HL7 v2 to FHIR Interface Mapping

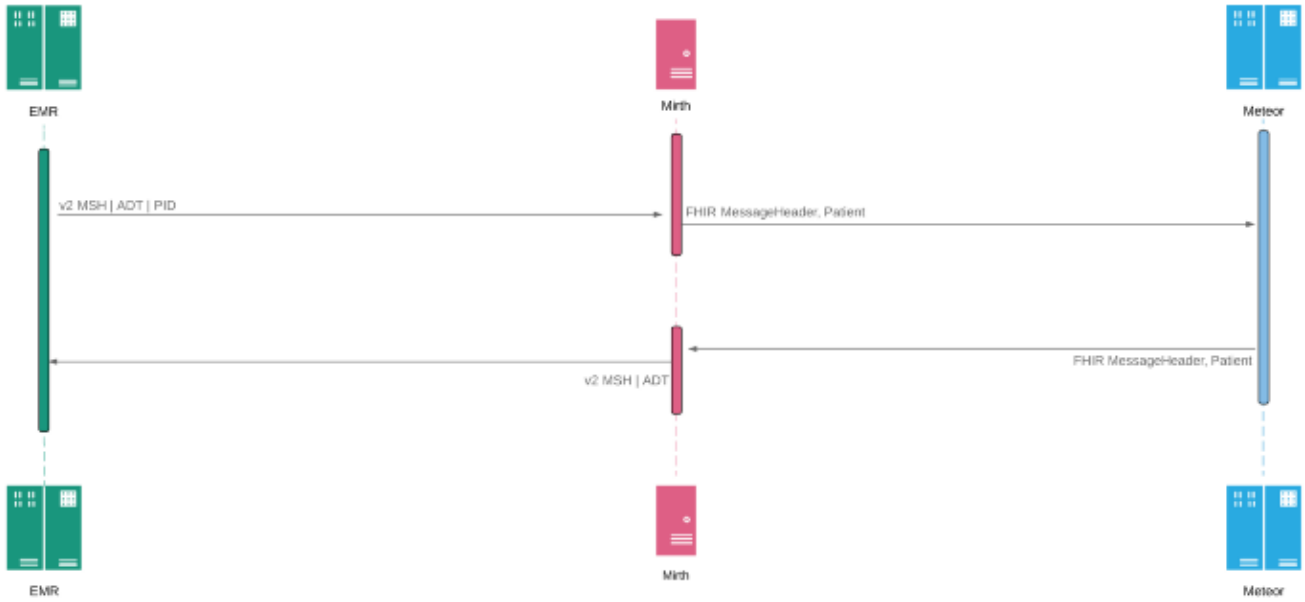
Abigail Watson
Feb 3, 2017 · 11 min read

In this tutorial, we're going to explain how to send data from an electronic health record (EHR) to a full-stack javascript webapp using an HL7 interface engine. We will be mapping data from the 1980s era HL7 v2 standard to the modern Fast Healthcare Interoperability Resources web standards.

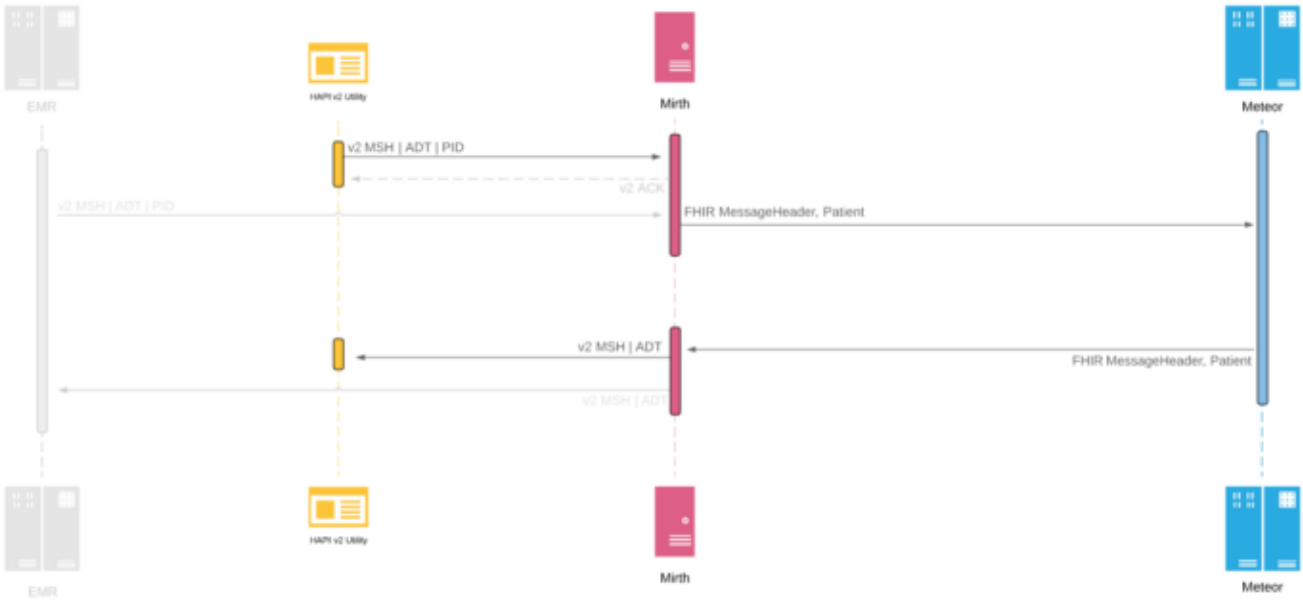
Architecture Overview

The most basic HL7 interface between two electronic health records is an Admit/Discharge/Transfer (ADT) interface. An ADT interface may be either *unidirection* or *bidirectional*, depending on whether a return message is sent.
For the purpose of this tutorial, we are interested in passing data from an older EHR that uses the HL7 v2 standard to a newer EHR that uses Fast Healthcare Interoperability Resources (FHIR). Assuming that EHR A only speaks v2, and EHR B only speaks FHIR, it's necessary to have an *interface engine* between the two, which is responsible for transforming messages from v2 to FHIR.

As such, we have a interface comprised of two *channels*, which includes four *links*, and eight *endpoints*.



app and this tutorial, we want to use a testing utilities to simulate the EHR interface.



Install the Tools

You'll need to install the following tools and utilities. This tutorial assumes that you have a vanilla installation, and haven't configured them yet.

Mirth Connect HL7 Interface Engine
Mirth FHIR Technology Preview 2
Google Chrome Web Browser
Meteor Javascript Platform
Meteor on FHIR Interface Engine
Postman HTTP Endpoint Utility
HAPI HL7 Endpoint Utility

> *Note: Due to the number of components involved in this tutorial, we recommend using multiple monitors. This is a task that is facilitated with screen real estate. Dual HDMI or Thunderbolt monitors will make this tutorial much easier.*

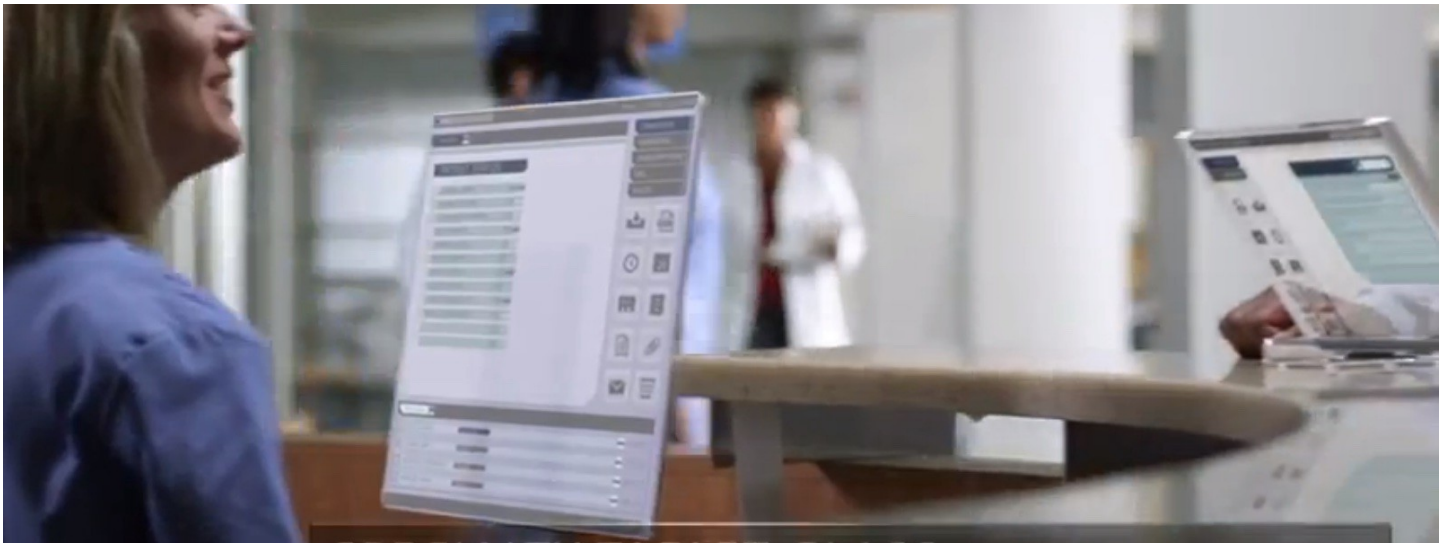
Meteor on FHIR Configuration

For running this tutorial, you will want to use the `settings.tutorial.json` settings file, and turn on debugging.

```
NODE_ENV=test DEBUG=true meteor --settings settings.tutorial.json
```

This tutorial relies on the default outbound interface being active; so be sure to set the status and endpoint if you decide to experiment.

```
{
  "public":{
    "interfaces": {
      "default": {
        "status": "active",
        "channel": {
          "endpoint": "http://localhost:3000"
        }
      }
    }
  }
}
```



Part 1 — Admitting a Patient from the EHR

The case sample begins with a patient Jane Doe who is registering at a walk-in clinic for the first time for an X-Ray of a broken arm. The front-desk receptionist is responsible for taking her information, Name, Age, Date of Birth, and adding it into the electronic health record, which will assign Jane a medical record number. When this happens, the EHR system will notify the Radiology system that Jane has arrived.

The Admit Discharge Transfer (ADT) Message

over Dual Port MLLP (Minimum Link Layer Protocol). It looks like this:

```
MSH|^~\&|EPICADT|DH|RADADT|DH|201701010915||ADT^A01|MSG000001|P|2.3
PID|||MRN12345^5^M11||Doe^Jane||19780101|F||C|||||||||123456789|
```

Unless you've been in the industry for a long time, this probably looks like gibberish to you. And you're not the only one. There's a reason we don't use this format anymore. Yet somehow it's still widely used, so it's important to be able to parse it.

The important things to remember are that this format was invented when RAM in computers was measured in kilobytes and modem speeds were 2,400 bit/s. It may seem terse, but all you need to remember is that the pipe character | is a field separator, which makes the entire message basically like a comma or tab separated file. The carrots ^ denote subfields.

If you're not familiar with modems, they're point-to-point communication devices used in the 70s and 80s. When we talk about Dual Port MLLP, we're talking about an Ethernet technology to simulate a modem link. It's a minimalist bare-bones TCP/IP connection. As simple as you can get.

Each HL7 v2 message has a segment header. In this case `MSH` which stands for Message Header. You'll also see that it's of type ADT A01. By using an HL7 Segment ID Lookup Table, you'll see the following segment corresponds to an Admit/visit notification.

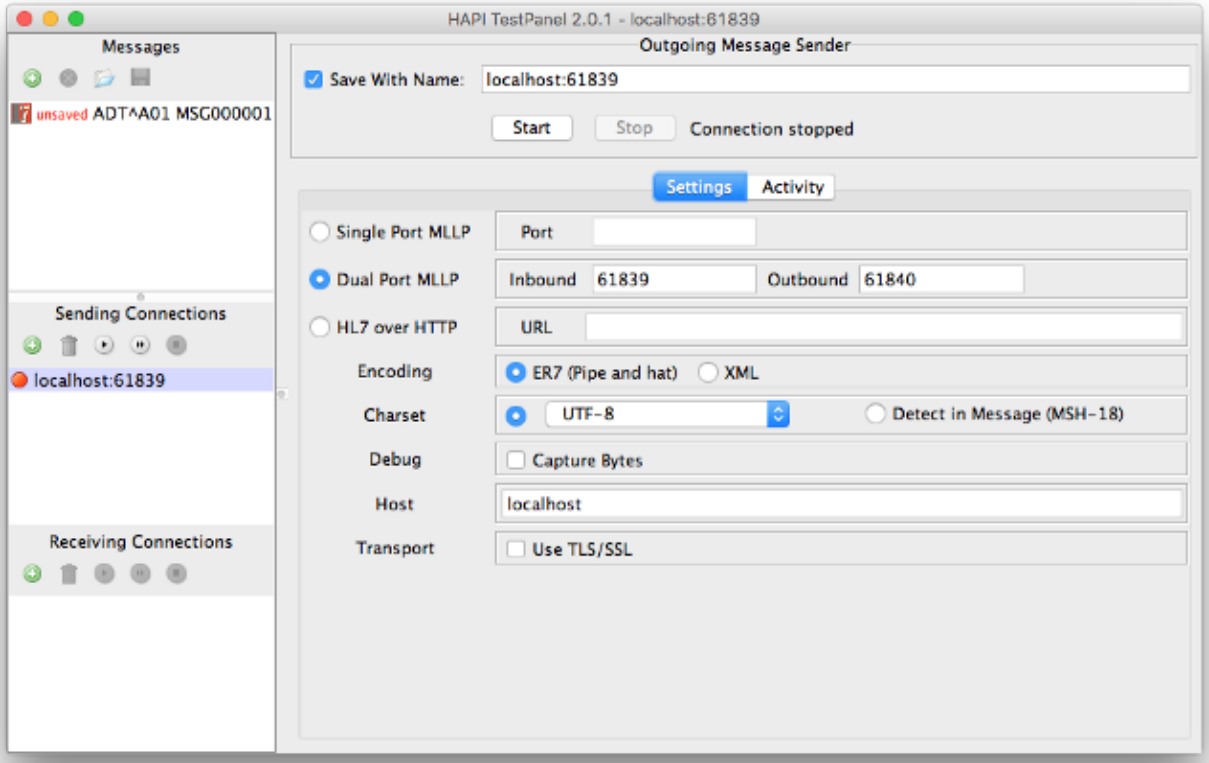
```
MSH|^~\&|EPICADT|DH|RADADT|DH|201701010915||ADT^A01|MSG000001|P|2.3
```

The second segment is a Patient Identification segment, which contains basic demographics for a patient, including medical record number, name, and date of birth.

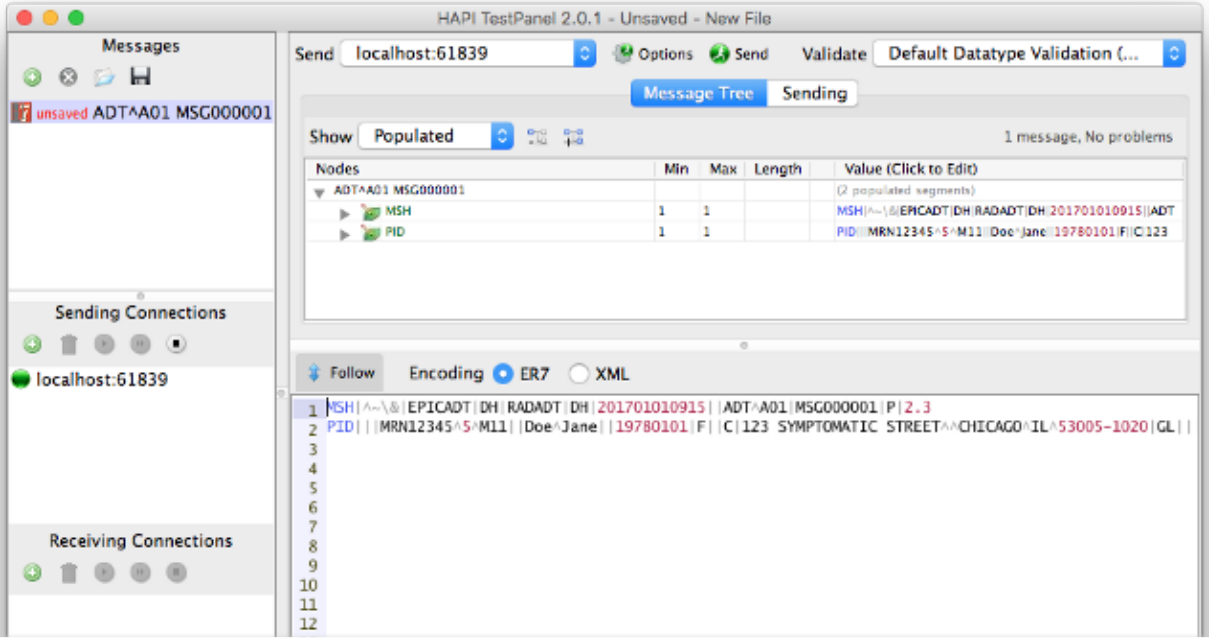
```
PID|||MRN12345^5^M11||Doe^Jane||19780101|F||C|||||||||123456789|
```

Configuring the Outbound v2 Endpoint

We'll want to take note of the inbound and outbound ports, which we'll use later when setting up Mirth.

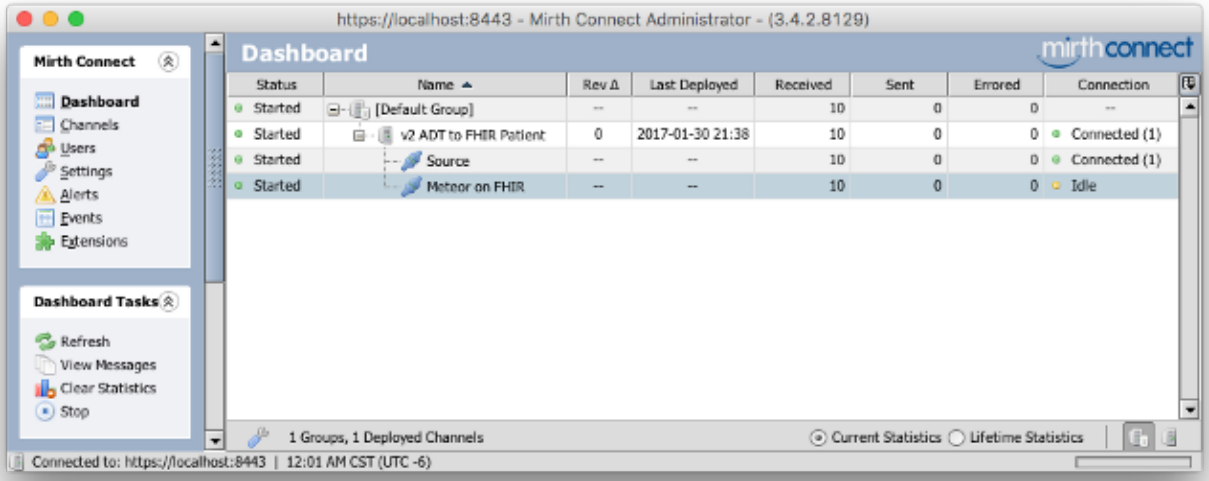


Click the 'Start' button, and you should see the `localhost:61839` light up green, and be routed to a new page where we can specify an ER7 message.

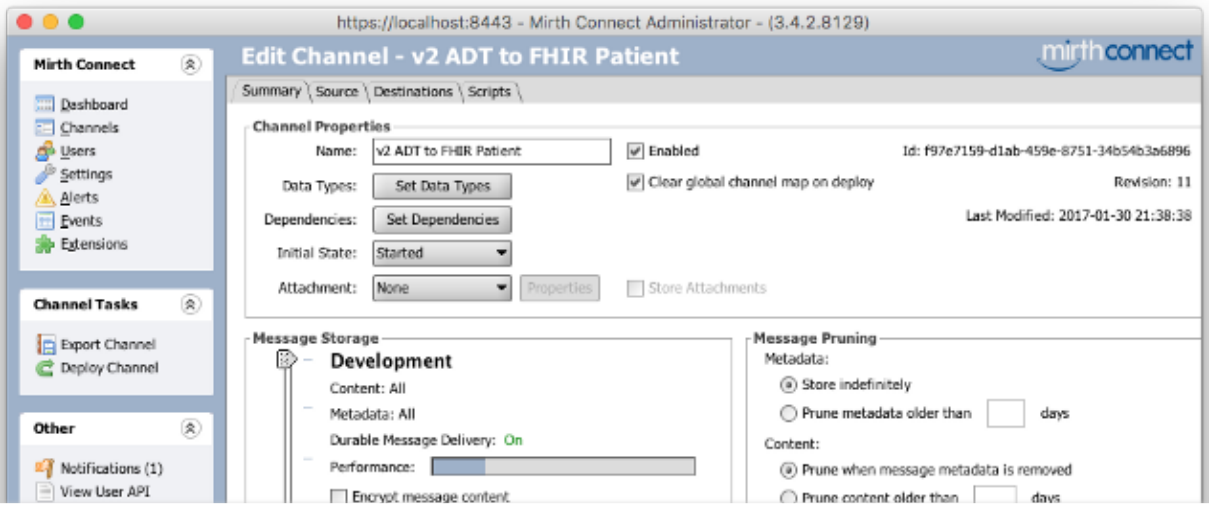


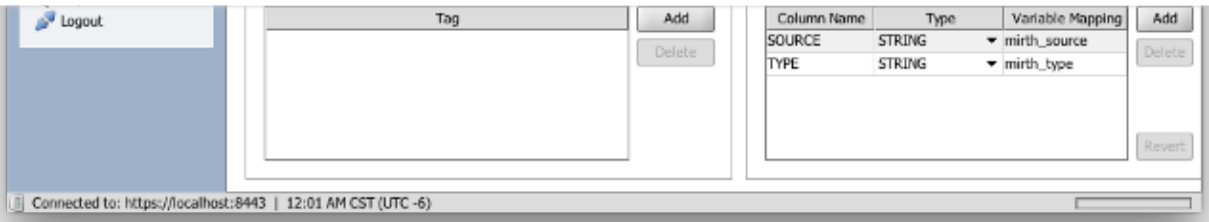
Configuring the Inbound v2 Endpoint

Before we can send the message, we need somewhere to send it *to*. We begin configuring the destination endpoint by logging into Mirth, and setting up a new channel.

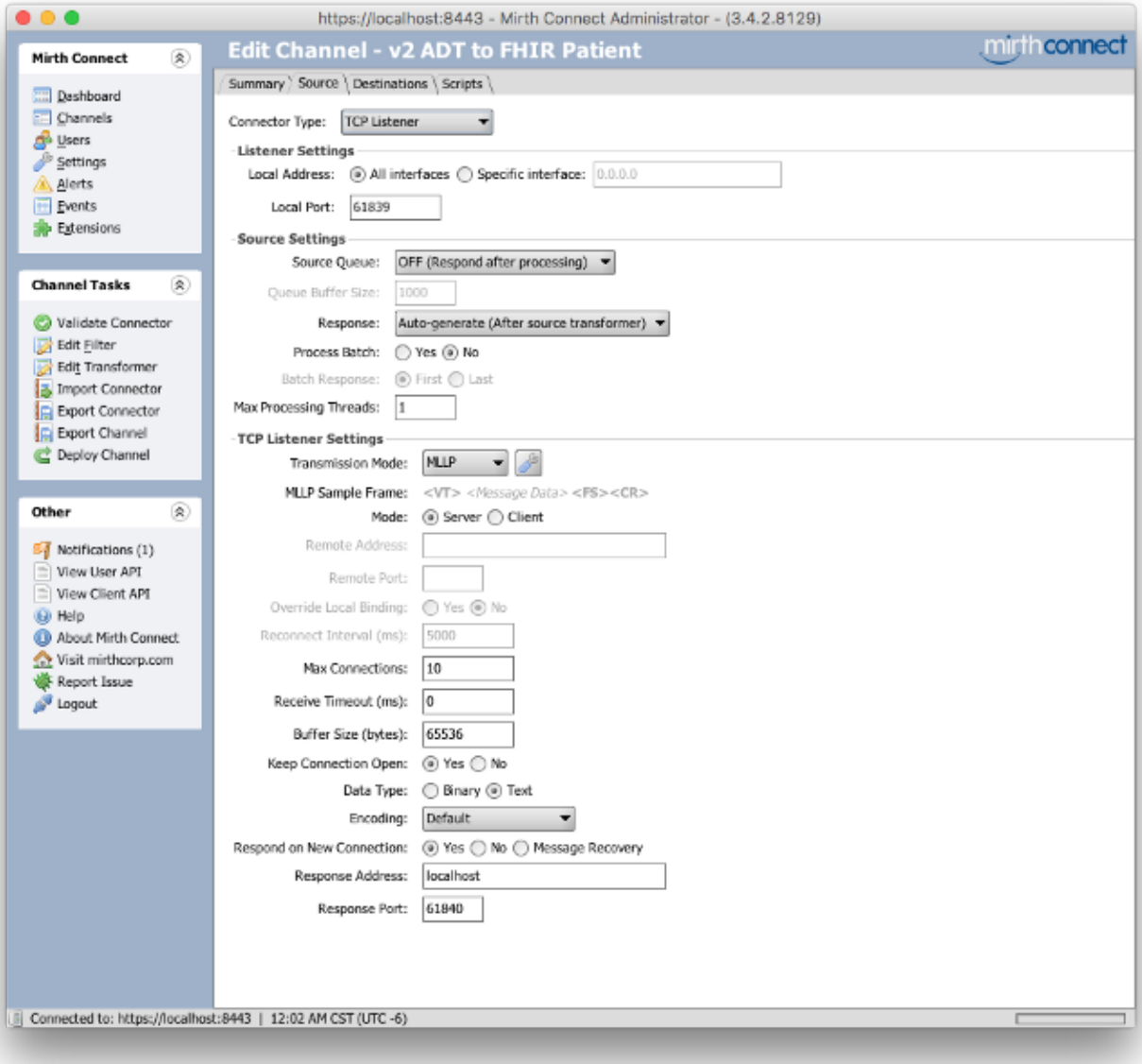


Most of the default channel settings are fine to use. These settings become more important when the interface is in production, and lots of data is flowing through the interface engine. That's when things like pruning metadata, caping storage, and encrypting the database become important.



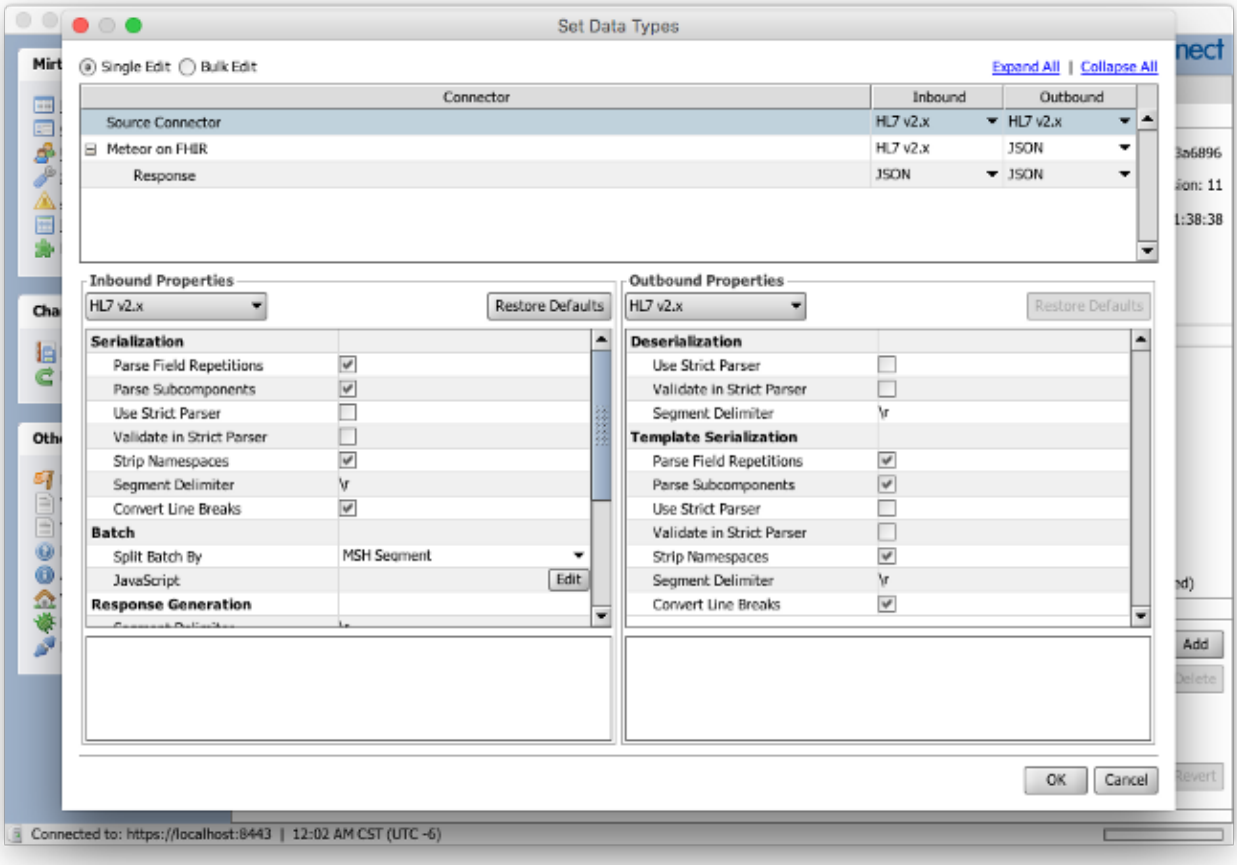


Click on the **Source** tab to begin configuring the other endpoint of our first link. Remember that port `61839` from the other endpoint? We want to set up a TCP listener on that port. Add an auto-generated response (which we call an *ACK* or *acknowledgement*), and set the transmission mode to MLLP. And we should be good to go. There are many more options available to fine-tune the interface in a production environment. But for now, simply Deploy Channel

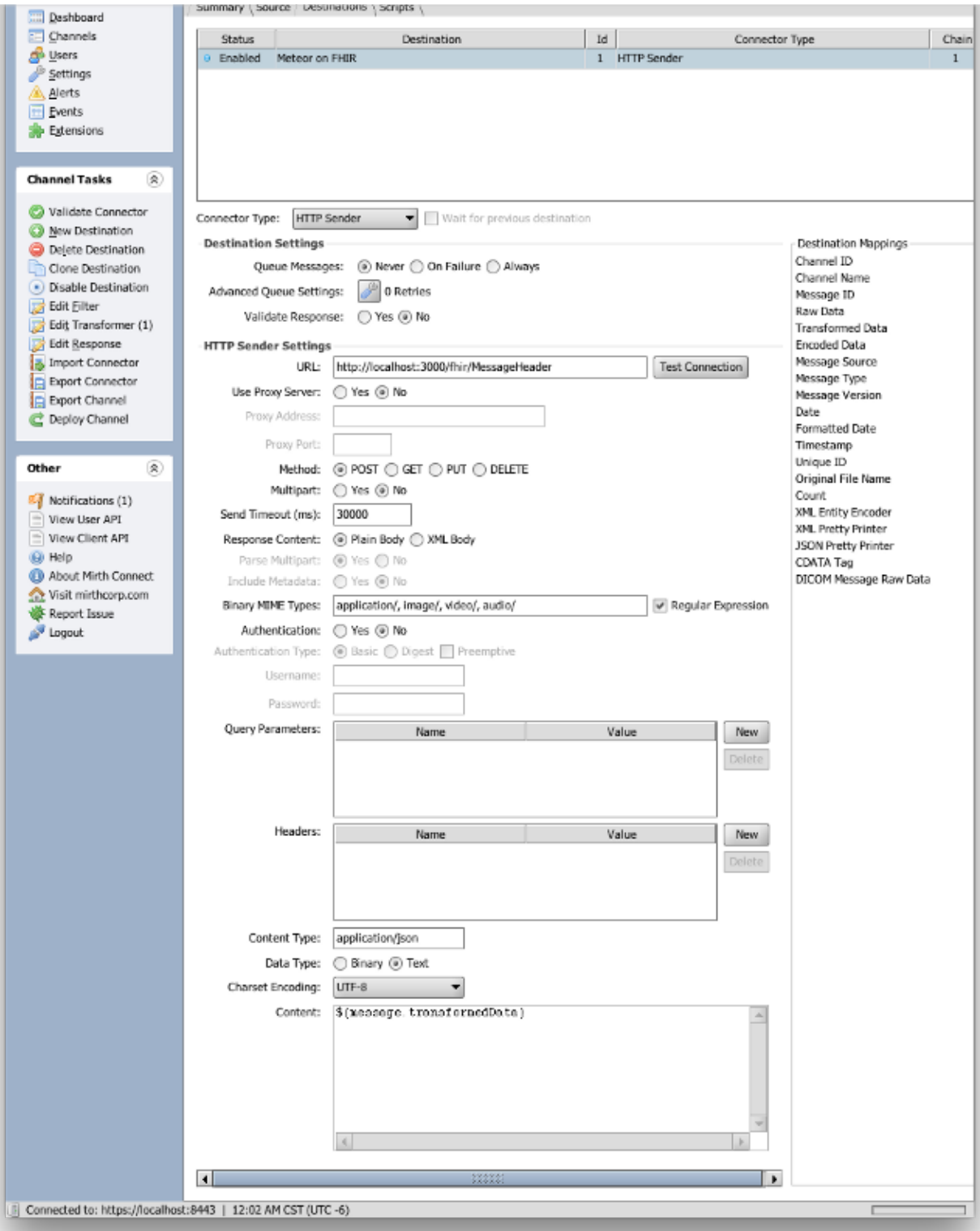


The idea behind an *interface engine* is that it connects inbound interfaces to outbound interfaces. Each of these connections is called a channel. And we begin defining a channel by specifying it's inbound and outbound datatypes.

For the purposes of this tutorial, we're interested in mapping legacy v2 messages to modern FHIR messages. And since FHIR is based on modern web standards, the format we're interested in is the Javascript Object Notation (JSON). So lets begin creating the channel by going back to the Data Types tab, and specifying an inbound datatype of HL7 v2.x, and an outbound datatype of JSON.



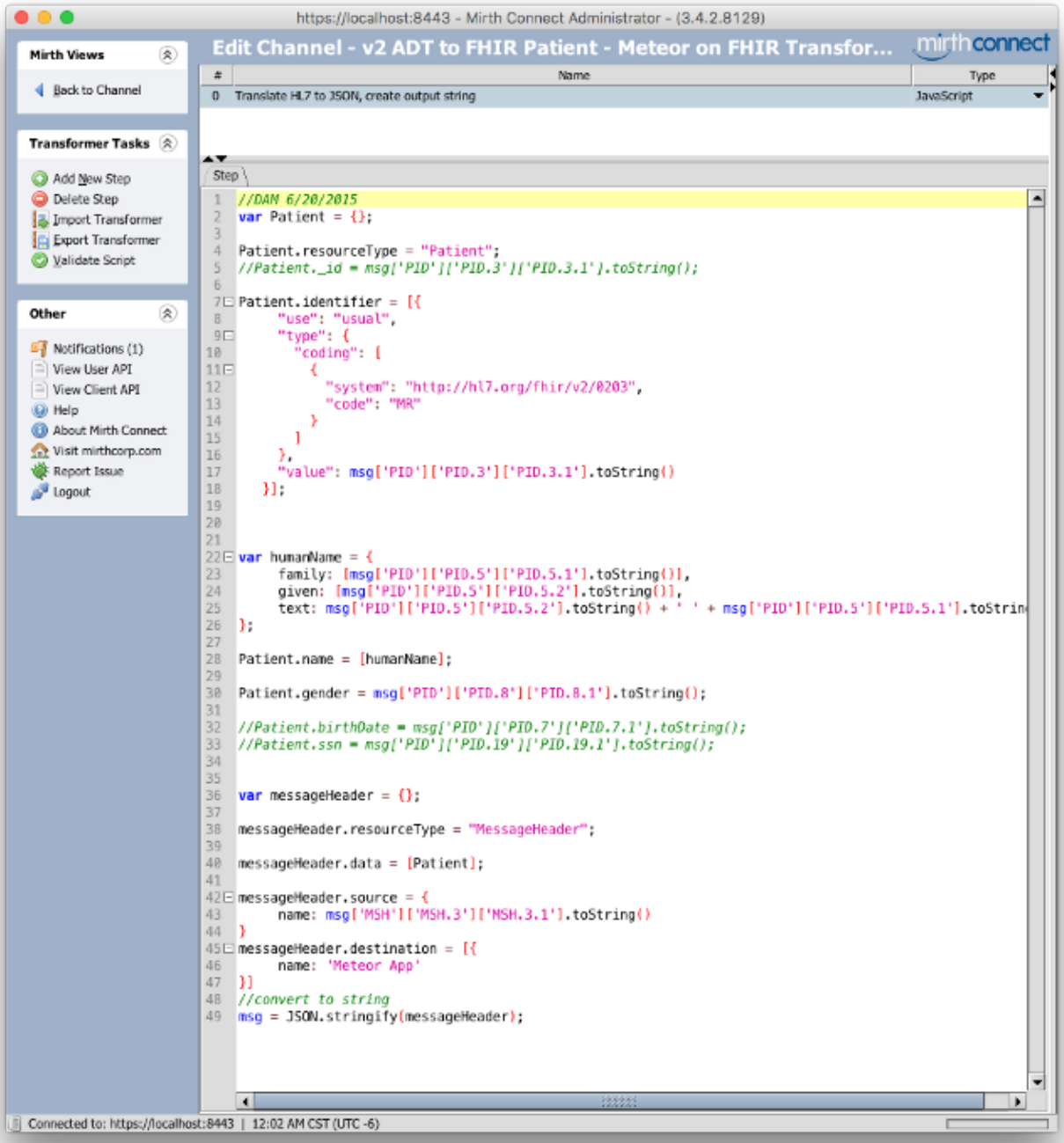
We're now ready to set up the outbound FHIR endpoint. Go to the **Destination** tab, and giving the destination a custom name and specifying an HTTP Sender connector type. We're interested in sending a `POST` message to the `/fhir/MessageHeader` route. Since HTTP doesn't support ED7 encoding, we'll also need to transform our message. So we'll need to specify the content as `${message.transformedData}`.



Mapping the ADT Message to a FHIR Patient Resource

To transform the data, click on **Edit Transformer**, which will take you to a scripting page. We need to create a `MessageHeader` object to map the `MSH` segment to, and a `Patient` object to map the `PID` segment to. Specific fields in the v2 message (delimited

guidance. When done, **Validate Script** and **Deploy Channel**.



> *See the entire v2 to FHIR Patient Channel Mapping File*

Sending an ADT Message from one App to Another

We're now ready to log into the Meteor on FHIR app, and receive an message. Open up Meteor on FHIR and log in as a System Administrator.

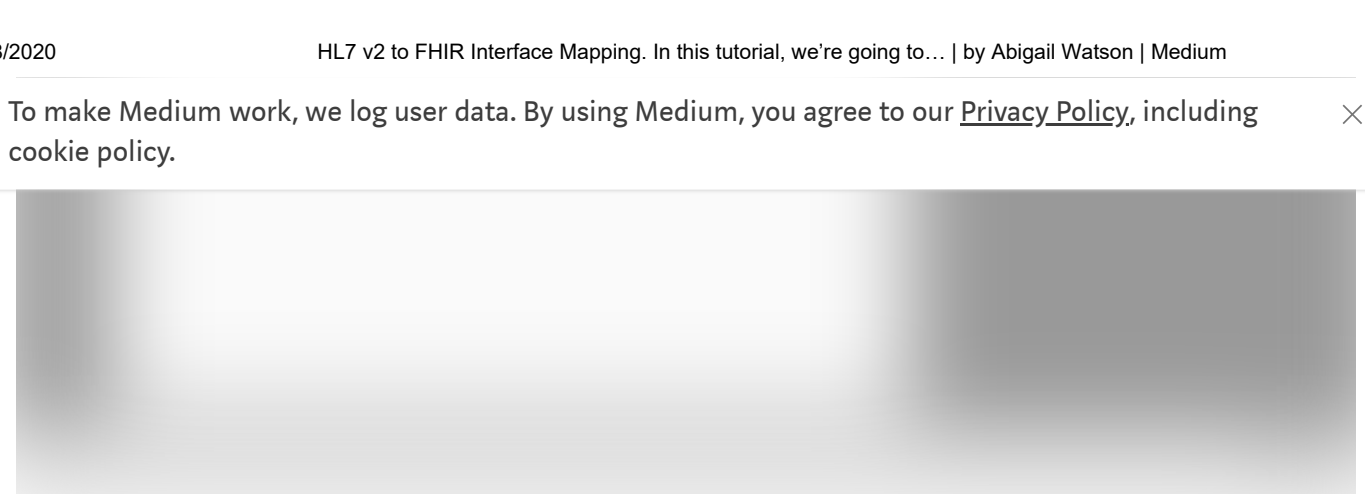
Confirm that there are no messages or patients in the system.

Now go to the HL7 v2 endpoint utility, make sure our ED7 message is correct, that the connection is started on localhost, and then click **Send**.

If everything works correctly, the interface engine dashboard should auto-increment, indicating that a message was both Received and Sent. You may need to refresh the app to get the latest metrics.

If you drill down on the channel, you'll see a message log where you can see the status of each message. Click on **Transformed** to see FHIR message that the transfomer script generated.

If the message was sent successfully, we should now be able to go to the Meteor on FHIR app, and see a message in the Inbound HL7 Message log.



We should also have a patient registered in the Patient index.

Now it's time for the return trip. Our case sample is that of sending the patient back to the original system with some edits to the demographic information.

The FHIR Patient Resources

On the return trip, we will be sending the same information; but with modern web-standards and using the Javascript Object Notation (JSON). As you can see, this format is more verbose and structurally formatted. JSON is comprised of a hierarchical data structure, so most JSON objects can be represented as a branching tree of some sort. This makes it very convenient for representing healthcare data; as the science of biology is fundamentally comprised of tree, graph, and web datatypes.

```
{
  "resourceType" : "Patient",
  "identifier" : [{
    "use" : "usual",
    "type" : {
      "coding" : [{
        "system" : "http://hl7.org/fhir/v2/0203",
        "code" : "MR"
      }]
    },
    "value" : "MRN12345"
  }],
  "name" : [{
    "family" : [
     "Doe"
    ],
    "given" : [
      "Jane"
    ],
    "text" : "Jane Doe"
  }],
  "gender" : "F"
}
```

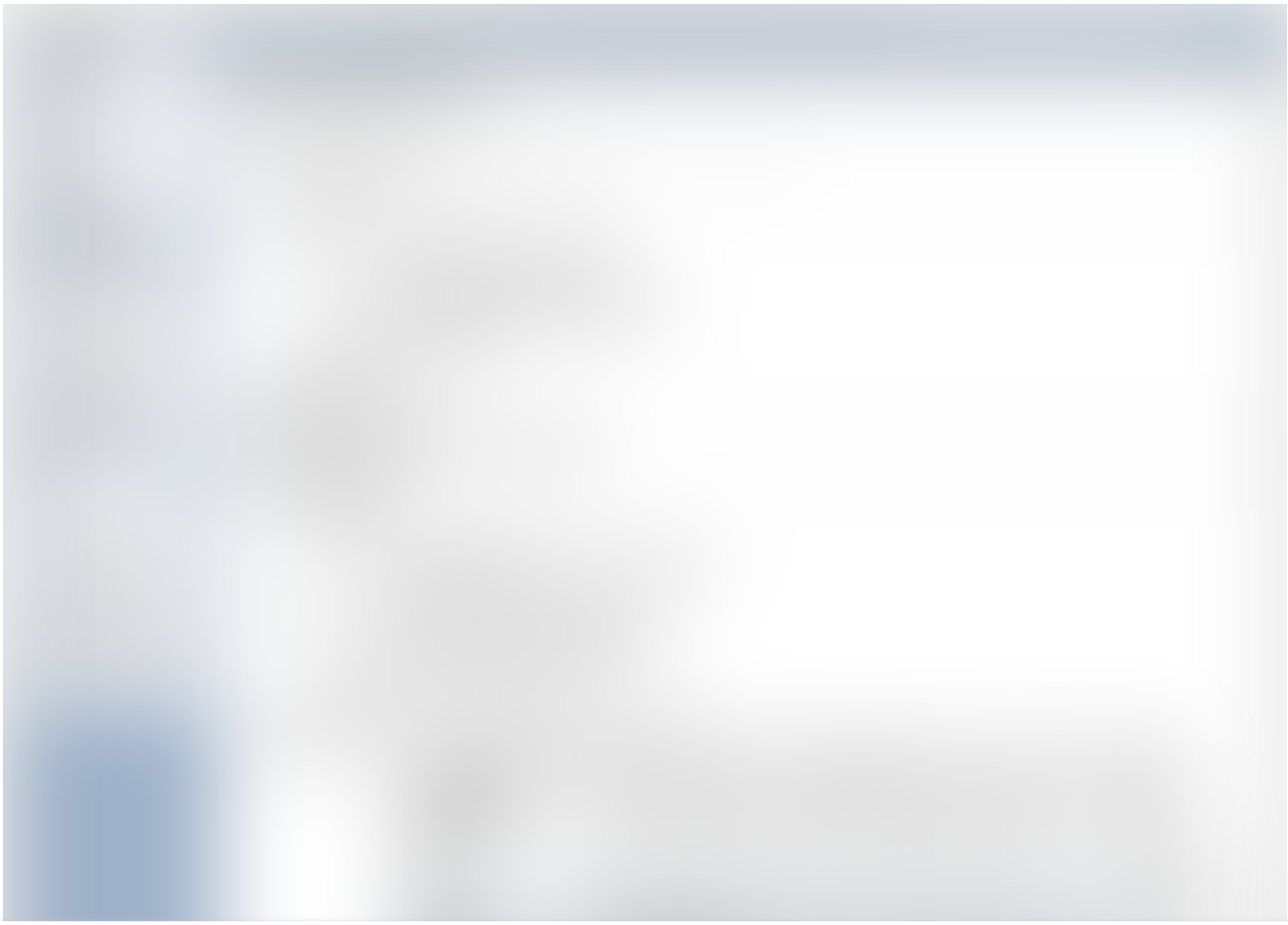
Configuring the Inbound v2 Endpoint

Setting up the inbound v2 endpoint is fairly straight forward. We simply need to configure a Receiving Connection with a new port number and start it. In this example, we will use Single Port MLLP, as the Mirth interface engine's outbound TCP Sender is a 'fire-and-forget' setup.

Once started, the interface will display an activity monitor, where we can watch for incoming messages.

EHR system's info.

For the next step, it's important that you've installed the Mirth FHIR Technology Preview 2, as we want to install a **FHIR Listener** as our Source connector type. We will also want to specify the JSON format, and enable the Patient Resource.



Next, configure the Destination connector type to be a **TCP Sender** with **MLLP Transmission Mode**. Use the address and port number from the v2 endpoint utility. Lastly, specify `${message.transformedData}` for the message content.

Now it's time for the return mapping. In this script, we're starting with a JSON object, and using it to build a string. The pattern used here is a stitching or decorator pattern; where we continuously decorate the string with one field after another. Be aware that we need to *escape* certain characters using the `\` character; that the date string has to be broken up into its component parts, segments are separated by the `\r` character,

string.

> *See the entire FHIR Patient to V2 Channel Mapping File.*

> *You can install default patient data with Meteor on FHIR by specifying the Patients environment variable, like so:* `Patients=true meteor`

If everything works, we should see a message Received over the channel, and Sent to the v2 endpoint.

Inspecting the message should show us the raw Patient object that was sent from the Meteor on FHIR server.

And the transformed ED7 message that was sent to the v2 endpoint.

Which, if everything worked correctly, should have arrived in our activity monitor. Success! It works!

The Received HL7 v2 Message in ED7 Format

If everything worked correctly, you should have received the following message.

```
MSH|^~\&|MeteorOnFHIR||HAPI||201722238||ADT^A01|20338|P|2.6|
PID|MRN12345|cTtAcSGnhE2cqApnQ||John Doe||1975-02-
03T00:00:00.000Z|M|||||||||||||||||||||
```

Conclusion

FHIR interface.

References

HAPI HL7 Endpoint Utility
Postman HTTP Endpoint Utility
Google Chrome Web Browser
Mirth Connect HL7 Interface Engine
Mirth FHIR Technology Preview 2
Meteor Javascript Platform
Meteor on FHIR Interface Engine

Catalyze.io — The HL7 ORM Order Entry Message
Catalyze.io — How to Integrate with Epic or any EHR
Catalyze.io — The Admission/Discharge/Transfer ADT Message
CorePointHealth — HL7 OBX Segment
CorePointHealth — HL7 ORU Segment
CorePointHealth — HL7 ORM Segment
Caristix — OBX Segment

Fast Healthcare Interoperability Resources — Overview
FHIR Resourcelist — DTSU2
FHIR Patient Resource
FHIR Message Header Resource

Day Made of Glass 2 (Medical Edit)

Hl7 Meteor Ehr Fhir JavaScript