n-
xDSESMGEH_1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIkwB
0xDIJOERB4EjRoWVWgYLGSduZbSpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jrm9
5ot48_Chs2G7A8JRXUl2sx9pbfayxktr0hBNmcVx3kPoww-
xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn9XLj8dD0aimhz6vPMOX4zAtdaWc03fe_neuU3B19A2UvTz
ymO4EnY4fA_dngBYzJPMDbC92G1oEc",
  "iv": "sjchiFykIZ9ab_OW",
  "tag": "4aT8aykEG7TAVrBfS27qaw"
}
```

JWE in Compact Serialized Form

1. BASE64URL(UTF8(JWE Protected Header)) || '.' || BASE64URL(JWE Encrypted Content Encryption Key) || '.' || BASE64URL(JWE Initialization Vector) || '.' || BASE64URL(JWE Ciphertext) || '.' || BASE64URL(JWE Authentication Tag)

```
eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NN
In0.JdC7ydHPnn-l
rrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLLD4D0FtLm6f_S
-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8f0_D3-JNkkNp
MLUYZAWSE8ZgRclwOHL1KD6zhBIF-uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-
CJcxYPoJmjAeSdooahB
rsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto0kDQMDf_Tk2i
gBfslXYiTA.sjchiFykIZ9ab_OW.lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1I
gtHSs_mhBpOv8cYM
sRi6_IWlbJYH7XwpvG6YoceqWYF7vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSr
gc80ZbYajZkaZZqf
TgY2QY3pEGmzpELnC65vQF2No6fHXdd3CewdoZHagyHFhBtqqoDRAw5q-
Naj5O36GJXT--l2D1yxBTN8
tZs7FNZKqRmFngPUj17wyM27vjqC8liKc3DJM3wzFNC98ixsqjlAsiMJJ7lT_tXW
-nTY-xm-fH6xUL6Z
QvFQnBa2jnJIsQ_WYIkEOddBd_0XNJMFf62l6g6IObrHSc1pEGsviIAFFvlOXzGC
ZO7WFFh580Af2cyO
e7vTD8irUm02qnxFGIMgOjLGbS871zJ_40_T3xfe_8e7nKgX4ti4PuwDmi5aBySY
HGIv70xJq1ZQQ1Mo
```

```
hxwJ_0jUkWcTVWa-n-
xDSESMGEH_1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIk
wB0xDIJOERB4EjRoWVWgYLGSduZbSpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jr
m95ot48_Chs2G7A8
JRXUl2sx9pbfayxktr0hBNmcVx3kPoww-xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn
9XLj8dD0aimhz6vPMOX4zAtdaWc03fe_neuU3B19A2UvTzymO4EnY4fA_dngBYzJ
PMDbC92G1oEc.4aT
8aykEG7TAVrBfS27qaw
```

Sending data to the Data Recipient

See: 4.1.3 Sending Nested JWT HTTP Response

RSA Consumption

Decrypting JWE Payload

1. Decode the Compact Serialized JWE

JWE Protected Header

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ
```

JWE Encrypted Key

```
KOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
psEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
v04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
qgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
UklfCpIMfIjf7iGdXKHzg
```

JWE Initialization Vector

```
8V1_ALb6US04U3b
```

JWE Ciphertext

```
eym8TW_c8SuK0ltJ3rpYIzOeDQz7TALvtu6UG9oMo4vpzs9tX_EFShS8iB7j6ji
diwkIr3ajwQzaBtQD_A
```

JWE Authentication Tag

```
FBoMYUZodetZdvTiFvSkQ
```

2. Look up private key (key encryption key) associated with the kid from the protected header ```

```
 -----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAuRK7Tfo+7OBUcuI/Tk4Qnh0lamAnwMXK/Kcd5Sz1P9DJl4QE
4tY5Wv6IKMCbdYKX5muo8DjahR/ZtAv7UvPobprmmWU2201uLaLROmuhTJzWWFqE
JGhvtabwdr7/7JsXywQI1ZchzON4RUJaukqu0MsUnfVZEJAedYvUZBfY5Pa6GCl+
dgSV9vZ6nsFGcH/hmQU7qKrk4PrTNV4lc5ehO7Tkckn0BA9soDjBGfdxaYwxm9kU
jgkJMQYYIkm4gzPfLk7UlLhMRy9UWBoRvajGXz5vrif/0KIFKVQtjbt108rwC+/u
KVlmEHqybkanSG8T1tDL1BLKpKxwtZo0fsAURQIDAQABAoIBAQCyuNwYWWka5yem
KcZooAp8JjlTmLfK+Tck9V1xSxErJz0GDH+LbsTEkrh6YkW+HPcDlUP3d2/Ozws1
S7zQRqpW5U97Irru3L9hYrIacIW7rllvyTmCzzfRe/0LLzeGDd/UNXkyilghvCkQ
+RsUe7qF9xdZ0uzHieVgLkBUyJKzx8k40VfJE/2FdPZafdDMv1p+4CkDpxZUS7F6
MPFSuAumLJ3lSUey0B1+IpUZhZBaSjHVyBfUOWluOEtz99NjQKS/8I8lYetAJ+Mo
GkkXLW5sHRwkdhc050cuKp41BEOeByCtESH30vJLEvwbF4o2/vj6Z+v4NvpSZMmA
0/p02goBAoGBAPCn0kRjMKOxmYJVkZkQFHpVs6LH/J0SZY3K3bdJfiNmOYdYVyAn
NZQ9IvQ712AJTuVawkOhjBnTeMD5TUQvjV0WAVp9AIu8oV3vTcl/8oubZ9D4COt9
S+aJOTkwD8voH2MkQc7tNl7Qw31wcE+MGIKh3qigdgwifkhFzmaqpt1xAoGBAMTf
phXzMmkwKEW5v/ULLsOHS96x89izjebu8P8fWbuVgU0sK6n46Gfxb798hEd7gjU/
EzOSbNRIFyertRIiUUWcmeo6d9aqxbzfkc2GqEJ+Jym5QFSlJgxqxMtmTPnO5Bat
S58/Scl7TSqk5bE3j2yBR4rJEjur5V1SldByQYoVAoGATKpJ8/tdbWiQrNKxtX9H
5skSlxL6yNcpfwhXpaJGCuTwAswDxXx4Nyda0U+XB0Mv3SUSqhT22uthlqhVExnL
ARKXj8ouuFV5WsF3mG+oRw1U/19lCBA8c87Xaf6DqcPi6+SLCm7LWV1MSdPeE5lf
3Y3PrwyfTrJWZJPIczB+RCECgYBYoVbkCthnAoce3MDOUHp9DCvb1cExjaQUkv1r
3XFIQcY0N+5wVt5J7SehzSzAAZpc3kiGryTPbKT/9w1NXKW58QZZrHjG65qZrQy2
uiiFxsVaw0tyz+aRMH/oEeYVkE6e5uVki9lsG1ZiHFpLrfejoY/TqzHKK1jW6pcH
gGiBAQKBgAIeLlx91P4DLHuHBSmAWElqVVJf+nNQYBX7uAnftGlD+/PcyquPE/wL
eKgQzj6g0/kDM6Erex08hcw2bkc0ZULfhFFMt0NwGGhVGyt+eQOR1pkIJFLy72TX
WrsXAt3Nkhzw59ZyMT2hxpdEsxdeosoioJkDBBXaNcdeck4OLTNN
-----END RSA PRIVATE KEY-----
```

3. Use the private key (key encryption key) to decrypt the Encrypted JWE Content Encryption Key (Base 64 encoded)

```
sSm8_yt2AdOpBtcMtqx7Wv7FSQqG9vEaoVyK-iA5GCI
```

4. Use JWE Content Encryption Key, Initialization Vector and Authentication Tag to decrypt the ciphertext. You should be left with plaintext that is a compact serialized JWS.

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyJ9
.eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJ
KYW5lIiwibGFzdCI
```

```
6IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWwuY29tIl19.n7zJKh
erBaBL4UACye3j41
AK48GZwGKHS383G2ZSBTqPUeh4_wXckngpRWqPP5y5O6DGaP34f_24LKAZ1KKpGh
V_HgWKSghTIDR6av
F_SHF0yMlHeU7PZZfyzy4OyLh0UCa92GQn3e_l7VK6m4rCjSBLS58vTx6g_QMoqY
Fuwbt9mGviO1aPbq
AZT-e-
NvQuZDwX1TJsqks7QG8zSEOB4KHgjHvLvh1wIcQAgXW5Y_qa2Txh3MqkUxAT3oGU
7ip9EFL6ZP
mwbVruf738cQVybmsCP3YH_AMF3twFPkJ56UzZBKj4HeFlFp-N3-
x6j8yHDdmaFypc5rQWWVTK9sxMug
```

Verifying JWS Payload

1. Decode the compacted payload extracting out the JWS Protected Header, Payload and Signature

JWS Protected Header

```
eyJhbGciOiJSUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyJ9
```

JWS Payload

```
eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJK
YW5lIiwibGFzdCI6
IlNtaXRoIn0sImVtYWlsIjpbImphbmVfc21pdGhAZW1haWwuY29tIl19
```

JWS Signature

```
n7zJKherBaBL4UACye3j41AK48GZwGKHS383G2ZSBTqPUeh4_wXckngpRWqPP5y5
O6DGaP34f_24LKAZ
1KKpGhV_HgWKSghTIDR6avF_SHF0yMlHeU7PZZfyzy4OyLh0UCa92GQn3e_l7VK6
m4rCjSBLS58vTx6g
_QMoqYFuwbt9mGviO1aPbqAZT-e-
NvQuZDwX1TJsqks7QG8zSEOB4KHgjHvLvh1wIcQAgXW5Y_qa2Txh
3MqkUxAT3oGU7ip9EFL6ZPmwbVruf738cQVybmsCP3YH_AMF3twFPkJ56UzZBKj4
HeFlFp-N3-x6j8yH
DdmaFypc5rQWWVTK9sxMug
```

2. Retrieve the public key associated with the kid from the protected header. The public key can either be in a local store or can be retrieved from signed certificate via a JWKS endpoint registered for the data provider.

```
-----BEGIN CERTIFICATE-----
MIIDhzCCAm+gAwIBAgIBADANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJCRTEN
```

```
MAsGA1UECgwEVGVzdDENMAsGA1UECwwEVGVzdDENMAsGA1UEAwwEVGVzdDAeFw0y
MTAxMjQyMjI0NTlaFw0yMjAxMjQyMjI0NTlaMDoxCzAJBgNVBAYTAkJFMQ0wCwYD
VQQKDARUZXN0MQ0wCwYDVQQLDARUZXN0MQ0wCwYDVQQDDARUZXN0MIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA40b3nY6KgCBekJVfj3MbbXEMvcDFz1ew
97LQ2B8AuQi3KhtlRpaSXr8VstL3PQ9WarGcWHKepor8dK+rgWAnpAM8UJdmW2jt
vUyE+7WjmEZkW9E+xPBAhRiLeGkkDdVJifaenqrHSLj222IKpw/16wRZ1HonV6Oj
XOjp5fF2og4OhL3S2EVx22m071VJ0f6mDK4Gg5F4iJWfuEL1Hie5AYbGEVoP2VKM
e2KKQIBypqawxBIlqVJdPGkK+V8YxRYrcZMzep9XIsvEAb90XC4+a7e66pWlk7Xf
ykZ5Bg9/dm+1aZPc653Dq0SWR0dtd5qKVWqgPgcz1i4LTJnwn3Zi4QIDAQABo4GX
MIGUMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFJ45lQ9T0P52egkfVdB6h/Wj
SIpUMGIGA1UdIwRbMFmAFJ45lQ9T0P52egkfVdB6h/WjSIpUoT6kPDA6MQswCQYD
VQQGEwJCRTENMAsGA1UECgwEVGVzdDENMAsGA1UECwwEVGVzdDENMAsGA1UEAwwE
VGVzdIIBADANBgkqhkiG9w0BAQUFAAOCAQEAvk3PD1Sbo/h2AR+twDv+LrQOyyN7
GkElF28goFhOiVBkGUmVvfy7bQEZa3q7RTUoe2pWqRdWQt1UkFcOGLWWtTdfDrak
9PJyMCIRQenD5DqdqwMmuPscy9y9GLJ2VV7KLzhTYQk5Og6LCn0PfWSLp9Yb3WCz
+AhBCs13N497DMBrLqwaLDITjnWNnXmv4AreiOqrvPJRY8acHeAVYmNph5Ine/Fe
nXTsGCHj7ERgZ8dD3+kvu7Ia8h4m+JbA1SeZajnKP34lKW7DucaHzgbUjVTj14oG
dZv+ZZAZ6QgS+xAHmVkKqX7SNNGEfmuzS4j6p9AMoy/77jkRnPSxy0Nnzw==
-----END CERTIFICATE-----
```

3. Confirm that the public key is valid.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA40b3nY6KgCBekJVfj3Mb
bXEMvcDFz1ew97LQ2B8AuQi3KhtlRpaSXr8VstL3PQ9WarGcWHKepor8dK+rgWAn
pAM8UJdmW2jtvUyE+7WjmEZkW9E+xPBAhRiLeGkkDdVJifaenqrHSLj222IKpw/1
6wRZ1HonV6OjXOjp5fF2og4OhL3S2EVx22m071VJ0f6mDK4Gg5F4iJWfuEL1Hie5
AYbGEVoP2VKMe2KKQIBypqawxBIlqVJdPGkK+V8YxRYrcZMzep9XIsvEAb90XC4+
a7e66pWlk7XfykZ5Bg9/dm+1aZPc653Dq0SWR0dtd5qKVWqgPgcz1i4LTJnwn3Zi
4QIDAQAB
-----END PUBLIC KEY-----
```

a. Confirm that it is signed by a trusted certificate chain

b. Confirm that it has not expired

c. Confirm that it has not been revoked

4. Verify the signature using the public key

a. Verify issued at date

b. Verify additional claims if applicable ([Examples](https://www.iana.org/assignments/jwt/jwt.xhtml))

4.1.2 Nested JWT using Elliptic Curve

Overview

This section provides implementation details for using EC algorithms for JSON Web Signatures and JSON Web Encryption

Recommended Curves

Per [Section 6.2.1.1 from RFC 7518] (https://tools.ietf.org/html/rfc7518#section-6.2.1.1), acceptable curves for ECDH-ES key agreement are P-256, P-384, P-521.

- P-256 uses prime256v1 elliptic curve
- P-384 uses secp384r1 elliptic curve
- P-521 uses secp521r1 elliptic curve

Required Elements/Minimum Standards

HTTP Headers

The following fields in the JOSE header for the JWS are allowed.

	Parameter	RFC	Description
1	alg	MUST	It identifies the cryptographic algorithm used to encrypt or determine the value of the CEK.
2	enc	MUST	It identifies the content encryption algorithm used to perform authenticated encryption on the plaintext to produce the ciphertext and the AuthenticationTag.
3	zip	OPT	Compression algorithm applied to the plaintext before encryption, if any.
4	jku	OPT	JWK Set URL is a URI that refers to a resource for a set of JSON-encoded public keys, one of which corresponds to the public key with which the JWE was encrypted An HTTP GET request to retrieve the JWK Set MUST use TLS (1.2 at the time of writing this document The identity of the server MUST be validated by JWS recipient should include it if multiple public keys are exposed in the JWK Set URL
5	jwk	OPT	JSON Web Key is the public key with which the JWE was encrypted
6	kid	OPT	Key ID is hint indicating the public key with which the JWE was encrypted. This parameter allows originators to explicitly signal a change of key to JWE recipients.
7	x5u	OPT	X.509 public key certificate or certificate chain [RFC5280](https://tools.ietf.org/html/rfc5280) contains the public key with which the JWE was encrypted.

8	x5c	OPT	X.509 public key certificate or certificate chain [RFC5280](https://tools.ietf.org/html/rfc5280) contains the X.509 certificate chain contains the X.509 public key certificate or certificate chain corresponding to the public key with which the JWE was encrypted. The certificate or certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64-encoded not base64url-encoded) DER PKIX certificate value. The certificate containing the public key corresponding to the key used to encrypt the JWE MUST be the first certificate. The recipient MUST validate the certificate chain and consider the certificate or certificate chain to be invalid if any validation failure occurs.
9	x5t	OPT	X.509 certificate SHA-1 thumbprint is a base64url-encoded SHA-1 message digest of the DER encoding of the X.509 certificate corresponding to the public key with which the JWE was encrypted. Certificate thumbprints are also known as certificate fingerprints.
10	x5t#S256	OPT	X.509 certificate SHA-256 thumbprint is a base64url-encoded SHA-256 message digest of the DER encoding of the X.509 certificate corresponding to the public key with which the JWE was encrypted
11	typ	OPT	type is used by JWE applications to declare the media type of this complete JWE. The "typ" value "JOSE+JSON" can be used to indicate that this object is a JWS or JWE using the JWS or the JWE JSON Serialization. Other type values can also be used by applications.
12	cty	OPT	content type is used by JWE application to declare the media type of the secured content (the plaintext).
13	crit	OPT	The "crit" (critical) Header Parameter indicates that extensions to this specification and/or [JWA] are being used that MUST be understood and processed. Its value is an array listing the Header Parameter names present in the JOSE Header that use those extensions. If any of the listed extension Header Parameters are not understood and supported by the recipient, then the JWE is invalid.
14			When ECDH-ES, ECDH-ES+A128KW or ECDH-ES+A256KW are used "epk", "apu", "apv" headers must be added.

JWS Claims

As JWS is used only for signature validation, no additional claims are added. This example shows how to create an EC certificate request used in ECDH key agreement.

How to create an EC Certificate Request

1.

```
# openssl.conf


[req]
```

```
      req_extensions = v3_req
      distinguished_name  = req_distinguished_name

      [v3_req]
      basicConstraints = CA:FALSE
      keyUsage=keyAgreement

      [ req_distinguished_name ]
      countryName                     = Country Name (2 letter
code)
      countryName_default             = US

      stateOrProvinceName             = State or Province Name
(full name)
      stateOrProvinceName_default     = IL

      localityName                    = Chicago

      0.organizationName              = Organization Name (eg,
company)
      0.organizationName_default      = Internet Widgits Pty Ltd

      organizationalUnitName          = Organizational Unit Name
(eg, section)

      commonName                      = Common Name (e.g. server
FQDN or YOUR name)
      commonName_max                  = 64

      emailAddress                    = Email Address
      emailAddress_max                = 64
```

2.

```
`openssl ecparam -out server.key -name prime256v1 -genkey -
config openssl.conf`
```

3.

```
`openssl req -new -key server.key -out server.csr -sha256 -
config openssl.conf`
```

4.

```
openssl req -text -noout -verify -in server.csr

     verify OK
     Certificate Request:
         Data:
             Version: 1 (0x0)
             Subject: C = US, ST = UT, O = Internet Widgits Pty
Ltd
             Subject Public Key Info:
                 Public Key Algorithm: id-ecPublicKey
                     Public-Key: (256 bit)
                     pub:

04:97:08:cd:34:34:8c:2b:7e:3e:db:c2:fc:4e:b3:

78:d5:7e:ee:4a:31:f9:4c:c6:d0:ac:b0:a4:2d:76:

3d:9c:62:03:16:69:26:ae:c8:4a:43:f9:0c:59:c5:

3d:79:a5:7f:9a:73:48:09:db:27:96:47:8a:97:c0:
                         b3:e0:13:d0:45
                     ASN1 OID: prime256v1
                     NIST CURVE: P-256
             Attributes:
             Requested Extensions:
                 X509v3 Basic Constraints:
                     CA:FALSE
                 X509v3 Key Usage:
                     Key Agreement
         Signature Algorithm: ecdsa-with-SHA256

30:46:02:21:00:a1:07:38:2f:7d:fa:31:5c:07:51:8c:23:52:

84:0e:c3:cb:ef:5c:5b:88:1d:5e:9a:59:50:77:84:55:b1:31:

49:02:21:00:b6:0b:39:f3:70:ec:28:f3:71:5f:d3:39:82:5f:

5c:26:9e:31:99:39:a6:2b:67:52:0e:f7:93:d7:55:1a:c4:4a
```

5. Follow the prompts and enter the information related to your organization

```
EC: JSON Web Signature
```

~~~
Per RFC 11 Message Encryption, an RSA signing algorithm or an EC
signing algorithm can also be used for the JWS that is then
encrypted to form the JWE using an EC encryption algorithm. This
example shows how to create a JWS with ECDSA using P-256 and
SHA-256 (ES256). The protected header and payload are Base64URL
encoded in the same way as the RSA example.

JWK:

```json
{
  "kty":"EC",
  "use":"enc",
  "crv":"P-256",
  "kid":"a152e75b-8e11-4f87-908c-366552610467",
  "x":"YoUQaMiqgeSmNMXSqcYiFRO4nZEXV47APaUnm4PyaPg",
  "y":"iUuwma37B6YFPKLA7tv_-cczuT4WJf2WmVfsgELHr1c"
}
~~~

EC Content Encryption Key & Key Encryption Key Generation

1. The Sender and Receiver agree on an elliptic curve

```
Curve: P-256
```

2. Data Recipient generates static EC keypair using an agreed upon elliptic curve with Data Provider. Receiver provides their public EC key as trusted CA signed cert via their JWK URL.

Receiver Private Key

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIFoxJaJ8UtevP3F+dNWkax6neQTjeipwAewO0K/N+8SqoAoGCCqGSM49
AwEHoUQDQgAEEWXLOwDqRhiiSHkd3GCzpVZ4d1ucLH9mzOWZNWunILdrS3vkVxf4
/odFGtCYEW+y05MhkVBGVg8OCWyQHrvoJA==
-----END EC PRIVATE KEY-----
```

Receiver Public Key

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEEWXLOwDqRhiiSHkd3GCzpVZ4d1uc
LH9mzOWZNWunILdrS3vkVxf4/odFGtCYEW+y05MhkVBGVg8OCWyQHrvoJA==
-----END PUBLIC KEY-----
```

3. Sender generates ephemeral (temporary) EC keypair using the same agreed upon elliptic curve.

Sender Private Key

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEILdwODe0rFBedZiW01KQEOZYnAwhR6k8jx9iRg8PuLN0oAoGCCqGSM49
AwEHoUQDQgAESW0RyC+JYmzdVolujnxEZDz+c8hgtUw8rfq5ELAT71IitgQ99l2h
hZvJ2GqM8nIq/TTwFJEfz39Pq1LB8ErWWA==
-----END EC PRIVATE KEY-----
```

Sender Public Key

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAESW0RyC+JYmzdVolujnxEZDz+c8hg
tUw8rfq5ELAT71IitgQ99l2hhZvJ2GqM8nIq/TTwFJEfz39Pq1LB8ErWWA==
-----END PUBLIC KEY-----
```

4. Sender generates ephemeral 256 bit key used to AES encrypt the Content Encryption Key(CEK)

```
sSm8_yt2AdOpBtcMtqx7Wv7FSQqG9vEaoVyK-iA5GCI
```

5. Sender computes a shared secret using their own ephemeral private EC key and Receiver's static public EC key.

```
EaYbmvnPGxXREBtu0o5X6OzQNZelwx2d-v_ty-2Ivrs
```

6. Sender uses the shared secret from step 4 and derives a 256 bit AES Key using Concat KDF. The key is used as the Key Encryption Key (KEK).

```
Ie1XYS7XcaqYztrtzot0SXmtXqLS40k1izIuq2RuVRY
```

EC Encryption

Note: This example uses an Elliptic Curve signed JWS. You may see signed payloads encrypted using other algorithms such as RSA.

1. Sender uses 256 bit AES key (KEK) derived in EC CEK & KEK Generation step 5 to encrypt CEK (Content Encryption Key) from EC CEK & KEK Generation Step 3.

```
6ZKl9aHBfd7hFFlUBmr16PGRHt2po8teuDBZzVTi0AYGpPHw87yBVg
```

2. Sender encrypts sensitive data in API response using 256 bit AES key from EC CEK & KEK Generation Step 3.

Plaintext

```
eyJhbGciOiJFUzI1NiIsImprdSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29t
L2tleXMuandrcyJ9
.eyJjdXN0b21lcklkIjoiYWJjLTEyMzQ1LXh5eiIsIm5hbWUiOnsiZmlyc3QiOiJ
Kb2huIiwibGFzdCI
6IlNtaXRoIn0sImVtYWlsIjpbImpzbWl0aEBlbWFpbC5jb20iXX0.1tQHVkd0aU5
EpE-oyY_R6tJjnYq
n2OQtprp8vKVZYtwYgzR6BdHPsi4xhgxoEq8iTAGJqjLdi1UnMLyPVABUYw
```

Encrypted

```
lA9EWgdnWrJ0pfHgR8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJYH
7XwpvG6YoceqWYF7
vIfpud6ZLW6Ma8sNVna00ptIAV5KvWeQIAO6-
bY8Ev47l7hPILaMawULTomzPg_HsWz2NvRqclrRn5nE
RNJOef0yjJWCgBfe2CRuj4r7XWpr1ei8xsCmA4nA__JmDiimFhhWtuYrL61SgSig
mCj9-Vfd5tL5rgGg
9XC4UX3JMDw6P-qHthR-hCZrjxUbDJcW_NWJnHvjkRjgT1qzJLCmRf0A-g-
kthE8sQn4Bfdkf7FbVOUs
SJQccKeAyFuUIKrhRM5xRFhnv64aQLxKaQCleujML15K8l8YweqCf4XSUc-
qNVsrgVRuF8UAOXLAYEM
```

3. Sender creates JWE containing encrypted sensitive data, encrypted CEK , ephemeral public EC key of Sender and some other additional material.

Protected Header

```
{
  "alg": "ECDH-ES+A256KW",
  "cty": "JWT",
  "enc": "A256GCM",
  "epk": {
    "crv": "P-256",
    "kty": "EC",
    "x": "SW0RyC-JYmzdVolujnxEZDz-c8hgtUw8rfq5ELAT71I",
    "y": "IrYEPfZdoYWbydhqjPJyKv008BSRH89_T6tSwfBK1lg"
  }
}
```

JWE JSON Serialization

```
{
  "ciphertext":
"lA9EWgdnWrJ0pfHgR8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJY
H7XwpvG6YoceqWYF7vIfpud6ZLW6Ma8sNVna00ptIAV5KvWeQIAO6-
bY8Ev47l7hPILaMawULTomzPg_HsWz2NvRqclrRn5nERNJOef0yjJWCgBfe2CRuj
4r7XWpr1ei8xsCmA4nA__JmDiimFhhWtuYrL61SgSigmCj9-
Vfd5tL5rgGg9XC4UX3JMDw6P-qHthR-
hCZrjxUbDJcW_NWJnHvjkRjgT1qzJLCmRf0A-g-
kthE8sQn4Bfdkf7FbVOUsSJQccKeAyFuUIKrhRM5xRFhnv64aQLxKaQCleujML15
K8l8YweqCf4XSUc-qNVsrgVRuF8UAOXLAYEM",
  "iv": "sjchiFykIZ9ab_OW",
  "encrypted_key":
"6ZKl9aHBfd7hFFlUBmr16PGRHt2po8teuDBZzVTi0AYGpPHw87yBVg",
  "tag": "JUhDoBNzNkDVZcCn2hwf8Q",
  "protected":
"eyJhbGciOiJFQ0RILUVTK0EyNTZLVyIsImN0eSI6IkpXVCIsImVuYyI6IkEyNTZ
HQ00iLCJlcGsiOnsiY3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJTVzBSeUM
tSlltemRWb2x1am54RVpEei1jOGhndFV3OHJmcTVFTEFUNzFJIiwieSI6IklyWUV
QZlpkb1lXYnlkaHFqUEp5S3YwMDhCU1JIODlfVDZ0U3dmQksxbGcifX0"
}
```

4. Sender sends JWE to Receiver

HTTP Header

```
Content-Type: application/jwt
```

HTTP Body

```
eyJhbGciOiJFQ0RILUVTK0EyNTZLVyIsImN0eSI6IkpXVCIsImVuYyI6IkEyNTZH
Q00iLCJlcGsiOnsi
Y3J2IjoiUC0yNTYiLCJrdHkiOiJFQyIsIngiOiJTVzBSeUMtSlltemRWb2x1am54
RVpEei1jOGhndFV3
OHJmcTVFTEFUNzFJIiwieSI6IklyWUVQZlpkb1lXYnlkaHFqUEp5S3YwMDhCU1JI
ODlfVDZ0U3dmQksx
bGcifX0.6ZKl9aHBfd7hFFlUBmr16PGRHt2po8teuDBZzVTi0AYGpPHw87yBVg.s
jchiFykIZ9ab_OW.
lA9EWgdnWrJ0pfHgR8Cq9uiDMyywZieYSp1IgtHSs_mhBpOv8cYMsRi6_IWlbJYH
7XwpvG6YoceqWYF7
vIfpud6ZLW6Ma8sNVna00ptIAV5KvWeQIAO6-
bY8Ev47l7hPILaMawULTomzPg_HsWz2NvRqclrRn5nE
RNJOef0yjJWCgBfe2CRuj4r7XWpr1ei8xvvDXpDA__JmDiimFhhWtuYrL61SgSig
mCj9-Vfd5tL5rgGg
9XC4UX3JMDwoP9C94BYJ9zptnzsqNIMS_tbs22fi-
xG6UVXoKMq0GtVgnCvzuAwckTzLY_NMcM4sD8YT
```

```
Jok5bNzL-TG0YYHgXeZNZ09WmL4SAYkOejGkde3WZWwT334e-
Pmhbr2FfvvLb3ZzvVRRMN4gGCz0aC90
4Qthpw.cYfrBMaS_NXftFKiGliS5Q
```

EC Consumption

JWE Decryption

1. Receiver retrieves its static private EC key from Step 1 and ephemeral public EC key from Sender to compute shared secret which will be same as shared secret in EC CEK & KEK Generation Step 4.

Data Recipient’s Private Key

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIFoxJaJ8UtevP3F+dNWkax6neQTjeipwAewO0K/N+8SqoAoGCCqGSM49
AwEHoUQDQgAEEWXLOwDqRhiiSHkd3GCzpVZ4d1ucLH9mzOWZNWunILdrS3vkVxf4
/odFGtCYEW+y05MhkVBGVg8OCWyQHrvoJA==
----END EC PRIVATE KEY-----
```

2. Receiver derives a 256 bit AES Key (KEK) from shared secret.
3. Receiver uses 256 bit AES key(KEK) to decrypt CEK.
4. Receiver uses CEK to decrypt sensitive data (e.g. PII data of customers

4.1.3 Sending Nested JWT HTTP Response

The JWE should be sent to the Data Recipient using the compact serialized form and with a content type of `application/jwt`

```
Content-Type: application/jwt


eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0NN
In0.JdC7ydHPnn-l
rrNctngVTawb1qmldPol8Fjn0cAVsRZvXFlj5OW6MSxoc6vIn7-
LRyDARY1vHAEr3dLLD4D0FtLm6f_S
-
wDpbf8VJ2qVQb3FAmZxbv8jKFDwWXXL6d_WtkYI6IBpIUVI7qbphGR6IqHqg0kft
gr8f0_D3-JNkkNp
MLUYZAWSE8ZgRclwOHL1KD6zhBIF-uDIXFEMYo48OuEE_bdZ7wTW-0_w8oWh-
CJcxYPoJmjAeSdooahB
rsPtX2OVts_6chgfhlyAXS9uh6Qc09M1MlU-HlyhwzK8-
wiN8MBnOW12T87XOw36-nto0kDQMDf_Tk2i
gBfslXYiTA.sjchiFykIZ9ab_OW.lA9EWgdnWrJ0pfH1R8Cq9uiDMyywZieYSp1I
gtHSs_mhBpOv8cYM
sRi6_IWlbJYH7XwpvG6YoceqWYF7vIfpud6ZLW6Ma8sNVna00ptIAhQttWqqECSr
gc80ZbYajZkaZZqf
TgY2QY3pEGmzpELnC65vQF2No6fHXdd3CewdoZHagyHFhBtqqoDRAw5q-
Naj5O36GJXT--l2D1yxBTN8
```

```
tZs7FNZKqRmFngPUj17wyM27vjqC8liKc3DJM3wzFNC98ixsqjlAsiMJJ7lT_tXW
-nTY-xm-fH6xUL6Z
QvFQnBa2jnJIsQ_WYIkEOddBd_0XNJMFf62l6g6IObrHSc1pEGsviIAFFvlOXzGC
ZO7WFFh580Af2cyO
e7vTD8irUm02qnxFGIMgOjLGbS871zJ_40_T3xfe_8e7nKgX4ti4PuwDmi5aBySY
HGIv70xJq1ZQQ1Mo
hxwJ_0jUkWcTVWa-n-
xDSESMGEH_1LI7MnToMaUoYb26l9zkHWVaFce8_qMK8a6RfZkuiR_uSwynysIk
wB0xDIJOERB4EjRoWVWgYLGSduZbSpOVyTJykX70SIszoUeBQC1IQkSM4Cgpb6jr
m95ot48_Chs2G7A8
JRXUl2sx9pbfayxktr0hBNmcVx3kPoww-xbhIP6t6r_M47LmT6SIv9_zX7r7Z-
ReQDF4qBeg9fN7kWjn
9XLj8dD0aimhz6vPMOX4zAtdaWc03fe_neuU3B19A2UvTzymO4EnY4fA_dngBYzJ
PMDbC92G1oEc.4aT
8aykEG7TAVrBfS27qaw
```

4.1.3.1 Error Handling

1. See table below

2. Errors are returned unencrypted (Assuming they do not include any PII or other sensitive information)

Nested JSON Web Token Java code using Nimbus

Introduction

A Nested JSON Web Token (JWT) is a token that involves a signed and encrypted payload. The inner token is a signed payload of the JSON Web Signature (JWS) type, and the outer token encapsulates the signed token by encrypting it into a JSON Web Encryption (JWE) payload.

An example Nested JWT is provided using the Nimbus library: Nested JWT - JWS within a JWE.

Sample Solution

This sample routine generates an elliptic curve key pair for signing and encryption before the function is called. It then builds the Nested JWT, and then breaks it down by decrypting and then validating the signature.

The code sample below uses Elliptical Curve cryptography (ECC) with P-256 curve ECC keys for signing and encryption.

Item	Algorithm
Signing	ES256 - Sign the payload with the ECC signing key and hash the signature output with SHA256
Asymmetric Encryption (Key encryption)	ECDH_ES_A256K - ECDH-ES using Concat KDF and CEK wrapped with "A256KW"
Symmetric Encryption (Payload)	AES-256 GCM

Nested JWT (JWS encapsulated in JWE) using Nimbus

Java version 11.

Imports

```
package com.jpmc.crypto;

import java.io.UnsupportedEncodingException;
import java.security.*;
import java.security.interfaces.RSAPublicKey;
import java.util.*;
import com.nimbusds.jose.*;
import com.nimbusds.jose.crypto.RSADecrypter;
import com.nimbusds.jose.crypto.RSAEncrypter;
import com.nimbusds.jwt.EncryptedJWT;
import com.nimbusds.jwt.JWTClaimsSet;
```

Dependency

In the pom.xml file add this dependency.

```
<dependency>
    <groupId>com.nimbusds</groupId>
    <artifactId>nimbus-jose-jwt</artifactId>
    <version>6.0.2</version>
</dependency
```

Sample code

```
protected static void RunNimbusNestedJWTSignFirstEC( ECKey
ECEncryptionKey,
  ECKey ECSigningKey) {
try {
  if (ECEncryptionKey != null && ECSigningKey != null) {
    // Nimbus implementation
    //Create a Signing key, see functions below.
    ECKey nimbusECSenderSigningKey = ECSigningKey;

    //Create an encryption key pair, see functions below for
generation.
    ECKey nimbusECEncryptionKey = ECEncryptionKey;
    //holds the public key of the asymmetric encryption key
    ECKey nimbusECEncryptionPublicKey =
nimbusECEncryptionKey.toPublicJWK();

    System.out.println("NimbusNestedJWT-SignThenEncrypt EC
Encryption JWK: "
      + nimbusECEncryptionPublicKey.toString());

    // Create Signed JWT with EC P-256 DSA with SHA-256
    SignedJWT signedJWT = new SignedJWT(
            new JWSHeader
                    .Builder(JWSAlgorithm.ES256)
                    .keyID(nimbusECSenderSigningKey.getKeyID())
                    .build(),
            new JWTClaimsSet.Builder()
                    .subject("sally")
                    .issueTime(new Date())
                    .issuer("secure.chase.com")
                    .build());

    // Sign the JWT
    signedJWT.sign(new ECDSASigner(nimbusECSenderSigningKey));

    System.out.println("NimbusNestedJWT-SignThenEncrypt EC: JWS:
"
      + signedJWT.serialize());

    // Create JWE object with signed JWT as payload
    JWEObject jweObject = new JWEObject(
            new JWEHeader
                    .Builder(JWEAlgorithm.ECDH_ES_A256KW,
EncryptionMethod.A256GCM)
                    .contentType("JWT") // required to indicate
nested JWT
                    .build(),
            new Payload(signedJWT));

    // Encrypt with the public key
    jweObject.encrypt(new
ECDHEncrypter(nimbusECEncryptionPublicKey));

    // Serialise to JWE compact form
    String jweString = jweObject.serialize();

    System.out.println("NimbusNestedJWT-SignThenEncrypt EC: JWE
Encryption method: "
      + " JWE algorithm " +
jweObject.getHeader().getEncryptionMethod() + " "
      + jweObject.getHeader().getAlgorithm() + " " + " JWE
string " + jweString);

    System.out.println("NimbusNestedJWT-SignThenEncrypt EC: JWE
Encryption method: "
      + "JWE header " + jweObject.getHeader());

    //Now consume the Nested JWT

    // Parse the JWE string
    JWEObject jweObjectEncrypted = JWEObject.parse(jweString);

    // Decrypt with private key
    jweObjectEncrypted.decrypt(new
ECDHDecrypter(nimbusECEncryptionKey));

    // Extract payload
    SignedJWT signedJWTAfter =
jweObjectEncrypted.getPayload().toSignedJWT();
    System.out.println("NimbusNestedJWT-SignThenEncrypt EC:
After decryption payload"
      + "  (will contain JWS): " + signedJWTAfter.serialize());

    assertNotNull("Payload not a signed JWT EC ", signedJWT);

    // Check the signature
    assertTrue(signedJWTAfter.verify(new
ECDSAVerifier(nimbusECSenderSigningKey)));
    System.out.println("NimbusNestedJWT-SignThenEncrypt EC:
After decryption check"
      + " signature of JWS: "
      + signedJWTAfter.verify(new
ECDSAVerifier(nimbusECSenderSigningKey)));

    System.out.println("NimbusNestedJWT-SignThenEncrypt EC:
After decryption subject"
      + " JWS: " +
signedJWTAfter.getJWTClaimsSet().getSubject());
```

```
  }
    }
    catch (Exception e) {
        System.out.println("NimbusNestedJWT-SignThenEncryptEC:
exception: " + e);
    }
}

protected static ECKey CreateNimbusECEncryptionKey() {
    ECKey jwkECEncryption = null;

    try {
        // Generate EC key pair in JWK format
        jwkECEncryption = new ECKeyGenerator(Curve.P_256)
                .keyUse(KeyUse.ENCRYPTION) // indicate the
intended use of the key
                .keyID(UUID.randomUUID().toString()) // give the
key a unique ID
                .generate();

    } catch (JOSEException e) {
        e.printStackTrace();
    }
    return jwkECEncryption;
}

protected static ECKey CreateNimbusECSigningKey() {
    ECKey jwkECSigning = null;

    try {
        // Generate EC key pair in JWK format
        jwkECSigning = new ECKeyGenerator(Curve.P_256)
                .keyUse(KeyUse.SIGNATURE) // indicate the
intended use of the key
                .keyID(UUID.randomUUID().toString()) // give the
key a unique ID
                .generate();

    } catch (JOSEException e) {
        e.printStackTrace();
    }
    return jwkECSigning;
}
```

Nested JWT output

Below is the output of the above routine. Note the epk, ephemeral public key.

```
NimbusNestedJWT-SignThenEncrypt EC Encryption JWK:
{"kty":"EC","use":"enc","crv":"P-256","kid":"a152e75b-8e11-4f87-
908c-366552610467",
"x":"YoUQaMiqgeSmNMXSqcYiFRO4nZEXV47APaUnm4PyaPg",
"y":"iUuwma37B6YFPKLA7tv_-cczuT4WJf2WmVfsgELHr1c"}
NimbusNestedJWT-SignThenEncrypt EC: JWS:
eyJraWQiOiI1ZDhhNTdiNi00ZjRmLTQ5NTItYTY5YS
1hYjcwNTU2NzkzYjAiLCJhbGciOiJFUzUxMiJ9.eyJzdWIiOiJzYWxseSIsImlzc
yI6InNlY3VyZS5jaGFz
ZS5jb20iLCJpYXQiOjE2MDIxNzQ1ODJ9.AYBIyjeNTPEZi_Vgfg2J9FpqL59coK5
xBU_YJaxwQg-5UCfE9p
MYwoBHR7gJZqYMMPtFpPkcxzZhIFFTElBoPz7mARRS-
r8N1iNUKbMP547qay4MIF-BtOxpyX8uTsint5cjx
hxtP8gsV-MEh-YXs89SNeZJo-cr_41KXr3_9XLUHrPo
NimbusNestedJWT-SignThenEncrypt EC: JWE Encryption method:  JWE
algorithm A256GCM
ECDH-ES+A256KW  JWE string
eyJlcGsiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsIngiOiIzbWt
nWjFDNXFxY19mQnhwc3NaeTRFSlMxVmFScGd2dS14aHFzenl1azYwIiwieSI6Im1
wQUVPRk1IamdCRk1GM
DN2bG84YnoweXFrR1NQajNna1RZUmtsV3BXX0kifSwiY3R5IjoiSldUIiwiZW5jI
joiQTI1NkdDTSIsImF
sZyI6IkVDREgtRVMrQTI1NktXIn0.YdWec3hyjvLSSDjjq4uZcBOG18BTZPO7L79
6v_3ytUmG9CyjWfjhZ
w.E8t4MircnwButjA1.a4w2_IRmxykljL7qfgXHEoz-my_g6qNFgJNdGTX_fi-
xAF8mJSG1gNSJzlLPq_X
_3npxPUmKbROmJmc3DOKxHukpElMVlWhhYo1XYHX9CwirZw77nO8McdTR9LNdPvF
bsjEwQ3QiamF56EPfl
zM7ABS7FBAqYaIwznBXJA_Vqvixtc-
KJOd1kIGcj9WqZ7xReFbk2KeRJEvya1l93Ajzia1eACcSyXEKnXR
wU23X_BPRQfv4eGHPoeU3tZwIiSAVV51oYHhcK8KSM4mmrHZ5doC0puJHqJnCgGS
OBiBcwNrBfSKYK7Y1i
djTwoRkHmWcvOOkJLfhFLTeixrPUVP0rFRdePItvb8w-
jrUBrA3yMhM5VbXCdrPkH7WwhIiHffdKDZ38eu
hoSwo3AyreUqBUA6OLu46J8vS0iGJHm4xCOTOXih9eZntrJhDv5klYg.i5IY3tmt
TDbqPdGGV4_H5g
NimbusNestedJWT-SignThenEncrypt EC: JWE Encryption method:
JWE header
{"epk":
```

```
{"kty":"EC","crv":"P-256",
"x":"3mkgZ1C5qqc_fBxpssZy4EJS1VaRpgvu-xhqszyuk60",
"y":"mpAEOFMHjgBFMF03vlo8bz0yqkGSPj3gkTYRklWpW_I"}
,"cty":"JWT",
"enc":"A256GCM",
"alg":"ECDH-ES+A256KW"}
NimbusNestedJWT-SignThenEncrypt EC: After decryption payload
(will contain JWS):
eyJraWQiOiI1ZDhhNTdiNi00ZjRmLTQ5NTItYTY5YS1hYjcwNTU2NzkzYjAiLCJh
bGciOiJFUzUxMiJ9.
eyJzdWIiOiJzYWxseSIsImlzcyI6InNlY3VyZS5jaGFzZS5jb20iLCJpYXQiOjE2
MDIxNzQ1ODJ9.AYBI
yjeNTPEZi_Vgfg2J9FpqL59coK5xBU_YJaxwQg-
5UCfE9pMYwoBHR7gJZqYMMPtFpPkcxzZhIFFTElBoP
z7mARRS-r8N1iNUKbMP547qay4MIF-BtOxpyX8uTsint5cjxhxtP8gsV-MEh-
YXs89SNeZJo-cr_41KXr
3_9XLUHrPo
NimbusNestedJWT-SignThenEncrypt EC: After decryption check
signature of JWS: true
NimbusNestedJWT-SignThenEncrypt EC: After decryption subject
JWS: sally
RunNimbusNestedJWTSignFirst EC: Total ns: 32099895- In ms
32.099895
```

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 includes 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=
Accept: application/json
Accept-Charset: UTF-8 Accept-Encoding: gzip
```

The FDX API server will use HTTP status codes to indicate the success or failure of a request. Response code details specific to FDX API follow. For status codes other than 200, the HTTP response body must contain an Error Entity.

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. We recommend that both the FDX API client and server use certificates. Additionally, FDX API server responses should include Cache-Control headers:

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

to prevent any caching or storing of the response.

6.1.2 Request Authorization

The FDX API client does 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 uses the Authorization request header with a Bearer or MAC token. Bearer tokens are recommended although the server has option to issue MAC tokens as an alternative if the client supports it. How to obtain this token was detailed in the Security Model section.

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

6.1.3 Content Negotiation

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

The FDX API client uses 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 uses 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 uses 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 uses 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 uses 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 will 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 returns a Date header with every response.

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

6.1.5 Host

The Host request header field specifies 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 supplies 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 can 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 optionally can supply the customer's IP address if this data is available or applicable.

```
CustomerIPAdress: 0.0.0.0
```

6.1.9 Interaction Tracking

The FDX API client uses the FDX `x-fapi-interaction-id` request header to inform server of an interaction tracing identifier. A FDX `x-fapi-interaction-id` is unique to an FDX API client instance. The FDX `x-fapi-interaction-id` allows support people 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: byx24A1a111
```

The FDX API server uses the FDX `x-fapi-interaction-id` response header to inform the client of the response FDX `x-fapi-interaction-id`. The FDX `x-fapi-interaction-id` value must be same as the corresponding client request header value.

```
x-fapi-interaction-id: byx24A1a111
```

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 the 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
```

6.2 Errors

When FDX API servers are unable to fulfill a request, they should send 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
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	401	Authenticated customer does not have the authorization to perform this action
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
703	Invalid date range	400	If the start date is not earlier than the end 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 we currently do not support. 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.

Error Code	Error Message	HTTP Status Code	Description
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	404	Amount for transfer is not in valid range, i.e. is not greater than 0
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.
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
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

7. Pagination

The FDX API uses cursor-based pagination by leveraging an opaque cursor in the response that can point to an 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 can use time based limits where the result set might be less than the limit and each page could contain a different number of elements.
- `offset` an optional field to specify an opaque cursor used to retrieve a portion of the requested data

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

- `page.nextOffset` is an opaque identifier used to retrieve the next page of results
- `page.prevOffset` is an opaque identifier used to retrieve the previous page of results
- `page.totalElements` contains the number of total elements matching the query
- `links.next.href` represents the URL used to retrieve the next page. 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. This URL should contain the original parameters of the search query such as `startDate` and `endDate` if applicable

The client should look for `page.nextOffset` 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 the key is no longer returned in the response.

> **Note**: The offset query parameter is a cursor for the collection to index from. It could be a number or an opaque string. If it is a number it SHOULD use a 0-based index.
>
> **Note**: The optional limit query parameter indicates the number of records that is requested by the client. The API server SHOULD have reasonable defaults.

7.1 Sample implementation

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

7.1.1 Initial Response

```
{
  "page":{
    "nextOffset": "nextoffset@10",
    "totalElements": 100
  },
  "links":{
    "next": {"href":"/accounts?offset=nextoffset@11&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":{
    "prevOffset": "prevoffset@81",
    "totalElements": 100
  },
  "links":{
    "prev": {"href":"/accounts?offset=prevoffset@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.nextOffset` and `links.next.href` are not returned in the response. This indicates to the client that no further pages are available.

The response attribute *prev* identifies the previous page url and offset.

8. Versioning

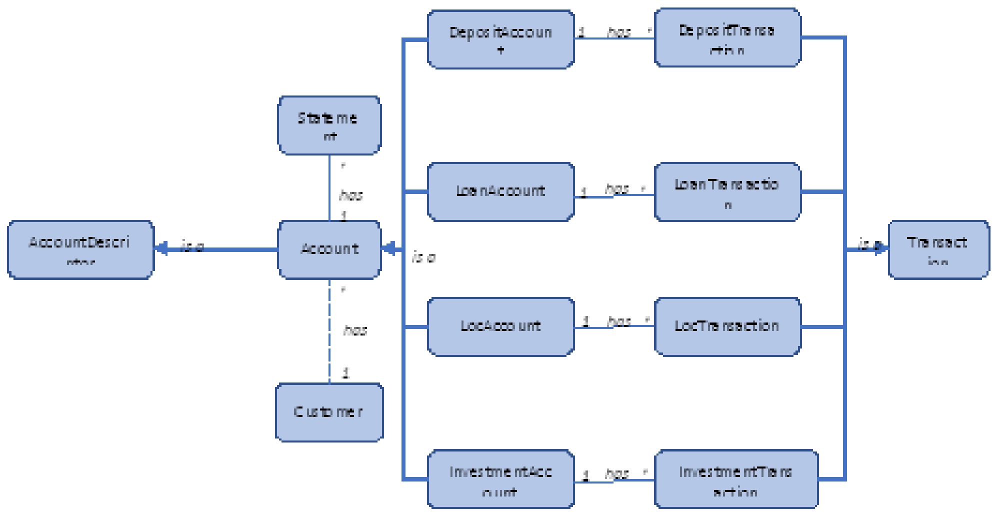
APIs should be designed for change in order to expose new behavior or data.

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 <i>in place of</i> 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, we recommend that <u>no more than three</u> approved major versions of a service exist at any one time. When a fourth major version is released into production, the oldest version will 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 Identity

The User entity is not expressed in FDX API messages.

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

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

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

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

The FDX API identity properties 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 Identity

OAuth creates a surrogate identity for a Login. An FDX API server does <b>not</b> expose the financial institution's principal identity of the Login. To limit the exposure of personally identifiable information, the other identities transmitted by the FDX API server should be a surrogate identity. Surrogates must provide the same uniqueness constraints on the entity relationships as described above. Any surrogate identities must be long term persistent.

If the Data Provider's account identity is considered confidential, a surrogate identity 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",
  "programId": "4FRCCQvGW0GZEMtsOQWlkQ",
  "programUrl": "https://www.marriott.com/loyalty.mi",
  "memberships": [
    {
      "accountIds": [
        "af0f8e58-9649-4c29-bab2-0295d522cd6f",
        "e75e31eb-bf04-4d87-9f20-4554f63a639e"
      ],
      "businessOrConsumer": "CONSUMER",
      "customerId":
"kBA5C3d7cBK9DuRngsQRwt6Ydo80bjYDR7n4O5yCKshizuS7hOZJ4cAev",
      "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 set qualifying to true indicate that those balances are used by the reward program to determine a member's progress towards 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":
"kBA5C3d7cBK9DuRngsQRwt6Ydo80bjYDR7n4O5yCKshizuS7hOZJ4cAev"
```

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:

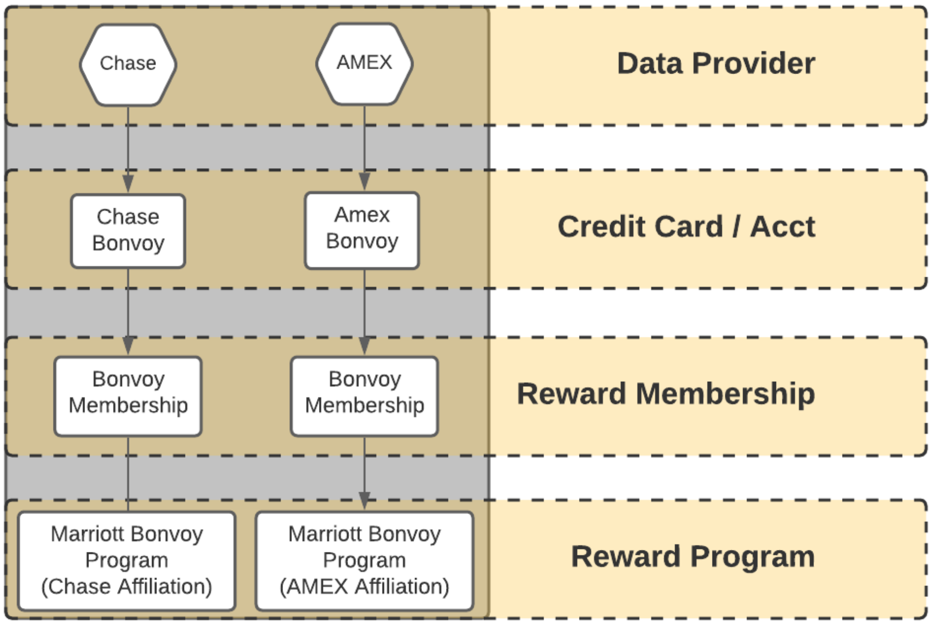
```
  "programId": "4FRCCQvGW0GZEMtsOQWlkQ",
  "programName": "Marriott Bonvoy",
    "programUrl": "https://www.marriott.com/loyalty.mi",
```

Field	Description
programId	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

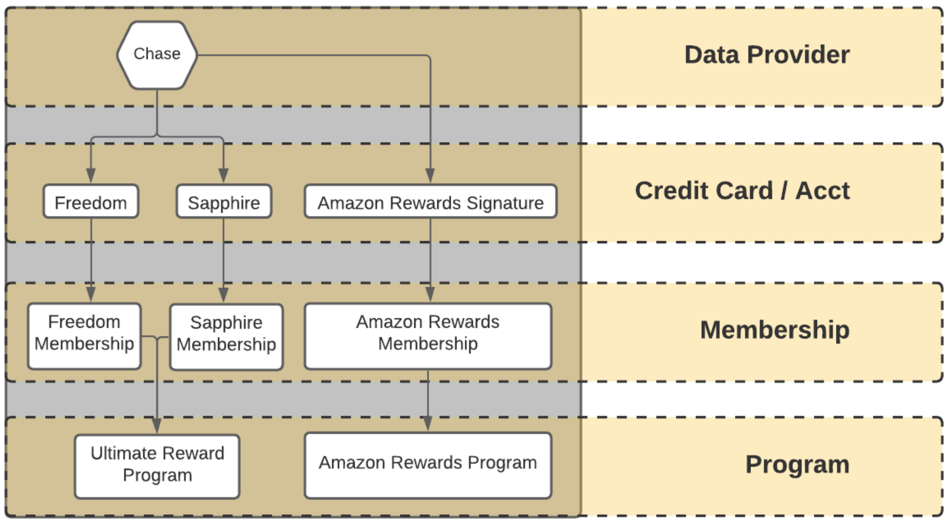
10.2.1 Example 1

The follow diagram shows that a single reward programs 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 programId.



10.2.1 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",
      "programId": "4FRCCQvGW0GZEMtsOQWlkQ",
      "programUrl": "https://www.marriott.com/loyalty.mi",
      "memberships": [
        {
          "accountIds": [
              "af0f8e58-9649-4c29-bab2-0295d522cd6f",
              "e75e31eb-bf04-4d87-9f20-4554f63a639e"
          ],
          "businessOrConsumer": "CONSUMER",
          "customerId":
"kBA5C3d7cBK9DuRngsQRwt6Ydo80bjYDR7n4O5yCKshizuS7hOZJ4cAev",
          "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",
      "programId": "iqOtPUEYb0Go6SCL8As4fQ",
      "programUrl": "https://www.starbucks.com/rewards",
      "memberships": [
        {
          "accountIds": [
              "89cf3262-ff38-4f6a-afbc-aafc50cac751"
          ],
          "businessOrConsumer": "CONSUMER",
          "customerId":
"kBA5C3d7cBK9DuRngsQRwt6Ydo80bjYDR7n4O5yCKshizuS7hOZJ4cAev",
          "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 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, categoryId's 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/Resaurants",
      "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 cross fit 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",
  "transactionCategory": "Hardware",
  "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 is 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 Bill Pay API makes use of 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 canceled, 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 Bill Pay 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 the Synchronization section 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 non-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 expiresOn field will be populated. Once the expiration date 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 expiresOn date passes.

As a payee is used within the payments system, the expiresOn date will update to reflect a new expiration date 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 date could have been updated between the last sync and the last recorded date of expiration.

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**, **In Process**, **Processed**, **No Funds**, **Failed** or **Cancelled** state. A newly created payment will be in the **Will Process** state until the scheduled time occurs, the payment is canceled or a failure occurs. The payment will move into the **In Process** 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 **No Funds** 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 the Synchronization section 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 non-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 paymentsForRecurringPayment (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 **Will Process** 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 **Will Process** 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 canceled 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 paymentsForRecurringPayment 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 the Synchronization section 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 "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 non-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 "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 canceling a recurring payment. The paymentsForRecurringPayment 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 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 transactions 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.

Duplicate Transaction Use Case 1a: Acknowledged successful transaction

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

Duplicate Transaction Use Case 1b: Failed to acknowledge a succeeded transaction 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 operation with the original transaction ID,
4. Then the server will return HTTP Code 201 Created, along with the payment reflecting the 5th of the month.

The operation was executed in step 1 and was not attempted in step 4

Duplicate Transaction Use Case 1c: Failure to receive original transaction

1. Given a client attempting to schedule a payment that is not received by the server.

2. When the client retries the operation with the original transaction ID,

3. Then the server will return HTTP Code 201 Created, along with the newly created payment

The operation was executed in step 3

Duplicate Transaction Use Case 2a: Acknowledged failed transaction

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

Duplicate Transaction Use Case 2b: Failure to acknowledge a failed transaction

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 operation with the original transaction ID,

4. Then the server will return HTTP Code 400 along with the original error response.

The operation was executed in step 1 and was not attempted in step 4

Duplicate Transaction Use Case 2c: Failure to receive original transaction

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 operation with the original transaction ID,

4. Then the server will return HTTP Code 201 Created, along with the newly created payment

The operation was executed in step 4. As compared to Use Case 2b, the operation 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 support synchronization. As of this writing Payees, Recurring Payments and Payments 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 pass the stored update ID back to the endpoint used to retrieve the original data set. The server will return created or updated entities using the update ID as reference. The client is expected to replace the local entities with their updated version. The server will return a new update ID that the client will store. The new update ID 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 Update ID

An update ID can expire. Typically due to the amount of time between queries. The server will indicate as such by returning the Update ID 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 a soft-deleted object is hard-deleted, the update ID should be considered expired to allow for the client to discover deleted objects via a reseed.

Update IDs are expected to remain valid unless necessary for purge, periodic cleanup and/or other data retention policies. If update IDs are not valid for a sufficient enough period, the client will need to reseed resulting in significant resource utilization.

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_v2021.0* document from FDX, published with release v5.0 in October 2021.

13. QR Codes

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_v2020.0* document from FDX, published with release v4.6 in May 2021.

14. Consent

14.1 Overview: Three-Party Model of Consent

Consent represents the following to each of the involved parties:

- Data Recipient ("DR") views the 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.
- Data Provider ("DP") sees consent as their permission to provide access to the Data Recipient for the End User's financial data. The Data Provider generally makes no assertions about Data Recipient's use of the data.
- End User ("EU") sees consent as their record that they have given permission for data access. The End User is not expected to disambiguate access vs. use of their financial data.

14.1.1 Considerations for Intermediaries

In the North American ecosystem, the DR's interface(s) for initiating 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" may 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 versions of this API.

14.2 Proposed Consent Lifecycle, v1

Consent occurs as follows in the three-party model:

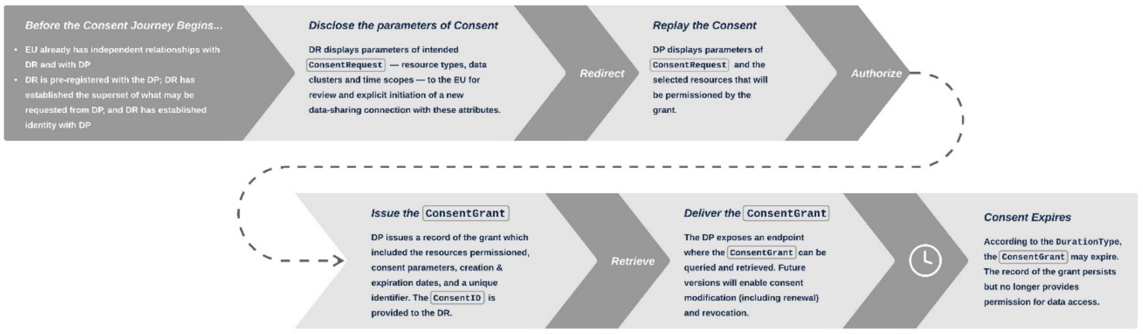
Data Recipient discloses the parameters of the consent request to the End User; Data Provider collects authorization from End User; Data Provider provides the record of consent to the Data Recipient.

These interactions are predicated on a number of conditions:

- Before the consent journey begins, End User MUST have independent relationships with Data Recipient and with Data Provider;
- Before the consent journey begins, Data Recipient MUST be pre-registered with the Data Provider. Resultantly, Data Recipient MUST have established a superset of what may be requested from Data Provider, and Data Recipient has established identity with Data Provider (both in terms of client authentication and in terms of an existing business agreement);

- End User MUST initiate their consent journey (requesting new data access) from within Data Recipient's experience/application:
- Data Recipient determines the types of data access it intends to access from Data Provider, and MUST disclose its intent to the End User;
- End User MUST actively authorize Data Provider to enable Data Recipient's access to End User's data.

Lifecycle of Consent Objects



Before the Consent Journey Begins...

- EU already has independent relationships with DR and with DP
- DR is pre-registered with the DP; DR has established the superset of what may be requested from DP, and DR has established identity with DP

Disclose the parameters of Consent

DR displays parameters of intended `ConsentRequest` — resource types, data clusters and time scopes — to the EU for review and explicit initiation of a new data-sharing connection with these attributes.

Replay the Consent

DP displays parameters of `ConsentRequest` and the selected resources that will be permissioned by the grant.

Issue the `ConsentGrant`

DP issues a record of the grant which included the resources permissioned, consent parameters, creation & expiration dates, and a unique identifier. The `ConsentID` is provided to the DR.

Deliver the `ConsentGrant`

The DP exposes an endpoint where the `ConsentGrant` can be queried and retrieved. Future versions will enable consent modification (including renewal) and revocation.

Consent Expires

According to the `DurationType`, the `ConsentGrant` may expire. The record of the grant persists but no longer provides permission for data access.

14.3 Consent Request, v1

In plain language, a `` `ConsentRequest` `` entity consists of the following elements:

1. **Time Scopes** — a collection of parameters asserting Duration Type, Duration Period of consent, and Lookback Period which asserts the 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; Data Clusters are described in FDX RFC 0167.

14.4 Consent Grant, v1

In plain language, a `ConsentGrant` entity consists of the following elements:

1. **ID** — an identifier for the record, to permit operations such as retrieval; the `ConsentId` MUST be unique to the Issuer (DP) and it MUST be a URL-safe string;
2. **Consent Status** — declaration of the current status of the grant;
3. **Time Scopes** — a collection of parameters asserting Duration Type, Duration Period of consent, and Lookback Period which asserts the 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 sharing chain. Descriptive information is collected during Data Recipient registration at Data Provider, 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; Data Clusters are described in FDX RFC 0167.

14.5 API Interactions, v1

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

- Granting consent
- Querying and/or retrieving the current state of consent

Modification and revocation of consent are not part of this API definition and will be addressed in a subsequent version.

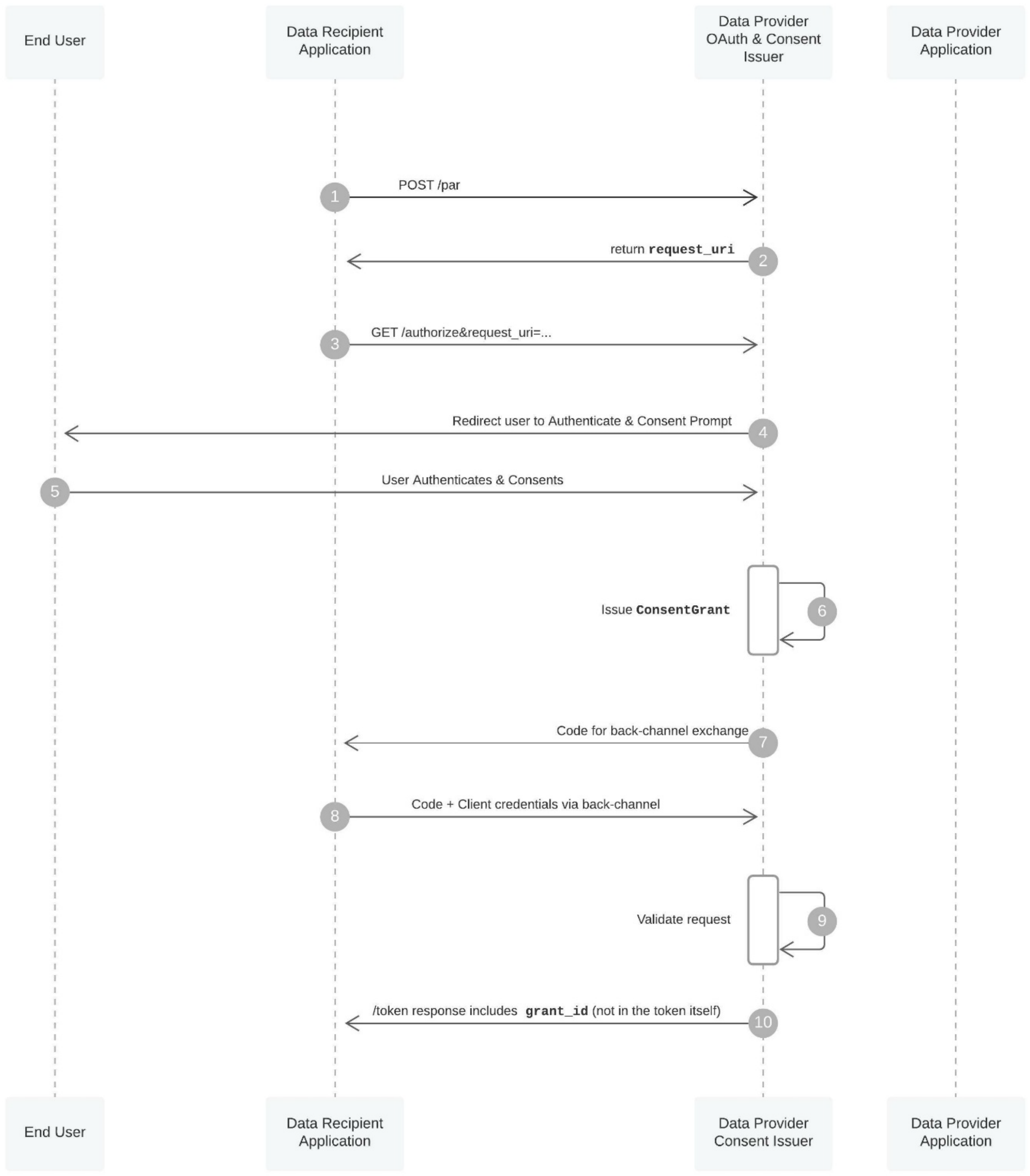
14.5.1 Granting Consent

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. When silent on a particular mechanism of authentication or authorization, this proposal intends to enable parties freedom to implement in a manner they see fit.

14.5.2 Create Consent Operation employing `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.

POST /par


End User
Data Recipient Application
Data Provider OAuth & Consent Issuer
Data Provider Application
1
POST /par
2
return request_uri
3
GET /authorize&request_uri=...
4
Redirect user to Authenticate & Consent Prompt
5
User Authenticates & Consents
Issue ConsentGrant
6
7
Code for back-channel exchange
8
Code + Client credentials via back-channel
Validate request
9
10
/token response includes grant_id (not in the token itself)
End User
Data Recipient Application
Data Provider Consent Issuer
Data Provider Application

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

The `authorization_details` request parameter MUST contain a JSON-formatted object with two members in compliance with the RAR format specified by RAR memo, § 2 the `type` parameter with value `fdx_v1.0`, and a `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"
            ]
          }
        ]
      }
    }
  ]
}
```

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

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

Endpoint behavior and responses defined in detail by PAR memo, § 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, § 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 grant

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, § 5

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

14.5.3 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:

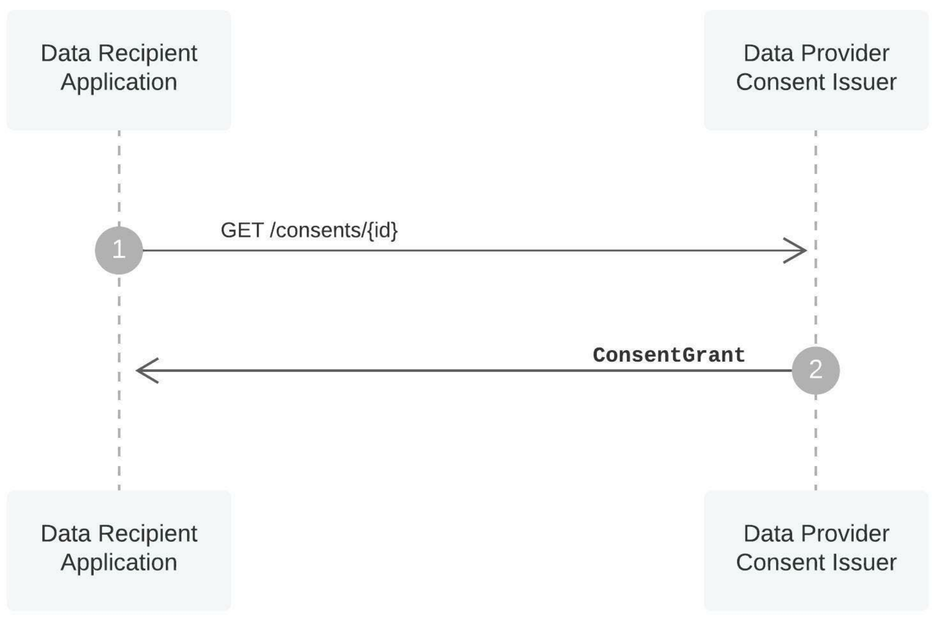
```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=invalid_scope
```

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

14.5.4 Retrieving Consent `GET /consents/{consentId}`

Request the record of Consent Grant. Request originates from the Data Recipient.

GET /consents


Data Recipient Application
Data Provider Consent Issuer
1
GET /consents/{id}
ConsentGrant
2
Data Recipient Application
Data Provider Consent Issuer

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.

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-notification.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 new transactions or an FI has a planned outage.

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. The server will respond with Access Token
    - Scope - `write: notification subscription`
    - Scope - `write: publish notification`
- 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 success response must be sent back to the requesting entity once the subscription is successful.
    - The notification types and categories need to be defined in the 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 call back URL should specify a link 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 success response.
    - Resiliency guidelines and best practice recommendation should be implemented for up to three retries on the /notification 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 outlined in the *fdxapi.event-notification.yaml*.

15.1 Event Notifications Implementation for Consent Updates

Payload Schema

The event notification framework is rich and extensible to support many scenarios. We only need to satisfy the following sub-schema

```
NotificationCategory:
  type: string
NotificationType:
  type: string
notificationPayload:
  type: object
  properties:
    id:
      type: string
    idType:
      type: string
    customFields:
      type: array
      items:
        $ref: '#/components/schemas/FiAttribute'
…. as follows:
{
  "category": "CONSENT",
  "type": "CONSENT_REVOKED" // or "CONSENT_UPDATED"
  "notificationPayload": {
    "id": "TheConsentID" // adheres to
'#/components/schemas/ConsentId', for example "9585694d3ae58863"
    "idType": "CONSENT",
    "customFields": {
      "name": "INITIATOR",
      "value": "TheInitiator" // adheres to
'#/components/schemas/PartyType', for example "INDIVIDUAL"
    }
 }
```

Using this and an optional HATEOAS URL defined in the *fdxapi.event-notification.yaml*, the receiver can fetch the 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.*

```
{
  /*
  ... other fields omitted for brevity ...
  */
  "category": "CONSENT",
  "type": "CONSENT_UPDATED",
  "notificationPayload": {
    "id": "9585694d3ae58863",
    "idType": "CONSENT",
    "customFields": [
      {
        "name": "INITIATOR",
        "value": "INDIVIDUAL"
      }
    ],
  }
  "url": {
    "href":
"https://api.xyz.com/fdx/v4/consents/9585694d3ae58863",
    "action": "GET"
  }
}
```

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 for any Data Recipient's which the user consented access for. The Data Provider will send a notification to Wonder Wallet alerting them that consent was revoked.*

```
{
  /*
  ... other fields omitted for brevity ...
  */
  "category": "CONSENT",
  "type": "CONSENT_REVOKED",
  "notificationPayload": {
    "id": "9585694d3ae58863",
    "idType": "CONSENT",
    "customFields": [
      {
        "name": "INITIATOR",
        "value": "DATA_PROVIDER"
      }
    ],
  }
  "url": {
    "href":
"https://api.xyz.com/fdx/v4/consents/9585694d3ae58863",
    "action": "GET"
  }
}
```

Calling back for detail:

Once consent changes, the sender will issue a 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 webhook containing CONSENT_UPDATED and the consent ID.

Appendix - Change Log - Prior Releases

Version	Date	Originator	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)	
			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:	

Version	Date	Originator	Reason for Change	Ratified
			* 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
		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	

Version	Date	Originator	Reason for Change	Ratified
			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:<ul><li>TelephoneNumber RFC</li><li>DeliveryAddress RFC</li><li>CustomerName RFC</li></ul>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. 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	

Version	Date	Originator	Reason for Change	Ratified
			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. <ul><li>_nickName_ is the user specified name for the account, used in user interfaces (UIs) to assist in account selection</li><li>_productName_ is the Marketed product name for this account, used in UIs to assist in account selection</li><li>_accountNumberDisplay_ is typically the masked account number that is displayed on a bank's UI.</li></ul>	
	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:<ul><li>"Th" has been corrected to "The" in section 6 Message Syntax".</li><li>Accounts DetailsRequest" (plural) has been changed to "AccountDetailsRequest" (singular).</li></ul>	
4.1	April 2020	Paul Allen	Implemented the final IRS 2019 tax form changes for Tax1065K1, Tax1120SK1	

Version	Date	Originator	Reason for Change	Ratified
		Clyde Cutting Bruce Wilcox		
			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, 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,	

Version	Date	Originator	Reason for Change	Ratified
			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 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	

Version	Date	Originator	Reason for Change	Ratified
			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 Bill Pay 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.	
		Jeff Johnson	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

Version	Date	Originator	Reason for Change	Ratified
		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 '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,	

Version	Date	Originator	Reason for Change	Ratified
			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 Clyde Cutting	RFC 0113 Updates schema documentation for Timestamp as string format date-time 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	

Version	Date	Originator	Reason for Change	Ratified
			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. 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.	

Version	Date	Originator	Reason for Change	Ratified
		Clyde Cutting	RFC 0147 Add FDX Tax1099ConsolidatedStatement	
		Camellia George Ben Simmons	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 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	RFC 0165 FDX US Tax v5.0 synch with OFX Tax 2020.0: Add Tax1099B.securityDetails.correctedCostBasis	

Version	Date	Originator	Reason for Change	Ratified
		Clyde Cutting		
		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	
		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_.	