User Guide

Creado y modificado por última vez por Nick Rupley el oct 16, 2015

Before continuing, read the overview if you haven't already:

- Overview

This guide is separated into the following sections:

Getting the FHIR Listener

Download the FHIR Technology Preview 2 here: https://info.mirth.com/Fast-Healthcare-Interoperability-Resources-FHIR.html

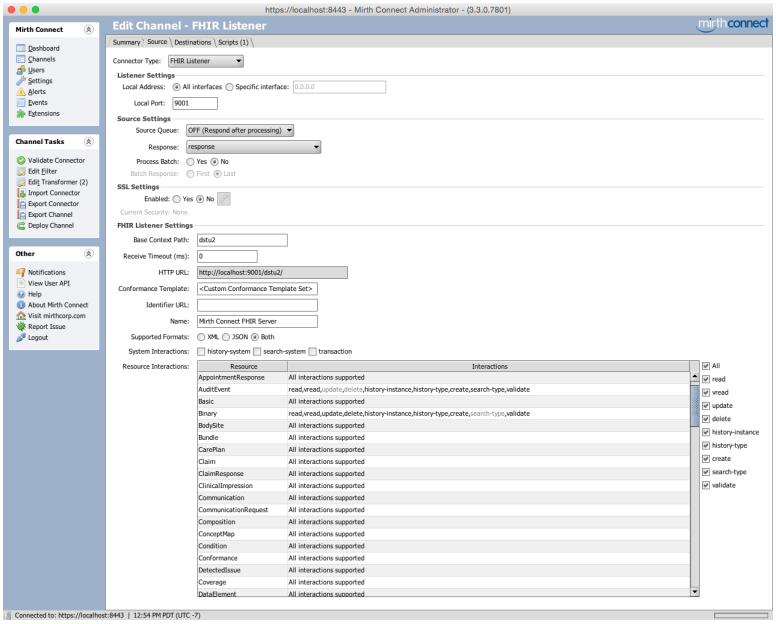
Installing the FHIR Listener

The FHIR Listener is installed like any other Mirth Connect extension. This can be done through the Mirth Connect Administrator in the Extensions view, or by extracting the contents of the provided ZIP file into the extensions directory inside the Mirth Connect installation directory.

> After restarting the server, make sure that you launch the Mirth Administrator by clicking the Launch Mirth Connect Administrator button and not using a previously downloaded JNLP file.

Setting FHIR Listener Properties

After installing, you should now have "FHIR Listener" as one of your source connector options. You can see that it's quite similar to the HTTP Listener:



A FHIR server provides a conformance statement that gives clients an overview of the application. This includes what resources and interactions the application supports, and any other requirements it has. The FHIR Listener connector settings affect what values show up in the conformance statement. They are as follows (see also — Resource Conformance - Detailed Descriptions):

- **Conformance Template:** If set, this will provide a base conformance statement that the server will use. The settings below this will

still override the matching items in the template. This allows you to configure advanced settings that aren't satisfied by the rest of the options in the UI.
- **Identifier URL**: An absolute URL that is used to identify this conformance statement when it is referenced in a specification, model, design or an instance.
- **Name**: A free text natural language name identifying the conformance statement.
- **Supported Formats**: Indicates in the conformance statement whether XML and/or JSON are formally supported by the server.
- **System Interactions:** These are system-level interactions that don't necessary operate on a single resource.

Finally, the **Resource Interactions** table determines which interactions are supported for each resource type. An "interaction" is just a pre-defined method of communicating with the server to send or retrieve a resource or some other metadata. For example, the "read" interaction accesses the current contents of a resource with a GET request. For more information, go here: http://www.hl7.org/fhir/http.html.

The selections you make in the Resources table will be reflected in the generated conformance statement, telling clients what is supported and what isn't. You can multi-select resources in the table, then use the check boxes to the right to toggle interactions on or off.

Developing a FHIR Listener Channel

After deploying a FHIR Listener channel, the "/metadata" endpoint will already be configured to return an auto-generated conformance statement. The FHIR server is now technically up and ready to receive requests. The next step is to use the source map information the connector provides, create your transformers and destinations, and return the appropriate responses.

If you send a FHIR request to your channel (for example, POSTing a Patient resource), you'll see in the message browser that certain FHIR-specific variables are automatically injected into the source map. These are the possible variables:

- **fhirInteraction**: The inferred interaction for this request (e.g. "create", "read", "update", "delete").
- **fhirType:** The type of resource the interaction/request is for (e.g. "Patient", "Binary"). This will be absent for system-level interactions.
- **fhirId**: The logical ID of the resource associated with the request. Not all interactions require the client to send an ID, so this may be absent.
- **fhirVid**: The version ID of the resource associated with the request. Not all interactions require the client to send a version ID, so this may be absent.
- **fhirCompartmentName:** The type of compartment the request is for (e.g. "Patient", "Encounter"). This will only be populated for compartment search interactions.
- **fhirCompartmentId:** The logical ID of the compartment resource associated with the request. This will only be populated for instance-level compartment search interactions.
- **fhirOperationName:** The name of the operation the request is for. For operation requests the interaction will be "operation" and this variable will contain the actual name of the operation (e.g. "$expand", "$lookup").

The above variables (depending on the type of request) will be available throughout your channel. So you can use destination filters (or destination set filtering in the source transformer) to, for example, only send to specific destinations for specific interactions. You might have a destination dedicated to creating resources, one dedicated to reading resources, etc.

Handling Responses

Let's say you've set up a destination that specifically handles the "create" interaction. Once you've created the resource and stored it (for example, in a database somewhere), the next step is to return a proper response back to the client. Each interaction is different and may require different response headers and status codes. You can view the summary table for responses here, but for ease of use, this technology preview also includes a couple of utility classes.

FhirResponse

The first is the FhirResponse class. This is an extension of the standard Response class that also allows certain FHIR-specific data to be stored so that the FHIR Listener can automatically return the correct information. Here is a list of the additional methods that can be used:

- **getId / setId**: The logical ID of the resource associated with the response.
- **getVid / setVid**: The version ID of the resource associated with the response.
- **getStatusCode / setStatusCode**: The HTTP status code that will be sent back to the client.
- **getLastModified / setLastModified**: The date that will be sent back to the client in the Last-Modified header.
- **getMimeType / setMimeType**: The MIME type that will be sent back to the client in the Content-Type header.
- **isIncludeLocationHeader / setIncludeLocationHeader**: Determines whether the Location header will be included in the HTTP response.
- **isIncludeETagHeader / setIncludeETagHeader:** Determines whether the ETag header will be included in the HTTP response.
- **getHeaders / addHeader / removeHeader**: Allows the user to add additional custom HTTP headers to include in the response.

In the example above, say you've successfully stored a resource in response to a "create" interaction. Now according to the FHIR specifications, your response should include the Location header, the status code 201 (Created), and the Last-Modified / ETag headers for versioning information. So you can create a FhirResponse object like so:

```
var fhirResponse = new FhirResponse();
fhirResponse.setId('The ID you created here');
fhirResponse.setVid('The version ID you created here');
fhirResponse.setStatusCode(201);
fhirResponse.setIncludeLocationHeader(true);
fhirResponse.setIncludeETagHeader(true);
fhirResponse.setLastModified(lastModifiedDate);
```

Then you can return that response to the source connector however you wish (response map, postprocessor script, etc.).

FhirResponseFactory

This utility class makes the above even easier, with the following methods:

- **getReadResponse**(message, vid, lastModified, statusCode, mimeType)
- **getVreadResponse**(message, vid, lastModified, statusCode, mimeType)
- **getUpdateResponse**(message, vid, lastModified, statusCode, mimeType)
- **getCreateResponse**(message, id, vid, lastModified, statusCode, mimeType)
- **getResponse**(message, id, vid, lastModified, statusCode, mimeType)

All of these are convenience methods for creating FhirResponse objects. Some methods have multiple versions depending on whether you're sending a payload back. So for example instead of the code above, all that would be needed is this:

```
1  var fhirResponse = FhirResponseFactory.getCreateResponse(id, vid, lastModified, 201);
```

Note that documentation for these classes is included in the Reference list in transformers, and also in the auto-completion dialog. You can also view the Javadoc by going to the extensions/fhir/docs/javadocs/ folder in your installation directory.

Once you send back an appropriate FhirResponse object, the FHIR Listener will take care of everything else (status code, custom headers) automatically.

The FhirUtil Utility Class

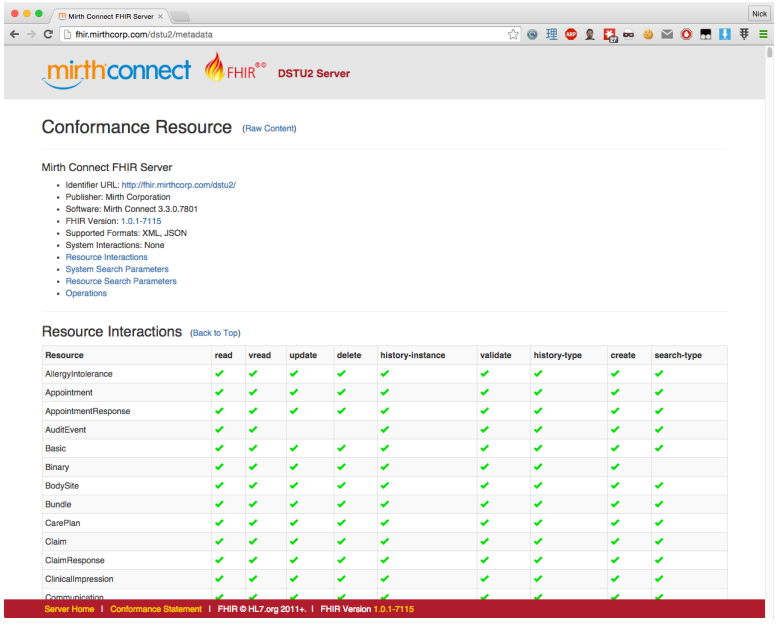
There is also a FhirUtil class accessible in JavaScript contexts. This contains methods to convert to and from XML/JSON, create special resource types like OperationOutcome, or generating resource narratives. Here are some of the most useful methods available:

- **toXML**(resource)
- **fromXML**(resourceXML)
- **toJSON**(resource)
- **fromJSON**(resourceJSON)
- **xmlToJSON**(resourceXML)
- **jsonToXML**(resourceJSON)
- **createOperationOutcome**(severity, code, details)
- **createBinaryResource**(byteArray, mimeType)
- **createBinaryResource**(base64String, mimeType)
- **generateNarrative**(resource)

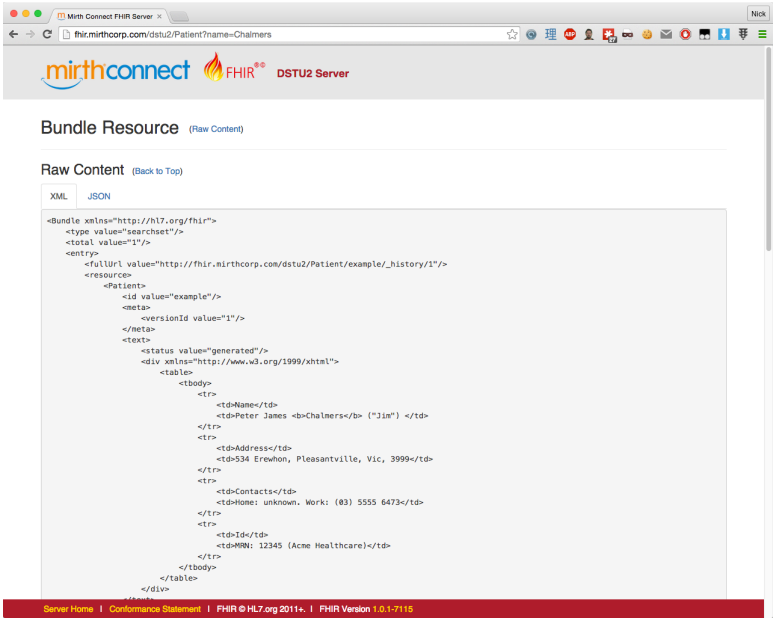
There are others too; as mentioned before you can look at the Reference list, the auto-completion dialog, or the Javadoc for more information.

Interacting With Your Server in a Browser

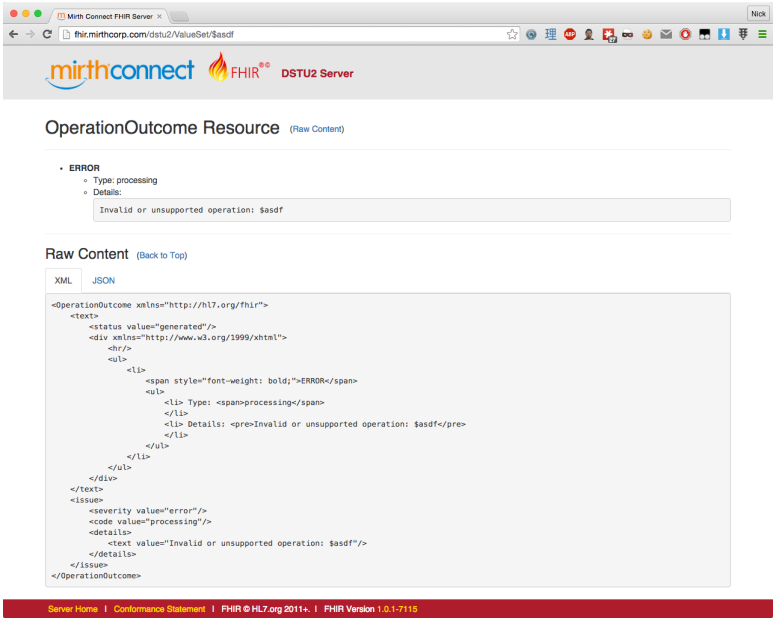
You can also view resources from your FHIR server in any modern browser. For example, the conformance statement:



Or, searching on a particular patient:



If anything goes wrong and the server returns an OperationOutcome, you will see that as well:



The web pages served to clients are generated from HTML templates in the extensions/fhir/public_html/ folder in your installation directory. These three templates are available:

- **template.html**: This is the main template for all pages that are served. Use this to include common headers/footers, CSS/JS assets, or anything else that you will show on all pages. The "mainContainer" div is where specific page content is populated.
- **resource.html:** This is the template used for any resource returned by the server. This includes virtually everything, the only exception being Binary resources (when you don't request a specific return format in the URL).
- **landingpage.html:** This is the template used for the base URL (GET [base]/). You can use this to return a custom "home page" for your FHIR server.

The "assets" folder is automatically served up by the server as well, so you can use it to include any resources (images, scripts, etc.) that your pages will need.

Generating Resource Narratives

Almost all resources can include a human-readable narrative describing the resource. This is an XHTML div element contained in the "narrative" node of the XML/JSON. More information: Narrative

This technology preview utilizes HAPI FHIR and Thymeleaf to help auto-generate resource narratives. Several templates are built-in, but you can override them and/or provide your own. The extensions/fhir/narratives/ folder in the installation directory contains all narrative-related resources that aren't built into HAPI. The narratives.properties file determines which HTML templates to use for specific classes. Two resources (Conformance and OperationOutcome) are overridden by default, and their template HTML pages are in that folder as well for reference.

As mentioned before, the FhirUtil class also has a convenience method for generating resource narratives.

For more information, check out the HAPI FHIR documentation: Narrative Generation

Debug Logging

The FHIR Listener logs out information for every request that comes in. In order to view these logs you can add the following to your log4j.properties file:

```
# FHIR Listener debugging
log4j.logger.com.mirth.connect.connectors.fhir.server = DEBUG
```

Example Channel

Next, check out an example channel we included to get you started: FHIR Technology Preview 2 - Example Channel

A 2 personas les gusta esto

2 Comentarios

HUMBERTO MANDIROLA

Hola Instale el conector de Escucha FHIR, me pidio reiniciar luego, reinicie el equipo y ahora no puedo hacer andar el Mirth Connect

me da un error de excepcion y no arranca, me podran ayudar

ouldNotLoadArgumentException[No se ha podido cargar la URL ni el archivo especificado:
C:\Users\MANDI\AppData\LocalLow\Sun\Java\Deployment\cache\6.0\16\5208e50-6a19df91]
at com.sun.javaws.Main.launchApp(Unknown Source)
at com.sun.javaws.Main.continueInSecureThread(Unknown Source)
at com.sun.javaws.Main.access$000(Unknown Source)
at com.sun.javaws.Main$1.run(Unknown Source)
at java.lang.Thread.run(Unknown Source)
Caused by: java.io.FileNotFoundException: C:\Users\MANDI\AppData\LocalLow\Sun\Java\Deployment\cache\6.0\16\5208e50-6a19df91 (El sistema no puede encontrar el archivo especificado)
at java.io.FileInputStream.open0(Native Method)
at java.io.FileInputStream.open(Unknown Source)
at java.io.FileInputStream.<init>(Unknown Source)
at java.io.FileInputStream.<init>(Unknown Source)
at com.sun.javaws.jnl.LaunchDescFactory.buildDescriptor(Unknown Source)
... 5 more

Eduardo Armendariz

Looks like you have to clear your java cache.