Patient corrections reference server

Note: in the middle of an update to the UI (specifically implementing separate Patient & EHR perspectives) so the documentation may not match the UI

Summary

A reference implementation that implements the Patient Corrections IG

Has both server (EHR end) and a client (patient end) User Interface

Expose a FHIR API (with some business logic) and UI to support patient correction requests.

API endpoint: http://corrections.clinfhir.com/fhir/

2 API's with business logic:

POST /Communication/ $process-medRecCxReq	Accepts a Bundle that must at least contain a Communication resource. If the 'about' element of the Communication is empty, then save the communication and create a Task resource. Otherwise save the Communication and update the task Returns a Communication
GET /Communication? about={communicationId}	Implement the 'about' query that returns all Communications that have an about reference to the initial Communication.

Other API's exposed that are proxied to the back end server

GET /Task?params	Calls the back end server with the params and returns the response. Most commonly useful when querying tasks for patients
GET /Task/{id}/_history	Returns a history bundle for the task

Currently only accepts Json

In this document:

- Patient is the person initiating the correction request, and responding to queries for further info. The term ‘client’ is also used. In theory this could be someone acting ‘on behalf of’ but this is not currently supported
- EHR is the system that the correction request is being made of
- The ‘trail’ represents the resources that are involved in a particular correction request, including their change over time.

Design work

My design notes

Use Cases

Patient submits Correction request

- Client app uses the $process-medRecCxReq operation to submit request bundle containing (at minimum) a Communication resource
-

EHR requests info

Patient responds to info request

EHR Accepts correction request

EHR Rejects correction request

Client (Patient) API

$process-medRecCxReq

Used by the client to send information to system
Is a bundle with one Communication and any other resources
? add a Patient as well? So the server can locate local record

Proposed behaviour to support denial disagree

- Uses transaction bundle - alternative is individual save of resources on server

- Receive bundle
- Extract Communication
    - Copy other resources to transaction bundle
- Validate Communication
    - Reject if invalid or missing
- If Communication is medReqCxReq (processing the original request)
    - If communication.about is empty (a new request)
        - Create Task
            - .code -> medRecCxReq
            - .status -> ready
            - .businessStatus ->new
            - .intent -> order
            - .focus -> communication
            - .description -> communication.topic or communication.payload.contentString
            - .for -> communication.subject
            - .requestor -> communication.requestor or communication.subject
            - .owner -> communication.recipient
            - .input -> communication
        - Update communication.about -> task
        - Copy task & communication to transaction bundle
        - POST transaction bundle & return
    - If communication.about is not empty (comment of original request
        - Validate
        - Retrieve task
- If Communication is medReqCxDenialDisagree (disagree with denial)
    - Retrieve the .about Communication
    - If that communication has a category of medReqCxReq (ie it was the original request)

        - Retrieve the Task that the .about communication refers to (this is the original Task
        - Create a new Task
            - Set .reasonReference -> original Task
            - Set .input to the newly received communication
            - Set the newly received communication.about -> new task
        - Add new task to transaction bundle
        - Add updated newly received communication to bundle
        - POST transaction bundle & return
    - If that communication has a category of medReqCxDenialDisagree (ie it is related to the disagree chain)
        - Add updated newly received communication to bundle
        - POST transaction bundle & return

Amendment accepted

Performed by EHR against the original request

- Retrieve primary (original) Communication
- Retrieve Task
    - From Communication.about
- Create Communication
    - .about -> original Communication
    - Save communication

Questions

Issue that arose during development

$process-medReqCxReq operation

How much detail in the spec. Can be complex as all interaction is through that operation (see description lower down).

How much Is part of the spec & how much vendor dependant

Other endpoints

Which other endpoints are part of the IG - or implementation dependant

How should Task status be managed

- In some cases automatically as Communication is processed. Eg Task is created (status=new), but what about transition to ‘in progress’
- In some cases manually - eg when EHR accepts / denies - closing task

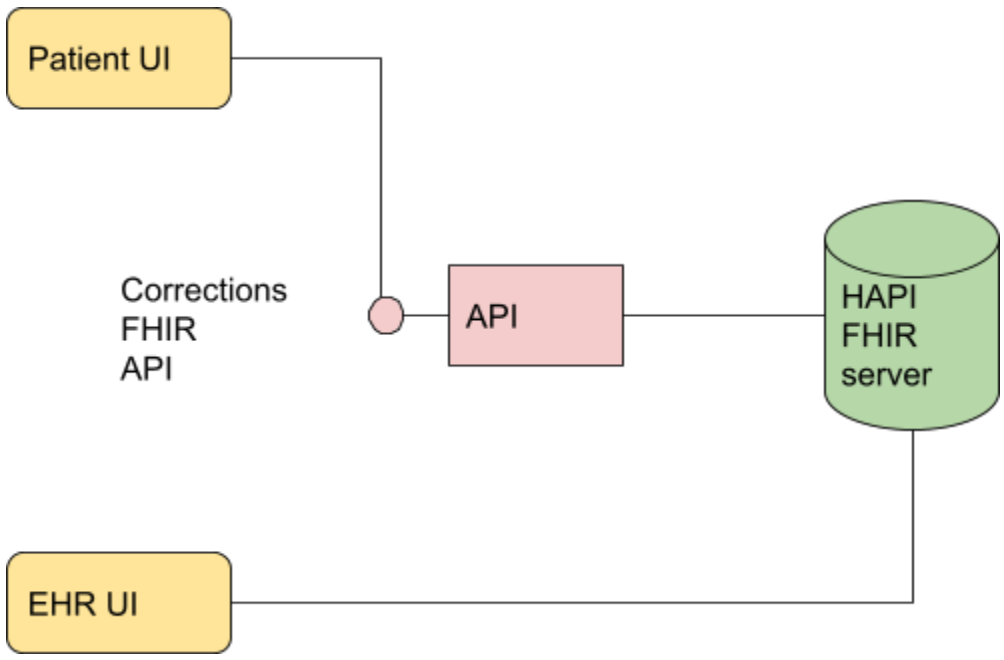
Confirm business statuses

When apply & relationship to task.status

Disagree Denial - can this be a conversation or only 1?

Architecture

Is a facade against a local instance of a HAPI server.



Endpoints

Url	Purpose
http://corrections.clinfhir.com/	The url of the User Interface to the app
http://corrections.clinfhir.com/fhir/	The root endpoint for the API exposed by the reference app. Details are described below
http://corrections.clinfhir.com:8091/baseR4/	The endpoint of the FHIR server that the reference app uses. Should really only query to this endpoint - making updates can upset the overall flow of the process. An exception will be to find (or create) a Patient resource to use Endpoint access may be restricted

Using the API

A client using the API

Some notes for the implementer

Basic flow

1. Find to create a patient on the FHIR server (http://home.clinfhir.com:8054/baseR4/). It is useful to set the identifier to facilitate debugging. As the custom 'about' query will only work if a patient has 50 or fewer Communication resources, using a different patient for each test run is advisable. I'll remove this limitation eventually
2. Create a Communication resource referencing that patient and with the .about element missing. Details of the elements are given below.
3. Add the Communication to a bundle, along with any other resources
4. POST the Bundle to the API endpoint (http://clinfhir.com/fhir/Communacation). Check the response - errors will be in an OperationOutcome. Some validation of the Communication is performed.
5. Use the UI (http://clinfhir.com/taskViewer.html) to view the Task created by the API as well as the Communication. Alternatively, query the FHIR server directly. There is alos a log view that may be helpful - see below.

6. Use the about query to poll for Communication resources. If the recipient is the patient, then it is a request for info (RFI). You can use the UI to generate the RFI Communication - see below.
7. To generate the response to the RFI, create an appropriate Communication and POST to the API endpoint. The .about element must be set to the initial Communication.
8. Repeat steps 3 -> 5 as required

Completing the overall correction request is done through the UI. The Task needs to be updated directly.

Finding conversations

If the client wishes to locate all the individual ‘conversations’ that they have initiated, the easiest is to query for Tasks for that patient. This works as the Task is automatically created when the first Communication for that conversation (.about is absent) is received.

```
GET Task/subject={patientId}
```

Getting Task versions

Each time there is an interaction with the API (ie posting a bundle containing a Communication at least to the operation, the associated Task resource is updated (Currently there is only a single Task per conversation as the ‘challenge’ workflow is not implemented).

To get a list of all the updates to the Task resource, use the version query:

```
GET Task/{id}/_history.
```

You will have needed to get the task id via the earlier query

Server interaction

Ie representing the request fulfiller. There are 2 options - the UI or direct POSTing of Communication resources

Direct POST

POST an appropriately crafted resource to the API. It’s up to you to set the elements correctly. Assuming the POST is accepted - and all the references are correct (especially the .about reference to the primary Communication), the Communication should be visible in the UI.

You can also access the FHIR server (http://home.clinfhir.com:8054/baseR4/) directly for debugging.

User Interface

Mocking a corrections conversation

You can use the UI to mock a corrections conversation between patient and EHR. The easiest way is to have the UI running on 2 browsers (can be in the same machine) with one browser representing the patient (and using the patient perspective in the UI) and the other the EHR perspective

You will need to refresh the browser manually to see the updated communications - plan is to automate this once the pub/sub functionality is implemented

Main UI

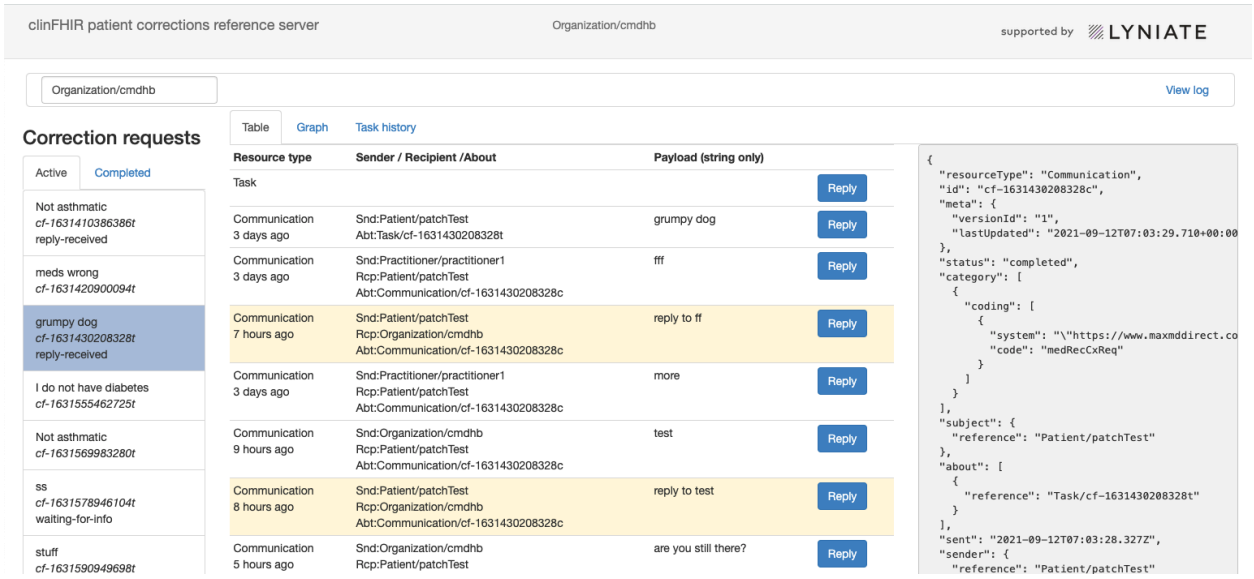
This is intended as an 'administrative' interface to the API. It has 2 key functions

- Displaying the resources that have been received and stored within the FHIR server. Kind of like the UI that an administration in the EHR might use to process correction requests.
- Creating 2 Communication resources, which are sent to the same API as external systems use. There are 2 sub-functions:
    - Those that the administrator would do - creating a query for further information and replying (if appropriate) to the responses to those requests.
    - Actions 'on behalf' of the patient. Although entered through this UI, they are created as if the patient had entered them (you could imagine that the patient asked the administrator to enter them on their behalf - eg entering data into the system from a phone call). This includes:
        - Creating a new correction request
        - Responding to a request for further info.

UI Address: http://clinfhir.com/taskViewer.html

The main screen of the UI presents the currently active Tasks (code = medRecCxReq, status is not completed) in a list to the left. Selecting a Task will display the Communifactions associated with the Task to the right.

The UI is, in effect, a Task centered system as a new Task is created for each new correction request (a Communication with .about absent)



Right now, all corrections are displayed. Eventually, different ‘EHR’s will be supported - allowing Correction requests to be targeted at different Organizations.

Main controls / tabs:

- Selecting a resource will display the Json to the right
- The ‘reply’ button that is to the right of a Communication will prompt the user for some text, then create a new Communication (via the API). The nature of the Communication depends on the sender
    - If the sender is the EHR (ie from the organization) then the user is in the role of the patient - they are replying to a Communication
    - If the sender is the patient (ie this is a response to an RFI) then the user is in the role of the EHR administrator
- The ‘Request more information’ button will create the RFI Communication - sender is the EHR, recipient the patient
- The ‘Close Task’ button will generate the Communication and set the task status to ‘completed’. This is an EHR role. A Communication is generated and saved through the usual API, and the Task is updated using a PUT to the Task endpoint.
- The graph tab displays a clinFHIR style graph of the resources in the interaction - including the patient. It is intended to show the references between the resources associated with this Task.

Note that the UI isn’t that great at refreshing (and can take a few seconds to do so) - just refresh the page if updates don’t seem to be there.

There is also currently a limit in the number of tasks that are shown in the left panel - only 50 Active and Completed Tasks are displayed. Working on that...

Task History

The Task history tab displays the history of changes to the currently selected Task. Each Communication will result in an update to the Task, so it's a useful timeline of activity. For each version the Task Json and the Communication json that caused that version to be created is shown (the link is Task.focus). Click the link to refresh the display.

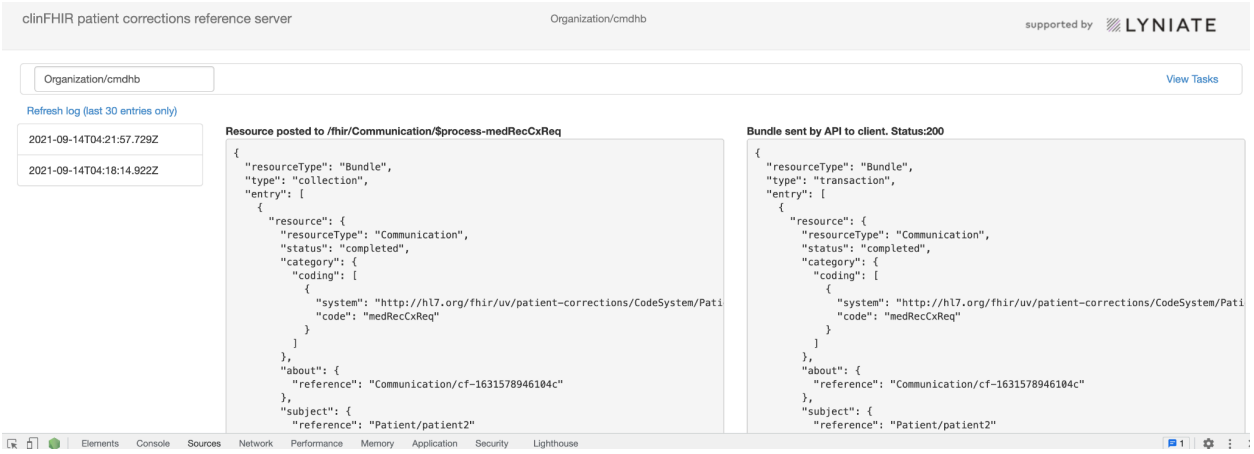
Debugging

The UI has a log option that is displayed by clicking the 'View Log' link to the upper right. Currently the logging is only done for bundles sent to the process-medRecCxReq operation, but more may be added later.

The last 30 logs are displayed, with a list to the left. Selecting the log displays details to the right. The list also shows any identifier in the Communication - this is to enable a vendor to 'tag' their resources so they can be (relatively) easily found.

Each log entry shows:

- The Bundle that was received by the operation
- The transaction bundle that is send to the FHIR server after business logic has been applied - eg the Task created
- The status of the transaction bundle processing



API details

POST Communication API

Both client and EHR can POST to this API . In this application the API is not simply ‘exposed by’ the EHR - rather it is an external entity with embedded business logic that both systems update.

Processing depends on the value of the .about element.

> If the .about is absent, then it indicates a new request for a correction (which will generally be made by the patient), and will result in the creation of a Task to manage interactions (exchange of Communication resources) concerning the correction request. This Communication resource will become the primary Communication for all subsequent interactions about this request.
>
> If .about is present, then it indicates that this Communication is part of the overall request - either a request for more information from the EHR, or from the Patient in response to such as request (or just supporting information)

Overall process of the API is:

- The bundle is received and parsed into Json. If this fails an OperationOutcome (OO) is returned. The only parts of the incoming bundle that are examined are the entry elements - other elements are ignored. This an incoming bundle only needs the entry elements (containing the .resource element). The entry.request element is ignored.
- A log entry is made with the submitted bundle. This is displayed in the log view of the UI. A correlationId is created, to allow this log to be associated with the log of the transaction sent to the fhir server (described shortly)
- If the bundle is not a FHIR bundle, the operation fails and an OO is returned
- A separate transaction bundle (the server bundle) is created that will be sent to the server if the request is valid.
- The Communication resource is located in the bundle. If there is none present the operation fails and an OO is returned. Other resources in the bundle are added to the server bundle and will be sent ‘as is’ to the server
- The contents of the Communication are validated (using specific code) - primarily those defined by the IG and/or needed for the API processing
- The Communication.about element is examined. Processing then forks depending on whether this is present or absent - details given next. In both cases, the server bundle is populated with the Communication and optionally the Task resources and sent to the FHIR server. A log entry for the server bundle is added to the log, including the correlation id.
- The status code from the FHIR response is examined:
    - If it is 200, then the Communication resource is returned
    - Otherwise, the FHIR server response - generally an OO - is returned.

.about is absent

The Communication resource will be assigned an id and saved on the server. Element values will be used to create the Task resource.

The Communication resource: (profile)

This will become the primary Communication for this trail

Element	Description
id	Will be ignored if present (shouldn't be on a POST). The ids are assigned by the API
status	Fixed to 'completed'
identifier	Not processed (just saved). Useful if you want to retrieve the communication directly from the FHIR server (as the id is created by the API) The identifier is also displayed in the log, to make it easier to find a log entry from a particular vendor.
category	Currently fixed to `medRecCxReq`. When 'log disagreement' is implemented there will be another.
reasonCode	Also fixed to `medRecCxReq`.
subject	The patient who the correction is about. This patient needs to exist on the FHIR server first.
sender	Also the patient.
payload	The message. Currently this must be in contentString.
recipient	Who the request is for
sent	This is set on the server to avoid timezone issues
topic	If present, will be the description in the Task. If absent, task.description will be from the payload contentString.
about	This is empty (or ignored by the server) in the submitted communication . It will be set as a reference to the Task that is created.
inResponseTo	Ignored when creating a new correction (.about is empty). For communications where .about is populated, it indicates that the communication is in response to a query for further information.

Other elements are simply saved on the FHIR server

When the Communication is processed, the API will add the .about element as a reference to the created Task

The generated Task resource (profile)

This is automatically created by the API

Element	Description and source
code	Currently fixed to medRecCxReq
status	Fixed to 'ready'
businessStatus	Set to http://clinfhir.com/cs/corrections|for-initial-review
intent	Fixed to 'order'
focus	A reference to the Communication that resulted in its creation
description	Copied from Communication.payload.contentString
for	Copied from Communication.subject
requestor	Copied from Communication.requestor, or from Communication.subject if requestor is absent.
owner	Copied from Communication.recipient. Need to add a default
input	A reference to the primary communication

This Task will be updated as overall processing of the correction request proceeds. It will be updated each time a new Communication is is received and the focus element will be a reference to that Communication (used when displaying the timeline)

.about is present

The about element is assumed to refer to the primary communication in the corrections trail. Ie no 'nesting' of communication resources - every communication in the trail will refer to the same primary communication.

The associated Task will be updated from the contents of the communication. This is applied in the order that the communication is received. The task is retrieved by first retrieving the primary

Communication (which is the contents of the .about element, and then retrieving the Task by examining the .about element of the primary Communication.

The following tables show the Task updates that are performed - the actual communication will be saved directly in the FHIR server.

Notes about the communication resource:

Element	Description
inResponseTo	Indicates that this communication is responding to another communication - presumably sent by the EHR requesting further information - though, in theory, it could also be a response by the EHR to a communication created by the patient. (It won't be a primary communication as the .about element will be present)
status	
recipient	Set to the EHR

Notes about the updates made to Task (resulting in a new version)

Task Element	Description & action
businessStatus	If communication.inResponseTo is set, then we assume that it is in response to a request for more info so set to: http://clinfhir.com/cs/corrections|reply-received Otherwise, assume it is a request for more info so set to: http://clinfhir.com/cs/corrections|waiting-for-info
status	Fixed to 'in-progress'
focus	A reference to the communication

GET Communication API

The only query (apart from the GET a single communication) implemented is the ‘about’ query which returns all the communication resources that have a reference to the primary communication resource. (It is always possible to make other queries directly to the underlying FHIR server if you need to).

Signature:

GET [host]/Communication?about={primary communication id}

The purpose of the query is to allow the client to retrieve all communication resources associated with the primary one - one may be a request for information (the .recipient element will be set to the patient)

Question: How does the client determine which communication resources require a response? If there are multiple communications then the .recipient alone won’t be enough…

Could an element on the ‘answer’ communication be updated by the app when saving? The client could then use that.

Implementing a subscription would help, but likely won’t be enough...

As the underlying FHIR server doesn’t support this parameter, this is done by the app as follows:

1. The communication resource (primary communication) that has the id of the about parameter is retrieved
2. All the Communication resources for the patient are retrieved (based on the .subject element)
3. The app iterates through all the resources to locate ones where the .about element refers to the primary communication, and adds them to the bundle.
4. The Patient is added to the bundle
5. The bundle is returned

Todo: Add the recipient resources (?generally Practitioner) resources to the list as well

This approach does have the limitation that the patient can’t have more than 50 communication resources in total as that is the maximum number that the FHIR server will return (without following the paging).

PUT Task API

The only use of this API is by the EHR to indicate that the correction thread has been finished and the process is complete. (This may change when the 'challenge' is implemented).

Only done by the EHR

Generally only the status will be updated

Dev stuff

Current behaviour - not supporting denial disagree

- Receive bundle
  - Reject if not bundle
- Create transaction bundle for server
- Iterate through bundle entry
  - If Communication (1 only) extract as communication resource
  - Copy others into transaction bundle - keep ids
- Extract communication .category
  - Reject if absent
- Set communication .sent to local server time
  - So consistent across timezones
- Validate other communication elements - using code - could use $validate against profile
  - Reject if errors
- Create communication .id
- If communication.about is present
  - Retrieve the primary task
    - Get the primary Communication from the .about then the task from that ones .about
  - If the category is medRecCxReq
    - Set task.status to 'in-progress' - is this the correct thing to do?
    - Set task.focus to the incoming communication
    - Add to the transaction bundle as a PUT (will update the task on the server
  - Add the communication to the transaction bundle
  - POST the transaction bundle to the server and return the Communication resource from the result
- If communication.about is absent
  - Create a new task (with id) and set communication.about to reference it
    - .code -> medRecCxReq
    - .status -> ready
    - .businessStatus ->new

        - .intent -> order
        - .focus -> communication
        - .description -> communication.topic or communication.payload.contentString
        - .for -> communication.subject
        - .requestor -> communication.requestor or communication.subject
        - .owner -> communication.recipient
        - .input -> communication
    - Add communication to transaction bundle
    - Add new task to transaction bundle
    - POST the transaction bundle to the server and return the Communication resource from the result
    -

Examples

Simple bundle with Communication

Assume all references exist on backend server

```
{
  "resourceType": "Bundle",
  "type": "collection",
  "entry": [
    {
      "resource": {
        "resourceType": "Communication",
        "status": "completed",
        "inResponseTo": [
          {
            "reference": "Communication/cf-1631656283330c"
          }
        ],
        "category": {
          "coding": [
            {
              "system":
"http://hl7.org/fhir/uv/patient-corrections/CodeSystem/PatientCorrectionTaskTypes",
              "code": "medRecCxReq"
            }
          ]
        },
```

```
        "about": {
          "reference": "Communication/cf-1631430208328c"
        },
        "subject": {
          "reference": "Patient/patchTest"
        },
        "recipient": [
          {
            "reference": "Organization/cmdhb"
          }
        ],
        "sender": {
          "reference": "Patient/patchTest"
        },
        "reasonCode": {
          "coding": [
            {
              "system":
"http://hl7.org/fhir/uv/patient-corrections/CodeSystem/PatientCorrectionTaskTypes",
              "code": "medRecCxReq"
            }
          ]
        },
        "payload": {
          "contentString": "yes I am"
        }
      }
    }
  ]
}
```

Bundle with contained resources

Things I plan to do sometime

Functionality

Test harness for server

Set of messages (communication resources) - send one, then query. repeat.

‘Rename’ comm to ‘messages’
    Especially for patient

‘Threaded’ display for Communications
    Tree type display

Remove ‘RFI’ button - can simply reply to the original Communication

Allow direct business status update
    Get vs from IG

Support different organizations
    Specify in recipient
    Build org list from active comms - server function - getActiveOrgs

Make graph object movement sticky

Processing new request
    Conditional update for patient

Creating a new correction request -
    have separate top level tabs - patient/EHR view with customized layout - perspective
    Patient tab
        Specify org (from active tasks)
            Have lookup of endpoint - vDir
        Allow attachments
        Enter text
        Subscribe to updates
            Eg email or text message number to receive notifications
        Need patient login
            For now, just list of patients with active tasks
                ? a query to Task (active, subject) - active & closed
        Specific display for messages
        New messages for me (ie recipient = patient)
            ?how to determine if already replied to?
                ?Reference to ‘replied’ to message (if it exists, you know it has been replied to

    EHR / System tab
        Select org to display tasks for

Subscriptions

Set up minimal subscriptions server