Article
Lucas Enard · May 13, 2022 8m read

Open Exchange

A simple example of a Fhir client in c#

1. Fhir-client-net

This is a simple fhir client in c# to practice with fhir resources and CRUD requests to a fhir server.

Note that for the most part auto-completion is activated.

GitHub

2. Prerequisites

Make sure you have git and Docker desktop installed.

If you are working inside the container, those modules are already installed.

if you are not, use nugget to install Hl7.Fhir.R4, we will be using Model and Rest from it :

Hl7.Fhir.Model

Hl7.Fhir.Rest

3. Installation

3.1. Installation for development

Clone/git pull the repo into any local directory e.g. like it is shown below:

```
git clone https://github.com/LucasEnard/fhir-client-net.git
```

Open the terminal in this directory and run:

```
docker build .
```

3.2. Management Portal and VSCode

This repository is ready for VS Code.

Open the locally-cloned fhir-client-net folder in VS Code.

If prompted (bottom right corner), install the recommended extensions.

3.3. Having the folder open inside the container

You can be *inside* the container before coding if you wish.

For this, docker must be on before opening VSCode.

Then, inside VSCode, when prompted (in the right bottom corner), reopen the folder inside the container so you will be able to use the python components within it.

The first time you do this it may take several minutes while the container is readied.

If you don't have this option, you can click in the bottom left corner and press reopen in container then select From Dockerfile

More information here

By opening the folder remote you enable VS Code and any terminals you open within it to use the c# components within the container.

If prompted (bottom right corner), install the recommended extensions.

4. FHIR server

To complete this walktrough you will need a FHIR server.

You can either use your own or go to InterSystems free FHIR trial and follow the next few steps to set it up.

Using our free trial, just create an account and start a deployement, then in the Overview tab you will get acces to an endpoint like https://fhir.000000000.static-test-account.isccloud.io that we will use later.

Then, by going to the Credentials tab, create an api key and save it somewhere.

Now you are all done, you have you own fhir server holding up to 20GB of data with a 8GB memory.

5. Walkthrough

Complete walkthrough of the client situated at /Client.cs.

The code is separated in multiple parts, and we will cover each of them below.

5.1. Part 1

In this part we connect our client to our server using Fhir.Rest.

```
// Part 1

// Creation of an htpclient holding the api key of the server as an header
var httpClient = new HttpClient();
httpClient.DefaultRequestHeaders.Add("x-api-key", "api-key");


var settings = new FhirClientSettings
    {
        Timeout = 0,
        PreferredFormat = ResourceFormat.Json,
        VerifyFhirVersion = true,
        // PreferredReturn can take Prefer.ReturnRepresentation or Prefer.ReturnMinim
al to return the full resource or an empty payload
        PreferredReturn = Prefer.ReturnRepresentation
    };
// Creation of our client using the right url
var client = new FhirClient("url",httpClient,settings);
```

In order to connect to your server you need to change the line :

```
httpClient.DefaultRequestHeaders.Add("x-api-key", "api-key");
```

And this line :

```
var client = new FhirClient("url",httpClient,settings);
```

The 'url' is an endpoint while the "api-key" is the api key to access your server.

Note that if **you are not using** an InterSystems server you may want to check how to authorize your acces if needed.

Just like that, we have a FHIR client capable of direct exchange with our server.

5.2. Part 2

In this part we create a Patient using Fhir.Model and we fill it with a HumanName, following the FHIR convention, use and family are string and given is a list of string. The same way, a Patient can have multiple HumanNames so we have to put our HumanName in a list before puting it into our newly created Patient.

```
// Part 2

// Building a new patient and setting the names
var patient0 = new Patient();
patient0.Name.Add(new HumanName().WithGiven("GivenName").AndFamily("FamilyName"));

// Creation of our client in the server
// It is to be noted that using SearchParams you can check if an equivalent resource
already exists in the server
// For more information https://docs.fire.ly/projects/Firely-NET-SDK/client/crud.html
var created_pat = client.Create<Patient>(patient0);

Console.Write("Part 2 : Newly created patient id : ");
Console.WriteLine(created_pat.Id);
```

After that, we need to save our new Patient in our server using our client.

Note that if you start Client.cs multiple times, multiple Patients having the name we choosed will be created.
 This is because, following the FHIR convention you can have multiple Patient with the same name, only the id is unique on the server.

Check the doc for more information.

It is to be noted that using SearchParams you can check if an equivalent resource already exists in the server before creating it.

For more information https://docs.fire.ly/projects/Firely-NET-SDK/client/crud.html

Therefore we advise to comment the line after the first launch.

5.3. Part 3

In this part we get a client searching for a Patient named after the one we created earlier.

```
// Part 3

// This gets all the Patient having the exact name "FamilyName" and we take the first
 one
// Note that if you have multiple patients with the same name, you will get only the
first one
// We advise to use the id, the names, and any other informations for the SearchParam
s to be sure to get the right patient
var q = new SearchParams().Where("name:exact=FamilyName");
Bundle bund = client.Search<Patient>(q);
patient0 = bund.Entry[0].Resource as Patient;

Console.Write("Part 3 : Name of the patient we found by searching : ");
Console.WriteLine(patient0.Name[0]);
```

```
// Creation of our patient telecom, here a phone number
patient0.Telecom.Add(new ContactPoint(new ContactPoint.ContactPointSystem(),new Conta
ctPoint.ContactPointUse(),"1234567890"));

// Change the given name of our patient
patient0.Name[0].Given = new List<string>() { "AnotherGivenName" };

Console.Write("Part 3 : Name of the changed patient : ");
Console.WriteLine(patient0.Name[0]);

Console.Write("Part 3 : Phone of the changed patient : ");
Console.WriteLine(patient0.Telecom[0].Value);

// Update the patient
var update_pat = client.Update<Patient>(patient0);
```

Once we found him, we add a phone number to his profile and we change his given name to another.

Now we can use the update function of our client to update our patient in the server.

5.4. Part 4

In this part we want to create an observation for our Patient from earlier, to do this we need his id, which is his unique identifier.

From here we fill our observation and add as the subject, the id of our Patient.

```
// Part 4

// Building of our new observation
Observation obsv = new Observation {

    Value = new Quantity(70, "kg"),
    Code = new CodeableConcept {
        Coding = new List<Coding> {
            new Coding {
                System = "http://loinc.org",
                Code = "29463-7",
                Display = "Body weight"
            }
        }},
    Category = new List<CodeableConcept> {
        new CodeableConcept {
            Coding = new List<Coding> {
                new Coding {
                    System = "http://snomed.info/sct",
                    Code = "276327007",
                    Display = "Body weight"
                }
            }
        }},
    Status = new ObservationStatus {},
    Subject = new ResourceReference {
        Reference = "Patient/" + update_pat.Id}

    };

// Creation of our observation in the server
var new_obsv = client.Create<Observation>(obsv);

Console.Write("Part 4 : Id of the observation : ");
Console.WriteLine(new_obsv.Id);
```

Then, we register using the create function our observation.

5.5. Conclusion of the walkthrough

If you have followed this walkthrough you now know exactly what Client.cs does, you can start it and check in your server your newly created Patient and Observation.

To start it, open a VSCode terminal and enter :

```
dotnet run
```

You should see some information on the Patient we created and his observation.

If you are using an Intersystems server, go to API Deployement, authorize yourself with the api key and from here you can GET by id the patient and the observation we just created.

7. How to start coding

This repository is ready to code in VSCode with InterSystems plugins.
Open Client.cs to start coding or using the autocompletion.

8. What's inside the repo

8.1. Dockerfile

A dockerfile to create a dot net env for you to work on.

Use docker build . to build and reopen your file in the container to work inside of it.

8.2. .vscode/settings.json

Settings file.

8.3. .vscode/launch.json

Config file if you want to debug

#.NET #FHIR #REST API #InterSystems IRIS #Open Exchange #VSCode

Check the related application on InterSystems Open Exchange