smile CDR

Smile Guide: RESTful FHIR

Table of Contents

What to Expect

🕘 *Reading time = 20 minutes*

This document is intended to teach you about FHIR's RESTful API. By the end of this guide, you'll be able to:

1. Identify the different parts that form a basic query
2. Describe the different parts of a query result
3. Create and execute FHIR queries
4. Customize FHIR queries with modifiers and custom parameters

You'll be relieved to know you don't need to be a coding expert to understand REST and use it successfully. This document will cover some of the basics of RESTful FHIR and coding techniques. While performing the exercises in this document, you will gain valuable experience in creating and executing queries against any FHIR repository—from the basics to many advanced features.

This is an interactive document and, as such, you should have access to a FHIR server—either your own, or reading how to access Smile CDR's playground. By the end of this document, you should be comfortable searching, accessing and manipulating FHIR data.

Welcome to the world of FHIR. We hope you enjoy your journey!

Note: If you already understand REST basics jump to "Exercises" on page 11.

Background

What Is REST?

API's are application programming interfaces. REST (Representational State Transfer) is an API coding style designed to provide easy uniform standards across the internet (More Information can be found here). RESTful systems are characterized by how they separate client and server requests. In RESTful API's, server implementations and client implementations can be done independently. In common practice, this means clients can take advantage of a server's data and methods (functions) without knowing the inner-workings of the server. For example, Google Maps publishes APIs with which any third-party client can interact. FHIR uses REST to expose and manipulate FHIR resources.

For a beginner's guide on REST, check out this video: What Is a RESTful API?

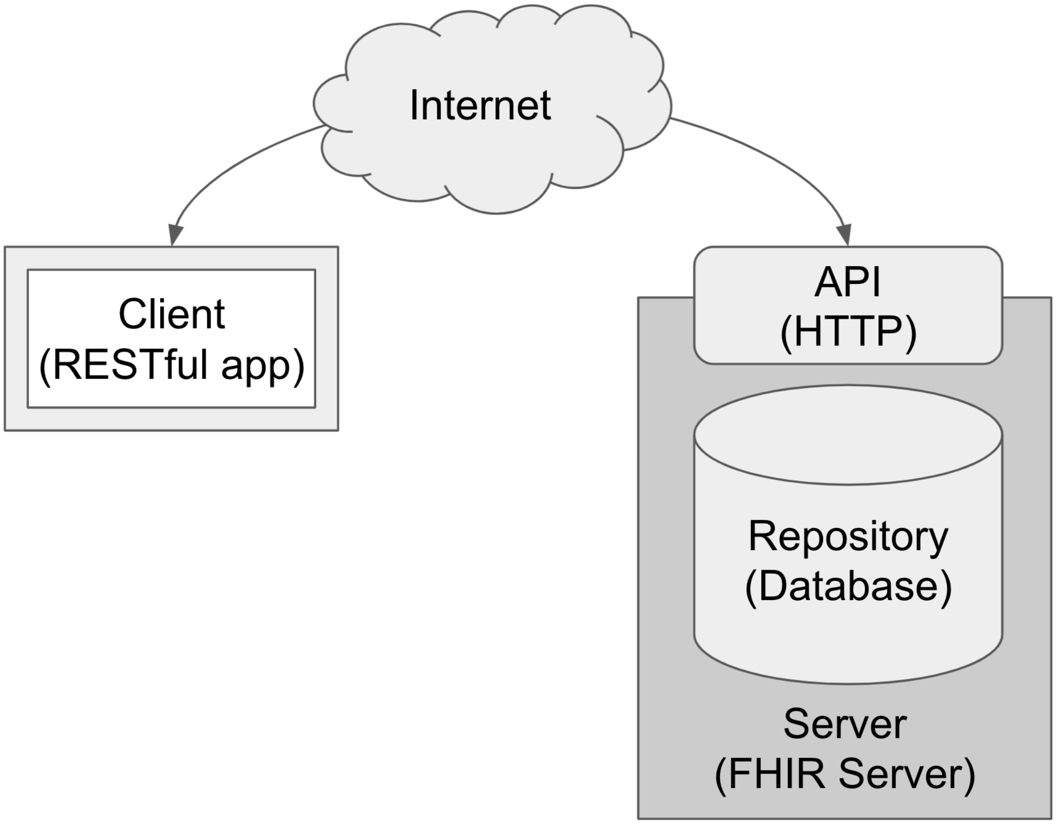


Figure 1: Overview of RESTful client-server interaction

Prerequisites

The following are required:

Knowledge Areas

- It's useful if Smile CDR is already installed. If it's not installed, check out this guide. Throughout this tutorial, we'll be using the open source endpoint *http://hapi.fhir.org/baseR4* which does not necessarily require a local instance of Smile running.

- It's assumed that there's an understanding of Smile CDR

Tools

- Either Postman or Insomnia[1] are downloaded and the reader has experience using the application. This document will use Postman for all the query examples. However, you can use Insomnia instead.[2] Postman and Insomnia are API clients that make it easy for developers to create, share, test and document restful APIs.

FHIR Queries

Parts of a Query

[Verb][Server][API Base][Resource Type][Optional Parameters]

- Verb: What you're telling the server to do. There are many HTTP verbs, with the most common being:
    - GET—retrieve a resource
    - POST—create a resource
    - PUT—update (replace) an entire resource
    - PATCH—update (alter) a minor aspect of a resource
    - DELETE—remove a resource
- Server: This is the computer, often in the cloud, that stores the data. (The server also manages access, security, backups and a number of other important functions.)
- API base (endpoint): The URL (path) to access the API.
- Resource: A single coherent collection of data. Ex: a single patient, a single medication, a single medical billing claim, etc...

[1] Certain aspects may be a little different in Insomnia. You may need to tweak.

[2] Certain aspects may be a little different in Insomnia. You may need to tweak.

- Resource Type: The category of resource where items can be found. Ex: Patient, Medication, Claim, etc...
- Optional Parameters: The optional parameters allow you to filter (refine) what you are searching for, or apply an operation on the server.

A query is built with individual factors from the list above that you assemble together. Think of it like a library. The verb is why you're at the library (getting a book, depositing a book, etc...). The server is the library's address. The API base is the floor of the library where the information resides. The resource type is the section you're searching (biographies, history, etc...), ID is unique to a specific book (remember the Dewey Decimal System?) and the optional parameters are any specific features in that section that you're searching for (language, books by a certain author, etc...).

The Prototype Query:

- This is an example of a common query. Using it, we're searching through the family names, a.k.a last names of the patients that exist on the server. We're looking for patients whose last name is Chalmers.

[Verb] [Server] [API Base] [Resource Type][Optional Parameters]

[GET] [http://hapi.fhir.org/baseR4/Patient?family=chalmers]

- The first four elements are relatively easy to learn. The parameters involve a bit of a learning curve. (Don't worry, we'll walk you through it.)
- In FHIR, base (endpoint) refers to the server and the API base together. Going forward, they'll be combined together. The base is static for all queries on a given server.
- The search parameter used above is an example of the type of optional parameter. With this example, all patients whose last name is Chalmers will be returned from the server.
- This is a search operation query

The Resource URL:

- The resource URL structure is similar to the prototype query above. The main difference between the two is that the resource URL points to a specific resource instead of a search parameter.

[Verb] [Server][Base][Resource Type/ID

[GET] [https://hapi.fhir.org/baseR4/Patient/123]

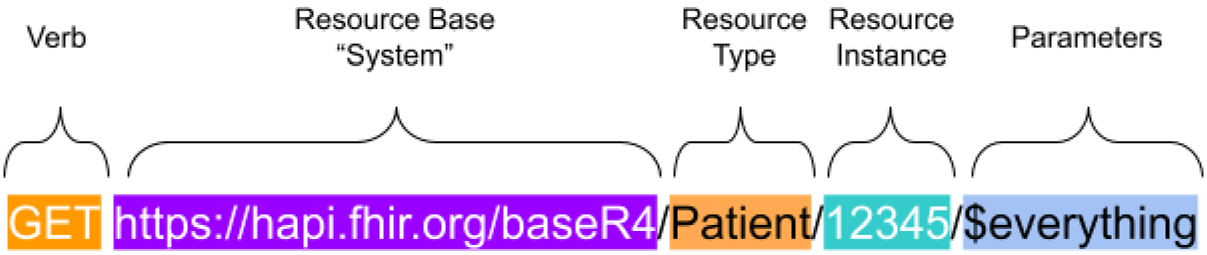
- In the example above, the resource is Patient/123[3].
- https://hapi.fhir.org/baseR4 is the resource base system
- "https://hapi.fhir.org/baseR4/Patient/xyz" is called the absolute path, where "xyz" can be any applicable ID. This query format can be called from any computer in the world and will return the same results.
- "Patient/123" is called the relative path as ID 123 only makes sense on the given server. (That is, the base is assumed.)
- The query above is a read operation

In Conclusion:

- The resource URL points to a specific resource (ex. http://hapi.fhir.org/baseR4/Patient/123). This is considered a read operation
- The Query URL is processed by the server's search engine to return any matching resource(s) (ex. http://hapi.fhir.org/baseR4/Patient?_id=123). This is considered a search operation.

In these examples, the server returns the same results (the Patient with ID 123). As we progress, you'll learn the differences between searching and reading.

Parts of a FHIR RESTful Query Overview



When performing a query, always check that you have the correct:

- Verb
- Server
- Base
- Resource type
- Spelling
- Punctuation and capitalization

[3] This value is likely to be different on your server

Data Types

Data types tell the computer how to store and process various classes of data. The primitive data types are:

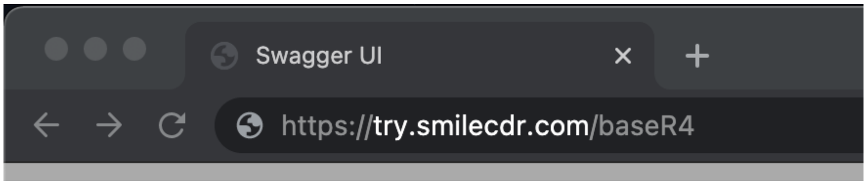
- Strings (text)
- Numeric
- Datetime
- Boolean (true/false)

Computers operate differently on different data types. For example, adding the two strings “one” + “ten” results in “oneten” and when sorting, “four” comes before “three.” While this may appear confusing at first, it serves the purpose of constraining the computer to operate on a data element in the way the programmer (usually) intended. (We’ll see the implications of this in later examples.)

FHIR has a rich set of expressive FHIR data types. Though we won’t go into them in this document, they will occasionally come up in our RESTful FHIR queries where there are data type specific methods and behaviors.

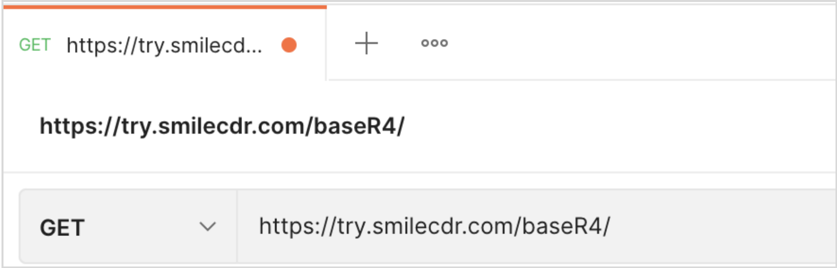
Your First Query

GET https://hapi.fhir.org/baseR4/



https://try.smilecdr.com/baseR4

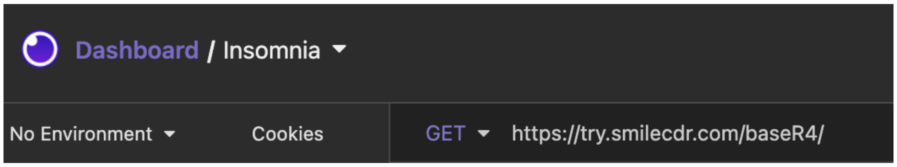
Figure 2 : The query in the address bar of a web browser



https://try.smilecdr.com/baseR4/

GET https://try.smilecdr.com/baseR4/

Figure 3: The query in Postman



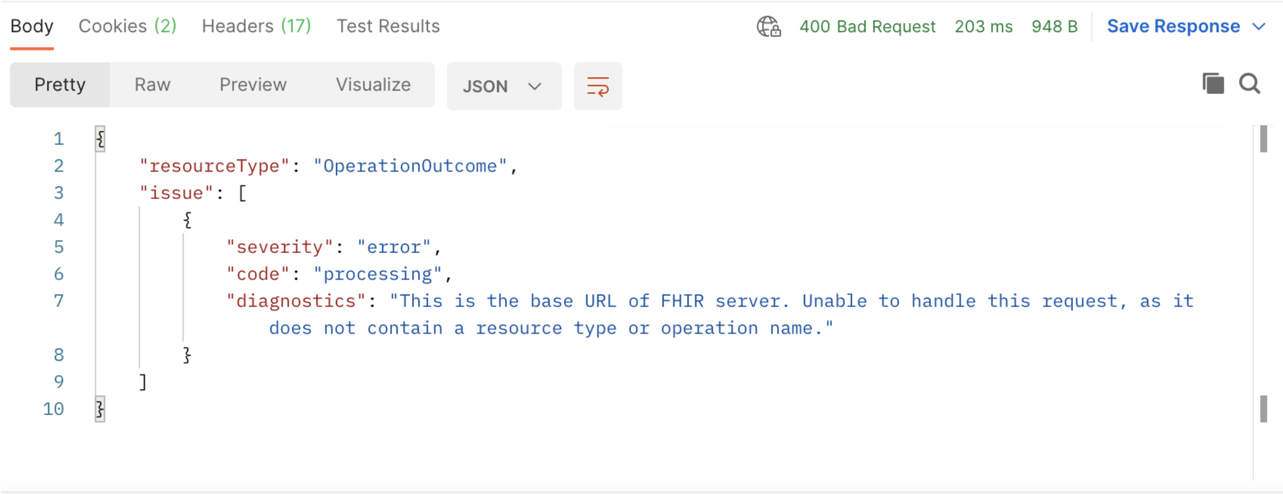
GET https://try.smilecdr.com/baseR4/

Figure 4: The query in Insomnia

Going forward, all the examples will be in Postman. But feel free to use the tool of your choice.

Errors

As you can see, this is an incomplete query. If an error occurs, the server returns an error message as shown. Mature FHIR servers will give you some insights into the cause of the error (which in this case is the missing Resource Type).



```
{
    "resourceType": "OperationOutcome",
    "issue": [
        {
            "severity": "error",
            "code": "processing",
            "diagnostics": "This is the base URL of FHIR server. Unable to handle this request, as it
                does not contain a resource type or operation name."
        }
    ]
}
```

Figure 5: Error Message

It’s also worth noting that in REST, as with most coding languages, punctuation, capitalization and spelling are incredibly important. Oftentimes a code will result in error messages, not because the code itself is wrong, but there is a misplaced punctuation, capitalization or spelling error. (These are the hardest errors to spot!)

Exercises

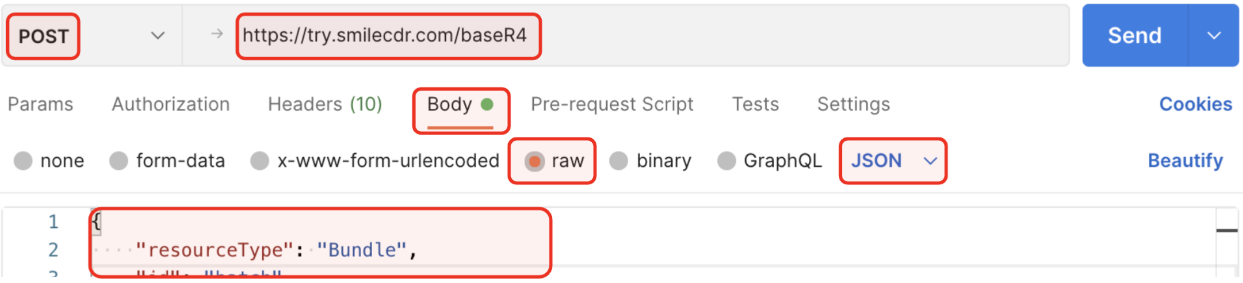
Are you ready to roll up your sleeves and work some magic? Coming up are a number of guided exercises to help you learn RESTful FHIR.

Preparations

We've prepared some sample data for these exercises. You'll want to load this data into your FHIR server. We'll submit two sets of sample data to a FHIR server using the RESTful POST operation:

First Set of Sample Data

1. **Open** the Bundle_fin.json file in any text editor
2. **Copy** the contents to your clipboard
3. **Open** *Postman* (see screenshot below, or API client of your choice) and **paste** the contents of your clipboard into the body. (Use body type Raw, JSON)
4. **Set** the action to *POST*
5. And ***set the URL*** to the server's base (e.x. https://hapi.fhir.org/baseR4)
6. **Click** *Send*



POST https://try.smilecdr.com/baseR4

Body: raw, JSON

```
{
    "resourceType": "Bundle",
```

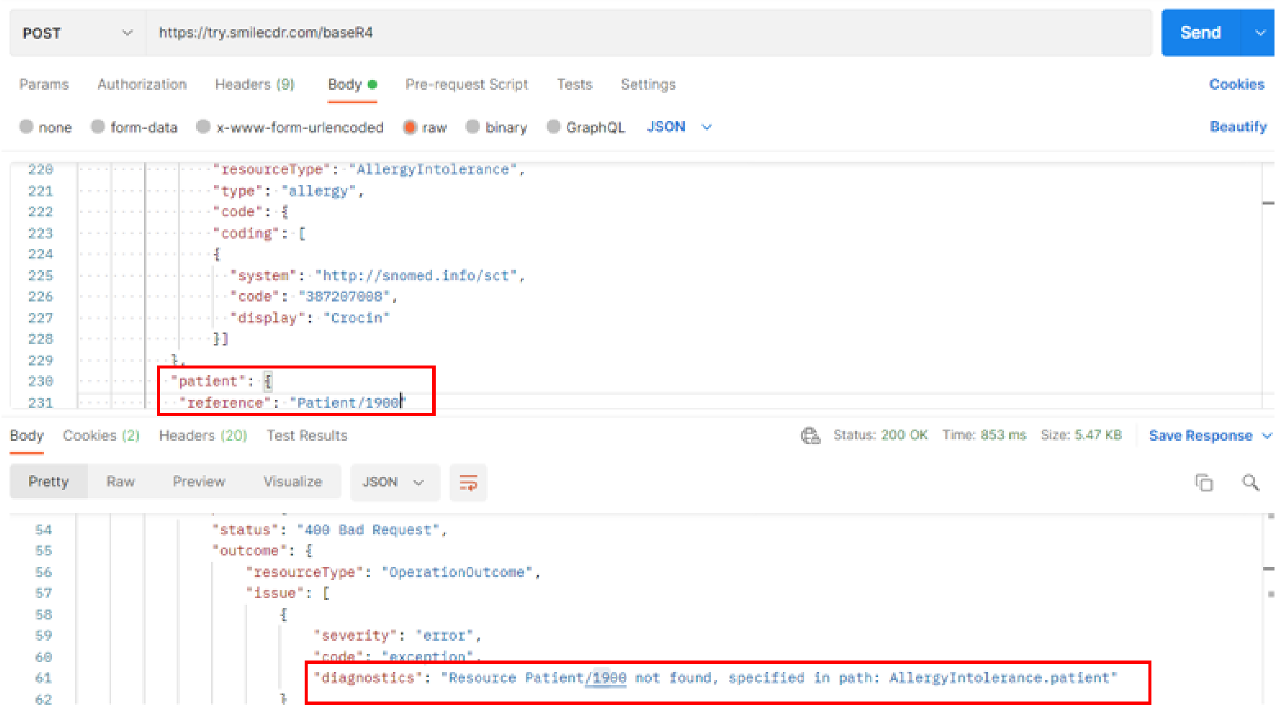
Figure 6: Syntax

❗ **Extremely Important Steps before Attempting the Exercises:**

1. When you post the bundle on Postman, as described above, you'll see that some resources in the result body posted successfully and some did not. This is because they refer to other resources through IDs that are not currently existing on the server.
2. After you post the bundle, you need to go through the result body, pick up the IDs of the resources already created and go back to edit the resources that haven't gone through in the query body.
3. Allergy Intolerance, List, Diagnostic Report, Patient, Observation and Medication Request are the resources that need to be tweaked. To tweak them, you need to go through the result body and note a few IDs of patients, organizations and practitioners and fill them up in the appropriate spaces in the request body.

For example, as Fig. 23 depicts, Allergy Intolerance is a resource that contains a reference to a patient. The error states that the ID referring to a patient doesn't exist. So we need to go back to the request body and add the appropriate ID in "Patient/xyz" by grabbing it from a patient (Fig. 24) that was already successfully posted from the bundle. Once we change the ID to 1422, the Allergy Intolerance resource is created successfully.

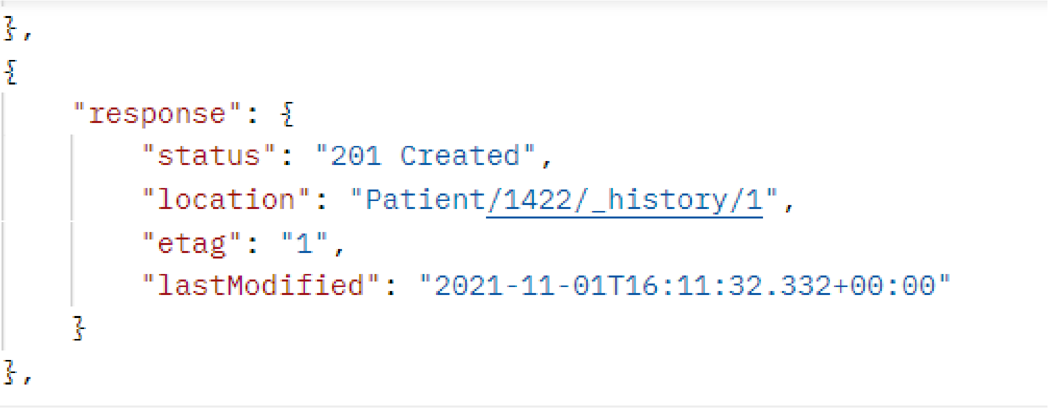
We need to do this because the server updates the IDs on its own. To work around that, we need to make adjustments to the code we send to the server.


POST
https://try.smilecdr.com/baseR4
Send
Params
Authorization
Headers (9)
Body ●
Pre-request Script
Tests
Settings
Cookies
○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON
Beautify

```
        "resourceType": "AllergyIntolerance",
        "type": "allergy",
        "code": {
        "coding": [
        {
          "system": "http://snomed.info/sct",
          "code": "387207008",
          "display": "Crocin"
        }]
    },
    "patient": {
     "reference": "Patient/1900"
```

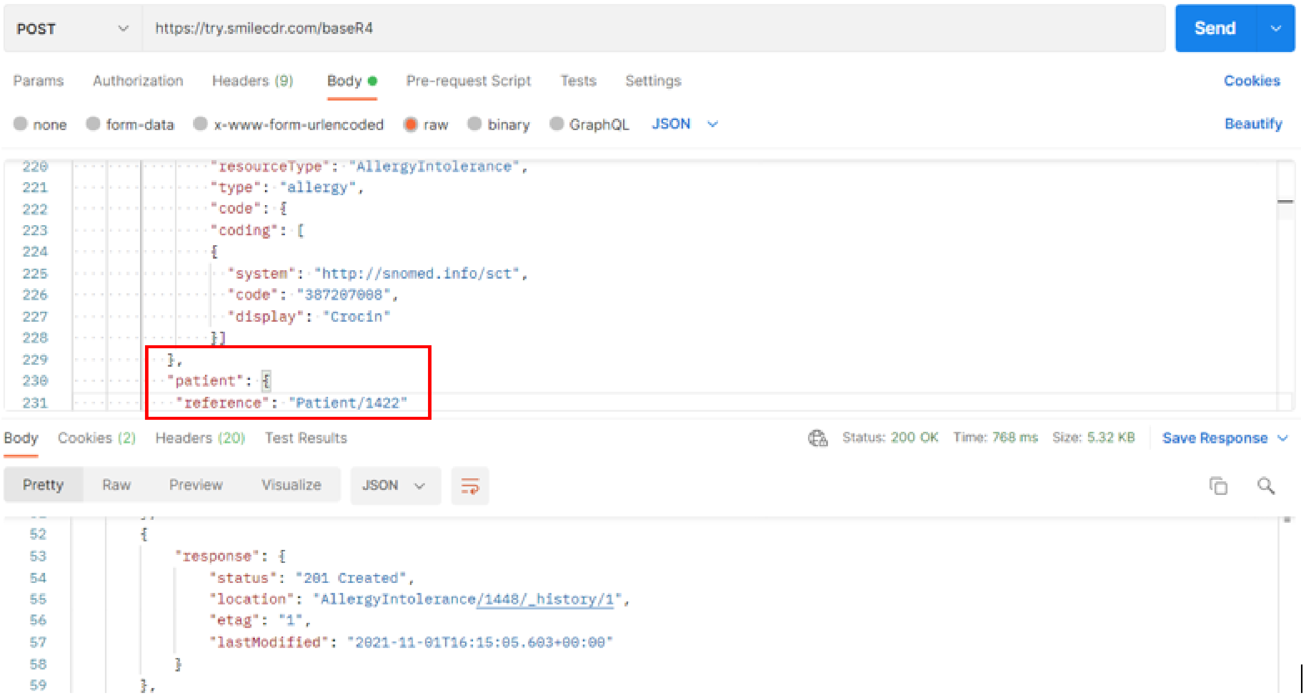
```
        "status": "400 Bad Request",
        "outcome": {
            "resourceType": "OperationOutcome",
            "issue": [
                {
                    "severity": "error",
                    "code": "exception",
                    "diagnostics": "Resource Patient/1900 not found, specified in path: AllergyIntolerance.patient"
                }
```

Figure 23 : Allergy Intolerance Patient reference error



```
},
{
    "response": {
        "status": "201 Created",
        "location": "Patient/1422/_history/1",
        "etag": "1",
        "lastModified": "2021-11-01T16:11:32.332+00:00"
    }
},
```

Figure 24 : Locating a successfully posted resource from the bundle



https://try.smilecdr.com/baseR4

```
                "resourceType": "AllergyIntolerance",
                "type": "allergy",
                "code": {
                "coding": [
                {
                  "system": "http://snomed.info/sct",
                  "code": "387207008",
                  "display": "Crocin"
                }]
          },
          "patient": {
            "reference": "Patient/1422"
```

Status: 200 OK Time: 768 ms Size: 5.32 KB

```
        {
            "response": {
                "status": "201 Created",
                "location": "AllergyIntolerance/1448/_history/1",
                "etag": "1",
                "lastModified": "2021-11-01T16:15:05.603+00:00"
            }
        },
```

Figure 25 : Grabbing the ID and pasting it in the Allergy Intolerance resource patient reference to post the Allergy Intolerance resource successfully

❗ While you post the same bundle of resources more than once to fix the IDs, you might see the following error with regard to the ValueSet resource: **"Can not create multiple ValueSet resources with ValueSet.url"**. You can ignore that. It's just saying that you cannot create the same resource twice, which we don't intend on doing.

❗ If you get stuck on any of the exercises, you can find sample answers at the end of this document.

❗ It's also important to keep in mind that, in some questions ahead, the resource ID mentioned will not be applicable for your query. There will be instances where you'll need to query for the appropriate ID to run the query for the exercise. This will be explained in more detail for each question.

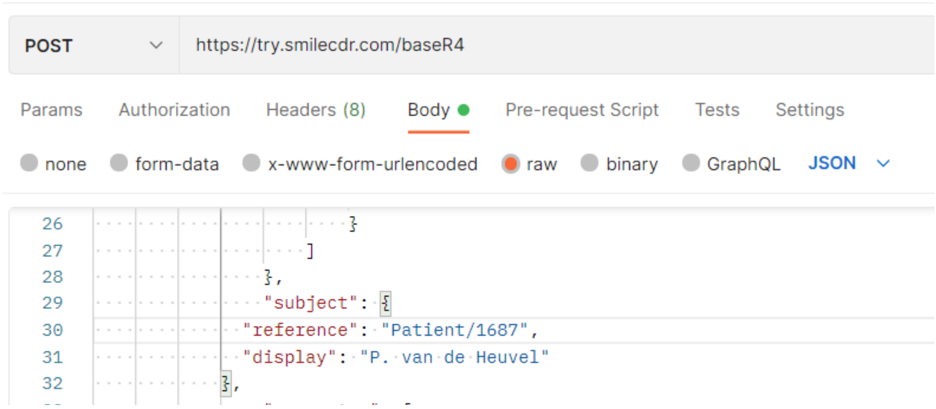
Second Set of Sample Data

1. **Open** the obser_height.json file
2. **Copy** the contents to your clipboard
3. **Open** *Postman* (or the API client of your choice) and **paste** the contents of your clipboard into the body (use body type Raw, JSON)

4. **Set** the action to *POST*
5. And ***set the URL*** to the server's base (e.x. https://hapi.fhir.org/baseR4)
6. **Click** *Send*

   As you saw in the "**Extremely Important Steps**," you'll need to replace the ID of the referenced patient with an already existing patient ID within this bundle as well, to post the resources correctly error free.

   For the '"reference": "Patient/1687"' below, the 1687 will need to be substituted with a patient ID that already exists on the server. You can use the same ID that you used in the step guided by Figure 24.



```
                    }
                ]
            },
            "subject": {
        "reference": "Patient/1687",
        "display": "P. van de Heuvel"
    },
```

Figure 26 : Replacing the Patient id in the obser_height.json file

Next, you'll need a FHIR server. There are a handful of public test FHIR servers you can use[4]. This guide uses:

1. http://hapi.fhir.org/baseR4
2. http://hapi.fhir.org/baseR4

For more publically available FHIR servers, refer to HL7 FHIR servers.

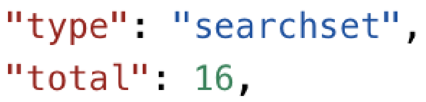
[4] Keep in mind that free cloud-based servers may occasionally reset their databases.

Ex. 1: Your First Exercise

Alright, let’s get started with your first FHIR query:

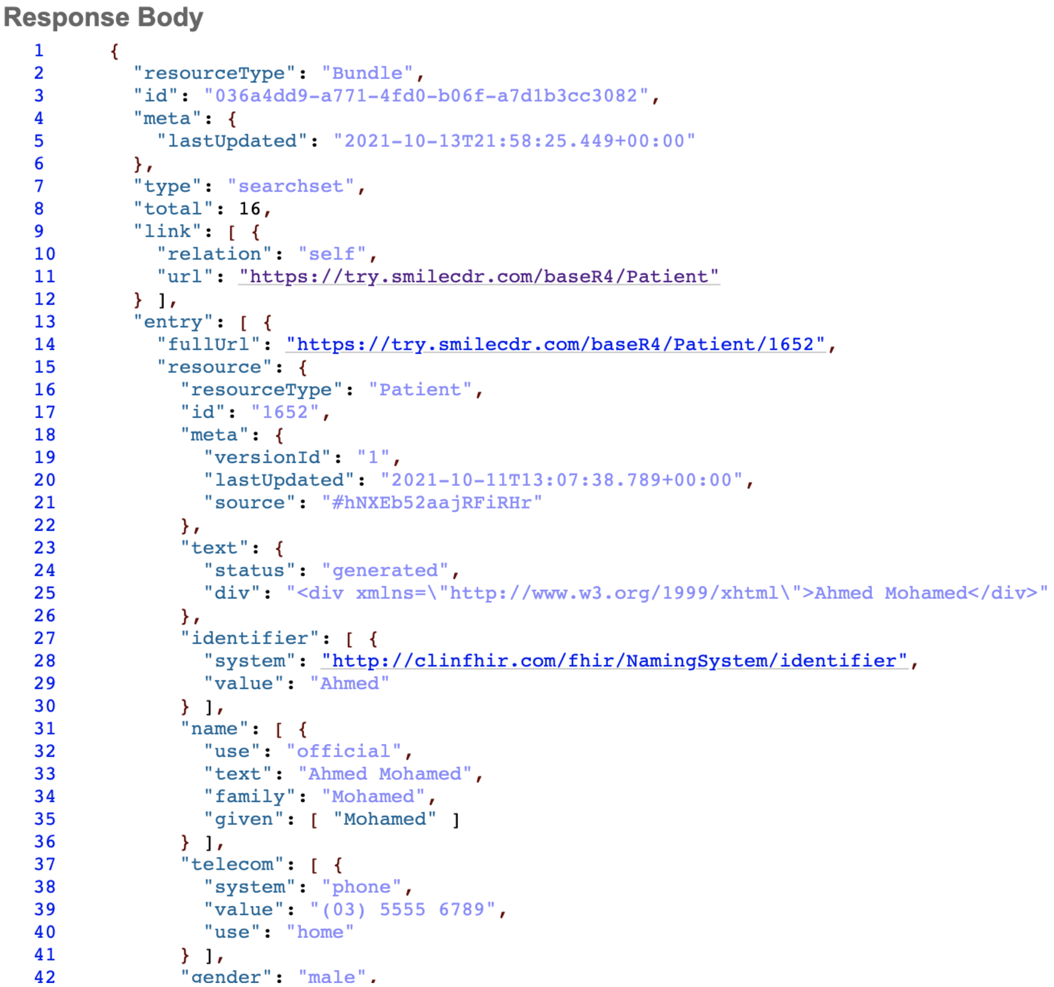
GET https://hapi.fhir.org/baseR4/Patient

This is a complete FHIR query. This query will return a bundle of all patients. You might see something similar to this in the Postman response body.



```
"type": "searchset",
"total": 16,
```

Click that URL above and see what happens. You’ll see something like this in your browser:

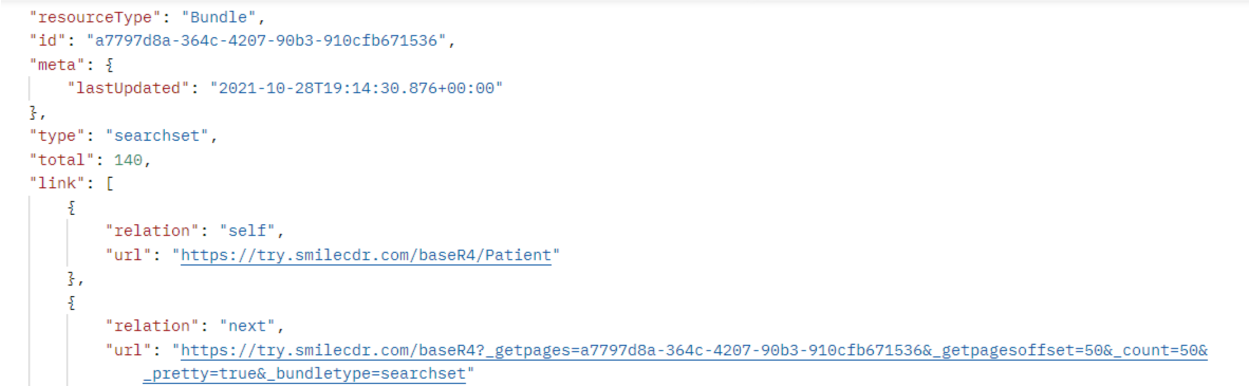


Response Body

```
{
  "resourceType": "Bundle",
  "id": "036a4dd9-a771-4fd0-b06f-a7d1b3cc3082",
  "meta": {
    "lastUpdated": "2021-10-13T21:58:25.449+00:00"
  },
  "type": "searchset",
  "total": 16,
  "link": [ {
    "relation": "self",
    "url": "https://try.smilecdr.com/baseR4/Patient"
  } ],
  "entry": [ {
    "fullUrl": "https://try.smilecdr.com/baseR4/Patient/1652",
    "resource": {
      "resourceType": "Patient",
      "id": "1652",
      "meta": {
        "versionId": "1",
        "lastUpdated": "2021-10-11T13:07:38.789+00:00",
        "source": "#hNXEb52aajRFiRHr"
      },
      "text": {
        "status": "generated",
        "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">Ahmed Mohamed</div>"
      },
      "identifier": [ {
        "system": "http://clinfhir.com/fhir/NamingSystem/identifier",
        "value": "Ahmed"
      } ],
      "name": [ {
        "use": "official",
        "text": "Ahmed Mohamed",
        "family": "Mohamed",
        "given": [ "Mohamed" ]
      } ],
      "telecom": [ {
        "system": "phone",
        "value": "(03) 5555 6789",
        "use": "home"
      } ],
      "gender": "male",
```

Figure 7 :Response Body

Sometimes the server could be holding a large bundle of patients stored. This makes it tough for it to spit out all the patients at one go or in one list. Within the bundle, you will be able to see a data element "next" and a url after that as specified in Figure 8. This URL indicates that you need to make an additional query to call for additional patient resources. This is an example of the query: "GET https://hapi.fhir.org/baseR4/Patient?&_count=150" that could result in a response like Figure 8. In some cases, even if you click the URL, you will automatically see another bundle of resources open.



```
"resourceType": "Bundle",
"id": "a7797d8a-364c-4207-90b3-910cfb671536",
"meta": {
    "lastUpdated": "2021-10-28T19:14:30.876+00:00"
},
"type": "searchset",
"total": 140,
"link": [
    {
        "relation": "self",
        "url": "https://try.smilecdr.com/baseR4/Patient"
    },
    {
        "relation": "next",
        "url": "https://try.smilecdr.com/baseR4?_getpages=a7797d8a-364c-4207-90b3-910cfb671536&_getpagesoffset=50&_count=50&
            _pretty=true&_bundletype=searchset"
```

Figure 8: GET response

At first, Figure 7 may look intimidating. But you'll quickly see how intuitive it really is. Let's break it down:

- The format of the data is **JSON**.
- Line 2 is a **bundle**. FHIR data is stored in discrete packages called **resources**. A collection of resources is a bundle. All search results are returned in a bundle, even the empty set.
- Line 7 is of type **searchset**. This is because we did a search for all patients. If we were submitting a bundle of resources to the server, we would typically use a type of transaction or batch.
- On line 8, the server found **16** resources that matched the search criteria (in this case the search criteria was: find all patients).
- Line 14 is the complete **URL** to the first matching resource in the bundle.
- Line 15+ are the contents of the first matching resource (i.e. the patient data).

💡 **Exercise**: Now you try it...

a) Write the API call that returns a bundle of resources of type organization.

b) For an ***advanced*** question: write two separate API calls each operating on the organization ID. One should be a read operation and the other a search operation.

Note on security: many FHIR servers have security frameworks which constrain the results returned by any given query. For example, it would not be wise to return a list of all patients as this action would have a high probability of being in violation of PHI rules.

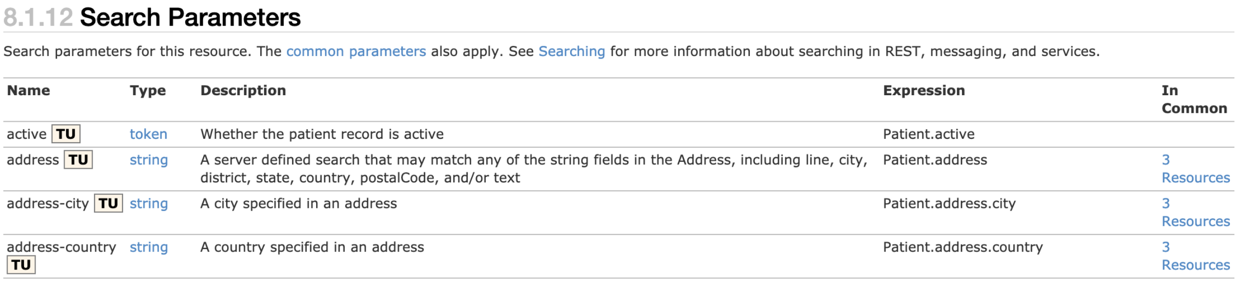
Smile CDR manages these constraints principally through permissions and consent. The test FHIR servers identified earlier typically have security disabled so you don't need to worry about security while you're learning.

Ex. 2: Searching: A Journey through GET Queries:

A simple parameter (key-value pair)

https://hapi.fhir.org/baseR4/Patient?family=chalmers

1. family=chalmers is a key-value pair:
   a. Family is the key
   b. Chalmers is the value
   c. Chalmers could match against Chalmerson
2. Each resource type has a list of searchable keys:
   a. Search keys for the patient resource are found here.
   b. Substitute "patient" with any other resource, and scroll to the bottom of the page.
   c. A comprehensive list of keys for all resources is found here.



8.1.12 Search Parameters

Search parameters for this resource. The common parameters also apply. See Searching for more information about searching in REST, messaging, and services.

Name	Type	Description	Expression	In Common
active TU	token	Whether the patient record is active	Patient.active	
address TU	string	A server defined search that may match any of the string fields in the Address, including line, city, district, state, country, postalCode, and/or text	Patient.address	3 Resources
address-city TU	string	A city specified in an address	Patient.address.city	3 Resources
address-country TU	string	A country specified in an address	Patient.address.country	3 Resources

Figure 9: Search Parameters

https://hapi.fhir.org/baseR4/Patient?family=c

2b. Here's a snippet of the first few search keys for the patient resource:

a. In a string search like the example above, the value "c" matches against the start of strings in the repository
b. This is not a case sensitive search
c. "c" will match all patients whose family name begins with a "C" or "c"

3.

Note that the **question mark sign** is used to separate the resource type and query parameters.

Exercise: You try now ...

a) Write the API call to find an organization named Peacehealth
b) Write the API call to find all patients whose last name begins with an "s"

Ex. 3: Modifying a Parameter—Exact

https://hapi.fhir.org/baseR4/Patient?family:exact=Chalmers

1. ":exact" is a modifier
   Modifiers allow you to modify the search parameter and tweak the query to return what you need:
   a. ":exact" helps matches the string precisely
   b. In this case, "Chalmers" will match
   c. "chalmers," "chalmer," "Chalmerson" and "C" will not match

2. The result will be all patients whose last name exactly matches Chalmers.
3. Modifiers generally begin with a **colon (:)**

💡 **Exercise**

Write the API call to find all patients whose last name is “Gold.”

Ex. 4: Modifying a Parameter—Contains

https://hapi.fhir.org/baseR4/Patient?family:contains=mers

1. “:contains” is a modifier
    a. “:contains” matches any part of the string
    b. “:contains” is not case sensitive
    c. In Smile CDR “:contains” is disabled by default as it is potentially a slow operation
        i. This can be enabled in the Persistence module, in the FHIR Search section

💡 **Exercise**:

(Note: You will likely get an error message because the modifier is disabled by default, however try this exercise for practice)

Formulate the API call to find all patients whose last name contains “Robin.”

Ex. 5: Two or More Parameters—&

https://hapi.fhir.org/baseR4/Patient?family=c&gender=male

1. This search finds every patient whose family name starts with “c” and whose gender is male
    a. The matching algorithm will search against all family names and all genders
2. This search generates multiple search parameters
    a. Each parameter is a key-value
    b. Each parameter is separated by an **“&”**

Note: to add multiple query parameters, **an "&" sign** is placed in between them to form what is known as a query string. In this case family=C&gender:not=female is the query string. Query strings can feature various object types with distinct lengths such as arrays, strings and numbers.

💡 **Exercise**:

Write the API call to find all active providers who reside in the state of Washington.

Hints:

1. The state abbreviation for Washington is *WA*
2. Similar to the patient resource, search keys for practitioners can be found here

Ex. 6: "Not and Not Equal " Parameters

https://hapi.fhir.org/baseR4/Patient?family=C&gender:not=female

1. The ":not" parameter is accurate for tokens. A token type is a parameter that provides a close to exact match search on a string of characters. It's mostly used against a code data type where the search is performed against the pair from a coding.
2. The linked search above finds every patient with a family name beginning with "C" that's not female.

https://hapi.fhir.org/baseR4/RiskAssessment?probability=ne0.8

1. The "ne" parameter is accurate for numbers. A number is a simple numerical value in a resource. Risk assessment, observation and immunization recommendation are a few examples of resources that often contain numerical values.
2. The linked search above searches for those risk assessment resources that contain a probability value of the risk not equal to 0.8.

💡 **Exercise**:

Use the "Not" and "&" function to write the API call to find all female patients whose name starts with "D."

Ex. 7: "Or" Parameters

https://hapi.fhir.org/baseR4/Patient?family=C&gender=male,female

1. The **comma ","** functions as an "or" search parameter
2. This search finds anyone whose family name starts with "C" and whose gender is only either male OR female

💡 **Exercise**:

Write the API call to find all practitioners residing in Alaska or Washington whose last name begins with an "R."

Hint:

1. Browse to find out the abbreviations for Alaska and Washington

Ex. 8: Equality Operators and Number/Date Precision

FHIR has a variety of equality-operators for searching[5]:

Mathematical notation	FHIR expression	Example
=	eq	Observation?value-quantity=eq30 (valid responses are 29.5 to 30.5)*
=	-	"No prefix" is the same as "eq"
<>, !=	ne	Observation?value-quantity=ne30

[5] See FHIR Search Filter Operators for more details, including additional operators.

		(valid responses are anything other than 29.5 to 30.5)*
>	gt	Observation?value-quantity=gt30 (valid responses: anything above 30)
<	lt	Observation?value-quantity=lt30 (valid responses: anything below 30)
>=	ge	Observation?value-quantity=ge30 (valid responses: 30 and anything above)
<=	le	Observation?value-quantity=le30 (valid responses: 30 and anything below)
~	ap	Observation?value-quantity=ap30 (valid responses: anything approximately equal to 30. Approximately means within 10% of the stated value, or for a date, 10% of the gap between now and the date. Note that different FHIR servers may calculate approximately differently.)

* Not what you expected? See the explanation below.

Figure 10 : Numerical Parameters

Heads up: decimals and dates are special.

When searching numbers or dates, nearby values will be included in the search result.

1. Numbers: the precision (number of decimals) determines the range of values that'll be included in the search results. For example, if you're searching for observations with a blood glucose value of 7.0 mmol/L, FHIR will include any results from 6.95 to 7.05. The general pattern for this range expansion is:

Search Value	Range of Results
100	99.5 to 100.5
100.0	99.95 to 100.05
100.00	99.995 to 100.005
123.4	123.35 to 123.45

Figure 11: Decimal Date Concepts

2. Dates: when searching dates and times, the precision included in the search parameter defines the range of values that will be returned. For example, if you only specify the year (e.x. Patient?birthdate=2000), then any patient who was born in 2000 will match. Similarly, if you specify the year and month (e.x. Patient?birthdate=2002-10), only patients born in October 2002 will match.

3. This range expansion only occurs when searching using equals or not equals (less than and greater than type searches do not expand numbers or dates).

💡 **Exercise**:

Write the API call to check for patients whose death date falls beyond August 20, 2013.

Hint:

1. Click here to see how to search for the death parameter and here for the upper or lower boundary.

Ex. 9 : "Above" and "Below" Parameters

A predominant use case for "Above" and "Below" can be used for partial matching in the case of Uniform Resource Identifiers (URIs), which are compact sequences of characters that identify an abstract or physical resource (eg. http://acme.org/fhir/).

For context:

https://hapi.fhir.org/baseR4/ValueSet?url=http://acme.org/fhir/ValueSet/123

1. The query above is a request to find any value set with the exact URL "http://acme.org/fhir/ValueSet/123"

https://hapi.fhir.org/baseR4/ValueSet?url:below=http://acme.org/fhir/

2. This query above performs a search that will return any value sets that have a URL starting with "http://acme.org/fhir/"

https://hapi.fhir.org/baseR4/ValueSet?url:above=http://acme.org/fhir/ValueSet/123/_history/5

3. This query above shows the converse search for any value set above a given specific URL. This will match on any value set with the specified URL, but also on http://acme.org/ValueSet/123

4. There are not many use cases where ”Above” is as useful as the “Below” search.

   Note: there are certain search parameters that begin with an **underscore “_”** —” _history” is one of them. Refer to this table for the rest.

   Refer to this link to know more about the “Above” and “Below” parameters.

💡 **Exercise**:

Write the API call to find ValueSets that starts with this URL: http://hl7.org/fhir/ValueSet.

Ex. 10: “Count” Parameters

“Count” is used to both count matching resources *and* limit the returned resources to the given count number.

Count the number of matching resources:

https://hapi.fhir.org/baseR4/Patient?gender=female&_summary=count

1. “_summary=count” will display the total number of resources. In the example above, this will return the total number of female patients.

Limit the number of returned resources to the “Count” value:

https://hapi.fhir.org/baseR4/Patient?gender=female&_count=10

2. Only the first 10 matching resources will be returned from the server. We can set the value of count to any desired number.

💡 **Exercise**:

Write the API call to derive the number of matching patient resources whose family name begins with “C.”

Ex. 11: “Sort” Parameter

https://hapi.fhir.org/baseR4/Observation?_sort=+date

1. To sort your queries in a specific order, use the “_sort: ” parameter
    a. If you want your results to be in ascending order, use the plus sign “+”
        i. E.x: “+date”
    b. If you want your results to be in descending order, use the minus sign“-”
        i. E.x: “-date”
2. The “_sort” parameter is often used with “_count=10” to specify which 10 resources to display.
    a. E.x: https://hapi.fhir.org/baseR4/Observation?_sort:valueQuantity:unit=cm&_count=10

    Note: While querying resources sorted with respect to their date(using the sort parameter), the **“+” sign** is used to retrieve a result that is in ascending order and the **“-” sign** indicates a result that is in descending order.

💡 **Exercise**:

Write the API call to search for a bundle of practitioner resources sequenced in ascending order with respect to date.

Ex. 12: Sort by Relevance Parameter

https://hapi.fhir.org/baseR4/Observation?_score:=+date

There are very few instances where this parameter will be used. Although it’s good to practice.

1. To sort by relevance the “_score” parameter is used
2. Relevance could be subjective to what you want to achieve with your sort. Dates are a predominant area of relevance in this search parameter
3. In the code above, the resources will be displayed according to the ascending date

Ex. 13: “Last Updated” Parameter

https://hapi.fhir.org/baseR4/Observation?_lastUpdated=2021

1. To see when the data in the resource was last updated, we use the code “_lastUpdated”

2. The code above will show all entities whose files were last updated in 2021
3. It's a syntax rule that "_lastUpdated" always begins with an underscore. Check the summary table in FHIR to see which parameters start with an underscore

💡 **Exercise**:

Write the API call to search for a bundle of patient resources last updated after 2020.

Hint:

1. Use the *gt* parameter we studied before.

Ex. 14: "ID" Parameter

https://hapi.fhir.org/baseR4/Patient?_id=1660

1. "_id" allows you to search for any resource by the ID
2. This search is specific
   a. If you search for "_id=3," only "3" will be displayed; "30," "300," "33," etc... won't be displayed
3. Alternatively you can search by ID with a forward slash[6]
   a. E.x: "http://hapi.fhir.org/baseR4/Patient/1660"

Yellow note: the section highlighted in yellow is a placeholder value, but this can be changed to search for various ID numbers. To obtain a valid ID instead of 1660, run the following:

1. https://hapi.fhir.org/baseR4/Patient on Postman/Insomnia
2. Now search for the "entry" data element in the payload. Within "entry," you'll see multiple "resource" child elements. Choose a resource and grab the ID.
3. Now place that ID in the query instead of 1660 above.

💡 **Exercise**:

Write the API call to search for a patient observation resource whose ID is 2001.
Hint: Implement the above note.

[6] This is a resource URL.

Ex. 15: "dateTime" Parameters

https://hapi.fhir.org/baseR4/Patient?_lastUpdated:date=gt2020-10-30T00:00

1. The dateTime parameter is used to search resource information regarding date
   a. This could be anything from "birth date" to "death date"
2. The above code will display all patients whose information was last updated October 30, 2020, after midnight, until now
   a. The format is always in yyyy-mm-dd format
   b. The "T" is case sensitive for time, with the format being hh:ss
   c. This is the standard XML format

   Date is an acting modifier when combined with lastUpdated. That's why it begins with a colon. An alternative way to structure the query and get the same result (without the date syntax) is: https://hapi.fhir.org/baseR4/Patient?_lastUpdated=gt2020-10-30T00:00

Ex. 16: "Identifier Value" Parameter

https://hapi.fhir.org/baseR4/Patient?identifier=12345

1. The "identifier" parameter will display all entities, with an identifier value containing the given value. This parameter can be paired with any identifier subtopic (use, system, value).
   - The above code will display all entities with an identifier containing "12345"

Note: the section highlighted in yellow is a placeholder value, but can be changed to search for various ID numbers. To obtain a valid ID, instead of 12345, run the following:

1. https://hapi.fhir.org/baseR4/Patient on Postman/Insomnia
2. Now search for the "entry" data element in the payload. Within "entry," you will see multiple "resource" child elements. Choose a resource and grab the ID
3. Now place that "ID" in the query instead of 12345 above.

Ex. 17: "Active" Parameter

https://hapi.fhir.org/baseR4/Patient?active=true

1. The "active" parameter is a boolean used to display if a patient (or other resource) is still active
2. "Active=true" will display all entities that are still active

   a. The “active” code must always be followed by “=true” for still active
   b. The “active” code must always be followed by “=false” or “:not=true” for no longer active

💡 **Exercise**:

Write the API call to search for all the active practitioners.

Ex. 18: “Text” Parameter

https://hapi.fhir.org/baseR4/AllergyIntolerance?code:text=crocin

1. The “:text” parameter is used to search for any text belonging to a resource. It’s often used to search for allergies, reactions, conditions, etc…
2. The above code will display any entities who have a text relating to allergies
   a. “code:text=___” can be used when a condition code is unknown or unspecified
   b. If the condition code is known “code=__” can be used
3. “:text” can be used for any resource type if applicable (where there is text listed under the resource)

💡 **Exercise**:

a) Write the API call to search for a bundle of resources that contain text related to allergic reactions on the intake of crocin

b)Write the API call to search for the Observation resource that refers to a patient of id 1531 and loinc code of 8302-2

Ex. 19: Lab Report Parameters

https://hapi.fhir.org/baseR4/DiagnosticReport?subject:patient.name=peter

1. To perform a series of search operations that cover multiple reference parameters, you can "chain" the series of reference parameters by appending them to the server request one by one using a period
2. For lab reports, the resource type used is "DiagnosticReport"
   a. For a specific patient, the code "subject:patient.name=___" can be used
3. This parameter can be used for any diagnostics report from blood test to procedures
4. In order to save a client from performing a series of search operations, reference parameters may be "chained" by appending them with a **period (.)**
5. This request returns all the lab reports that have a subject which is a patient, whose name includes "Peter." Since the diagnostic report subject can be a set of different resources, it's necessary to limit the search to a particular type like how the query is formed above

https://hapi.fhir.org/baseR4/DiagnosticReport?subject.name=peter

1. Here's a slightly different but similar query. Given that the resource DiagnosticReport has a search parameter named subject—which is usually a reference to a patient resource—and the patient resource includes a parameter name which searches on patient name, this search is a request to return all the lab reports that have a subject whose name includes "Peter"

💡 **Exercise**:

a) A patient with an ID of 1985 had a diagnostic report. Find out his name and make another API call to find out what type of report it was.
b) Tune in for an ***advanced*** question. Write the API call to search for patients who are treated by a practitioner named "Lisa" or by a doctor located in New York.

Note a): 1985 is a placeholder ID that's not accurate. To populate an accurate ID, there are certain steps required to run. We'll need to do the front and backend work for this question, which means:

1. After having already posted the bundle before, run https://hapi.fhir.org/baseR4/Patient or https://hapi.fhir.org/baseR4/Patient?&_count=140
2. Search for "Chalmers" in the patient payload. Within that patient resource grab the ID "abc"
3. Now search for "DiagnosticReport" in the bundle. Within this resource, you'll find a child element "reference" under the "subject" element. Populate "patient/abc" as the value of the reference

element with the quotations. By doing this, we're referencing "Chalmers" within the diagnostic report resource for us to be able to reference later

4. Now, post the bundle again and create and run the query in question

Note b): chained parameters are applied independently to the target resource. For example, a chained query may return patients cared for by a doctor named Lisa and patients cared for by a doctor named Jane from New York. No one practitioner needs to satisfy both conditions

For better context on chained parameters refer to this link.

Ex. 20: "Search by Lists" Parameter

1. The "_list" parameter allows for the retrieval of resources that are referenced by a list resource. The list parameter always begins with an underscore. Check which other parameters begin with an underscore in FHIR's summary table. Refer to this link to know more about the list parameter or click here: https://hapi.fhir.org/baseR4/Patient?_list=5

2. This request above returns all patient resources that are referenced from the list found at [base]/List/5). Using a list as a search criterion allows for additional search criteria to be specified: https://hapi.fhir.org/baseR4/Patient?_list=42&gender=female

3. This request will return all female patients in the list

💡 **Exercise**:

A list with an ID of 1693 exists on the server. Find out how many male patient resources exist on this list.

Note: the section highlighted in yellow is a placeholder value, but this can be changed to search for various ID numbers. To obtain a valid ID instead of 1693, run the following:

1. https://hapi.fhir.org/baseR4/List on Postman or Insomnia.
2. Now search for the entry data element in the payload. Within "entry," you'll see multiple "resource" child elements
3. Now place that ID in the query instead of 1693 above

Ex. 21: “Include” Parameter

https://hapi.fhir.org/baseR4/MedicationRequest?_include=MedicationRequest:patient

1. “_include” is used to request resources related to search results (i.e. asking for the subject's details along with the medication request). The returned result would include the patient resource as well
2. The related resource that is requested to be included in the response should begin with a **colon “:”** (in this case “:patient”)
3. “_include” is often used when looking for medications and conditions
4. The “_include” parameter must always have an **underscore(_)** in front of it

Note: if the example query above doesn’t work, you might need to go into the bundle and change the patient reference to a valid patient resource ID that already exists on the server. Refer to the above sections for guidance to search for an applicable patient ID

💡 **Exercise**:

a) Write the API call to search for Peter’s patient resource that includes information about his referenced organization

b) Write the API call to search for all the observation resources related to height of a patient whose ID is 1531. Be sure to include the referenced patient resource. The loinc code of Height is 8302-2

c) Write the API call to search for all “MedicationRequest” resources including all referenced patient and practitioner resources

d) Write the API call for all the “MedicationRequest” resources including all referenced resources

Note: while structuring a query it is important to look at the “SearchParameters” section to understand what names and expressions to use when searching for a particular thing. For example, in the case of “MedicationRequest,” referring to this section will help define how we should structure the query to include practitioner resources. Check the list to see what syntax is defined to include “practitioner” resources in the “MedicationRequest” resource.

Ex. 22: Reverse Include Parameter

https://hapi.fhir.org/baseR4/Practitioner?_revinclude=MedicationRequest:Practitioner

1. If you want to find resources related to the parameter being searched, the code is "_revinclude"
    a. For example, if you're searching for a prescription and want to find resources related to the prescription, use: "_revinclude"
2. In the example above, the query is asking for the medication requests made by different practitioners. Since the practitioner resource doesn't include medication request as a reference within its resource, we use revinclude to search for such information
3. "_include" and "_revinclude" queries separate the source and target resources using a **colon ":"**
4. Refer to YouTube for more information

💡 **Exercise**:

a) Write the API call to search for all the patient resources for "Peter" and the referenced allergy intolerance resources
b) Write the API call to search for all the referenced resources of a patient whose ID is 1531.

Note: the section highlighted in yellow is a placeholder value, but this can be changed to search for various ID numbers. To obtain a valid id instead of 1531, run the following:

4. https://hapi.fhir.org/baseR4/Patient on Postman or Insomnia
5. Now search for the entry data element in the payload. Within "entry," you'll see multiple "resource" child elements
6. Now place that "ID" in the query instead of 1531 above

Ex. 23: "Quantity" Parameter

https://hapi.fhir.org/baseR4/Observation?value-quantity=6.3

1. To search for the quantity of a resource, the parameter "quantity" is used
    a. The amount and unit are always separated by **the pipe "|"** if it's provided
2. This parameter is often used for medication, observation and dosage queries

3. Here we are searching for observation resources where the quantity 6.3 is used

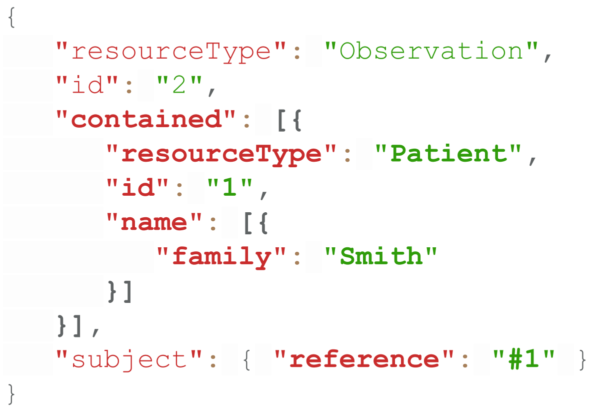
💡 **Exercise**:

Run a query to search for an observation resource of a patient whose glucose quantity is 6.3 moles.

Ex. 24: “Contained Resource” Parameter

https://hapi.fhir.org/baseR4/Observation?subject.name=Smith&_contained=true

1. By default, search results only include resources that are not contained in other resources
2. This behavior can be changed by adding a “_contained**”** query parameter
3. For example, to find observations where the patient’s name is “Smith,” we can write a query as `/Observation?subject.name=Smith` which returns observations where subject references non-contained (separate) patient resource with name “Smith”
4. But this will not return the following observation where the patient resource is contained, even though the subject’s name matches the query



```
{
   "resourceType": "Observation",
   "id": "2",
   "contained": [{
      "resourceType": "Patient",
      "id": "1",
      "name": [{
         "family": "Smith"
      }]
   }],
   "subject": { "reference": "#1" }
}
```

Figure 12: Observation payload example

To find such resources, **“_contained=true”** should be passed to the URL

Note: Smile CDR has “index_contained_resources” configuration disabled by default. It must be enabled in order to use the “ _contained” parameter.

💡 **Exercise**:

Find conditions resources where asserters's last name is "Cole" (result should return resources where asserter is referencing to contained resources)

Note: the configuration could be disabled, but try anyway.

Ex. 25: "Summary" Parameter

https://hapi.fhir.org/baseR4/ValueSet?_summary=data

1. The "_summary" parameter will return a portion of the resource. There are five summary values:
    a. "_summary=data" removes the text elements from the resource
    b. "_summary=text" displays only "text," "metadata," "id" and top-level mandatory elements
    c. "_summary=true" displays a limited subset of elements from the resource
    d. "_summary=false" returns the whole resource
    e. "_summary=counts" displays the number of resources, not the resources themselves
2. When "_include" and "_revinclude" are searched with the "_summary=text" parameter and error code will be displayed

💡 **Exercise**:

Use a query above to return the text content of the ValueSet resources

Ex. 26: "Elements" Parameter

https://hapi.fhir.org/baseR4/Patient?_elements=identifier,active

1. The "_elements" parameter is used when only specific elements are desired to be displayed as part of the resources
    a. The values should always be separated by a **","**
2. The "_elements" parameter is used when the "_summary" parameter is not appropriate

💡 **Exercise**:

Write a query that returns only “addresses” for all practitioner resources.

Ex. 27: “Missing” Parameter

https://hapi.fhir.org/baseR4/Patient?gender:missing=true

1. If an element is missing, the parameter “:missing” can be used to pull up records of the missing element

Note: this parameter won’t work on the hapi.fhir.org server

Ex. 28: “History” Parameter and vRead

Every time a resource is updated, the old data is retained as history. The original version of a resource is called version 1. An incremental version number/VersionID is given to each update. When searching or retrieving a resource, unless you specify otherwise, the server will always return the most recent version. However, if you wish to retrieve a historical version you can do so using the “_history” parameter.

https://hapi.fhir.org/baseR4/Observation/obs114/_history

1. To see the history of a resource we use the parameter “_history”
    a. To see the resource history you need the exact resource ID
2. The “_history” parameter displays all past and current versions of a resource. including the verb used (POST, PUT, etc...) and the creation date of each version

https://hapi.fhir.org/baseR4/Observation/obs114/_history/2

3. This will return version 2 of the given observation resource
4. This is known in FHIR as “vRead” (short for read a given version)

The two queries above might not work since an observation with an ID of “obs114” might not exist on the server. You can replace the ID with the steps mentioned ahead in green to successfully retrieve some information.

💡 **Exercise**:

Write a query that returns the recent and historical versions of an observation resource of ID " o123"

Note: the section highlighted in yellow is a placeholder value, but can be changed to search for various ID numbers. To obtain a valid id instead of "o123," run the following:

1. https://hapi.fhir.org/baseR4/Observation on Postman or Insomnia
2. Now search for the entry data element in the payload. Within "entry," you'll see multiple "resource" child elements. Choose a resource and grab the ID as displayed in Figure 23 below. Figure 23 is just an example, don't use "o123." You will need to manually search for an accurate ID on the server
3. Now place that "ID" in the query instead of "o123" above

Ex. 29: "Code" Parameter within Resource

There's an added capability to search for a specific resource using specific information. Using "code" makes it easier to search for only those specific resources that have included within it that code.

For example, in the following query we're using "code" to search for only those allergy intolerance resources that contain "387207008" as a code. "387207008" is the code for crocin. Our goal is to find all those resources that claim crocin as an allergy.

https://hapi.fhir.org/baseR4/AllergyIntolerance?code=387207008

Try replacing "1687" with the ID you used in the step guided by Figure 26 in the following query. Search for "code" in the response and get a feel of the different clinical codes existent.

https://hapi.fhir.org/baseR4/Observation?patient=1687

Here's an example of finding all the observations for a patient that has a LOINC code of 1234-5. Sometimes servers don't have all search parameters configured to search for information this way, so a query like this will not always result in any responses. But this can be an example of how you would combine the code and the system in a query.

Note: the pipe (|) in this case is used to separate the system with the code because it is a token type of search parameter. Tokens utilize the pipe (|) for its syntax.

https://hapi.fhir.org/baseR4/Observation?code=http://loinc.org|1234-5

💡 **Exercise**:

a) Write a query that searches for 30 observation resources whose patient ID is “1162698.” For this exercise, use http://hapi.fhir.org/baseR4 instead of https://hapi.fhir.org/baseR4.

b) Now write a query that searches for observation resources whose patient ID is “1162698” **and code is “8302-2**.” Use http://hapi.fhir.org/baseR4 instead of https://hapi.fhir.org/baseR4.
Search through the response to see what code “8302-2” represents.

Note: Sometimes servers are configured in a certain way that restricts the results to a particular query. The http://hapi.fhir.org/baseR4 server is configured to only return 20 resources at a time. So it’s important to use the “_count” to request 50 so we can see all up to 50 resources.

Ex. 30: The “_has” Operation

a) As the name suggests, this operation is used to query resources based on the properties of resources that refer to them

b) This is called “reverse chaining"

https://hapi.fhir.org/baseR4/Patient?_has:Observation:patient:code=1234-5

c) Here’s an example of the “_has” parameter above. This requests the server to return patient resources, where the patient resource is referred to by at least one observation and where that observation has a code of “1234-5"

In other words, this query is pulling all the patient resources that were referenced in the observation resources that contain code “1234-5”

Note: it’s syntax to chain it like (observation:patient:code). The base resource (i.e patient) should be in the center. But the following value in this case code should be a search parameter of observation—for a complete list, click here.

💡 Exercise:

a) Write a query that searches for the organizations who have patients living in Kingston, Ontario

b) Write a query that searches for the patients who have observations that contain either “15074-8” or “718-7” as a code

c) Write a query that searches for the specific patient that has an observation identifier of “6327” and status as final

Note: the section highlighted in yellow is a placeholder value, but can be changed to search for various ID numbers. To obtain a valid ID instead of “1541/1544,” run the following:

1. https://hapi.fhir.org/baseR4/Patient on Postman or Insomnia.
2. Now search for the entry data element in the payload. Within “entry,” you’ll see multiple “resource” child elements. Choose a resource and grab the ID as displayed in Figure 23 below. Don’t use “1541/1544,” you’ll need to manually search for an accurate ID on the server

   Now place that “ID” in the query instead of 1660 above

Ex. 31: FHIRPath

a) FHIRPath is an expression language defined by FHIR. It typically takes the form of a single dotted path. For example: Patient.name.given

b) If this path above is used as the expression in a custom search parameter for a particular patient like the one below, this is how the query will be formed:

https://hapi.fhir.org/baseR4/Patient/1541?_fhirpath=Patient.name.given

c) This query will return the given name of the patient whose ID is “1541”

Refer to this link and this link to learn more about FHIRPath.

💡 **Exercise**:

a) Write a query using FHIRPath to search for the gender of the patient whose ID is “1541”

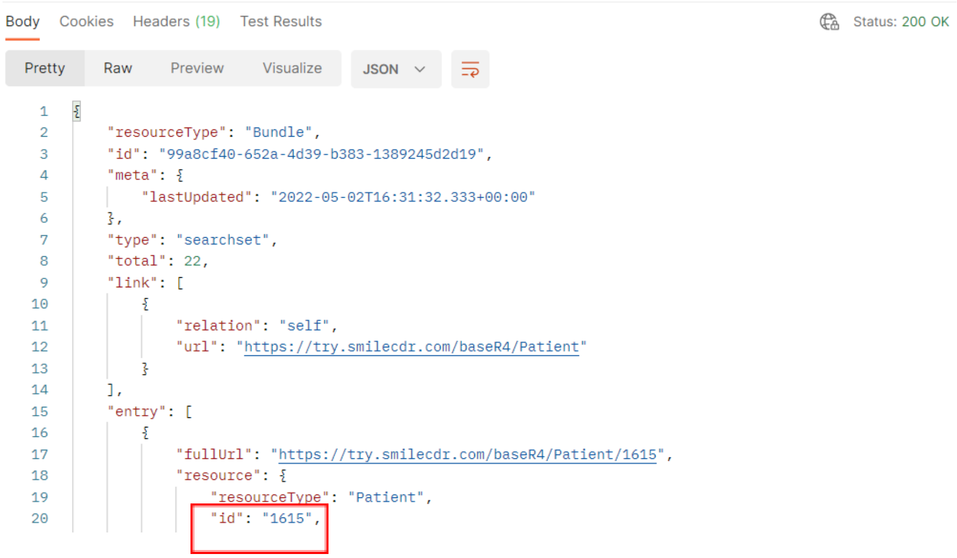
b) Write a query using FHIRPath that searches for the value system of the observation resource whose ID is “1544”

c) Write a query using FHIRPath that returns a true or false if the patient whose ID is “1541” contains information about their email ID

Note: the section highlighted in yellow is a placeholder value, but can be changed to search for various ID numbers. To obtain a valid ID instead of “1541/1544,” run the following:

1. https://hapi.fhir.org/baseR4/Patient on Postman or Insomnia
2. Now search for the entry data element in the payload. Within “entry,” you’ll see multiple “resource” child elements. Choose a resource and grab the ID as displayed in Figure 23 below. Don’t use “1615,” you’ll need to manually search for an accurate ID on the server

3. Now place that “ID” in the query instead of “1660” above



```
{
    "resourceType": "Bundle",
    "id": "99a8cf40-652a-4d39-b383-1389245d2d19",
    "meta": {
        "lastUpdated": "2022-05-02T16:31:32.333+00:00"
    },
    "type": "searchset",
    "total": 22,
    "link": [
        {
            "relation": "self",
            "url": "https://try.smilecdr.com/baseR4/Patient"
        }
    ],
    "entry": [
        {
            "fullUrl": "https://try.smilecdr.com/baseR4/Patient/1615",
            "resource": {
                "resourceType": "Patient",
                "id": "1615",
```

Figure 23: Searching for the ID

Ex. 32: The “Everything” Command

1. This operation is used to return all the information related to one or more patients described in the resource or context on which this operation is invoked

2. The response is a bundle of type "searchset"

3. At a minimum, the patient resource(s) is returned, along with any other resources that the server has that are related to the patient(s) and available for the given user. The server also returns whatever resources are needed to support the records (e.g. linked practitioners, medications, locations, organizations, etc...)

http://hapi.fhir.org/baseR4/Patient/1405775/$everything

4. The query above will pull the patient resource whose ID is “1405775.” It’ll also return other referenced resources related to that patient

Refer to this link to know more about this command

Creating and Updating a Resource:

Tools required for this section:

- Postman/Insomnia or similar application.

Resource Types Explained:

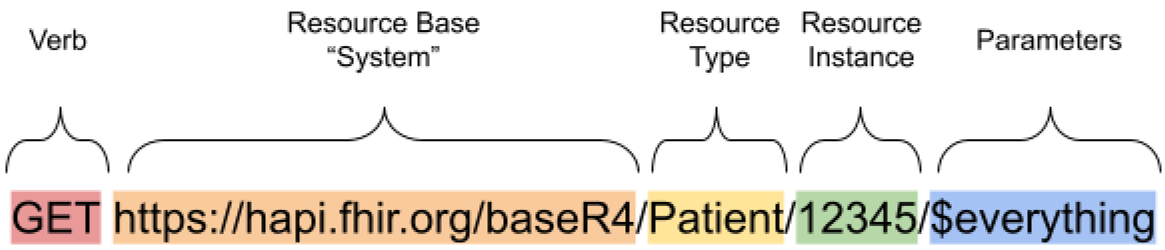
FHIR resources are the data packages used to send or receive data from a FHIR repository. The HL7 organization wants enough resources to handle 80% of the healthcare workflows, but doesn't want so many that the standard would become too complex. Resources are meant to capture a meaningful amount of data. Things like patient demographic data or allergies provide the perfect amount of data. For those familiar with HL7 V2 messaging, a resource would typically contain about the same amount of data as a segment in a HL7 V2 message. Click here for a full list of resources.

There are many resource types that are used in RESTful FHIR. This document will cover a few of the common ones that often result in confusion:

1. "Code-systems" describe an enumerated list of values. For example, the FHIR code system for gender is {male, female, other, unknown}. Some code systems are quite large such as LOINC (list of laboratory observations) or SNOMED (medical terms and symptoms)
2. "Value set" specifies a set of codes from one or more code systems
3. When sharing medical information, one of the common challenges is ensuring everybody is speaking the same lingo. "Terminology" is the umbrella term for code systems, value sets, codes, etc.... This is often one of the trickier concepts for beginners. Some good resources on this topic are:
    a. https://build.fhir.org/terminologies.html
    b. Terminology Services on FHIR (Rob Hausman)
4. "Group" defines a collection of resources which can be addressed all together by their group
5. "Operation outcome" is the query status returned from the server. You'd like to see "success," but when there's an error, the operation outcome can provide helpful insights

Creating Your First Resource:

As you may recall, searching or reading a resource is done with the GET verb:

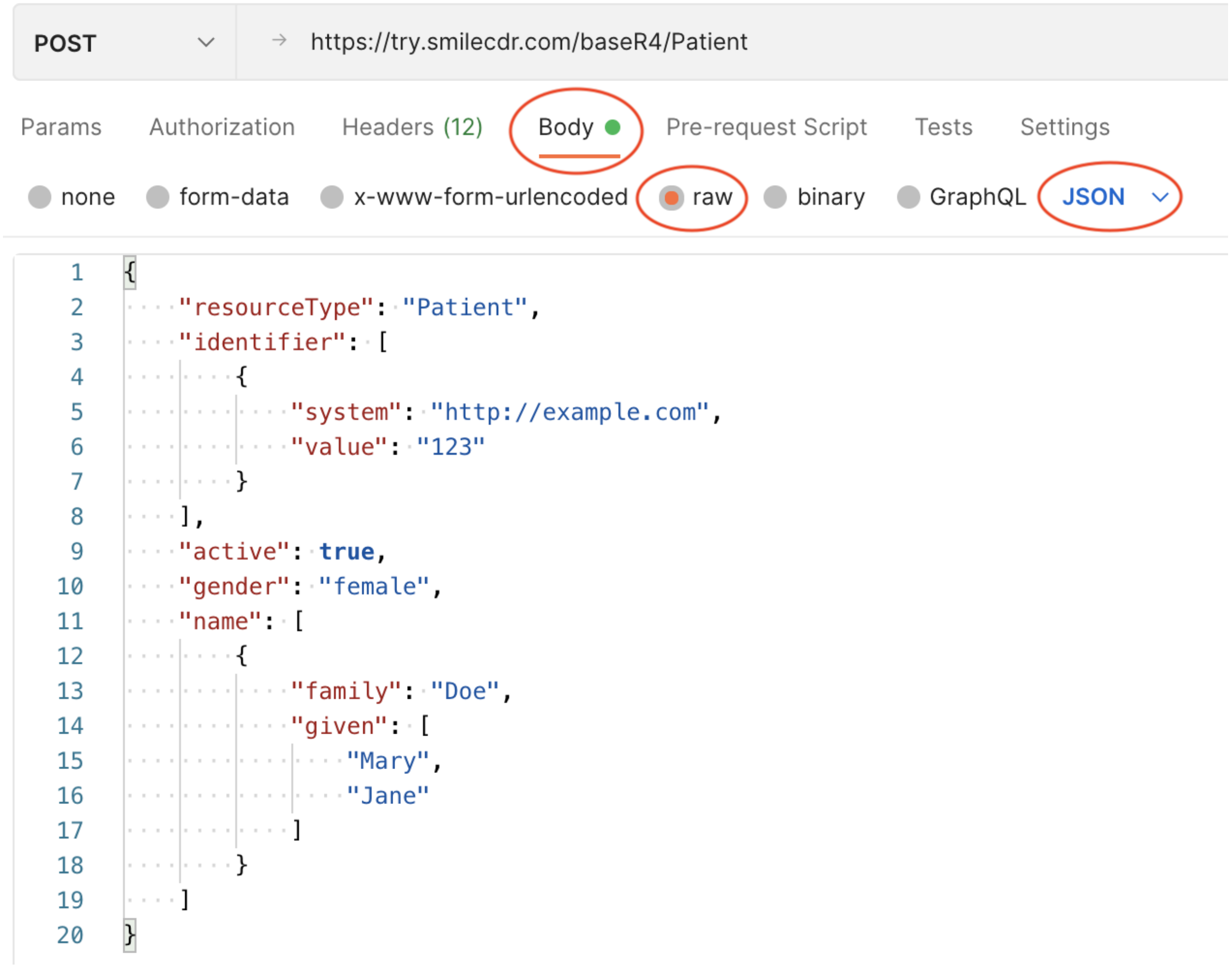


Creating a resource is performed with the POST verb:

[Verb] [Server][API Base][Resource Type]

[POST] [https://hapi.fhir.org/baseR4]/[Patient]

1. We don't specify the resource ID; rather we let the server auto-assign the ID
2. The resource contents are detailed in the resource body
    a. To edit the resource contents in Postman, **select** "*Body*," "*raw*" and "*JSON*" just below the search bar. These selections allow you to edit the data as you would in any other editor.
    b. Once you're ready to commit your resource to the repository, **press** *send*. If there are no errors, the resource is successful and will be returned from the server including the server assigned ID. (Errors with details will also be returned if the server could not commit the resource to the repository.)



POST https://try.smilecdr.com/baseR4/Patient

Body: raw, JSON

```
{
    "resourceType": "Patient",
    "identifier": [
        {
            "system": "http://example.com",
            "value": "123"
        }
    ],
    "active": true,
    "gender": "female",
    "name": [
        {
            "family": "Doe",
            "given": [
                "Mary",
                "Jane"
            ]
        }
    ]
}
```

Figure 13a : POST Body

```
{
   "resourceType": "Patient",
   "active": true,
   "gender": "female",
   "name": [
       {
           "family": "Doe",
           "given": [
               "Mary",
               "Jane"
           ]
       }
   ]
}
```

Figure 13b: POST result

3. To create a resource you use the **POST** verb and "*$validate*"

    a. Validate isn't needed when creating a resource but strongly recommended because it'll highlight any issues with the code without committing the resources to the repository (Figure 4)

    b. For example, the exercise for this playbook depends on a bundle of resources fed into the server. To make that bundle, we used "$validate" to ensure that the bundle is free from errors.

    In the case of a patient resource, we can send an empty patient resource to the server and there won't be an error. But we can still validate it to gain tips of improvement for the resource. For that purpose, we'll first POST the resource, then send a GET request with "$validate" at the end to assess any errors or areas of improvement . The two queries would be:

    POST https://hapi.fhir.org/baseR4/Patient

    GET https://hapi.fhir.org/baseR4/Patient/1664/$validate

Ex. 32: Updating a Resource

Updating a resource can be a little tricky. There are two different ways to update a resource. There's PATCH and PUT. PATCH updates the one element you wish to change[7], for example, it only changes the patient's gender from "female" to "male." Whereas PUT updates the resource as a whole, for example updating the entire patient resource.

There are 4 steps for PATCH:

[Verb] [Server][API Base][Resource Type][ID]

[PATCH] [https://hapi.fhir.org/baseR4/Patient/1833487]

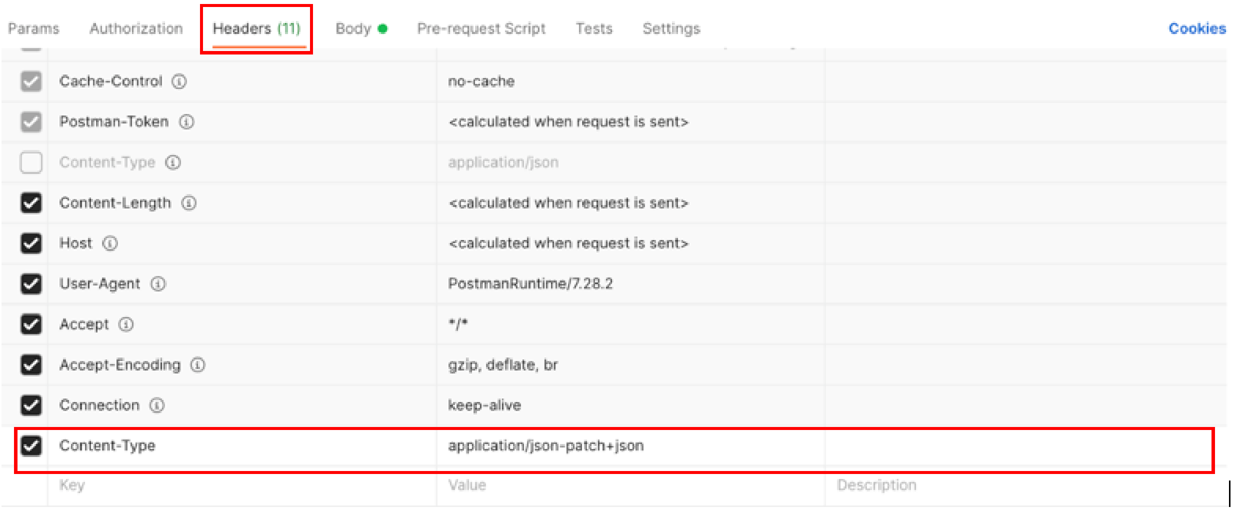
1. For a PATCH operation, tweak the verb to Patch, paste the above query in the search bar and be sure to use an appropriate patient ID. "1833487" is just a placeholder.

2. You'll then need to change the header from "*content-type=application/json"* to "*content-type=application/json-patch+json*."[8] The other headers in the diagram below are not necessary for the patch.

[7] This document will only cover PATCH.

[8] This may not be needed for different servers of mime types.

3. This may not be needed for different servers.



	Key	Value	Description
☑	Cache-Control	no-cache	
☑	Postman-Token	<calculated when request is sent>	
☐	Content-Type	application/json	
☑	Content-Length	<calculated when request is sent>	
☑	Host	<calculated when request is sent>	
☑	User-Agent	PostmanRuntime/7.28.2	
☑	Accept	*/*	
☑	Accept-Encoding	gzip, deflate, br	
☑	Connection	keep-alive	
☑	Content-Type	application/json-patch+json	

Figure 14: PATCH headers

4. Let's say we want to change the patient's gender to male. The search bar should have the full resource URL, including the patient ID. Then **paste** the code below in the body section of the request:

FYI: **op** stands for the operation you'd like to run. In this case it's "replace." Alternatively, the operation could be "add." This link specifies the different operations.

```
[
{
"op": "replace",
"path": "/gender",
"value":      "male"
}
]
```

Figure 15: Gender change, PATCH

5. It's important to note that whenever "PATCH" is being used, the "path" value always follows a forward slash "/"

    a. If there are multiple paths, they're also separated by a forward slash (ex: "/name/family")

6. To **add** a patient's address, we'd use the code:

```
[
    {
   "op": "add",
   "path": "/address",
    "value": [
              {
                   "use": "home",
                   "text": "55 Happy st",
                   "line": ["55 Happy st"],
                   "city": "Boring",
                   "state": "Oregon",
                   "country": "U.S.A"
              }
         ]
    }
]
```

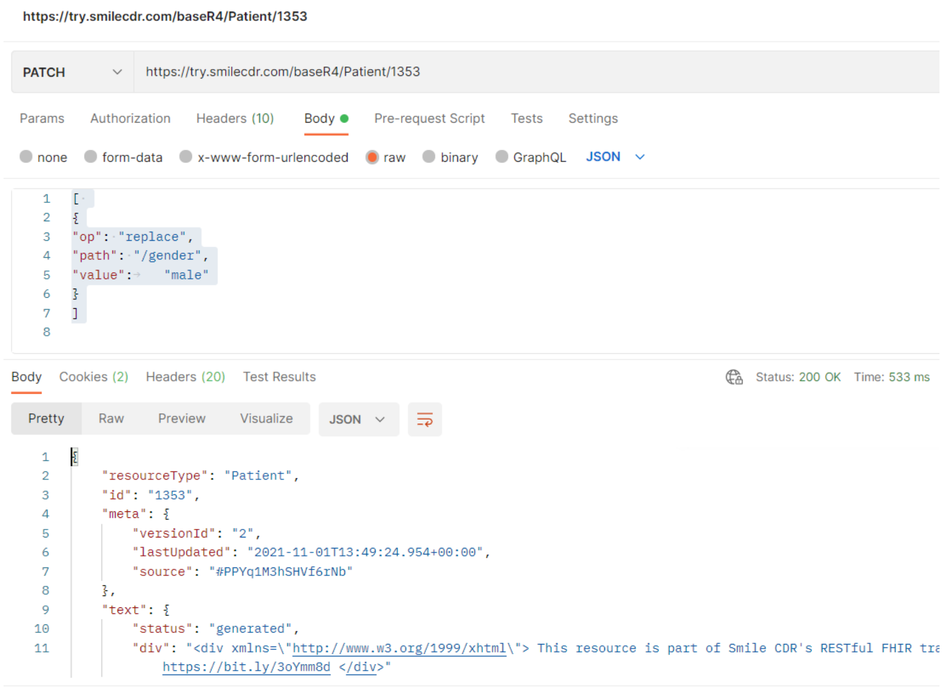
Figure 16: Add patient address, PATCH

7. To replace the patient's country from "America" to "U.S.A". you'd use the code:

```
[
   {
        "op": "replace",
        "path": "/address/0/country",
        "value":   "U.S.A"
   }
]
```

Figure 17: Replace country, PATCH

   a. The "0" symbolizes the first address. Because a patient can have multiple addresses, they will be numbered beginning with "0." This applies to any resource with multiple options. Figure 22 is a display of the working of the PATCH request.



https://try.smilecdr.com/baseR4/Patient/1353

PATCH https://try.smilecdr.com/baseR4/Patient/1353

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
[
{
"op": "replace",
"path": "/gender",
"value":    "male"
}
]
```

Status: 200 OK Time: 533 ms

JSON

```
{
    "resourceType": "Patient",
    "id": "1353",
    "meta": {
        "versionId": "2",
        "lastUpdated": "2021-11-01T13:49:24.954+00:00",
        "source": "#PPYq1M3hSHVf6rNb"
    },
    "text": {
        "status": "generated",
        "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"> This resource is part of Smile CDR's RESTful FHIR tra
            https://bit.ly/3oYmm8d </div>"
```

Figure 22: Patch example

💡 **Exercise**:

> Write a query that replaces patient Danisha Syed's last name to SyedAli.

Note: To find the accurate ID for patient Danisha Syed, there are certain steps required.

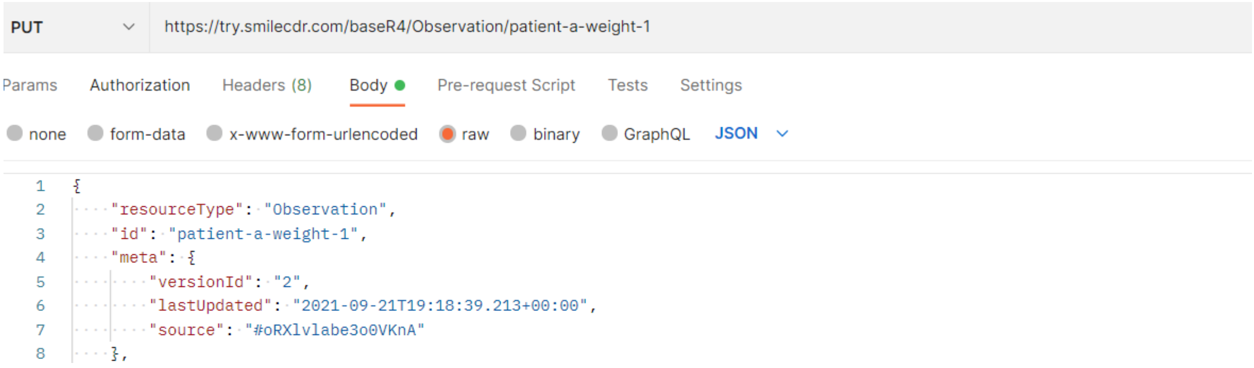
1. After having already posted the bundle before, go through the result body and search for the ID created for Danisha.
2. Use that ID for the patch request.

Here's an example of "PUT"

[Verb] [Server][API Base][Resource Type][ID]

[PUT][https://hapi.fhir.org/baseR4/Observation/patient-a-weight-1]

3. You begin by **changing** the verb to *PUT*
4. **Paste** the observation resource in the body that you retrieved using *GET*. **Change** the body headers to *raw* and *JSON*

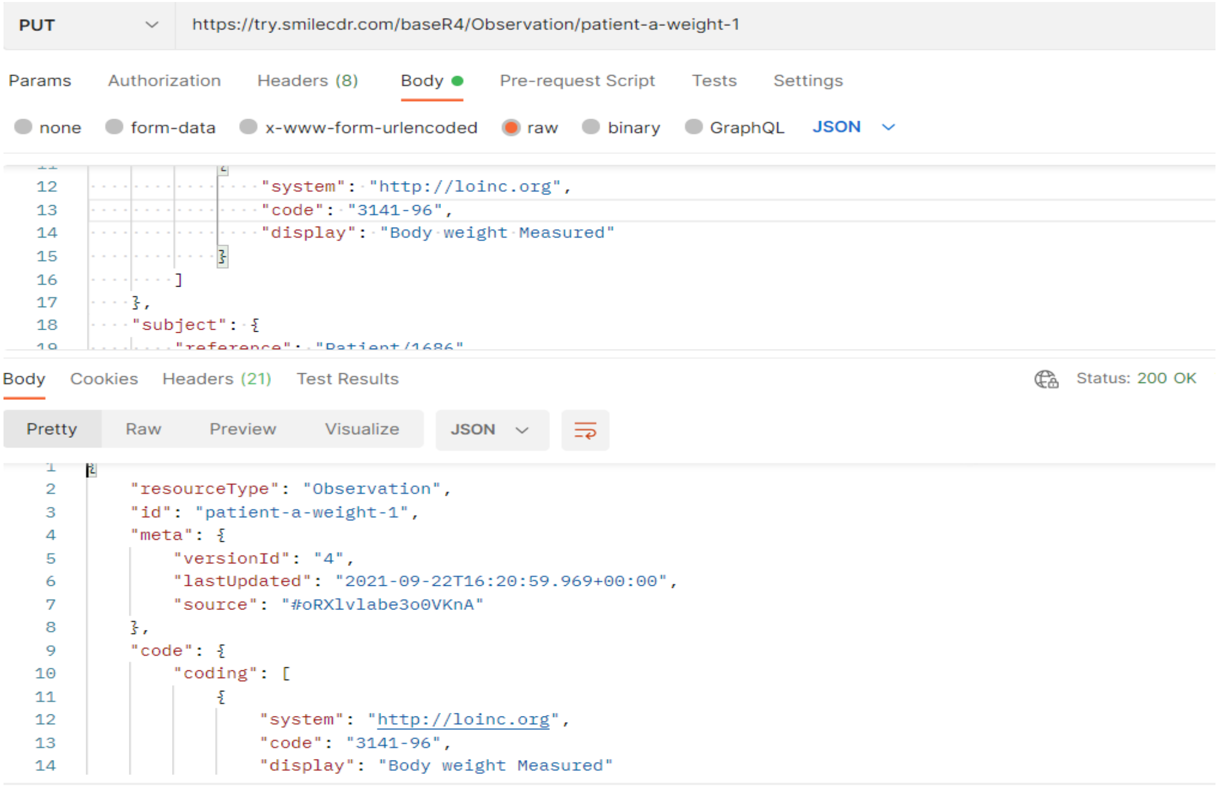


PUT https://try.smilecdr.com/baseR4/Observation/patient-a-weight-1

```
{
    "resourceType": "Observation",
    "id": "patient-a-weight-1",
    "meta": {
        "versionId": "2",
        "lastUpdated": "2021-09-21T19:18:39.213+00:00",
        "source": "#oRXlvlabe3o0VKnA"
    },
```

Figure 18a: PUT Body

5. Let's say we want to change the system.code of the "observation" to "3141-96," originally being "3141-95." Editing the code in the body and executing the "PUT" query by **clicking** *send* will make the change.



PUT https://try.smilecdr.com/baseR4/Observation/patient-a-weight-1

```
                "system": "http://loinc.org",
                "code": "3141-96",
                "display": "Body weight Measured"
            }
        ]
    },
    "subject": {
        "reference": "Patient/1686"
```

Status: 200 OK

```
{
    "resourceType": "Observation",
    "id": "patient-a-weight-1",
    "meta": {
        "versionId": "4",
        "lastUpdated": "2021-09-22T16:20:59.969+00:00",
        "source": "#oRXlvlabe3o0VKnA"
    },
    "code": {
        "coding": [
            {
                "system": "http://loinc.org",
                "code": "3141-96",
                "display": "Body weight Measured"
```

Figure 18b: PUT result

Updating an "Observation:"

If you want to update a patient resource to display a heart rate, body weight, pregnancy, history with smoking, etc…, we'd use the observation resource type.

[Verb] [Server][API Base][Resource Type][Optional Parameter]

[PATCH] [https://hapi.fhir.org/baseR4/Observation/$validate]

- Let's say we want to display that patient 1699 has a heart rate of 64 BPM. We'd use the code:

```
{
   "resourceType": "Observation",
   "status": "final",
   "valueQuantity":
   {
       "value": 64,
       "unit": "beats/minute",
       "system": "http://unitsofmeasure.org",
       "code": "/min"
   },
   "code": {
   "coding": [
           {
               "system": "http://loinc.org",
               "code": "8867-4",
               "display": "heart rate (beats/minute)"
           }
       ]
   },
   "subject": {
       "reference": "Patient/1699"
   }
}
```

Figure 19: Updating an Observation, PATCH

1. This creates an observation that now points to patient 1699
2. When creating an observation, always make sure the units and code are exact and make sure status is correct
    a. Status is always required for observation resource
3. If the code is successful the observation will be given an "ID"

Creating and Updating a Practitioner:

1. Creating it with a post:
   If you want to add a practitioner resource and link the resource to a patient, perform the following steps. This process has quite a large learning curve and is very tedious. We start by using the practitioner resource type:

   [Verb] [Server][API Base][Resource Type][Optional Parameter]

   [POST] [https://hapi.fhir.org/baseR4/Practitioner/$validate]

a. We use the “POST” verb and the “Practitioner” resource type
    i. It's optional to use the “$validate” parameter, but it’s recommended to do so
b. Let's say we want to create a resource for Dr. Curious F. George. We’d use the code below:

```
{
    "resourceType": "Practitioner",
    "identifier": [
        {
            "system": "http://www.acme.org/practitioners"
        }
    ],
    "active": true,
    "name": [
        {
            "family": "George",
            "given": [
                "Curious",
                "Fifi"
            ],
            "prefix": [
                "Dr."
            ]
        }
    ],
    "address": [
        {
            "use": "billing",
            "line": ["10 Times Square ave"],
            "city": "New York",
            "state": "New York",
            "country": "U.S.A",
            "postalCode": "9053",
            "text": "10 Times Square ave"
        }
    ],
    "qualification": [
        {
            "identifier": [
                {
                                                                            "system":
    "https://Interestingonlineuniversity.com/BachelorofScience",
                        "value": "8752947295"
                }
            ],
            "code": {
                "coding": [
                    {
                                                                            "system":
    "http://terminology.hl7.org/CodeSystem/v2-0360/2.7",
                            "code": "BS",
                            "display": "Bachelor of Science"
                    }
                ],
                "text": "Bachelor of Science"
            },
            "period": {
                "start": "2008"
            },
            "issuer": {
                "display": "Interesting Online University"
            }
        }
    ]
}
```

Figure 20: Validate resource

c. An important point to note is when imputing “active” is that the response must always be true or false
   i. When coding the value for “active” must be a boolean
d. If the code is successful the practitioner will be given an “ID”

2. Link the practitioner to a patient resource:

[Verb] [Server][API Base][Resource Type][Id]

[PATCH] [https://hapi.fhir.org/baseR4/Patient/1701]

a. We use the “PATCH” verb, the “Patient” resource type and the patient ID.
b. Since we’re using “PATCH,” it's important to change the header from “Content-Type” to

“application/json-patch+json” (Figure 5).

c. We use the code below to link the “practitioner” to the “patient:”

```
[
   {
       "op": "add",
       "path": "/generalPractitioner",
       "value": [
           {
               "reference": "Practitioner/1715",
               "type": "Practitioner",
               "display": "Dr. Curious George"
           }
       ]
                                        }
                                      ]
```

Figure 21: Link the “Practitioner” to the “Patient

d. If a patient’s seen multiple practitioners, don’t forget to have the dates that the patient has been seeing/saw the practitioner.

Challenge Yourself

A great way to understand RESTful FHIR is to challenge what you have learned. Here are some simple starter challenges:

1. Create three patients
2. Create three practitioners and link them to patients
3. Create two observations
4. Make four updates to a patient resource

Exercise Sample Answers

Ex.1. Your First Query
https://hapi.fhir.org/baseR4/Organization
https://hapi.fhir.org/baseR4/Organization/1502
https://hapi.fhir.org/baseR4/Organization?_id=1502

Ex.2. A Single Parameter
https://hapi.fhir.org/baseR4/Organization?name=Peacehealth
https://hapi.fhir.org/baseR4/Patient?family=s

Ex.3. Exact
https://hapi.fhir.org/baseR4/Patient?family:exact=Gold

Ex.4. Contains
https://hapi.fhir.org/baseR4/Patient?family:contains=robin

Ex.5. Two or More Parameters—And
https://hapi.fhir.org/baseR4/Practitioner?active=true&address-state=wa

Ex.6. Not Parameter
https://hapi.fhir.org/baseR4/Patient?given=D&gender:not=male

Ex.7. Or Parameter
https://hapi.fhir.org/baseR4/Practitioner?family=R&address-state=AK,WA

Ex.8. Equality Operators and Number/Date Precision

https://hapi.fhir.org/baseR4/Patient?death-date=gt2013-08-20

Ex.9. Above and Below Parameter

https://hapi.fhir.org/baseR4/ValueSet?url:below=http://hl7.org/fhir/ValueSet

Ex.10. Summary and Count Parameter

https://hapi.fhir.org/baseR4/Patient?family=C&_summary=count

Ex.11. Sort Parameter
https://hapi.fhir.org/baseR4/Practitioner?_sort:=+date

Ex. 13. Last Updated Parameter
https://hapi.fhir.org/baseR4/Patient?_lastUpdated=gt2020

Ex. 14. ID Parameter
https://hapi.fhir.org/baseR4/Observation/2001

Ex. 17. Active Parameter
https://hapi.fhir.org/baseR4/Practitioner?active=true

Ex. 18. Text Parameter
https://hapi.fhir.org/baseR4/AllergyIntolerance?code:text=crocin
https://hapi.fhir.org/baseR4/Observation?patient=Patient/1531&code=http://loinc.org|8302-2

Ex. 19. Lab Report Parameter
https://hapi.fhir.org/baseR4/Patient/1985
https://hapi.fhir.org/baseR4/DiagnosticReport?subject:patient.name=chalmers
https://hapi.fhir.org/baseR4/Patient?general-practitioner:Practitioner.name=Lisa&general-practitioner:Practitioner.address-state=NY

Ex. 20. List Parameter
https://hapi.fhir.org/baseR4/Patient?_list=1693&gender=male

Ex. 21. Include Parameter
https://hapi.fhir.org/baseR4/Patient?name=Pieter&_include=Patient:organization

https://hapi.fhir.org/baseR4/Observation?patient=1531&code=8302-2&_include=Observation:patient
https://hapi.fhir.org/baseR4/MedicationRequest?&_include=MedicationRequest:patient&_include=MedicationRequest:requestor
https://hapi.fhir.org/baseR4/MedicationRequest?_id=1759&_include=*

Ex 22. RevInclude Parameter
https://hapi.fhir.org/baseR4/Patient?name=Peter&_revinclude=AllergyIntolerance:patient

https://hapi.fhir.org/baseR4/Patient?_id=1531&_revinclude=*

Ex. 23. Quantity Parameter
https://hapi.fhir.org/baseR4/Observation?value-quantity=6.3|mmol/l

Ex. 24. Contained Resource Parameter
https://hapi.fhir.org/baseR4/Condition?subject.name=Cole&_contained=true

Ex. 25. Summary Parameter

https://hapi.fhir.org/baseR4/ValueSet?_summary=text

Ex. 26. Elements Parameter
https://hapi.fhir.org/baseR4/Practitioner?_elements=address

Ex. 28. History
https://hapi.fhir.org/baseR4/Observation/o123/_history

Ex. 29. Code
http://hapi.fhir.org/baseR4/Observation?patient=1162698&_count=50
http://hapi.fhir.org/baseR4/Observation?patient=1162698&code=8302-2

Ex. 30. Has

https://hapi.fhir.org/baseR4/Organization?_has:Patient:organization:address-city=Amsterdam
https://hapi.fhir.org/baseR4/Patient?_has:Observation:patient:code=15074-8,718-7
https://hapi.fhir.org/baseR4/Patient?_has:Observation:patient:identifier=6327&_has:Observation:patient:status=final

Ex.31 FHIRPath

https://hapi.fhir.org/baseR4/Patient/1541?_fhirpath=Patient.gender
https://hapi.fhir.org/baseR4/Observation/1544?_fhirpath=Observation.value.ofType(Quantity).system
https://hapi.fhir.org/baseR4/Patient/1541?_fhirpath=Patient.telecom.where(system='email').exists()

Ex.32. PATCH

https://hapi.fhir.org/baseR4/Patient/135

```
[
  {
"op": "replace",
"path": "/name",
 "value": [
{

          "use": "official",
          "text": "Danisha Syeda",
          "family": "Syeda",
          "given": [
            "Danisha"
          ]
        }
    ]
  }
]
```

Glossary

API: Application Programming Interface. A software middleman that allows multiple applications to communicate (Mulesoft).

Bundle: a collection of resources. A common use of bundles is grouping a patient and their related resources together (FHIR).

Chained and Reverse Chained Parameters:
Think of chained (and reversed chained) parameters as similar to SQL join operations.

In order to save a client from performing a series of search operations, reference parameters may be "chained" by appending them with a period (.) followed by the name of a search parameter defined for the target resource. This can be done recursively, following a logical path through a graph of related resources, separated by a period.

Crud: Create, Read, Update and Delete. These are the four primitive actions you can apply to data in a repository.

Formats: REST most often represents data in the JSON format. RESTful FHIR principally represents data in either JSON or XML. (If you're not familiar with either of these, then go with JSON—it has become the more popular choice.)

HTTP: HyperText Transfer Protocol. The basis of online data exchange that results in the gathering of resources (retrieving a YouTube video, Mozilla, etc...).

ID: normally a sequential number. Databases are likely to assign similar IDs that can result in a collision.

Identifier: in coding, an identifier is something that is unique to a resource and follows it wherever it goes. A phone number is a good example of an identifier and, of course, the resource in this example would be the person.

Mime Type: Multipurpose Internet Mail Extension (MIME). Indicates the format of a document file or assortment of bytes. Common MIME types in RESTful FHIR are xml, turtle and JSON (Mozilla).

PHI: Personal Health Information (Canada) or Protected Health Information (US) (Lutevich et al.).

Resource: any online data (web-page, document, etc…) (Tutorials Point).

URI: Uniform Resource Identifier. A sequence of variables that are unique to a single resource (Fitzgerald).

URL: Uniform Resource Locator. A specific identifier that not only identifies a resource but also tells you how to access the resource (Fitzgerald).

UUID: Universally Unique Identifier. A randomly generated alphanumeric string commonly 24 characters long with an extremely low chance of collision (as in, you never need to worry about two systems randomly assigning the same UUID) and consequently can be used over multiple databases.

Helpful Sources

- Swagger
- Web Console
- Download Postman
https://www.postman.com/downloads/
- Download Insomnia
https://insomnia.rest/download
- Explanation of “include” and “revinclude”
https://youtube.com/watch?v=W8F4cGmseo0&feature=share
- Full list of error codes
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
- Full list of Resource types
https://www.hl7.org/fhir/resourcelist.html
- FHIR Searches
https://www.hl7.org/fhir/search.html
- FHIR Search filters
https://www.hl7.org/FHIR/search_filter.html
- FHIR Report Codes
https://www.hl7.org/fhir/valueset-report-codes.html
- FHIR Substance Codes
https://www.hl7.org/fhir/valueset-substance-code.html
- FHIR Medical Finding Codes
https://www.hl7.org/fhir/valueset-clinical-findings.html
- FHIR Terminology
https://build.fhir.org/terminologies.html

Prefixes to Keep in Mind:

gt	The parameter is greater than the given value. The range above the parameter intersects with the wanted value range.	(parameter)= gt100 value= >100
lt	The parameter is less than the given value. The range below the parameter intersects with the wanted value range.	(parameter)= lt100 Value= <100
eq	The parameter is equal to the given value. The parameter range. The parameter range contains the full range of the wanted value.	(parameter)= eq100 Value= 100
ne	The parameter is not equal to the given value. The parameter range does not contain the full range of the wanted value.	(parameter) = ne100 value= </=100
ge	The parameter is greater than or equal to the given value. The range above the parameter intersects with the range of the wanted value, or fully contains the range of the wanted value.	(parameter)= ge100 value= >/=100
le	The parameter is less than or equal to the given value. The range below the parameter intersects with the given value, or fully contains the range of the wanted value.	(parameter) = le100 value= </=100
sa	The parameter starts after the given value. The range of the parameter does not overlap with the range of the given value, and the range above the parameter contains the wanted value.	(parameter)= sa100 value= 101+
eb	The parameter starts before the given value. The range of the parameter does overlap with the range of the given value, and the range below the parameter contains the wanted value.	(parameter)= eb100 value= 99-
ap	The parameter is approximately the same as the given value. The range of the parameter overlaps with the range of the wanted value.	(parameter)= ap100 value= 99.5-100.5

Value range	Limit based on value precision	Range of 2.0 = 1.95-2.05
Range above value	The value and up	Range of 2.0 = =/>2.00000
Range below value	The value and below	Range of 2.0 = =/<2.00000

JSON Important Terms

Term	Explanation	Example
String	A string is any arbitrary text. The text can be composed of alphanumeric and special characters. Strings are encapsulated in quotes.	"Mary Jane," "32 years old," "12"
Number	A number is computable. Math functions can be performed on numbers, however, math problems cannot be performed on numbers in strings. Numbers can have decimals.	1, 2, 3.14, -5, 7.6E3
Date	Date is a special type of number that represents date and time. There are functions that are specific to dates such as add 30 days, subtract 4 months, convert time zones.	DOB: **30/10/20**, observationTime: **2015/11/07T10:32:06+2:00**
Boolean	Boolean is also a special type of number- either a 0 or a 1. A boolean is a binary variable with the only values being true or false. Booleans are never in quotes.	"active:" **true**, "active:" **false**, "active:" **0**
Key	A key is used to describe the value. Keys and values are separated by ":" (Interchangeable with name).	"**gender**:" "male," "**age**:" "42," "**active**:" "false"
Value	The value is the information being presented.	"gender:" "**male**," "age:" **18**, "active:" **true**
Object	An object is a series of	{name: value, name: value, name:

	information often called a class (classroom, classes, etc…).	value...}
		{ “name:” “Mary Jane,” “age:” 32 **}**

Array	An array is an order of elements that are all contained in the same variable. There can be any number of objects inside an array.	[value, value, value...] **[** { “given:” “Mary,” “family:” “Jane” } **]**
Element	An element is anything being declared.	Numbers, boolean, strings, objects, etc... 1, “1”, “one”, true

Operator Summary

	Strings	Numbers	Tokens
And	[parameter]= “string”&[parameter]=”string”	n/a	-
Or	[parameter]= “string”,”string”	n/a	-
Not	-	[parameter]=ne”number”	[parameter]:not= “code” [parameter]:not=”code”
Is	[parameter]:exact=”string” [parameter]:contains=”string” [parameter]=”string”	[parameter]=”number”	[parameter]:[sub-element]=”code”
Greater than	n/a	[parameter]=gt”number”	[parameter]:above=126851005
Less than	n/a	[parameter]=lt”number”	[parameter]:below=126851005

Signs and Syntax to Keep in Mind

1. **Question Mark “?”**
   The question mark symbol **“?”** is used to separate the resource type or the queryable object and the query parameters.

E.g. GET
http://hapi.fhir.org/baseR4/Patient?family=chalmers

2. **And “&”**
   To add multiple query parameters, an ampersand **“&”** is placed in between them to form what is known as a query string. It can feature various object types with distinct lengths such as arrays, strings and numbers. It could be used to add different values to improve the search. For example, to search for patients that speak both French or German, the values will be separated by an “**&**.”
   E.g. GET Patient?general-practitioner.name=Joe&general-practitioner.address-state=MN
   E.g. GET
   /Patient?language=FR&language=NL
3. **Comma “,”**
   If the search is to find resources that are two or more characteristics, then the comma **“,”** is used. For example, to search for patients that speak either French or German, then this is a single parameter with multiple values, separated by a “**,**.”
   E.g. GET
   /Patient?language=FR,GN
4. **Underscore “_”**
   It’s a syntax rule that a few parameters always begin with an underscore **“_.”** Refer to the list in the summary table
5. **Colon “:”**
   Modifiers used in the query generally begin with a colon **“:”** like “_include” and “_revinclude” queries.
   E.g. GET
   https://hapi.fhir.org/baseR4/Patient?family:contains=mers
6. **Period “.”**
   In order to save a client from performing a series of search operations, reference parameters may be "chained" by appending the operations with a period “.”
   E.g.
   GET
   [base]/DiagnosticReport?subject.name=peter
   GET
   Patient?general-practitioner.name=Joe&general-practitioner.address-state=MN
   Chained parameters can also be applied independently to the target resource as above.
7. **Pipes “|”**
   i) To query quantitative values along with the unit, they must be separated with a pipe “|” if the unit is provided in the resource.
   E.g. GET
   https://hapi.fhir.org/baseR4/Observation?value-quantity=6.3|mmol/l
   ii) Pipes are used in the syntax for token search parameters or in other words queries that involve code and code/codesystem (to query on codes) and label, system and key (to query on identifier).
   E/g/ GET

https://hapi.fhir.org/baseR4/Observation?component-code-value-quantity=http://loinc.org|8480-6

8. **"+" positive sign and "-" negative sign**
   While querying resources sorted with respect to their date(using the sort parameter), the plus sign is used to retrieve a result that is in ascending order and the negative sign indicates a result that is in descending order
   E.g. GET
   https://hapi.fhir.org/baseR4/Observation?_sort=+date
9. **Dollar sign "$"**
   i) Composite queries are queries that contain multiple parameters for the purpose of running more precise searches. These queries are too complex to make use of "&" or ",". Instead they use the dollar sign **"$"** to attach parameters together.
   E.g. GET
   [base]/Group?characteristic-value=gender$mixed
   The above query can be used to search for all groups that have a characteristic "gender" with a text value of "mixed."
   E.g. GET
   [base]/Questionnaire?context-type-value=focus$http://snomed.info/sct|408934002
   The above query can be used to search for all questionnaires that have a clinical focus: "Substance abuse prevention assessment (procedure)" Use this link to cross check the Snomed code 408934002.

   ii) The "_list" parameter allows for the retrieval of resources that are referenced by a list resource. FHIR defines some functional lists like current problems, current medications, current allergies and current drug allergies. The syntax of these lists begin with a dollar **'$'** sign.
   Eg GET
   [base]/AllergyIntolerance?patient=42&_list=$current-allergies
   This is a request to fetch all the allergies in patient 42's "Current Problem List"
10. **Backslash "\"**
    If these characters(**"$"**, **","** and "**|**") appear in an actual parameter value, they must be differentiated from their use as separator characters.
    When any of these characters appear in an actual parameter value, they must be prepended by a "\," which also must be used to prepend itself.
    E.g. GET
    [base]/Observation?code=a\,b
    The above query is a request for any observation that has a code of "a,b."
11. **Star "*"**
    Revinclude and include queries make use of the star "*" to substitute the idea of returning all resources or everything.
    E.g. GET
    https://hapi.fhir.org/baseR4/MedicationRequest?_id=1759&_include=*
    This query will return all resources that are referenced in all the medication request resources on the server.

Sources Used

Code Academy. “What is REST?” Code Academy, https://www.codecademy.com/articles/what-is-rest. Accessed August 30, 2021.

FHIR. “Bundle.” HL7 FHIR, https://www.hl7.org/fhir/bundle.html#:~:text=A%20bundle%20is%20a%20collection,directly%20using%20the%20RESTful%20API. Accessed August 30, 2021.

Fitzgerald, Anna. “URI vs URL: What's the difference?” Hubspot, 6 January 2021, https://blog.hubspot.com/website/uri-vs-url. Accessed August 30, 2021.

Lutevich, Ben, et al. “What is PHI?” TechTarget, April 2021, https://searchhealthit.techtarget.com/definition/personal-health-information. Accessed August 30, 2021.

Mozilla. “MIME types.” MDN, https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types. Accessed August 30, 2021.

Mozilla. “An overview of HTTP.” MDN, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. Accessed August 30, 2021..

Mulesoft. “What is an API?” Mulesoft, https://www.mulesoft.com/resources/api/what-is-an-api. Accessed August 30, 2021.

Reid-Miller, Margaret. “Boolean Data Types.” CMU.EDU, https://www.cs.cmu.edu/~mrmiller/15-110/Handouts/boolean.pdf. Accessed August 30, 2021.
Tutorials Point. “RESTful web services.” TutorialsPoint, https://www.tutorialspoint.com/restful/restful_resources.htm. Accessed August 30, 2021.

Smile CDR Inc.

622 College Street, Suite 401
Toronto, Ontario M6G 1B4, Canada
info@smilecdr.com
1 (800) 683-1318

www.smilecdr.com

All product names, logos, and brands are the property of their respective owners. All company, product and service names used are for identification purposes only. The use of these names, logos, and brands does not imply endorsement.

Version: 1.0
Last updated: November 12, 2021
Principle Authors: Alexis Cole, Tanvi Shah