



FINANCIAL[™]
DATA EXCHANGE

Alternate Authentication
for Tax Documents

Version 2023.0
December 2023



Legal Notice

FDX is a standards body and adopts this Alternate Authentication for Tax Documents specification for general use among industry stakeholders. Many of the terms, however, are subject to additional guidance under prevailing laws, industry norms, and/or governmental regulations. While referencing certain laws that may be applicable, readers, users, members, or any other parties should seek legal advice of counsel relating to their particular practices and applicable laws in the jurisdictions where they do business. See FDX's complete Legal Disclaimer located at <http://www.financialdataexchange.org> for other applicable disclaimers.

Revision History

| Document Version | Notes | Date |
|------------------|--|---------------|
| 2021.0 | Initial Document Release | October 2021 |
| 2023.0 | Updates for QR Code transmission of Basic Auth credentials | December 2023 |

Contents

| | |
|---|-----------|
| WHY AN ALTERNATE API AUTHENTICATION FOR TAX DOCUMENTS? | 4 |
| <i>Data Hosting Services</i> | 5 |
| EXPLANATION OF ALTERNATIVE API AUTHENTICATION | 6 |
| <i>User Experience</i> | 6 |
| <i>Overall Software Flow</i> | 7 |
| <i>API Token Generation</i> | 9 |
| <i>Communicating Document ID and Passcode via QR Code</i> | 10 |
| IMPLEMENTATION REQUIREMENTS | 11 |
| <i>Techniques for Risk Mitigation</i> | 11 |
| MITIGATION OF SECURITY RISKS | 13 |
| <i>Password-Based Threat Model</i> | 13 |
| APPENDIX 1: SAMPLE CODE FOR JAVA DEVELOPERS | 15 |
| <i>Create token and related request header</i> | 15 |
| <i>Read request header and decode token</i> | 17 |

Why an Alternate API Authentication for Tax Documents?

Requests to FDX API servers are secured by tokens in the request header.

For example, an HTTP tax document request might appear as follows:

```
GET /fdx/v5/tax-forms?resultType=details HTTP/1.1
Accept: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJjbGkiOiJ0YXhjbGkiLCJ1c2VyIjoiodEYmZQ1Njc4NSIsInBhc3MiOiJGWko1NTY0TkIzMCI6MTYxMjQ2NTQ2NSwiZXhwIjo
xNjEyNDY5MDY1fQ.kjl4U1x10a4yrkws08IBclW4IXheFA1jxhE70cGzdJU
User-Agent: Jersey/2.29 (HttpURLConnection 1.8.0_252)
Host: api.taxdocserver.com
Connection: keep-alive
```

The token shown in the Authorization header record is generally obtained by the client in an OAuth2 flow. See the “FDX Financial-Grade API Security Specification V3.4” file published by FDX.

However, an OAuth2 flow is often **not available** in the case of annual tax documents. There are multiple use cases. The following section details one use case where OAuth2 tokens are not available.

IMPORTANT: Financial Data Exchange always recommends that the API should be properly secured per the FDX API Security Model v 3.4. The approach in this document must be adopted within the very narrow scope of the constraints and risk mitigation strategies described below. FDX encourages the involved parties to work towards standing up the requisite infrastructure to move to secure approaches per the FDX API Security Model.

Data Hosting Services

The production and hosting of annual tax document data is frequently outsourced by a tax document issuer (Data Provider) to a trusted third party (Data Hosting Service).



The hosting arrangement has the following characteristics:

- The tax document issuer (Data Provider) has established a data sharing agreement with a tax software company (Data Recipient)
- The tax document issuer informs the tax software company that API requests for their tax data should be sent to the designated Data Hosting Service
- The taxpayer has no relationship with—nor knowledge of—the Data Hosting Service
- The Data Hosting Service either produces the hosted data or receives it from the tax document issuer and then hosts it on their behalf

In this scenario, an OAuth2 flow is not possible because the Taxpayer has no credentialed relationship with the Data Hosting Service. An alternative authentication mechanism is required. The next section will explain the alternative authentication to be used.

Explanation of Alternative API Authentication

User Experience

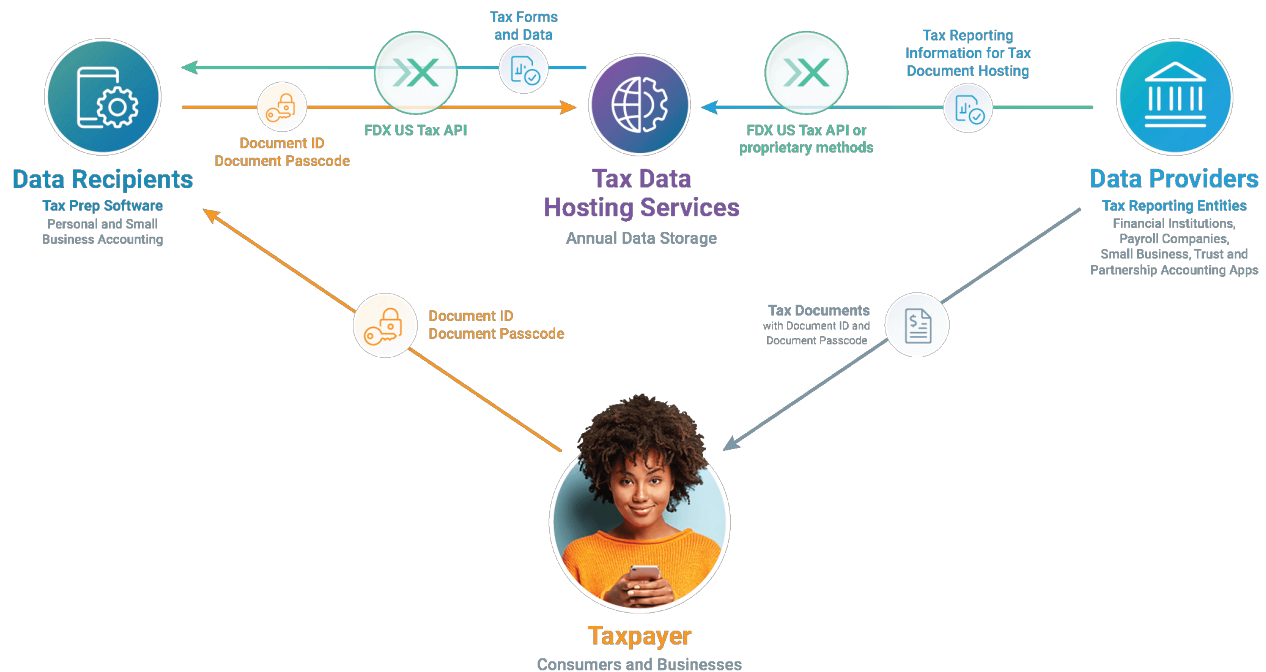
In their income tax software, taxpayers select a screen which has the name and/or logo of their tax document issuer. The screen instructs the taxpayer to enter two pieces of information visible on their tax document received in the mail or email or downloaded from the issuer website. The tax software then sends these two data items in a request to the API server of the Data Hosting Service. The detailed process is as follows:

The Data Hosting Service then retrieves and returns the taxpayer's data as shown on the paper or PDF tax document. The tax software can then import it into the tax return of the user. This import process saves the taxpayer the tedious and error-prone effort of manually entering their data from the tax document.

The two items to be entered from the tax document are determined by the producer of the document. In general, one item provides a unique id for the document and the second item is complex string of characters that is unlikely to be guessed. Together they function as a unique and secret "document name" and "password" combination, which we will call here the "Document ID" and "Document Passcode" or just "ID" and "Passcode".

Overall Software Flow

The FDX ecosystem flow for this alternative authentication method is as follows:



1. Data Provider / Tax Reporting Entity (tax document image provider):

- Displays “Document ID” and “Document Passcode” data on tax document images (generally in header area)
- “Document Passcode” must appear on no more than one tax document image
- Neither “Document ID” nor “Document Passcode” can contain a colon (":") character

2. Tax Preparation Software (the API client):

- Has established contractual and technology relationships with Provider / Tax Reporting Entity
- Uses the “Basic” HTTP authentication scheme as defined in [IETF RFC 7617](#), which transmits Document ID and Document Passcode as an “ID”:“password” pair, encoded using base64
- Encryption is provided through normal SSL encryption of headers over HTTPS
- Collects name of Provider / Tax Reporting Entity from taxpayer
- Collects the “Document ID” and “Document Passcode” for use as user-id and password from taxpayer
- Constructs the “user-pass” by concatenating “Document ID”, a single colon (":") character, and “Document Passcode”
- Encodes the “user-pass” into an octet sequence
- Creates the basic credentials by encoding this octet sequence using Base64 (IETF RFC 4648, Section 4) into a sequence of US-ASCII characters (IETF RFC 0020)

- Submits the credentials in header of request to GET /tax-forms API of Provider / Tax Reporting Entity

3. Data Hosting Service (the API provider):

- Retrieves header credentials from /tax-forms API request, decodes from Base64
- Validates that “Document ID” and “Document Passcode” reference a single known tax document
- Returns the data from referenced tax document

API Token Generation

The tax software composes an HTTP request like the below:

```
GET /fdx/v5/tax-forms?resultType=details HTTP/1.1
Accept: application/json
Authorization: Basic MTIzNDU2Nzg5OkZaSjU1NjROQjMw
User-Agent: Jersey/2.29 (HttpURLConnection 1.8.0_252)
Host: api.taxhostingserver.com
Connection: keep-alive
```

The Authorization header is composed using the Basic authentication technology described in [Internet Engineering Task Force \(IETF\) Request for Comments 7617](#).

The steps are as follows:

1. Collect the two pieces of data. For example, **123456789** and **FZJ5564NB30**, respectively.
2. Concatenate the items with a colon between them. (Therefore the Document ID may not contain a colon. Subsequent colons within the Document Passcode are acceptable.)
1123456789:FZJ5564NB30
3. Encode the above string using Base 64 encoding:
1MTIzNDU2Nzg5OkZaSjU1NjROQjMw
4. Include the encoded value in the Authorization header record:
Authorization: Basic 1MTIzNDU2Nzg5OkZaSjU1NjROQjMw

To complete the secure request, this header will also be fully encrypted when it is subsequently submitted over HTTPS using SSL or TLS.

Communicating Document ID and Passcode via QR Code

When creating the recipient's copy of a tax form, instead of or in addition to printing the Document ID and Document Passcode upon it in human-readable form, those values can also be delivered by placing them in a QR Code. For full details, see the Tax Document QR Code Specification document included in the FDX API.

For a Basic Auth QR Code, generate JSON for the BasicQuthForQR entity containing the basic auth credentials as seen here:

```
{
  "basicAuth" : {
    "taxYear" : 2023,
    "taxFormType" : "Tax1099B",
    "id" : "00677560089990B1",
    "passcode" : "PK2Z-0QP-L6EF"
  },
  "version" : "V5.0",
  "softwareId" : "OakTreeSecurities"
}
```

This JSON is then encoded as a QR code and placed on a page of the recipient's copy of the tax document. Here is an example.

Oak Tree Securities
implements electronic tax data import technologies defined by
the Financial Data Exchange (FDX) organization.


You may scan the below QR code
with your tax software smartphone app that supports this technology.


Using the encoded information,
your tax software will securely retrieve your data from our server
and import the data into your tax return.

**QR CODE WITH TAX
DOCUMENT DATA**

The Financial Data Exchange
standards-setting organization defines
the use of QR codes for US Tax data.

For more information about how this code can be used visit:
<https://financialdataexchange.org/us-tax>

 **FINANCIAL**
DATA EXCHANGE



Implementation Requirements

The following constraints **MUST** be followed by implementers of this authentication method, in order to reduce or eliminate the risks inherent in not follow the FDX standard OAuth security model.

Techniques for Risk Mitigation

One document:

- “Document Passcode” **MUST** appear on no more than one tax document image
- “Document Passcode” **MUST NOT** give access to any other documents than that one

Static data:

- The data retrieved for the single document image **MUST NOT** change in the future
- Therefore, corrected tax forms **MUST** receive a new, different “Document Passcode”
- Provider API response for original “Document Passcode” requests **SHOULD** also notify caller that its data has been superseded by a Corrected document

One-time full access:

- Provider **MUST** return all the data on a document, i.e. everything that is protected by the “Document Passcode”, in a single API call
- Therefore calling application **SHOULD NOT** call the API more than once for a given “Document Passcode”

Temporary access:

- Provider **MUST** have an expiration date for a “Document Passcode”, past which it can no longer be used
- Expiration **CAN** be beyond current tax year, if Provider will make tax document data available for taxpayer preparation of returns for prior years

Visual data only:

- Data response **MUST NOT** contain more information than is visually provided on the tax document image itself. For example, if the image displays only the last 4 digits of an account number, the API call **MUST NOT** return the full account number.
- Response for a “Document Passcode” **MUST NOT** give access to any other user data, which is not visible on the form

Confidential ID:

- "Document ID" CAN be the customer account number or other customer-specific data used as the username for the document. It CAN be a natural data element known to the customer, but does not need to be
- "Document ID" MUST be an already confidential data element or be treated as confidential

Passcode security:

- The "Document Passcode" is a Restricted data element and MUST be treated as a password when creating it and storing it internally
- Each "Document Passcode" MUST be unique and randomized to be "unguessable"
- The "Document Passcode" MUST only be stored internally as a hash as recommended by NIST and [IETF](#) guidelines on password storage.

Count & limit access:

- Provider MUST count the requests made for each taxpayer document
- Provider MUST set a hard limit for number of requests made for a document (perhaps 3?), beyond which requests will be prevented
- Repeated requests do not return any additional data for a document (taxpayer might choose to use multiple providers of tax preparation software for the same year tax return)
- Provider MUST count the failed requests made for each taxpayer document
- Provider MUST set a hard limit for number of failed requests made for a document, beyond which requests will be prevented, i.e. limited number of failed password attempts

No colon:

- "Document ID" MUST NOT contain a colon (":") character
- This is simply a requirement of the Basic Authorization protocol

Mitigation of Security Risks

Password-Based Threat Model

“Document Passcode” is effectively a password for the document. Even though this credential does not represent the user, all risks of password-based authentication still apply. However, only to the data on this specific document. This proposal describes below how all these risks can be mitigated and will potentially define certification criteria for use of this authentication method.

The following are common and well-understood problems and risks of password-based authentication schemes, and our mitigation recommendations here.

Mitigation Techniques

| | | | |
|------------------|-----------------------|----------------------|----------------------|
| One document | Static data | One-time full access | Temporary access |
| Visual data only | Confidential username | Password security | Count & limit access |

Risk Mitigations

| Issue / Risk | Description | Mitigations |
|---|---|---|
| User Consent | The user sharing a password with a piece of software is a form of consent. However, the software has no way to prove that the user gave consent, as the software could have obtained the password via other (malicious) means. | <ul style="list-style-type: none"> • Confidential ID • One document • Count & limit access |
| Phishing | Users may be tricked into entering passwords into malicious software or giving it to attackers in other ways (e.g. SMS). The potential damage of a phishing attack depends on what data is accessible using the password. | <ul style="list-style-type: none"> • One document • Visual data only |
| Brute Force / Guessability | If the possible character set and length of the password is known, attackers may attempt to try all combinations until they find a password that works. | <ul style="list-style-type: none"> • Confidential ID • Count & limit access |
| Password Reuse | If users can choose passwords, it is possible (and likely) that the user will reuse the password at other services. | <ul style="list-style-type: none"> • One document |
| Inability to Revoke Access | Once a password is shared with an application, there is no way to revoke that application's access to the data accessible with the password. | <ul style="list-style-type: none"> • Static data • Temporary access • Count & limit access |
| Inability to Limit Scope of Access | When sharing a password with an application, the application has the ability to access all the same data with that password that the user can access. There is no ability to limit the types of data the application can access, and there is no ability to enforce a time limit on this access either. | <ul style="list-style-type: none"> • One document • Static data • Temporary access • Count & limit access |
| Secure Storage of Passwords | Password database leaks at providers are a concern. In this case, attackers may get access to all passwords or password hashes within a system at once. | <ul style="list-style-type: none"> • Confidential ID • Passcode security |

Appendix 1: Sample Code for Java Developers

Create token and related request header

```
import java.util.Base64;

import java.nio.charset.StandardCharsets;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Invocation;
import javax.ws.rs.client.WebTarget;

import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

/**
 * API Client
 * to do GET request of
 * FDX Tax Document server
 */
public class BasicAuthFdxAPIClient {

    /**
     * Do request.
     *
     * @param targetUrl The complete URL
     * @param item1 Item 1 of 2
     * @param item2 Item 2 of 2
     * @return Response object
     */
    public static Response performRequest(
        String targetUrl,
        String item1,
        String item2
```

```

    ) {

        // Concatenate the data items with colon in between
        String concat = item1 + ":" + item2;

        // Do Base 64 encoding
        String asBase64 = Base64.getEncoder( )
            .encodeToString(concat.getBytes(
StandardCharsets.UTF_8 )
        );

        // Incorporate into HTTP request
        Client client = ClientBuilder.newClient( );

        WebTarget webTarget = client.target( targetUrl );

        Invocation.Builder builder
            = webTarget.request( MediaType.APPLICATION_JSON )
                .header( HttpHeaders.AUTHORIZATION, "Basic " +
asBase64 )
                .accept( MediaType.APPLICATION_JSON );

        Response response = builder.get( );

        return response;

    }
}

```


Read request header and decode token

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.util.Base64;
import java.util.Enumeration;
import java.util.logging.Logger;

@WebServlet( "/header-reader" )
public class HttpHeadersReader extends HttpServlet {

    public static final Logger LOGGER =
        Logger.getLogger( HttpHeadersReader.class.getName( ) );

    @Override
    protected void doGet(
        HttpServletRequest request,
        HttpServletResponse response
    ) {

        Enumeration<String> headerNames =
request.getHeaderNames( );

        while ( headerNames.hasMoreElements( ) ) {

            String headerName = headerNames.nextElement( );

            Enumeration<String> headers = request.getHeaders(
headerName );

            while ( headers.hasMoreElements( ) ) {

                String headerValue = headers.nextElement( );

                switch ( headerName ) {

                    case "Authorization":
```

```

String noLabel = headerValue.replace(
"Basic ", "" );

String decoded =
    new
String(Base64.getDecoder().decode(noLabel));

String items[] = decoded.split( ":" );
String item1 = items[0];
String item2 = items[1];

String message = String.format(
    "Item 1: %s, Item 2: %s",
    item1,
    item2
);

LOGGER.info( message );

break;

default:

String outLine = String.format(
    "Header name: %s, Header value: %s",
    headerName,
    headerValue
);

LOGGER.info( outLine );

    }
    }
    }
}

```