# HL7 Gateway Programming Exercise

This exercise is designed to let the candidate showcase their skills with both JavaScript and Mirth development, as well as knowledge of HL7.  The exercise involves developing two distinct components that collectively form a rudimentary HL7 gateway solution. Each component should be designed as two channels within Mirth. Component #1 is an HL7 producer Mirth channel.  Component #2 is an HL7 consumer Mirth channel.  Details for both components follow below.

Several criteria are required for a successful submission of these components:

* The candidate must make both Mirth channels available for review via a publicly accessible repository (e.g., GitHub).
* The candidate must be prepared to demonstrate and discuss the two components working together in a live demonstration via screenshare.
* Each component must be the candidate’s own work.  It is acceptable to use open-source libraries and frameworks but the principal functionality of each component must be original work.
* The candidate should be mindful that the goal of this exercise is to demonstrate not only technical fluency, but also their level of professionalism as a software engineer.  The candidate should expect that their work will be assessed with the same rigor as a pull request.

## Component #1 – HL7 Message Producer

This component should ingest a delimited file and create HL7 messages from each row of data. A minimum of 5 HL7 messages should be produced. The messages can be rudimentary in terms of HL7 structure and fictionalized data: it must have a message header, patient demographics, and visit information. It may include any additional segments that the candidate chooses to use to highlight the functionality described below for Component #2.

Component #1 should then send these HL7 messages along with the channel name to Component #2. The channel name of Component #1 should be available to retrieve in Component #2. The method at which to send them will be up to the candidate. As always, creativity is a plus!

The messages that are produced should have sufficient variability to allow Mirth channel features to be employed including filtering, transforming, and dealing with successful and errored message sources.

## Component #2 – HL7 Consumer

This component consumes the HL7 messages written by Component #1 and must be implemented as a Mirth channel.  (The candidate must download and install the freely available NextGen Connect Integration Engine to complete this task.)

The channel must demonstrate that it uses the correct mechanism as the message source, with message filtering and transformation applied to that source.  The channel must have at least one destination as well.  The details of the destination(s) are up to the candidate, but one suggestion would be to write to a custom file.  More interesting destinations are always appreciated, creativity is a plus! The only requirement is to display the channel name of Component #1 is some aspect when sending the files through the destination.

## HL7 Questionnaire

1. What is an ADT message?

🡪 ADT stands for Admit Discharge and Transfer. This interface contains patient demographic related information along with Insurance and Diagnosis. There are different trigger events present in ADT message that tells us if the patient is admitted, new patient got registered, Patient demographic updates so on and so forth. Mostly used or known trigger events are

A01- Patient admitted

A04: New patient registered

A08: Patient demographics update

These Trigger events will be present in MSH segment of ADT message to be more precise MSH 9

1. What are HL7 Separator characters?

🡪

Field Separtor: |

Component Separator : ^

SubCOmponent Separator: &

Repetion Character: ~

1. What function would you use to update the date and time to current in MSH segment in Mirth?

🡪Dateutil.getCurrentDate(‘yyyyMMddhhmmss’): This will provide the current datetime in the format year,month,date,hour,minutes and seconds

1. A sending application sends all ADT types. The client only accepts ADT-A01. How would you handle that in Mirth?

🡪 Mirth provide mechanism to filter the messages based on Message Type present in MSH segment. Following way we can add the filter to only accept message that are ADT A01 and filter out rest other messages

1)GO to channel and select source

2) Under channel task click on edit filter and then add new Rule

3) Then we can write a Javascript code to accept messages if MSH9.1 is ADT and MSH 9.2 is A01.

1. Where in Mirth would you write code that is used in multiple channels?

🡪 Code Template

1. Where would you set a certain type of data for a channel to receive and send out?

🡪 When you create a channel under Summary tab we will see Datatype . Click on Set Data Types and then select what type of data will be inbound and outbound

1. How would you allow the code, written in question 5, to be used by a channel?

🡪 Go to channels and then select edit code templates. Select the library that you want to apply to the channel and then select the channel that will use the code template from channels list in the same window.

1. Inbound and outbound message templates are used for what?

* Inbound and outbound message templates is the part of the transformer. The inbound message Template helps to create the Message tree that shows how is the structure of the inbound message is . You can refer to the inbound message structure using keyword msg . Similarly outbound message templates will provide the structure of the message that will go out from the transformer. The structure of the outbound message is referred by keywork tmp.

1. How would you map a variable without the use of JavaScript?

* Go to Transformer , Create new step and select mapper under type instead of javascript.

1. Using JavaScript, loop through any 3 element array and log/write out the individual elements.

🡪 Looping through PID 3 values

For each(pid3 in msg[‘PID][‘PID.3’]){

Logger.info(‘All PID3 values’+pid3);

};

1. What is the Velocity Template Language? Can you provide an example?

* Not used . Willing to learn.