FHIR - Hands On Guide

Rik Smithies, NProgram Ltd.

Objectives

Beginners to be able to use FHIR and REST

Browse servers and understand FHIR URLs

Use a REST client to read and write data

Create and validate your own resources

Perhaps start coding (or try another day)

For those that already code with FHIR, jump to slide Coding FHIR (1)

Slides are in the form of exercises. Not all the answers are here ☺

Other resources, for today or later

Lots of useful links on Confluence
https://confluence.hl7.org/display/FHIR

In particular the FHIR Chat group (chat.fhir.org aka "zulip")

Let’s go

You can read the specification (and it’s worth doing)

But a good way to understand FHIR is to look at some FHIR servers

A list is here

https://confluence.hl7.org/display/FHIR/Public+Test+Servers

One that is good for learning the basics is:

http://nprogram.azurewebsites.net

Browsing FHIR (1)

FHIR "RESTful" servers are structured like websites

There is a "root"
e.g. http://nprogram.azurewebsites.net

The FHIR "resources" are pages underneath the root, each with its own address (URL).

e.g. http://nprogram.azurewebsites.net/Patient/1 is the address of the patient with id 1.

Go there now and see.

Servers…

These exercises use the public FHIR test servers (see link earlier).

They are provided free by various individuals and companies. There is no guarantee that they are bug free, available, support all features or have all types of data.

But in general they are reliable and a great help when learning FHIR.

Be prepared to switch servers if need be.

Since data on servers can vary, you can adapt searches in these exercises to try to find useful data.

Tip – if you can’t find a patient with name e.g. “Hobbs”, try just name “H”, or any other letter. It works like a wild-card.

Browsing FHIR (2)

"Patient" is the name of a type of FHIR resource, and is also the page of the site where those resources are stored.

Look at patients 2 and 3 and so on.

Back at the root page, there are links that work to show JSON instead of XML, and that go to different types of resources (e.g. Organization), and that do searching.

Can you see how the website tells the server that you want JSON, and how you use a URL to search?

Get patient 5 to display in XML - then manually change the URL to show JSON

Get the XML for all patients with name “Hobbs”

Browsing FHIR (3)

FHIR servers use HTTP to communicate (that is what REST is), just like any other website, and that is why they can be browsed with a normal internet browser.

However since they are mainly intended for use by machines rather than people, some FHIR servers are more human friendly than others.

e.g. at http://nprogram.azurewebsites.net, there is a page full of links. Because this server is meant for demonstrations, it needs a human friendly front end.

Other servers such as https://hapi.fhir.org/baseR4 (a fully functional FHIR server) have home pages, but some don't.

Browsing FHIR (4)

If the FHIR server recognises that it is being read with a browser (and so it is really a person rather than a machine), some servers will show HTML, or XML or JSON formatted as text (more on this later).

Others will send out pure XML but you will be prompted to save it as a file (and then open it with an application). This works, and is what machines expect, but isn't so easy to browse as a person.

When browsing a server, it is not always easy to know if it is running or not. How might you be able to tell?

You can’t look for a particular patient id, it may not exist.

Browsing FHIR (5)

Something that works on all servers is the metadata page - which tells you what the server supports.

http://nprogram.azurewebsites.net/metadata

https://server.fire.ly/metadata

FHIR Bundles

Look again at the XML for Patient/1

(http://nprogram.azurewebsites.net/Patient/1)

Now do a search for patients with name “Hobbs”. There is only patient, and it’s the same one.

Note that like any website you can open different tabs or windows to make it easy to compare things.

Compare the XML results of the search to the Patient obtained from “Patient/1”.

Search for name “Bradley", and compare the results again.

Links between resources (1)

Resources exist at FHIR end points such as

{name of site}/Patient/1

But resources can also link to each other.

e.g. the patient here http://nprogram.azurewebsites.net/Patient/1

references a generalPractitioner of Organization/1

You can see that Organization itself at http://nprogram.azurewebsites.net/Organization/1

Links between resources (2)

This is easier to see if the linked HTML version of the site

is used: e.g. http://nprogram.azurewebsites.net/html/Patient/1

and click the Organization link (near the end).

More complex resources such as DiagnosticReport will link to several resources.

See http:/nprogram.azurewebsites.net/html/DiagnosticReport/10

This has links to Patient id 1, Organization id 10 and Observations 1 to 17 (which provide the data on which the report is based).

Links between resources (3)

Note though that there are times when you want to send a report in its entirety. The previous example just has links, so the receiver would need to fetch all the other observations one by one.

And this only works in a REST situation. If this same XML was transferred as a message, the observations may not be accessible.

Resources can also be “contained” within others.

see http://nprogram.azurewebsites.net/html/DiagnosticReport/1

Note that the "<contained>" resources, after the <text>. These are in-line resources.

Links between resources (4)

The observations are referenced in the results at the end of the report, as with the previous DiagnosticReport example.

But note how the references have a # to indicate that they are local to this resource (local to the DiagnosticReport).

FHIR data formatting (1)

You may be wondering how the same resource can show in different ways

e.g. http://nprogram.azurewebsites.net/html/DiagnosticReport/1

shows formatting,

vs. http://nprogram.azurewebsites.net/DiagnosticReport/1

which is plain.

To explain, you need to understand that FHIR is intended for machine to machine integration.

But since it is usually implemented as "REST" based (meaning http and browser technology) it is easy to use a browser to explore things. This is useful for introducing people to FHIR, and for testing (and debugging).

FHIR data formatting (2)

There are strict rules about how a FHIR server formats its data.

Specifically it must return data using an http "Content-Type" (MIME type) of "application/fhir+xml" (or in the case of JSON "application/fhir+json").

By contrast, normal websites use "text/html".

It is not easy to look directly at "application/xml+fhir" content with a browse - your browser may offer to save the file to disk.

This works fine, but is inconvenient. To see this go to:

http://nprogram.azurewebsites.net/raw/DiagnosticReport/1

FHIR data formatting (3)

Web servers can detect the difference between another machine asking for a data, and a person using a web browser. To make things easy for humans, the server can detect this case and format the data so that it is directly viewable in the browser.

This is fine and doesn't break the specification, as long as a machine always sees the proper format, and only humans get the pretty printed version (or several different variations).

Knowing about Content-Types e.g. "application/fhir+json" becomes important when writing data back to FHIR servers.

REST clients

FHIR servers can be read with a browser, but to really explore them you need to be able to write data back.

Browsers are not good at this on their own.

For sending data you need a REST client tool.

REST client programs use the same HTTP language as a browser but just give you more control.

Examples are Fiddler and Postman

Get Postman from here:

https://www.getpostman.com/

Using Postman to read (1)

Pick a server from the list of public FHIR servers (was on one of the links earlier :-). Choose one that is not read only (so not http://nprogram.azurewebsites.net)

(The http://hapi.fhir.org/baseR4 server tends to be reliable.)

Task 1 is to use GET to read a patient by id.

But first you need to find a valid patient id.

To do this, get a list of all patients from: {server URL}/Patient

In Postman use the GET option - in the drop down list - and the Send button to read from (for example) http://hapi.fhir.org/baseR4/Patient

Using Postman to read (2)

In the response window (scroll down if need be) you should get a large set of patients in JSON or XML (depends on the server), inside a "Bundle"

Search for a "resourceType": "Patient" within the Bundle JSON (or <Patient> for XML).

Find the id for that Patient (normally right after the Patient tag), and see the number (or string).

Use that id as a new URL and read that patient directly, again using Postman and GET (tip - you can open another tab so you can keep the Bundle open).

You should see just that patient JSON/XML alone, and not in a Bundle.

Using Postman to read (3)

Task 2. Now search patients by name

You can use any name or part of a name (e.g. “Brooks”, or just “B”), until you get some hits.

Task 3. Now get the same patient as in 1 by id but this time as XML, if you had JSON before (or vice versa), by changing the URL.

Task 4. (harder) Now get the same patient as in 1 by id, again as the other format, but this time using the original URL (i.e. don’t add the "?" part to the URL).

(clue, use the headers, and see https://www.hl7.org/fhir/http.html, content-type)

Using Postman to write (1)

Now you will write the patient from the last section back to a server, as a new patient.

For this you need to use POST rather then GET (from the dropdown).

Use the Body "tab", (below the URL, looks like a menu), to actually contain the XML or JSON that you want to send.

You will need to set the Body type to "raw" - it will probably say "none" initially.

Copy and paste the patient data you fetched earlier.

Using Postman to write (2)

You also need to set the Content-Type header to "application/fhir+xml" or "application/fhir+json" using the Headers tab

Use key = "Content-Type" and value = "application/fhir+xml" (or "...+json")

Do the POST and check the Response. You should see an HTTP 201 code ("Created")

The Patient will have been assigned a new id.

(Some servers may not like that the Patient XML/JSON you sent had an id already - if so, remove it and try again)

Using Postman to write (2)

Check the Content-Location of the response (in the Headers tab - make sure you are in the lower set of tabs).

The part after "Patient/" is the new id that got assigned. (There may also be a history id, but that can be ignored).

Also check the actual id inside the Patient resource XML/JSON that you just created and was echoed back to you. It should match the Content-Location.

Now read this patient back again, by id. (Remember to change PUT back to GET). Again, you can also use another Postman tab to make it easier.

Now edit that patient (change the name maybe), perhaps using notepad (cut and paste it out of Postman), and save it back over the *same* resource - don’t create a new patient (i.e. not POST)

Using Postman to write (4)

Now read that edited patient back, to check it worked, but as the other type JSON vs XML. Note how your XML was converted to JSON, or vice versa, by the server on the fly.

You have now covered reading, creating and updating resources using REST “verbs”.

Create your own resource (1)

In this step you will create and validate your own resource XML data.

This requires an XML tool, such as a trial version of Oxygen XML, or XML Spy. https://www.oxygenxml.com/xml_editor/register.html

http://www.altova.com/simpledownload1.html

Download and install the FHIR schemas

https://www.hl7.org/fhir/fhir-all-xsd.zip

Create your own resource (2)

Now create an empty XML file, and start it off:

```
<?xml version="1.0" encoding="utf-8"?>
<Patient xmlns="http://hl7.org/fhir"></Patient>
```

Validate it with the schema “fhir-all.xsd”. (In Oxygen, you can create a "validation scenario", browse to the xsd, then save and apply it to your file.)

Now add in some data. Use the schema to guide you, but look at https://www.hl7.org/fhir/patient.html for details.

When it comes to POSTing, bear in mind that any references you use (e.g. an Organization), must already exist on the server.

Create your own resource (3)

A next step is to create an Observation resource, and POST it so that it links to your Patient using Observation.subject.

A more advanced task would be to create and POST a FHIR document.

See https://www.hl7.org/fhir/documents.html

Once you have written an Observation that links to your Patient, write 2 queries:

1. Query for the Patient, by name, but also retrieve the linked Observation at the same time
2. Query for the Observation, but using the name of the Patient

Coding FHIR (1)

Those who can program are encouraged to write code that reads a patient from a FHIR server.

FHIR has library code you can use to get you started quickly – though you can also start from scratch using your environment’s HTTP functions.

There are guides to using C# and Java here:

C#: https://fire.ly/blog/make-your-first-fhir-client-within-one-hour/

Java: http://fhirblog.com/2014/07/31/fhir-connectathon-7-for-java-dummies/

Coding FHIR (2)

For coding objectives, anything you achieve is a good result. But we can use the FHIR Connectathon tracks, specifically the Patient one:

1. Register a new patient
2. Update a patient
3. Retrieve patient history
4. Search for a patient on name

More exercises

For those who don’t code, there is much to explore with the FHIR querying features, using only a browser.

FHIR allows searches that use “and” and “or”.

Also you can “chain” searches, over different resources, for instance to search for diagnostic reports for a named patient (even though the patient’s name is not in the DiagnosticReport but in the separate Patient one).

Not all servers support all types of searches.
Try the “hapi ” or “firely” servers for full searching features.

See https://www.hl7.org/fhir/search.html