Reference Client

Overview

NCHS is working to modernize the national collection and exchange of mortality data by developing and deploying new Application Programming Interfaces (APIs) for data exchange, implementing modern standards health like HL7's Fast Healthcare Interoperability Resources (FHIR), and improving overall systems and processes. This repository provides a reference implementation and documentation describing the Reference Client, which supports the client side exchange of mortality data between vital records jurisdictions and NCHS. For the server side implementation, see NVSS API server.

Reference Client

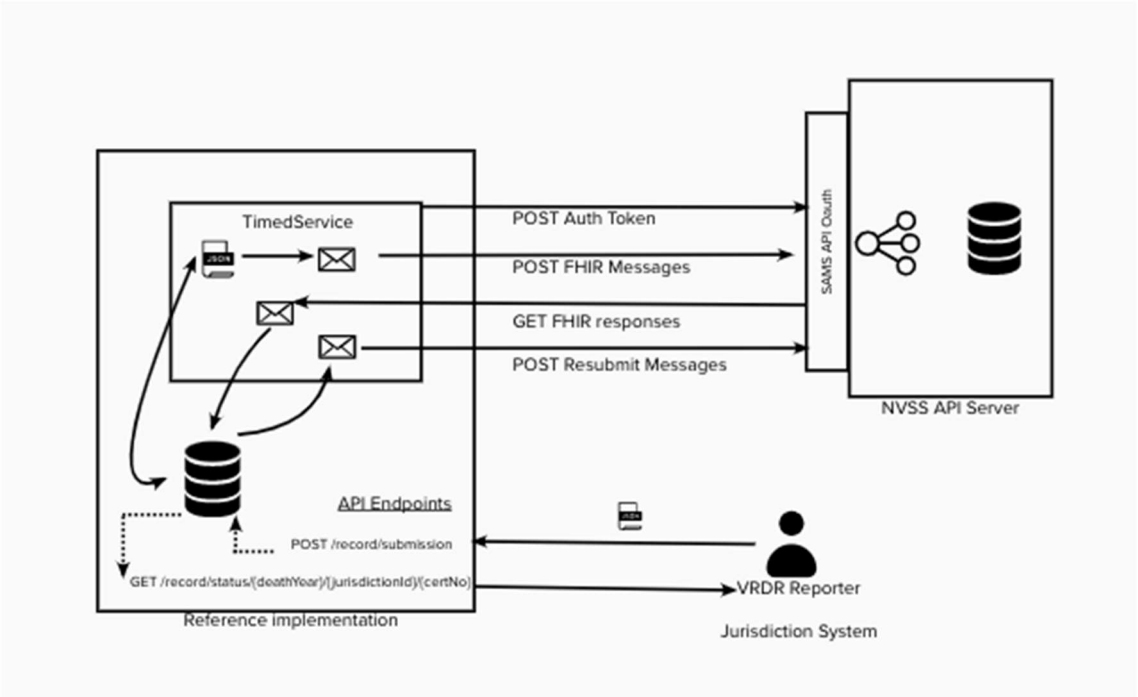
The Reference Client is an example implementation for jurisdications that handles submitting VRDR FHIR Messages, reliable delivery (acknowledgements and retries), and retrieving message responses from the NVSS API server. The implementation follows the FHIR Messaging IG. The reference implementation is developed for .NET using C# and leverages the vrdr-dotnet library to create and parse FHIR Messages. This implementation uses the VRDR.Client library to make calls to the NVSS API server. If you are looking to build your own client, see the VRDR.Client library for basic library functions to authenticate, post records, and retrieve responses from the NVSS API server.

The client is responsible for interacting with the NVSS API server so jurisdictions do not need to duplicate work when building their own client. The client abstracts away the reliable message delivery and retrieval between the client and NVSS API server. It provides its own API interface to jurisdictions to simplify their workflow.

Use Cases

This client can be used to test and demo messaging between the client and the NVSS API Server using the FHIR Messaging IG format. It can also be used as a reference for jurisdictions building their own client implementation.

Architecture Diagram



Architecture Description

The client implementation runs a service that submits messages to the NVSS API and retrieves responses to make available to jurisdictions. The service handles reliable delivery, resends, and parsing so jurisdictions do not need to implement a custom service. The service sits between the jurisdiction's system and the NVSS API Server.
Example workflow

- The VRDR reporter sends a JSON vrdr record via POST to the client's /record/submission, /record/update, or /record/void endpoint
- Upon receipt, the client converts the json to a VRDR record, wraps it in a FHIR Message and inserts it in the MessageItem table
- The client's TimedService pulls new messages from the MessageItem table every X-configured seconds and POSTs the message to NVSS API Server
- Next, the TimedService makes a GET request for any new messages from the NVSS API Server
- The TimedService parses the response messages and stores them in the ResponseItems table
    - If there was an acknowledgement or error, it updates the corresponding message in the MessageItem table with the new message status
- Finally, the TimedService checks for any messages that have not received an acknowledgement in Y-configured seconds and resubmits them
    - The TimedService runs these three steps in sequence every X-configured seconds
    - The frequencies of X and Y are configurable

- The VRDR reporter sends a GET request to the /record/{deathYear}/{jurisdictionId}/{certNo} endpoint at any time to get the status of the message with the provided business identifiers for death year, jurisdiction id, and certificate number

Note that if a submission message was sent, followed by an update message with the same business identifiers, the returned status will be for the latest message inserted into the database, in this case the update message status.

API Endpoints

The client implementation has endpoints to submit VRDR records, update records, and void records. It also has an endpoint to retrieve the status and response of a given record.

Sending VRDR Records

Submission Records

1. POST /record/submission
    i. Parameters: The POST /record/submission endpoint accepts a VRDR record as json
    ii. Function: Wraps the record in a FHIR Submission message and queues the message to be sent to the NVSS API Server
    iii. Response: A successful request returns 204 No Content
2. POST /record/submissions
    i. Parameters: The POST /record/submissions endpoint accepts a list of VRDR records as json
    ii. Function: Wraps each record in a FHIR Submission message and queues the message to be sent to the NVSS API Server
    iii. Response: A successful request returns 204 No Content

Update Records

1. POST /record/update
    i. Parameters: The POST /record/update endpoint accepts a VRDR record as json
    ii. Function: Wraps the record in a FHIR Update message and queues the message to be sent to the NVSS API Server
    iii. Response: A successful request returns 204 No Content
2. POST /record/updates
    i. Parameters: The POST /record/updates endpoint accepts a list of VRDR records as json
    ii. Function: Wraps each record in a FHIR Update message and queues the message to be sent to the NVSS API Server

iii. Response: A successful request returns 204 No Content

Void Records

1. POST /record/void
    i. Parameters: The POST /record/void endpoint accepts a VRDR record as json and a block_count parameter
    ii. Function: Wraps the record in a FHIR Void message, sets the block count, and queues the message to be sent to the NVSS API Server
    iii. Response: A successful request returns 204 No Content
2. POST /record/voids
    i. Parameters: The POST /record/voids endpoint accepts a list of VRDR records and associated block_count as json
    ii. Function: Wraps each record in a FHIR Void message, sets the block count, and queues the message to be sent to the NVSS API Server
    iii. Response: A successful request returns 204 No Content

Alias Records

1. POST /record/alias
    i. Parameters: The POST /record/alias endpoint accepts a VRDR record as json as well as the following parameters
        a. alias_decedent_first_name
        b. alias_decedent_last_name
        c. alias_decedent_name_suffix
        d. alias_father_surname
        e. alias_social_security_number
    ii. Function: Wraps the record in a FHIR Alias message, sets the alias values, and queues the message to be sent to the NVSS API Server
    iii. Response: A successful request returns 204 No Content
2. POST /record/aliases
    i. Parameters: The POST /record/aliases endpoint accepts a list of VRDR records and associated parameters below as json
        a. alias_decedent_first_name
        b. alias_decedent_last_name
        c. alias_decedent_name_suffix
        d. alias_father_surname
        e. alias_social_security_number

ii. Function: Wraps each record in a FHIR Alias message, sets the alias parameters, and queues the message to be sent to the NVSS API Server

iii. Response: A successful request returns 204 No Content

Checking Responses

1. GET /record/{deathYear}/{jurisdictionId}/{certNo}
    i. Parameters:
        a. deathYear: the year of death in the VRDR record
        b. jurisditionId: the jurisdiction Id in the VRDR record
        c. certNo: the 6 digit certificate number in the VRDR record
    ii. Function: Retrieves the most recent MessageItem with business identifiers that match the provided parameters
    iii. Response: A successful request returns 200 OK and a JSON object with the MessageItem and it's Extraction Error or Coded Response if available

Getting Started

The client implementation can run with a local development setup where all services are run locally, or an integrated development setup that connects to the development NVSS API Server.

Requirements

This project uses dotnet and docker to run the local database.

NVSS Development Setup

1. Set up the database docker containers
    i. Run docker-compose up --build to initialize the client db (postgres)
    ii. From the reference-client-api/nvssclient directory, run dotnet ef database update to intialize the client's db
2. Configure the client implementation to connect to the development NVSS API Server
    i. Create an appsettings.json file from the appsettings.json.sample file
    ii. In appsettings.json set "LocalTesting" : false
    iii. In appsettings.json set "AuthServer": "https://apigw.cdc.gov/auth/oauth/v2/token"
    iv. In appsettings.json set "AuthServer": "https://apigw.cdc.gov/OSELS/NCHS/NVSSFHIRAPI/<jurisidction-id>/Bundles"
    v. In appsettings.json fill out the "Authentication" section to authenticate to the server via oauth, contact admin for your credentials

vi. In appsettings.json set "ResendInterval" to your desired interval. The recommended length in production is 4 hours, or 14400 seconds. If you are testing the resend implementation then you may want to set it to a shorter interval like 10 seconds to see results quickly.

vii. In appsettings.json set "PollingInterval" to the frequency you want to poll for new responses. In production, 1 hour may be a good interval to use, but for testing an interval like 30 seconds may be better for seeing results quickly.

3. Set up OAuth

    i. OAuth is required to authenticate to the NVSS API Server. Contact the NVSS server team to acquire your client id, client secret, username, and password.

    ii. In appsettings.json set "ClientId": "<your-client-id>"

    iii. In appsettings.json set "ClientSecret": "<your-client-secret>"

    iv. In appsettings.json set "Username": "<your-username>"

    v. In appsettings.json set "Password": "<your-password>"

4. Once the client db is running and the configuration is complete, go to the reference-client-api/nvssclient project directory and run

5. dotnet run

Local Development Setup

1. Set up the database docker containers

    i. Run docker-compose up --build to initialize the client db (postgres) and the NVSS API Server db (mssql)

    ii. From the reference-client-api/nvssclient directory, run dotnet ef database update to intialize the client's db

2. Download the reference-nchs-api code
   from https://gitlab.mitre.org/nightingale/reference-nchs-api

    i. make sure the db password in appsettings.json matches the one set in the docker compose file

    ii. from the reference-nchs-api/messaging project directory run dotnet ef database update to initialize the NVSS API Server's db

    iii. from the reference-nchs-api directory run the NVSS API Server with dotnet run --project messaging

3. Configure the client implementation to connect to the local NVSS API Server

    i. Create an appsettings.json file from the appsettings.json.sample file

    ii. In appsettings.json set "ClientDatabase" to your database connection string

iii. In appsettings.json set "LocalTesting" : true

iv. In appsettings.json set "LocalServer":"https://localhost:5001/bundles"

v. In appsettings.json set "ResendInterval" to your desired interval. The recommended length in production is 4 hours, or 14400 seconds. If you are testing the resend implementation then you may want to set it to a shorter interval like 10 seconds to see results quickly.

vi. In appsettings.json set "PollingInterval" to the frequency you want to poll for new responses. In production, 1 hour may be a good interval to use, but for testing an interval like 30 seconds may be better for seeing results quickly.

4. Now that the client db, NVSS API Server db, and the NVSS API Server are all up and running, go to the reference-client-api/nvssclient project directory and run

5. dotnet run

Interacting with the Client API

Sending VRDR Records

Submission Records

1. Create a FHIR Record. The standard that specifies this format can be found here. There are also two public library implementations available to assist in the creation of FHIR Records, VRDR-dotnet and VRDR_javalib.

2. Submit the record using a POST request to the /record/submission endpoint. The following example demonstrates how to make the request using curl:

```
curl --location --request POST 'http://localhost:4300/record/submission' \
--header 'Content-Type: application/json' \
--data "@path/to/file.json"
```

The API will return a 204 No Content HTTP response if the request was successful.

Retrieving Responses

1. After submitting a record, use the GET /record/{deathYear}/{jurisdictionId}/{certNo} endpoint to check the status of the message and the coded response if available. The json response will include the status of the most recent message with the provided business identifiers deathYear, jurisdictionId, and certNo. The following example demonstrates how to make the request using curl:

```
curl http://localhost:4300/record/status/2018/MA/001
```

The API will return a json response if the request was successful. Example Response:

```
{
  "message": {
```

```
      "id": 4,

      "uid": "bd734a03-8a5c-4ee7-a994-779f58f4433f",

      "stateAuxiliaryIdentifier": "42",

      "certificateNumber": 1,

      "deathJurisdictionID": "MA",

      "deathYear": 2018,

      "message": "{\"resourceType\":\"Bundle\",\"id\":\"b443f717-96cc-4f72-8e62-
06119d54111d\",\"type\":\"message\",\"timestamp\":\"2021-12-13T10:51:32.770941-
05:00\",\"entry\":[{\"fullUrl\":\"urn:uuid:bd734a03-8a5c-4ee7-a994-
779f58f4433f\",\"resource\":{\"resourceType\":\"MessageHeader\",\"id\":\"bd734a03-8a5c-
4ee7-a994-
779f58f4433f\",\"eventUri\":\"http://nchs.cdc.gov/vrdr_submission\",\"destination\":[{\"en
dpoint\":\"http://nchs.cdc.gov/vrdr_submission\"}],\"source\":{\"endpoint\":\"https://exam
ple.com/jurisdiction/message/endpoint\"},\"focus\":[{\"reference\":\"urn:uuid:27c1f73d-
59a7-4c52-bb20-7073c1778535\"}]}},{\"fullUrl\":\"urn:uuid:d1911214-a3fc-452f-bb9d-
91d4b38d96c6\",\"resource\":{\"resourceType\":\"Parameters\",\"id\":\"d1911214-a3fc-
452f-bb9d-
91d4b38d96c6\",\"parameter\":[{\"name\":\"cert_no\",\"valueUnsignedInt\":1},{\"name\":\"
state_auxiliary_id\",\"valueString\":\"42\"},{\"name\":\"death_year\",\"valueUnsignedInt\":2
018},{\"name\":\"jurisdiction_id\",\"valueString\":\"MA\"}]}},

      ...


,{\"value\":\"42\"}],\"status\":\"current\",\"type\":{\"coding\":[{\"system\":\"http://loinc.org\"
,\"code\":\"64297-5\",\"display\":\"Death certificate\"}]},\"date\":\"2020-08-
07T16:41:57.698163-04:00\",\"author\":[{\"reference\":\"urn:uuid:7afbe14f-932f-4639-
966a-84c4e9d2f7c6\"}],\"content\":[{\"attachment\":{\"url\":\"urn:uuid:cf96ffd9-fb38-
479f-ba3c-5f3f36752fdb\"}}],\"context\":{\"related\":[{\"reference\":\"urn:uuid:cf96ffd9-
fb38-479f-ba3c-5f3f36752fdb\"}]}}}]}}]}",

      "retries": 0,

      "status": "Acknowledged",

      "expirationDate": "2021-12-13T10:52:09.439643",

      "createdDate": "2021-12-13T10:51:32.781559",

      "updatedDate": "2021-12-13T10:51:49.376313"

   },

   "response": null

}
```


Client Developer Documentation

This section provides useful information to developers working on the client implementation or referring to it as a reference for their own custom implementation.

Migrations

1. When applying a new migration, update the Models to reflect the desired changes.
2. Stop and remove the container running the current client db
3. Run docker-compose up --build to recreate the db
    i. You may need to comment out this block in Program.cs
4. // .ConfigureServices((hostContext, services) =>
5. // {
6. // services.AddHostedService<TimedHostedService>();
7. // });
8. ```
9. Run dotnet ef migrations add <Your-Migration-Name> to create the new migration
10. Run dotnet ef database update to update the db schema

Testing

1. Tests are located in nvssclient.tests
2. cd into the nvssclient.tests folder
3. Run dotnet test

Developer Notes and Justifications

- To persist data and make it available to the Jurisidction upon request, the full response message is stored in the ResponseItems table, rather than just the ID in a Message Log. The ResponseItem table serves as the Message Log, see FHIR Messaging IG
- The POST /record end point does not return data because the submission and coding process takes too long to provide a synchronous response. The user can request the message status via the GET /record endpoint.

Down the road tasks and questions

- Existing issue when applying new migrations requires temporarily commenting out the TimedService
- Need to consider the state jurisdictions will keep track of, ex if using a native format, there may be two versions of the same record
    - Consider a potential adapter for handling different formats
- Request feedback on the status endpoint, should messages be cleared out over time