iRx Control Center - HealthShare API

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Contents

Part A - Communication

1. Overview

2. Main API

3. Security

4. Control API

Part B - HealthShare

1. Components & Configuration

2. Deployment

3. Participant Identification

1. Overview

This note defines an interoperability standard between iRx Reminder and HealthShare. Communication between the iRx Control Center (iRxCC) and HealthShare (HS) will occur over HTTP(S). Usually, requests will use POST methods to send JSON serialized JavaScript objects. Likewise, responses will contain JSON strings in the response body. Other HTTP(S) methods are supported for certain "control" functions and in such cases special- ized HTTP Headers are used.

2. API details

The endpoint for sending messages to HS will be

http(s)://<server>:<port>/irx

where:

server = hostname of HS server

port = port number for irx service

Functionally, the API is nothing more than POSTs to the above end-point. The payload of the POSTs will contain information as to the type of request along with any application data.

For example, here is a sample message:

{ participant: "1FA09",

study: "01AF0",

medication: "Liadla",

timestamp: 1369236083,

action: "taken" }

The 'action' property specifies the event and based upon this value the message routed through HS.

Responses are of the following form:

{ "action":"taken\_response",

"control\_id":"<env>;<guid>"

"timestamp":1369926525,

"status":"ok" }

Where the return 'action' value will be the input action with "\_response" appended. The 'timestamp' is in UNIX format. The 'control\_id' is generated and attached to all the transactions associated with the message in the system, and thus useful to track and trace messages in the system. <env> is the environment name and <guid> is a 16 byte globally unique identifier.

Value Tables

action status

------ ------

taken ok

missed error

skipped

Errors are encoded as;

{ "action":"taken\_response",

"timestamp":1369926525,

"control\_id":"<env>;<guid>",

"status":"error",

"error": { "code":1234,

"description":"Unable to match participant '1FA09'" } }

See Appendix A for list of error codes.

3. Security

Production systems require SSL. Testing environments can use unencrypted HTTP connections.

TODO: Do we need mutual certificate validation?

HS maintains it's own security infrastructure. All inbound requests will require a HS username and password be inserted in the HTTP headers.

TODO: What additional authentication/authorization does iRxCC need when getting messages from HS?

Every request requires an 'api-key' in the request headers. The name of this header is: API-KEY the value of which is determined at HS installation time.

Console access to the HS system is required to configure the api-key. Once the HS system is access, enter into a terminal session with HS, move to the "IRXSYS" environment and run;

IRXSYS>write $$apiKey^irx.SYS()

EB99A714-CC4F-11E2-BEFC-206610175400

To reset the api-key, call $$apiKey^irx.SYS(1) and a new key will be generated.

4. Control API

To facilitate certain system-level functionality such as debugging, logging, and configuration a set of features will be available by inserting special HTTP headers in requests. These features will respond to GET and POST HTTP requests where appropriate.

Each control function requires the special HTTP header X-IRX-CONTROL be present, the value of which determines the desired function. The values of the headers are NOT case-sensitive. Sample exchanges for some of the control functions are provided; omitted samples follow the same pattern.

1. Echo - Returns the message sent into HS, with a control-id inserted.

X-IRX-CONTROL:Echo

HTTP POST

Request (read content from file):

curl -v -X POST -d @$1 -u \_system:SYS \

http://server:57774/irx \

--header "Content-Type:application/json" --header "X-iRx-Control:echo"

POST /irx HTTP/1.1

Authorization: Basic X3N5c3RlbTpTWVM=

User-Agent: curl/7.24.0 libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5

Host: server:57774

Accept: \*/\*

Content-Type:application/json

X-IRX-API-KEY:EB99A714-CC4F-11E2-BEFC-206610175400

X-iRx-Control:echo

Content-Length: 107

{

participant: "1FA09", study: "01AF0", medication: "Liadla", timestamp: 1369236083, action: "taken"

}

Response:

HTTP/1.1 200 OK

Date: Mon, 03 Jun 2013 13:05:00 GMT

Server: Apache

CACHE-CONTROL: no-cache

CONTROLID: ENSDEMO;32E82B88-CC4E-11E2-8C49-206610175400

EXPIRES: Thu, 29 Oct 1998 17:04:19 GMT

HS-CONTROL: echo\_response

PRAGMA: no-cache

REQUESTID: 19

SET-COOKIE: CSPWSERVERID=063fc5dbbfe2ae19a9138071bffc2ab5a18b3c55; path=/; CONTENT-LENGTH: 118

Connection: close

Content-Type: application/json

{ "ControlId":"ENSDEMO;ACDA0A42-CC4E-11E2-8C49-206610175400",

"action":"taken",

"medication":"Liadla",

"participant":"1FA09",

"study":"01AF0",

"timestamp":1369236083

}

2. Flush - clear out the message log

X-IRX-CONTROL:Flush

HTTP GET

3. Config - get or set configuration

X-IRX-CONTROL:Config

If HTTP GET, then returns JSON with current configuration

If HTTP POST, then expects to read JSON from request body with desired configuration. Returns 'ok' or 'error' if validation does not succeed.

The configuration object is:

{

ControlCenterEndpoint : "http://irx:9393/healthshare",

SSLConfig : <name of HS SSL config

....more }

4. Logging - retrieve logs of messages sent

X-IRX-CONTROL:LOG

HTTP GET - return JSON array of requests/responses

Part B - HealthShare

1. Introduction

This section describes the implementation aspects of support of iRx. The primary role of HS here is to provide connectivity and interoperability between iRx and various healthcare entities (hospital, laboratory, etc). Such connectivity will be accomplished through industry standard mechanisms, such as HL7 or CDA whenever possible. Since the specific end-point systems HS will connect to are yet to be determined, they are treated in general terms with the pri- or 'standards-based' assumption in context.

2. Components and Configuration

There are 2 main data-flows within the system. First, messages will be sent from iRx into HS and then on to EMRs. Conversely, EMRs will generate messages that are of interest to iRx and HS will act as the intermediary forwarding on such messages when appropriate.

Each data flow is comprised of input & output end-points along with a set of processes for handling the data. All of these components are housed within HS.

Input end-points are Business Services. Output end-points are Business Operations. The logic in the middle is a Business Process.

Services, Processes, and Operations are terms from the messaging infrastructure of Ensemble upon which HS is based.

Data flows are decoupled in the sense that when iRxCC sends a message to HS the message is consumed, put on a queue, and then a response is sent to iRxCC. Asynchronous processes work these queues, applying business rules and transformations before finally sending outbound messages to HISs. The reverse flow is similar. When HISs send HS mes- sages, these are consumed, queued, and responses are sent. Other pro- cesses work the outbound queues appropriately before send messages on to iRxCC.

[] Flow 1: iRx --> HS --> HIS

Business Service: irx.ControlCenter

This service handles accepting HTTP requests as described above. Business Process: irx.OutboundProcess

This process handles message routing (which HIS cares about this mes- sage) and message translation (transform message to desired HIS stan- dard). A mapping of events to HIS transformations will be stored by this process. At deployment times various operations can subscribe to the HIS messages.

Business Operations: names to be determined by customers

There will be an operation defined for each HIS endpoint. These ops will be configured according to the HIS's desired protocol (tcp, ftp, file, etc).

[] Flow 2: HIS -> HS -> iRx

Business Service: t.b.d.

Same as Business Ops for Flow 1. Business Process: irx.InboundProcess

Map of events to iRx events

Business Operation: irx.ControlCenterSender Operation to send HTTP POST requests to iRxCC.

2. Deployment

One instance of HS will be deployed in the cloud with some to be determined provider (e.g. AWS). This instance of HealthShare will be configured with 2 namespaces (environments) per customer. 'Customer' here means a set of EMRs which send data to and from iRx. This pair of namespaces per customer will be comprised of a test and production environment, and named accordingly.

For example, if the customer is the University of Cincinnati, and short mnemonic name will be established, such as 'uoc'. Then the testing/development environment would be called 'uoc-test' and the production environment 'uoc'.

A unique environment called 'irxsys' will be established to house system-wide configuration and services to be shared among all the customer namespaces.

3. Participant Identification

The messages from iRxCC will contain de-identified codes that map to a particular participant (patient). EMR systems, however, will send names, medical record numbers, SSN's, etc. The system needs a way to reconcile these different identifiers.

For example, ParticipantMap

-------------------------------------------

iRx | identifier | source | type

-------------------------------------------

1FA09 | 123-23-1232 | HospitalA | SSN

1FA09 | 03-ADS-293293| HospitalA | MRN

2B23F | Smith, Jimmy | ClinicFoo | Name

2B23F | 555-32-2020 | HospitalB | SSN

Business rules can then be defined to inspect incoming HIS messages, pull data items out (from the standards the 'type' is known) and issue a query against the above structure to determine the iRxCC participant id.

There will be one unique instance of the ParticipantMap within HS. This means that the multiple customer environments will share the same mapping. Since the iRx identifier is globally unique though this will ensure consistency across multiple customers.

A control-level API is provided for the management of the participant map.

X-IRX-CONTROL:PARTICIPANT

To Add a mapping: POST Body:

{ participant: 1FA09,

identifier:'123-23-1232',

source:'HospitalA',

type:'SSN'

}

To retrieve mappings: GET

Default returns full map (all participants)

Optional, send body:

{ participant: 1FA09 }

to retrieve the mapping for one participant

To delete mapping: DELETE

Requires body:

{ participant: 1FA09 }

which specifies which participant to delete the mapping for.

Appendix A - Error Codes

Code & Description

UnknownError - Some error without a specific code

UnknownParticipant - Can't locate participant in the map

HSProcessFail - Process failure (in the event HS process is down, the service could not submit the message)

UnknownAction - Invalid action sent - unable to route message

Configuration - Something is not configured right, details contain more specifics

Irx Ensemble REST Service Implementation Design

Extends EnsLib.HTTP.Service, EnsLib.Util.JSON

OnProcessInput()

1. Process Headers – if any “control” headers route to Control handler
2. Fetch request body
3. Build response from request
4. Send request to irx.Operation which will persist the request and make available for further downstream processing.
5. Return response

The irx.Operation is the key to the asynchronousness of things. Basically iRx can send whatever messages into the operation and they are stored. Later ensemble components can process the queue and do things like generate HL7, etc.

For reverse data-flow, from EMRs to iRx, a similar concept is used. Various ensemble post messages to a service and these are queued. An outbound process sits on this queue and sends items to an OutputOperation which calls a REST api in the iRx control center.

iRx ----- HS ----- EMRs

---> inbound

🡨 outbound

irx.InboundRESTService

irx.InboundOperation

irx.OutboundRESTService

irx.OutboundOperation

irx.ControlProcessor (handles all X-IRC-CONTROL header operations)

irx.ParticipantMapOperation – manages the participant map

Queue is a persistent class irx.ActionQ

irx.ActionQ

------------

Source – from who – in the inbound case this will be ‘irx’ for the outbound case it can be the source EMR, really any string (index on this field)

Participant – who (index on this)

Timestamp – when

Action – what

Raw – raw data from iRx

Scenario A: Generate PDF report for all the stuff a patient did during a day.

Select \* from irx.ActionQ where Source = ‘irx’ Timestamp = $today and Participant = ?

Then, from these rows build out a report and embed it into an HL7 message