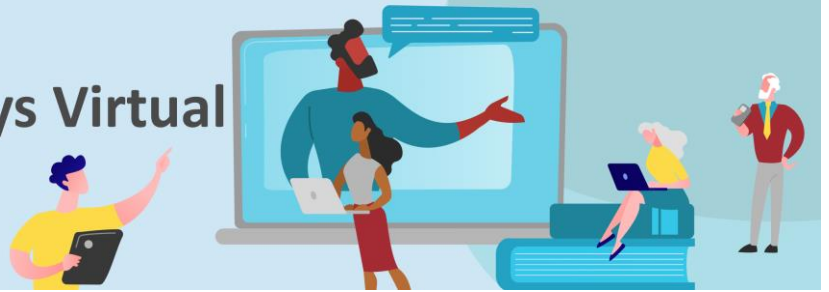




HL7® FHIR® DevDays Virtual

June 7-10, 2021



Exercise

Let's Build .Net June 2021

Track : Marco Visser

During the Let's Build! session, you will learn:

- Use the validation options to validate the FHIR resources
- Communicate with a FHIR server to send and retrieve FHIR resources

After completing this tutorial, you will be able to:

- Using the validation methods of the Firely .Net SDK
- Using REST to communicate with a FHIR server

Validating FHIR resources

For this exercise, we are going to validate our resources before sending them to a FHIR server. We will use the Validator of the Firely .NET SDK.

Exercise steps

- Create a new C# Console project
- Add the NuGet Package `Hl7.Fhir.R4` (latest version is 3.3.0) to your C# project
- In the main code, create a static object of type `Patient` and fill in some properties of this patient, for example a Name, Active and Birthday:

```
private static Patient _patient = new()
{
    Name = new List<HumanName> { new HumanName { Family = "Visser" } },
    Active = true,
    BirthDate = "2001-03-01",
};
```

- Let's serialize this patient to a string so we can display it in our console. For that we can use the extended method `ToJson()`. When don't want to have to whole json on 1 line, you can use the `FhirJsonSerializationSettings` and set `Pretty` to true.

```
// pretty print the json
var jsonSerializationSettings = new FhirJsonSerializationSettings { Pretty = true };
_patient.ToJson(jsonSerializationSettings);
```

- So we have our (in memory) patient ready. Now let's validate this patient to make sure it meets the FHIR rules.
- Add the NuGet Package `Hl7.Fhir.Specification.R4` (latest version is 3.3.0) to your C# project
- Add to your main code, the following using statement to include the validator:

```
using Hl7.Fhir.Validation;
using Hl7.Fhir.Specification.Source;
```

ORGANIZERS





HL7® FHIR® DevDays Virtual

June 7-10, 2021



- Create a new Validator instance:

```
var validator = new Validator();
```

- Try to validate a single patient:

```
var outcome = validator.Validate(_patient);
```

- The outcome of the validator is of type `OperationOutcome` and is also a FHIR resource. We can serialize this to a string and write this to the console:

```
// print the outcome
Console.WriteLine($"Success: {outcome.Success}
\n{outcome.ToJson(jsonSerializationSettings)}");
```

- Now, run the console application and check the outcome of the validation operation. This outcome has some properties that you can use:
 - Success: a Boolean which indicates whether the validation was successful or not
 - Issue: a list of issues that were raised during validation
- See [this link](#) for more information about the `OperationOutcome`.
- You will notice that the validation fails. The message `[ERROR] Unable to resolve reference to profile 'http://hl7.org/fhir/StructureDefinition/Patient'` is shown.

The validator needs the standard Patient profile (`StructureDefinition`) to validate the instance. So, we must tell the validator where to find this profile. We do this by passing a `ResourceResolver` to the validator. For all the standard HL7 FHIR resources, the SDK has a special `ResourceResolver` already made for you:

```
ZipSource.CreateValidationSource();
```

```
var resolver = ZipSource.CreateValidationSource();
var settings = ValidationSettings.CreateDefault();

settings.ResourceResolver = new CachedResolver(resolver);
var validator = new Validator(settings);
```

- Note that we wrap the standard `ResourceResolver` in a `CachedResolver`. This will speed up the validation when you validate more than 1 resource.
- Run the program again and you will see that the validation of Patient is successful.
- The field `language` in `Communication` is mandatory (see also <https://www.hl7.org/fhir/patient.html>). When we add a communication item to patient and leave out the language, the validator should report this. Try this out.

The validator can also use other profiles to validate against. For example the profile `'Us-core-patient'`, see [here](#) for the definition.

In the next steps we are going to validate our in memory patient to this us-core-patient profile.

- Downloading the profile (`StructureDefinition-us-core-patient.xml`) would be not enough, because this particular profile is also dependent on other profiles. It would be better to download the [FHIR package](#) of us-core, which contains all the us-core profiles.
- Extract the package in a subdirectory (`'profiles'`) of your solution.



HL7® FHIR® DevDays Virtual June 7-10, 2021



- In order to use these profiles we have to tell the validator where to find this profile. We do this with a `DirectorySource`:

```
var directoryResolver = new DirectorySource("profiles");
```

- This will read all profiles in the subdirectory profiles. To combine this profile with the standard profiles we use the class `MultiResolver`. The code would be then:

```
var resolver = ZipSource.CreateValidationSource();  
var directoryResolver = new DirectorySource("profiles");  
  
var settings = ValidationSettings.CreateDefault();  
settings.ResourceResolver = new CachedResolver(  
    new MultiResolver(resolver, directoryResolver));  
  
var validator = new Validator(settings);
```

- Let's validate our patient against this new us-core profile:

```
var outcome = validator.Validate(obs, new[] {  
    "http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient" });
```

- You will see that the validation fails, because an identifier and gender is mandatory.
- Change your in-memory patient so that it validates again.
- You can also add the profile in the meta part of the patient. The validator will pick that up and uses this profile to validate your instance. Try this out.

Session 2b – Communicate with a FHIR server

Exercise: For this exercise, we will be using the in-memory patient from session 2a and send this FHIR resource to a FHIR server. In addition, we will cover a few search methods to find resources on a FHIR server.

Exercise steps

- Open your previously created app (session 2a)
- To your main code, add a using statement to include the FHIR RESTful client:

```
using Hl7.Fhir.Rest;
```

- Create a new `FhirClient` object, pointing it to the public Firely test server "<https://server.fire.ly/r4>". With the `FhirClient`, you can use methods for the RESTful interactions:

```
var client = new FhirClient("https://server.fire.ly/r4");
```

- Let's see what the Firely Server is capable of. We can do this by retrieving the capability statement of the server:

```
var capability = client.CapabilityStatement();
```



HL7® FHIR® DevDays Virtual

June 7-10, 2021



- This object is also a FHIR Resource (<https://www.hl7.org/fhir/capabilityStatement.html>) and can be serialized to a string. For now we are only interested in a few properties, like the name and FhirVersion. Let's print those to the console:

```
Console.WriteLine($"capability: Name: {capability.Name}, Fhir Version: {capability.FhirVersion} ");
```

- It should return the following:

```
capability: Name: Firely Server 4.2.0 CapabilityStatement, Fhir Version: N4_0_1
```

- So we made sure that our FHIR server can handle FHIR R4 (4.0.1) resources. Now let's try to validate our in-memory patient on the FHIR server. Actually the same procedure what we have done in session 2a, only now the validation is done by the server. To accomplish this, we use the method `ValidateResource` of the `FhirClient`. This function will upload the resource to the server and the server will validate this resource and gives back an `OperationOutcome` to the client:

```
var oo = client.ValidateResource(_patient);
```

- You will notice that the resource is not validate, because we still have the profile of `us-core-patient` in the meta section of our patient, and this profile does not exist on the FHIR server, only the standard FHIR profiles.
- Remove the meta section in our patient instance and try again.
- Now let's try to upload our in-memory patient and persist it on our FHIR server. We will use the method `Create` for that:

```
var pat = client.Create<Patient>(_patient);
```

- If you use the `Create` method, the server will assign a new technical ID to the resource. The return value of the function `Create` will have this patient and technical ID. You can write this id to the console.
- Using the `Update` method, your Patient will be created with the technical ID you have assigned, or updated if a Patient with that technical ID already exists – please note that a production server will not always allow this.
- We can now try to read this patient back from the server. We use the function `Read<Patient>` for that:

```
var patFromServer = client.Read<Patient>($"Patient/{pat.Id}");
```

- Note that the meta section now also contains `versionId` and `lastUpdated` fields.
- The FHIR client has several operations to do basic search. For example to search for all patient with family Name = "Visser" we can do the following:

```
Bundle results = client.Search<Patient>(new string[] { "family:exact=Visser" });
```

- For more complex searches you can use the Firely .NET SDK `SearchParams` class. Like so:

```
// using the class SearchParams
var q = new SearchParams()
    .Where("family:exact=Visser")
```



HL7® FHIR® DevDays Virtual

June 7-10, 2021



```
.OrderBy("birthdate", SortOrder.Descending)
.summaryOnly()
.include("Patient:organization")
.limitTo(5);
results = client.Search<Patient>(q);
```

- This will search for patients with an exact family name = 'Visser', ordered by birthdate and include organization (if present). Only the summary is returned (so not all fields). And also limit the results to 5 patients.
- The example above limits the results to only 5 patient. To retrieve the 5 next patients, just use the FhirClient function Continue:

```
// continue with the next page:
results = client.Continue(results);
```

Extra exercise

The public test server is open to everyone, and does not use authorization/authentication. In production systems you will run into needing this, and often it is implemented with OAuth2. Servers that support the SMART on FHIR app launch will be able to recognize this and use a SMART on FHIR token. The next steps in this exercise will guide you to add such a token to the FhirClient, in order to be used on a request.

We assume you have a valid token. If not, there is a good topic on that to be found on [chat.fhir.org](https://chat.fhir.org/#narrow/stream/179171-dotnet/topic/SMART.20on.20FHIR.20app.20sample.20code.20in.20dotnet), pointing to a couple of code projects to help you with

that: <https://chat.fhir.org/#narrow/stream/179171-dotnet/topic/SMART.20on.20FHIR.20app.20sample.20code.20in.20dotnet>

- Adding a token to be sent with every request, is done by implementing your own message handler to be used by the FhirClient. An example for authorization would be:

```
public class AuthorizationMessageHandler : HttpClientHandler
{
    public System.Net.Http.Headers.AuthenticationHeaderValue Authorization { get; set; }

    protected async override Task<HttpResponseMessage> SendAsync(HttpRequestMessage
        request, CancellationToken cancellationToken)
    {
        if (Authorization != null)
            request.Headers.Authorization = Authorization;
        return await base.SendAsync(request, cancellationToken);
    }
}
```

- Then add that to the FhirClient – note the different server base!:

```
var handler = new AuthorizationMessageHandler();
var bearerToken = "AbCdEf123456"; //example-token, replace with provided token
handler.Authorization = new AuthenticationHeaderValue("Bearer", bearerToken);
var client = new FhirClient("https://labs.vonk.fire.ly/r4", null, handler);
```



HL7® FHIR® DevDays Virtual June 7-10, 2021



- Now you can request a Patient:

```
var pat = client.Read<Patient>("Patient/test");
```

- You can also perform a search for the Patient's white blood cell count Observations, resulting in a Bundle resource:

```
var q = new SearchParams("code", "http://loinc.org|6690-2");  
var result = client.Search<Observation>(q);
```

- Add those results to an Observation list, and display the Patient and Observation data
- Do the same for red blood cell count, and hemoglobin Observations Have fun, and remember to ask for help if you get stuck!

Further information

Some useful links:

- Extra documentation for FhirClient: <https://docs.fire.ly/projects/Firely-NET-SDK/client.html>
- HL7 Fhir Restful API specification: <https://www.hl7.org/fhir/http.html>
- Zulip stream for asking questions about Firely .NET SDK: <https://chat.fhir.org/#narrow/stream/179171-dotnet>
- List of Fhir Test servers: <https://confluence.hl7.org/display/FHIR/Public+Test+Servers>
- SMART overview: <https://github.com/GinoCanessa/FhirDevVideoNotes/tree/main/03-Getting-SMART>

The tutorial and code for this exercise can be found at:

<https://github.com/FirelyTeam/DevDays2021June-LetsBuild/tree/session2>

Have fun, and remember to ask for help if you get stuck!