ReportStream

Programmer's Guide for Organizations and Testing Facilities

VERSION 4.7 – October 2024

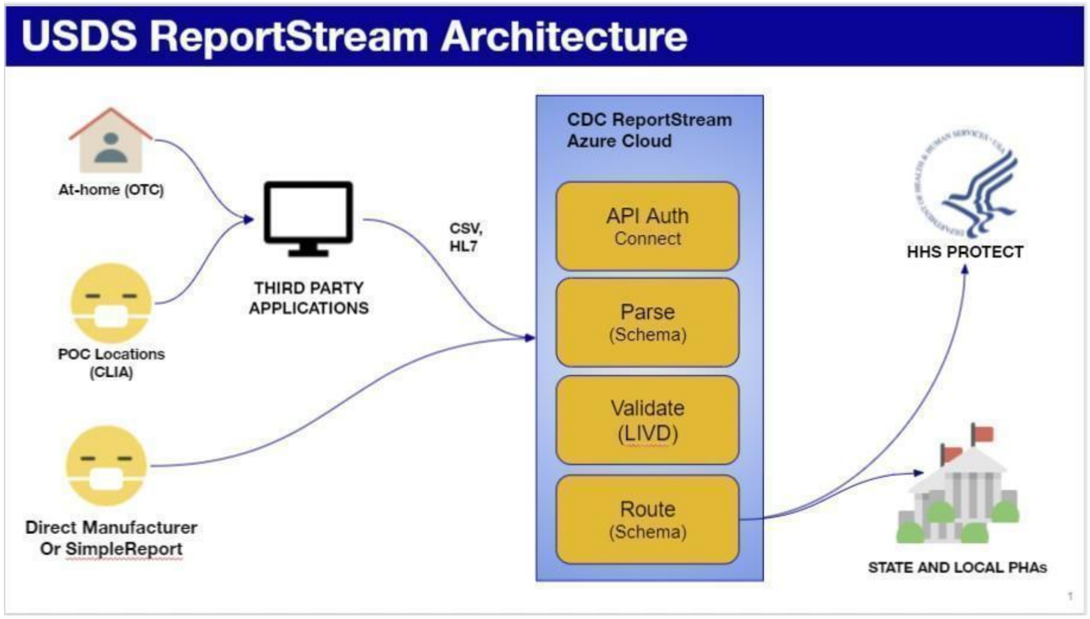
Table of contents

Introduction

About ReportStream

ReportStream is a free, open-source data platform that makes it easy for public health data to be transferred from testing facilities to public health departments.

ReportStream will automatically filter, transform, batch, and forward data to local, state, and federal jurisdictions based on both geographical and data quality filters provided by those jurisdictions.



ReportStream is not a permanent repository, EMR, or registry for health data. We only keep the data long enough to ensure it gets to the proper local, state, and federal jurisdictions.

About this guide

This programmer's guide enables those who are writing automated systems and tools to send laboratory and other health-related data to local, state, and federal jurisdictions. It helps you, the technical user at the testing facility or sending location, learn how to send data using the ReportStream Restful (REST) API.

Examples in this guide use curl commands for simplicity with the assumption you'll be coding these calls into your sending system. You can also use a program like Postman to test submissions.

About our API

The Waters API — the primary secure entry point to ReportStream — is named in memory of Dr. Michael Stephan Waters (1973-2020). His tireless work at the U.S. Food and Drug Administration championed diagnostic data interoperability efforts nationwide. ReportStream honors Dr. Waters through continuation and elevation of his work.

Release notes

Onboarding

Overview

1. **Kickoff**: As you get started, a ReportStream engineer will set up a kickoff call to review the process outlined in this guide and answer any questions.

2. **Validate and test your data**: There are three rounds of testing: validating formatting with fake data, testing the API connection with fake data, and testing PII data in production.

3. **Start sending your data**: Now that we know your files and connection will work, you can start sending your data through the API.

1. Kickoff

Before setting up your data, you will:

1. Attend a kickoff call with a ReportStream engineer
2. Get an Okta account
3. Agree to our Terms of Service

If you haven't connected with us yet, reach out so we can help you begin onboarding.

2. Validate and test your data

Test 1: Testing your formatting with fake data

To prepare your file for testing, set up a sample file with fake data (artificially created, non-PII data).

Currently, ReportStream can accept either HL7 or FHIR data. We can provide a sample HL7 file with fake data to review upon request.

For this step, you can refer to formatting requirements for HL7v2.5.1 ORU-R01 or RADxMARS for OTC (over the counter) test data.

When you've formatted your fake (non-PII) data file, test your data model using the NIST Electronic Lab Reporting (ELR) Validation Suite. (Note: If you're testing OTC data, use the NIST HL7 General Validation Tool and select "RADx MARS HL7v2" from the tool scope dropdown. Navigate to the "Context-free" tab and select the "Onboarding" profile group.) Correct any errors you receive.

Test 2: Set up authentication and test your API connection

After you have finalized the formatting of your data, the ReportStream team will begin onboarding you to our staging environment.

As part of the onboarding process, the ReportStream team will assign your unique client-id and set up your ReportStream account with the type of data you will be submitting. ReportStream will use the client-id to look up the associated data model and format (FHIR or HL7) and validate the attached payload.

Your first step in this phase is to set up your authentication.

Set up authentication with a public/private key pair

ReportStream uses token-based authentication with a public/private key pair.

The example below uses the fake client-id healthy-labs, that you will change for your submissions. The example submits the payload contained in the file `./healthy-labs-nonPII-data.hl7`. In the example, data are submitted via an HTTP POST to the ReportStream staging system `reports` endpoint. The data submitted are sent as the payload of the POST, as is, with no changes.

Step 1: Prior to submission, send your public key to ReportStream.

1. Prior to connecting to the endpoint, you'll need a public/private keypair. There are many ways to do this. The steps below show how to create a key pair using `openssl`.

   EC

   ```
   openssl ecparam -genkey -name secp384r1 -noout -out my-es-keypair.pem
   openssl ec -in my-es-keypair.pem -pubout -out  my-es-public-key.pem
   ```

   RSA

   ```
   openssl genrsa -out my-rsa-keypair.pem 2048
   openssl rsa -in my-rsa-keypair.pem -outform PEM -pubout -out my-rsa-public-key.pem
   ```

2. Send the public key to the ReportStream team using our public key tool. Note: you'll need to login to use that feature. If you do not have a login contact ReportStream support at reportstream@cdc.gov. ReportStream will associate the key with your configuration within ReportStream.

You only need to do this step once, not every time you submit reports. If you need to change your keys at any time, contact ReportStream support.

Step 2: At the time of submission, generate a signed JWT using your private key.

A JWT is a base64 encoded string that has three parts: `header`, `payload`, and `signature`.

You can find an example python program to generate a valid JWT on GitHub.

If you receive errors, reference this list of error types for explanations and instructions.

Here is an example, using the fake `client-id healthy labs`, of header and payload data that should appear in a ReportStream JWT, prior to signature:

```
{
  "header": {
    "kid": "healthy-labs.default",
    "typ": "JWT",
    "alg": "RS256"
```

```
  },
  "payload": {
    "iss": "healthy-labs.default",
    "sub": "healthy-labs.default",
    "aud": "staging.prime.cdc.gov",
    "exp": 1660737164,
    "jti": "4b713fcd-2514-4207-b310-620b95b749c5"
  }
}
```

Note:

- The exp (expiration time) should be a Unix time, five minutes after the time the token was generated.
- The jti (JWT ID) should be a random unique string, new with every call.
- Generate the signed JWT using your private key.

Step 3: Send the signed JWT to ReportStream to get a temporary bearer token

POST to the token URL, as in the example below, noting the following:

1. Use Content-Type: application/x-www-form-urlencoded.
2. In the `scope` parameter, replace the dummy string '`healthy-labs`' with your client-id, as assigned to you by ReportStream staff.
3. The `grant_type` and `client_assertion_type` parameters are always fixed values. The `grant_type` should be `client_credentials` and `client_assertion_type` should be `urn:ietf:params:oauth:client-assertion-type:jwt-bearer`, as in the example curl below.
4. In the client_assertion parameter, replace the <token-signing-secret> below with your JWT from above.
5. All the parameters are sent in the body/payload of the post (when using curl, via the `-d` option), not in the URL.

Here is an example 'curl' POST:

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded" -d "scope=healthy-
labs.default.report&grant_type=client_credentials&client_assertion_type=urn:ietf:params:oauth:cl
ient-assertion-type:jwt-bearer&client_assertion=<token-signing-secret>"
"https://staging.prime.cdc.gov/api/token"
```

You should get something like this back, which will be valid for five minutes:

```
{"access_token":"<long-access-
token>","token_type":"bearer","expires_in":300,"expires_at_seconds":1625260982,"scope":"healthy-
labs.default.report"}
```

Step 4: Submit data to ReportStream using the bearer token.

Use the access token returned above as the bearer token for the submission:

HL7 example

```
curl -H "authorization:bearer <long-bearer-token>" -H "client:healthy-labs"  -H "content-
type:application/hl7-v2" --data-binary "@./healthy-labs-nonPII-data.hl7"
"https://staging.prime.cdc.gov/api/waters"
```

Again, always remember to replace the healthy-labs client-id with the client-id supplied to you by ReportStream staff.

Test your automation

Once authentication is complete, you can test your automation code as well as your code that handles responses using the staging API. Data is sent in the HTTP payload, either in FHIR or HL7 2.5.1 format. You can use curl commands, Postman or another method of your choosing to post test submissions to the staging environment.

> **Note:** Do not send any PII or PHI to the staging system — only fake (dummy, example, synthetic) data is acceptable.

Let us know when you send submissions to the staging environment. We'll review that data and work with you to correct any issues. You may send as many fake data submissions to staging as is helpful.

For troubleshooting on your own, here is the complete endpoint input and response

OpenAPI specification.

Test 3: Testing real data in production

The ReportStream team will onboard you to the production system in training mode. ReportStream won't forward or transport data received in training mode. However, the response message provides detailed information on where your data would have flowed if production mode was active.

3. Start sending your data

When you are ready, the ReportStream team will move you out of training mode and enable full production mode. Once in production, you can send a single record or up to 10,000 records in a single submission.

Data will automatically flow to appropriate state, local, and federal jurisdictional systems.

Note: Some jurisdictions require additional validation before sending data to their systems. If this affects your data submission, the ReportStream team will assist you in the process. Currently, the following states require additional validation:

- California
- Illinois
- Washington

Responses from ReportStream

ReportStream responds to each API call with a response (JSON formatted) about the disposition of your data.

Response messages

Asynchronous processing

ReportStream uses asynchronous (async) processing. Upon submitting data via ReportStream async processing, the REST endpoint returns almost immediately. However, ReportStream doesn't return information about where the tests will be sent.

In exchange for speed, the async submission response provides less initial information in the JSON. The initial response will provide errors and warnings, but no destination or filter information. The History Details API can be queried later to get full information about expected and actual destinations.

Example ReportStream response to an async submission:

```
{
  "submissionId":1604,
  "timestamp":"2022-02-10T13:50:19.162694Z",
  "sender":"simple_report.default",
  "httpStatus":201,
  "id":"3597ad7d-b92c-4bc0-a8fc-d909ed87bc90",
  "reportItemCount":2,
  "destinationCount":0,
  "destinations": [],
  "errors": [],
  "warnings": [],
  "topic":"covid-19",
  "warningCount":0,
  "errorCount":0
}
```

ReportStream features a History Details API that can be later queried to obtain the actual destinations and relevant detail using your existing private/public key pair.

The request is made with the submissionId in the earlier example.:

```
https://prime.cdc.gov/api/history/simple_report/submissions/1588
```

JSON Error responses

In error cases, no report "id" UUID is returned, because no report was created based on the submission.

Example failure response and identical HistoryAPI response (Note the "id" is null, and the “httpStatus” is not 201):

```
{
    "submissionId": 1594,
    "timestamp": "2022-02-09T20:44:55.055545Z",
    "sender": "simple_report",
    "destinationCount" : 0,
    "httpStatus": 400,
    "id": null,
    "destinations": [],
    "errors": [
        {
            "scope": "item",
            "index": 1,
            "trackingId": "abcde",
            "type": "error",
            "message": "Blank value for element 'Patient_last_name' ('patient_last_name')"
        }
    ],
    "warnings": [],
    "topic": null,
    "warningCount": 0,
    "errorCount": 1
}
```

Example of a report level error:

```
{
    "submissionId": 1599,
    "timestamp": "2022-02-09T20:56:16.82117Z",
    "sender": "strac",
    "httpStatus": 400,
    "id": null,
    "destinationCount" : 0,
    "destinations": [],
    "errors": [
        {
            "scope": "report",
            "index": null,
            "trackingId": null,
            "type": "error",
            "message": "CSV file has an inconsistent number of columns on row: 3"
        }
    ],
    "warnings": [],
    "topic": null,
    "warningCount": 0,
    "errorCount": 1
}
```