Lunaford

Training Lab Book

Version 1.2
December 2009

Contents

Lab 1 Manually Parse an HL7 Message

Objectives

In this lab, you will manually parse an HL7 message. You are given an HL7 message and a list of fields to find. For each field indicated, find the segment in the message, then count from the segment name the fields, then field components indicated by the HL7 field notation. Don't forget that for the MSH segment, the first field is the field delimiter (|).

Estimated Time to Complete

2-3 minutes.

Step-By-Step

For the following HL7 message and requested fields, manually parse the message to find the values for the fields.

MSH|^~\&|MIRTHCONNECT|SMMC|||200907131519||ORM^O01|12345678|P|2.4|||AL|NE<CR>
PID|1|4223161584^^^Mirth^PN|4223161584|4223161584|Fry^Philip^J^^^||19720515|M|||
123 Von Karman^Apt. 10B^Irvine^CA^92612||949 555-1234|||||815-16-2342|<CR>
PV1|1|O|SMMC|OP|||ShephardJ^Shephard^Jack^G^^^MD|BurkeJ^Burke^Juliet^F^^^MD||CARE||||
PHYSICIAN||||CLINIC||BC|||||||||||||||||SMMC||REG|||20090712|||||<CR>
ORC|XO|801887.001|||R|N|||||||||<CR>
OBR|1|801887.001||MRS^Shephard^Jane MRI W/&W/O CONTRAST^70553|
|200907131230|||||||||ShephardJ^Shephard^Jack^G^^^MD||MR^20061213-0007|
|||||||1^^^200907131230^^R|||||||<CR>
OBX|1|NM|84295^SODIUM^GH|1|145|mmol/L|||||F|||20090422152300|GH<CR>
OBX|2|NM|84132^POTASSIUM^GH|2|5.2|mmol/L|||||F|||20090422152300|GH<CR>

MSH.4 (Sending Facility) SMMC

MSH.9 (Message Type) ORM^O01

MSH.12 (HL7 Version) 2.4

PID.5 (Patient Name) Fry^Philip^J^^^

PID.11.3 (Patient Address City) Irvine

PV1.44 (Admit Date/Time) 2009 07 12

ORC.2 (Order Number) 801887.001

OBX.3.2 (Observation Text) Sodium
Potassium

Results

MSH.4 = SMMC, MSH.9 = ORM^O01, MSH.12 = 2.4, PID.5 = Fry^Philip^J^^^, PID.11.3 = Irvine, PV1.44 = 20090712, OBX.3.2 = SODIUM *and* POTASSIUM

Lab 2 Start/Stop Mirth Connect Server

Objectives

In this lab, you will start and stop the Mirth Connect Server, using both the Mirth Connect Server Manager, as well as from the command line. The Mirth Connect Server should already be running as a service on your computer.

Estimated Time to Complete

5 minutes

Tools Used

- Mirth Connect Server Manager
- Windows Command Prompt

Step-By-Step

1. Open the Mirth Connect Server Manager by double-clicking its icon in the Windows notification area (system tray)

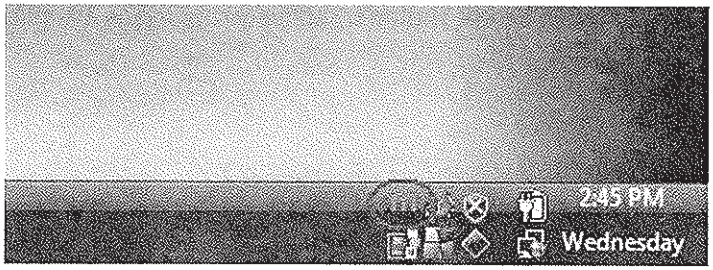


Figure 2-1 Mirth Connect Server Manager Icon in Notification Area

2. The Mirth Connect Server should show that the service is currently running (by the fact that the Start button is disabled). Click the Stop button to stop the server.

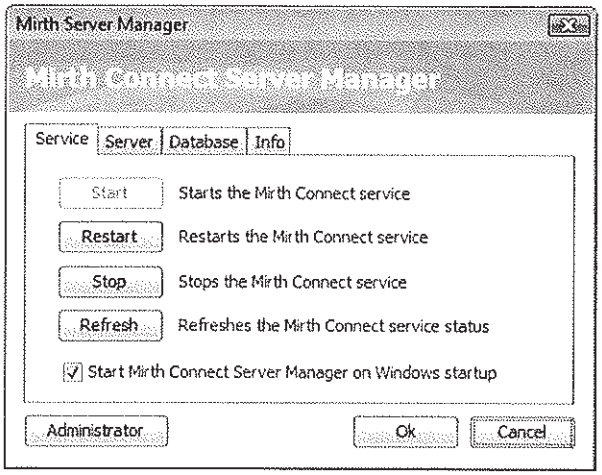


Figure 2-2 Mirth Connect Server Manager - Service Tab

3. After a few seconds, a notification window should appear indicating that the service was successfully stopped. Click OK
4. Open a Command Prompt window (Windows menu > Programs > Accessories > Command Prompt)
5. At the command prompt, change to the Mirth Connect installation directory (usually C:\Program Files\Mirth on a 32-bit computer or C:\Program Files (x86)\Mirth on a 64-bit computer). For example:
   `cd "\Program Files (x86)\Mirth`
6. To start the Mirth Connect Server, type `mirth` at the command prompt and press Enter. This will begin the startup process. Once the server is running, you will see a line such as:
   `Running Java 1.6.0_15 on Windows Vista (6.0, x86) with charset windows-1252.`
7. Stop the Mirth Connect Server by pressing Ctrl-C with the Command Prompt window active. Once the server has shut down, you will be returned to the command prompt Ctrl-C x2
8. Open the Mirth Connect Server Manager once again.
9. Restart the Mirth Connect Server by clicking the Start button
10. Once the server has finished starting, you will see a notification window indicating that the server started successfully. Click OK

Results

After completing each of the above steps, the Mirth Connect Server should by actively running as a service.

Can also stop/start from control panel → services

Lab 3 Launch Mirth Connect Administrator

Objectives

In this lab, you will launch the Mirth Connect Administrator using Java Web Start from any web browser. As this will be the first attempt to launch the application after installation, you will also need to customize the default user information after logging in.

Estimated Time to Complete

3-4 minutes.

Tools Used

- Any web browser, such as Internet Explorer or Firefox
- Mirth Connect Administrator

Step-By-Step

1. Launch Mirth Connect Administrator from a web browser using Java Web Start
   a. Open a web browser. Both Internet Explorer and Firefox should be installed on your computer.
   b. In the browser's address field, type the following URL:
      http://localhost:8080
   c. Press Enter key. This will display the Java Web Start page for the Mirth Connect Administrator (Figure 3-1).

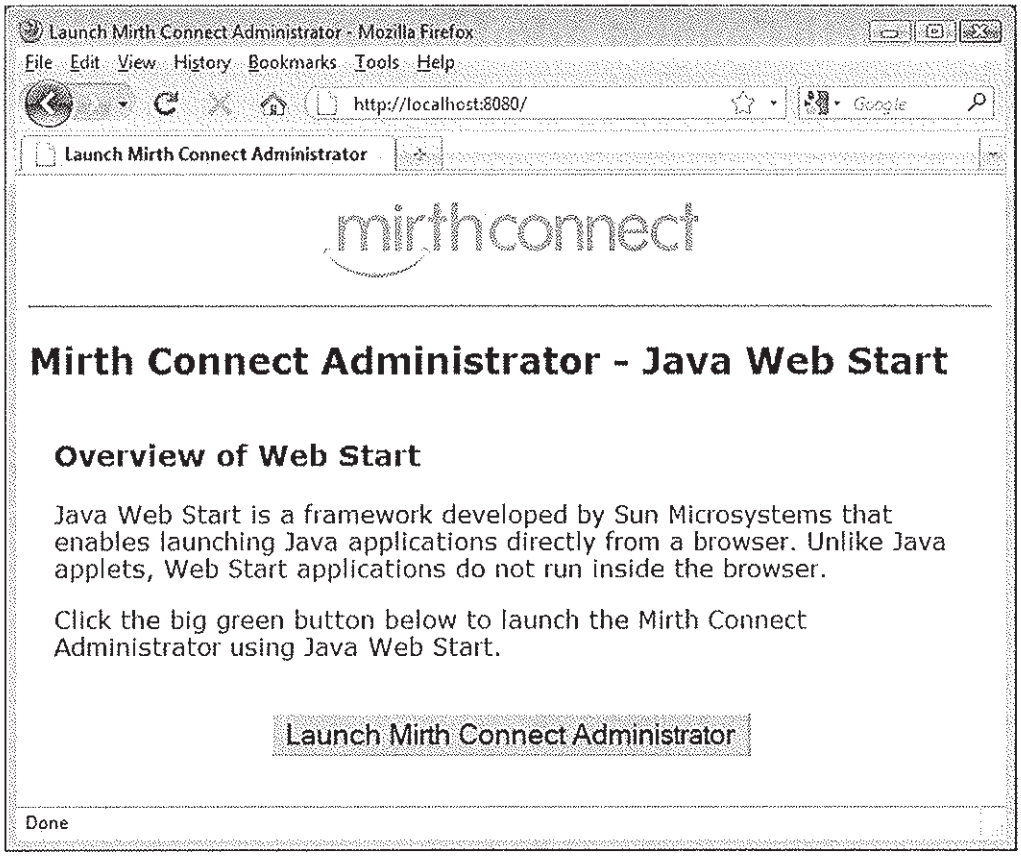


Figure 3-1 Java Web Start Page

d. Click the green Launch Mirth Connect Administrator button.
e. As the application is downloaded and started, you will likely see one or more warnings appear indicating that the application's digital signature cannot be verified and asking if you want to run the application (Figure 3-2). In each case, click the Run button.

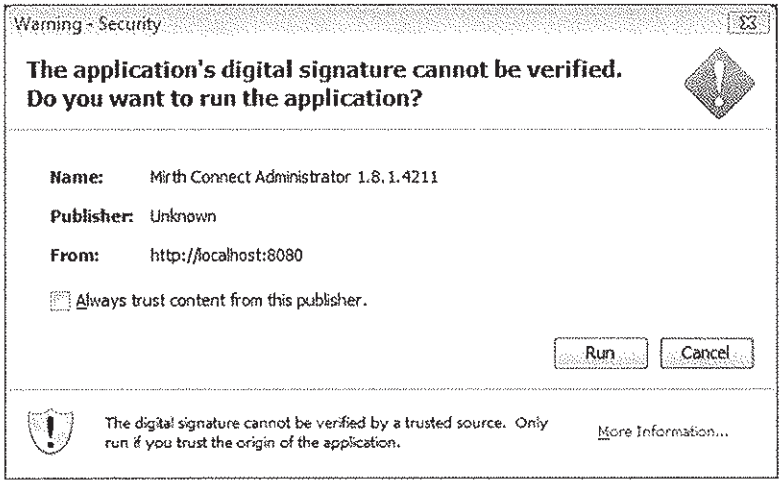


Figure 3-2 Digital Signature Warning

2. Login to the Mirth Connect Administrator

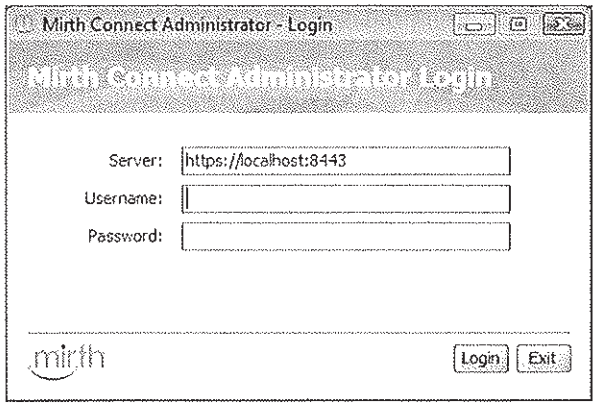


Figure 3-3 Mirth Connect Administrator Login

a. For the Server field, leave the default address: https://localhost:8443
b. For the Username field, type `admin`
c. For the Password field, type `admin`
d. Click the Login button

3. In the Welcome to Mirth dialog, customize the default user information by entering information in each of the required fields (marked with a red asterisk). Note that you can keep the default username and password as admin/admin by re-entering those values in the corresponding fields, or you can enter a different username and password to overwrite admin/admin.

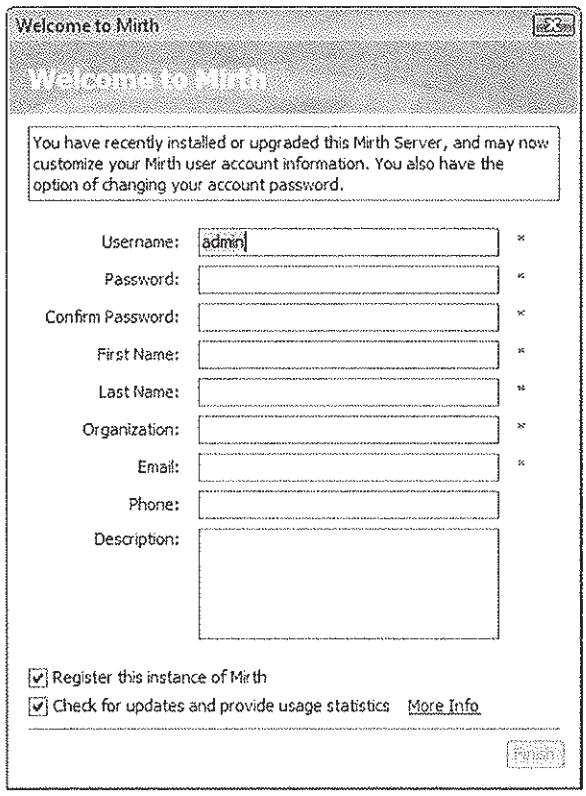


Figure 3-4 Welcome to Mirth Dialog

a. In the Username field, type `admin` to keep the username as admin, or type a new username to change it
b. In the Password field, type `admin` to keep the password as admin, or type a new password to change it
c. In the Confirm Password field, re-type the same password that you entered in the Password field
d. Note your username and password for future reference (optional)
   Username: _____
   Password: _____
e. In the First Name field, type your first name
f. In the Last Name field, type your last name
g. In the Organization field, type the name of your organization or company
h. In the Email field, type your email address
i. Click the Finish button

Results

You should now be logged into the Mirth Connect Administrator. If you view the Users list by clicking Users in the Mirth Connect menu (top, left-hand side of the Administrator window), you should see a single user with the information entered in the Welcome to Mirth dialog.

Lab 4 Create and Deploy a Simple Channel

Objectives

In this lab, you will:

- Create a simple channel with the default source and destination connectors (LLP Listener and File Writer, respectively), using the default connector settings whenever possible
- Save and deploy the channel
- Send a test message using HL7 Inspector
- Verify the results in both the Dashboard and Channel Messages views, as well as in the HL7 Inspector

Estimated Time to Complete

10-15 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector
- Windows Explorer

Step-By-Step

1. Create a New Channel
   a. In the Mirth Connect menu, click Channels to switch to Channels view.
   b. In the Channel Tasks menu, click New Channel.
2. Set the channel's Summary tab properties
   a. In the Channel Name field, type a name for the channel (for example, *Simple Channel - LLP to File*). Note that only alphanumeric, space, hyphen and underscore characters are allowed.
   b. Leave all other settings on the Summary tab as their defaults.
3. Set the source connector properties
   a. Click the Source tab near the top of the Channel view to display the source connector properties
   b. Leave all settings as their defaults
   c. Note the default Listener Port value of 6661. This will be used to configure the HL7 Inspector to send to the channel.
4. Set the destination connector properties
   a. Click the Destinations tab near the top of the Channel view to display the destination connector properties
   b. Rename the default destination connector by double-clicking the cell that displays "Destination 1", selecting the text, typing a new name (for example, Output File) and pressing the Enter key.
   c. In the Directory field, type `/Mirth Training/outbox`
   d. Define the output file name as a unique ID followed by a .txt extension
      i. From the Destination Mappings box to the right, drag-and-drop the Unique ID mapping into the File Name field by clicking Unique ID, and while holding down the left mouse button, drag the mouse cursor to the File Name field then release the mouse button. This inserts the ${UUID} placeholder.

ii. After the ${UUID} placeholder, type `.txt`
iii. The value in the File Name field should now be ${UUID}.txt

e. For the Template field, drag-and-drop the Encoded Data destination mapping using the same method as the Unique ID mapping in the previous step

5. Save the channel
   a. Click Save Changes in the Channel Tasks menu
6. Deploy the channel
   a. Click Channels in the Mirth Connect menu to return to Channels view
   b. Click Deploy All in the Channel Tasks menu
7. Send a message using HL7 Inspector
   a. Open HL7 Inspector from the Windows Programs menu
   b. If a dialog appears asking to check for updates, click No
   c. Open HL7 Inspector's Message Sender pane by clicking the toolbar button with the red, upward-pointing arrow
   d. Click the button with the screwdriver and wench in the toolbar for the Message Sender pane to open the Send Options dialog
   e. In the Send Options dialog, replace the default port value of the 2100 with 6661 (the default port for the LLP Listener connector)
   f. Click OK
   g. Open a file to send by selecting File Open... from the File menu and browsing to one of the HL7 files in the C:\Mirth Training\Messages\HL7 folder
   h. In the Import Options dialog that opens when opening a file, click the OK button
   i. Click the message in the message tree (in the top pane of the HL7 Inspector window) to select it
   j. In the Message Sender pane, click the Send button (blue triangle) to send the message

Results

To verify the message was sent correctly, you should now see the following results:

- HL7 Inspector received an ACK message in response to the message sent
- The statistics displayed for the channel in the Administrator's Dashboard should show one message received and one message sent
- The Channel Messages view for the channel (click the channel in the Dashboard, then click View Messages from the Status Tasks menu) should show two entries: one for the source connector and one for the destination connector. The status for the source connector should be TRANSFORMED, while the status for the destination connector should be SENT.
- In the /Mirth Training/outbox folder, you should see a file with a name consisting of a unique ID followed by a .txt extension. The contents of this file should be the HL7 message sent by HL7 Inspector.

Lab 5 Create Filters with Rule Builder Steps Using Drag-and-Drop

Objectives

In this lab, you will create a new channel to demonstrate the use of filters with Rule Builder steps to not only filter out unwanted messages, but to route messages to different destinations based on data within the message. The channel will have an LLP Listener source connector in two File Writer destination connectors. Rule Builder filter steps will be added to the source connector to filter out any messages that are not ADT messages, as well as any messages that are not being sent from "Hospital A" or "Hospital B".

Each File Writer destination connector will write to a different folder based on the sending facility, using Rule Builder filter steps to essentially route the message to the correct folder by filter out messages where the sending facility does not correspond to the folder to write to.

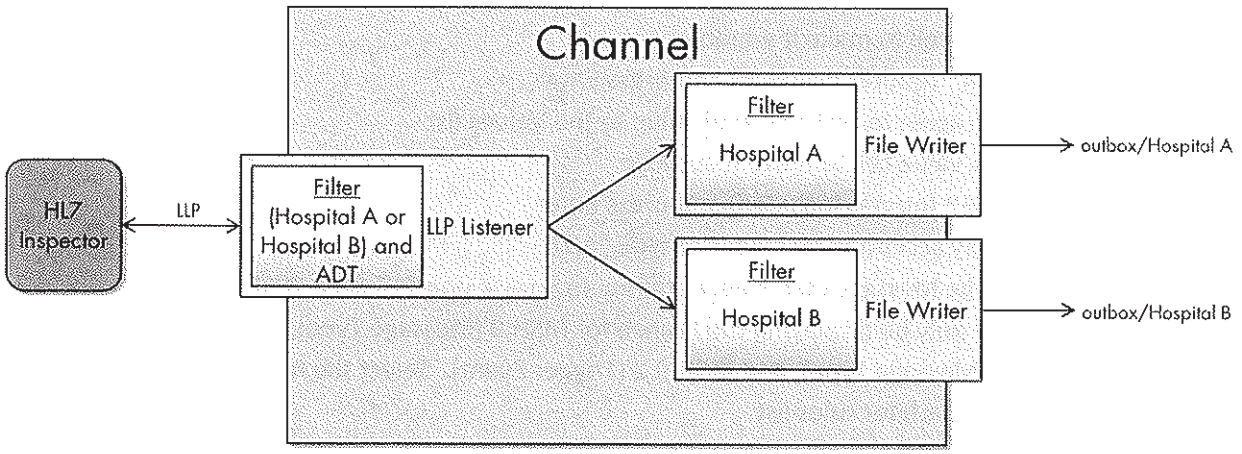


Figure 5-1 Filters Channel Flow

Estimated Time to Complete

15-20 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector
- Windows Explorer

Step-By-Step

1. Create a new channel named *Filters Drag-and-Drop*
2. Configure the LLP Listener source connector
   a. Click the Source tab to view the source connector settings
   b. Since the default Listener Port value of 6661 was used by a channel in a previous lab, change the setting to a new value, such as 6662
3. Define an inbound message template for the source connector

a. In the Channel Tasks menu, click Edit Filter
b. In the right-hand pane of the window, click the Message Templates tab
c. Click the Open File button (button with file and magnifying glass)
d. Browse to and open any ADT message in the C:\Mirth Training\Messages\HL7 folder

4. Create filter rule for only accepting messages with a sending facility value of "Hospital A" or "Hospital B"
   a. Click the Message Trees tab (to the immediate left of the Message Templates tab) to view the Inbound Message Template Tree
   b. In the Filter field, type `sending facility` and press Enter
   c. The tree should now only be displaying the branch for MSH.4.1 (Sending Facility). Look for the lower-most node on the tree, designated by a green dot, and situated below the "MSH.4.1 (Namespace ID)" branch
   d. Select this node with your mouse and drag and drop it into the filter rule list area at the top center part of the window
   e. In the rule detail area (below the filter rules list), select the Equals radio button
   f. Click the New button to add a new value to the Values list
   g. Double-click the value cell to make it editable
   h. Type `"Hospital A"` (quotes included) and press Enter
   i. Click the New button again to add a second value to the Values list
   j. Double-click the new value cell to make it editable
   k. Type `"Hospital B"` (quotes included) and press Enter
5. Create a filter rule for only accepting messages of type "ADT"
   a. In the Filter field of the Messages tab tree, enter `message type`
   b. The tree should now only be displaying the branch for MSH.9 (Message Type), which includes two sub-branches, MSH.9.1 (Message Type), and MSH.9.2 (Trigger Event). Select the node containing "ADT" and drag and drop it into the filter rules list
   c. Click the Equals radio button
   d. Click the New button to add a new value to the Values list
   e. Double-click the value cell to make it editable
   f. Type `"ADT"` (quotes included) and press Enter
   g. Click Back to Channel in the Mirth Views menu to return to the Edit Channel view
6. Configure File Writer destination for "Hospital A"
   a. Click the Destinations tab to view the channel's destination connectors
   b. Rename the default destination from "Destination 1" to "Hospital A"
   c. In the Directory field, enter `/Mirth Training/outbox/Hospital A`
   d. Define the output file name as a unique ID followed by a .txt extension, as was done in Lab 4
   e. In the template field, set the output to the message encoded data by dragging-and-dropping the Encoded Data mapping from the Destination Mappings box
7. Define an inbound message template for the Hospital A destination connector
   a. In the Channel Tasks menu, click Edit Filter
   b. In the right-hand pane of the window, click the Message Templates tab
   c. Click the Open File button (button with file and magnifying glass)
   d. Browse to and open any ADT message in the C:\Mirth Training\Messages\HL7 folder

8. Add a filter rule to the Hospital A destination connector to only accept messages with a sending facility value of "Hospital A"
   a. Click the Message Trees tab to view the Inbound Message Template Tree
   b. In the Filter field, type `sending facility` and press Enter
   c. The tree should now only be displaying the branch for MSH.4.1 (Sending Facility). Look for the lower-most node on the tree, designated by a green dot, and situated below the "MSH.4.1 (Namespace ID)" branch
   d. Select this node with your mouse and drag and drop it into the filter rule list area at the top center part of the window
   e. In the rule detail area (below the filter rules list), select the Equals radio button
   f. Click the New button to add a new value to the Values list
   g. Double-click the value cell to make it editable
   h. Type `"Hospital A"` (quotes included) and press Enter
   i. Click Back to Channel in the Mirth Views menu to return to the Edit Channel view
9. Create a second File Writer destination that will write to a file only messages that have a sending facility value of "Hospital B"

   Because the connector settings and filter rules for this destination will be identical to those of the first destination, you can just clone the first destination and change all instances of "Hospital A" to "Hospital B"
   a. In the Channel Tasks menu, click Clone Destination
   b. When prompted to save your channel before cloning the destination, click Yes
   c. Select the new destination, "Destination 1" in the destinations list
   d. Rename the destination to "Hospital B" (Note: after pressing Enter to accept the new name, the first destination, "Hospital A", may become the selected destination. If this happens, make sure you re-select the "Hospital B" selection)
   e. In the Directory field, change the destination folder to:
      `/Mirth Training/outbox/Hospital B`
   f. Click Edit Filters in the Channel Tasks menu
   g. Double-click the value cell containing `"Hospital A"`, change it to `"Hospital B"` and press Enter to accept the change
   h. Click Back to Channel in the Mirth Views menu to return to the Edit Channel View
10. Save the channel
11. Deploy the channel
12. Send some messages to the channel using HL7 Inspector to test the filter settings
    a. In HL7 Inspector, change the port to which to send to the Listener Port value in your LLP Listener (e.g., 6662). Click the button with the screwdriver and wrench to change this setting
    b. Open the following message files in HL7 Inspector (you will need to open them one-at-a-time):
       - ADT-A08 - Hermes Conrad - Hospital A.hl7
       - ADT-A08 - John Panucci - Sacred Heart.hl7
       - ADT-A08 - Lars Fillmore - Hospital B.hl7
       - ORM-O01 - Ogden Wernstrom - Hospital B.hl7
       - ORU-R01 - Hermes Conrad - Hospital A.hl7

c. Select all of the imported messages by clicking the first one, holding down the Shift key, and clicking the last one
d. Click the Send button to send all of the selected messages to the channel

Results

The statistics for the channel in the dashboard should display the following counts: 2 received, 5 filtered, 2 sent. Note that the 2 messages received/sent plus 5 messages filtered totals 7, which is two more than the number of messages sent by HL7 Inspector. This is because the filter count includes messages filtered out by the source connector as well as those filtered out by the destination connectors. For the two messages that were accepted by the source connector, each was filtered by *one* of the destination connectors.

The following table shows the filter status and reason for each of the five messages sent.

Message File Name	Source Connector Reject/Accept	Reject /Accept Reason
ADT-A08 - Hermes Conrad - Hospital A.hl7	Accepted	Both message type (ADT) and sending facility (Hospital A) requirements met
ADT-A08 - John Panucci - Sacred Heart.hl7	Rejected	Sending facility (Sacred Heart) requirement not met
ADT-A08 - Lars Fillmore - Hospital B.hl7	Accepted	Both message type (ADT) and sending facility (Hospital B) requirements met
ORM-O01 - Ogden Wernstrom - Hospital B.hl7	Rejected	Message type (ORM) requirement not met
ORU-R01 - Hermes Conrad - Hospital A.hl7	Rejected	Message type (ORU) requirement not met

For the three messages that were filtered by the source connector, in the HL7 Inspector you should see the ACK message received with an acknowledgement code of AR (application reject). Likewise, for the two that were successfully received and sent, you should see the ACK message with an acknowledgement code of AA (application accept).

In the Channel Messages view for the channel, you should see a status of FILTERED in the Source row for the three messages rejected by the Source Connector. For the two messages accepted by the source connector, you should see a status of TRANSFORMED for the Source row. For the two destination connector rows, you should see one with a status of SENT and the other with a status of FILTERED, depending upon the value of the MSH.4.1 (Sending Facility) field.

Finally, in the /Mirth Training/outbox folder, you should now see two new folders, Hospital A and Hospital B. Each folder should contain one file, with the contents of each file the message with the Sending Facility value corresponding to the folder in which it was written.

Lab 6 Create Transformers with Mapper and Message Builder Steps Using Drag-and-Drop

Objectives

In this lab, you will create a new channel to demonstrate the use of transformer steps to both extract data from the inbound message as well build a new outbound message. The channel will have an LLP Listener source connector and two File Writer destination connectors. The first File Writer destination will use Mapper transformer steps to extract patient data from the inbound message and create a plain-text patient report. Included in the transformer steps, you will use some JavaScript code to convert a date string format, and use regular expressions for string replacement. The second File Writer destination will use an XML outbound template and Message Builder steps to generate an XML document file containing patient data from the inbound message.

Estimated Time to Complete

25 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector
- Windows Explorer
- Any text editor, such as Notepad++

Step-By-Step

1. Create a new channel named *Transformers - Text Report and XML*
2. Update the Listener Port value for the LLP Listener to a new value, such as 6663, since the default value was used in a previous lab
3. Configure first File Writer destination
   a. Rename the destination to "Patient Report"
   b. In the Directory field, type `/Mirth Training/outbox`
   c. Leave the File Name field blank for now
   d. In the Template field, type the following with a space character at the end of each line:

```
Patient Report - Created
----------------------------------
Patient ID:
Last Name:
First Name:
Middle Name:
Date of Birth:
Gender:
SSN:
```

   Note: this text can also be copied from the file "Text Patient Report Template.txt" in the /Mirth Training/Channels/Miscellaneous folder
4. Create the Mapper transformer steps to extract patient data from the message

a. Click Edit Transformer in the Channel Tasks menu
b. Set the Inbound Message Template to any ADT message
c. Click the Message Trees tab to view the Inbound Message Template Tree
d. In the Filter field, type `patient id` and press Enter
e. Select the value in the PID.2.1 (ID) branch next to the green dot
f. While holding down the left mouse button, drag and drop the value into the transformer steps list (release the mouse button to drop). This will create a new mapper variable with a default variable name of patientIdentification_patientId_id
g. In the Variable field, rename the variable to patientId
h. Repeat steps d through g, above using the information from the table below

Filter Search String	HL7 Field ID	Rename Variable To
family name	PID.5.1	lastName
given name	PID.5.2	firstName
second and further	PID.5.3	middleName
date/time of birth	PID.7.1	dateOfBirth
sex	PID.8.1	gender
ssn	PID.19.1	ssn

i. For the dateOfBirth mapper variable created in step h, add code to convert the value to a more user-friendly format
    i. Select the dateOfBirth variable in the transformer steps list
    ii. Click the Reference tab next to the Message Trees and Message Templates tabs
    iii. In the Filter drop-down list, select Date Functions
    iv. Select Convert Date String, and drag-and-drop it into the beginning of the Mapping field, placing it before the value that is currently in that field
    v. Delete the `var dateString =` part of the inserted code, including the space that follows the equals sign
    vi. Delete the `date);` part of the inserted code
    vii. Add a closing parenthesis at the very end of the code.
    viii. Replace `inpattern` with `"yyyyMMdd"` (including the quotation marks)
    ix. Replace `outpattern` with `"MMMM dd, yyyy"` (including the quotation marks)
    x. The code in the Mapping field should now look like:

```
DateUtil.convertDate("yyyyMMdd", "MMMM dd, yyyy",
msg['PID']['PID.7']['PID.7.1'].toString())
```

j. For the gender mapper variable created in step h, create regular expression string replacement rules to make the gender value more user-friendly
    i. Select the gender variable in the transformer steps list
    ii. Click the New button to the right of the String Replacement list
    iii. Double-click the cell in the Regular Expression column to make it editable
    iv. Type `"M|m"` (including quotation marks) and press Enter to accept changes
    v. Double-click the cell in the Replace With column to make it editable
    vi. Type `"Male"` (including quotation marks and press Enter to accept changes
    vii. Repeat steps ii through vi, above, using the information from the following table:

Regular Expression	Replace With
"F|f"	"Female"
"U|u"	"Unknown"

k. Click Back to Channel from the Mirth Views menu

5. Finish configuring the Patient Report File Writer destination connector
   a. Set the File Name field to the ID of the patient in the message followed by a .txt extension
      i. Drag-and-drop the patientId mapping from the Destination Mappings list to the File Name field
      ii. After the ${patientId} placeholder, type `.txt`
   b. Drag the Formatted Date mapping from the Destination Mappings list to the end of the first line in the Template field
   c. Change the value within the single quotes of the Formatted Date mapping that you just dragged over to `MM/dd/yyyy`
   d. For each of the remaining fields in the template for the patient report, drag the corresponding mapping from the Destination Mappings list and drop it at the end of the line. For example, for "Patient ID: ", drag over the patientId mapping
   e. The Template field for the Patient Report destination should now look like this:

```
Patient Report - Created ${date.get('MM/dd/yyyy')}
---------------------------------

Patient ID: ${patientId}
Last Name: ${lastName}
First Name: ${firstName}
Middle Name: ${middleName}
Date of Birth: ${dateOfBirth}
Gender: ${gender}
SSN: ${ssnNumber}
```

6. Create and configure a second File Writer destination to generate an XML document file with patient data from the message
   a. In the Channel Tasks menu, click New Destination
   b. Rename the destination to `XML Document`
   c. In the Directory field, type `/Mirth Training/outbox`
   d. Set the File Name field to the ID of the patient in the message followed by a .xml extension
      i. Drag-and-drop the patientId mapping from the Destination Mappings list to the File Name field
      ii. After the ${patientId} placeholder, type `.xml`
   e. For the Template field, drag-and-drop the Encoded Data mapping from the Destination Mappings list
7. Set the outbound template and create the Message Builder transformer steps for the XML Document destination
   a. Click Edit Transformer in the Channel Tasks menu
   b. Set the Inbound Message Template to any ADT message
   c. Set the Outbound Message Template
      i. In the Data Type drop-down list for the outbound template, select XML
      ii. Open the file \Mirth Training\Messages\XML\Templates\ Patient Template.xml
   d. Click the Message Trees tab to view the message templates

e. Expand the Outbound Message Template Tree by selecting the top node, XML-Message (1.0), right-clicking, and selecting Expand from the context menu
f. For each node in the Outbound Message Template Tree, find the corresponding field in the Inbound Message Template Tree (as defined in the table below), and drag the value from the Inbound Message Template Tree to the node in the Outbound Message Template Tree. This will create a Message Builder step that maps the field in the inbound message to the corresponding field in the outbound message. Repeat until all fields are mapped.

Inbound Message Template Field	Outbound Message Template Field
PID.2.1 (ID)	patientId
PID.5.1 (Patient Name – Family Name)	name – last
PID.5.2 (Patient Name – Given Name)	name – first
PID.5.3 (Patient Name – Second and Further Given Names or Initials Thereof)	name – middle
PID.5.4 (Patient Name – Suffix)	name – suffix
PID.7.1 (Date/Time of Birth)	dateOfBirth
PID.8.1 (Administrative Sex)	gender
PID.19.1 (SSN Number – Patient)	ssn
PID.11.7 (Patient Address – Address Type)	address – @type
PID.11.1 (Patient Address – Street Address)	address – line1
PID.11.2 (Patient Address – Other Designation)	address – line2
PID.11.3 (Patient Address – City)	address – city
PID.11.4 (Patient Address – State or Province)	address – state
PID.11.5 (Patient Address – Zip or Postal Code)	address - postalCode

g. Click Back to Channel in the Mirth Views menu

8. Save the channel
9. Deploy the channel
10. Send one or more ADT messages to the channel using HL7 Inspector (don't forget to change the port to which to send to match the port number used by the channel's LLP Listener)

Results

For each message that you send to the channel, the Received count should increase by one and the Sent count increase by two in the Dashboard. In the /Mirth Training/outbox folder, you should see two new files. Each file should have the ID of the patient specified in the PID segment of the message sent, one with a .txt extension and the other with a .xml extension.

The txt file should contain the patient report, displaying the patient data extracted from the message placed in the corresponding fields. The Date of Birth and Gender fields should be populated with the re-formatted data, as opposed to the unformatted data straight from the message.

The xml file should contain the XML document structure defined by the outbound message template, with each element populated with data from the inbound message.

Lab 7 Create a Channel that Reads a File from a Local Folder and Writes to an FTP Server

Objectives

In this lab, you will create a channel with a File Reader source connector and a File Writer destination connector. In previous labs, you've used the File Writer to write a file to the local file system. This channel will demonstrate one of the other file connector capabilities—to connect to an FTP server. You will also use a File Reader source connector for the first time to poll for new files from your local file system.

Estimated Time to Complete

10 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer
- FileZilla FTP Client (optional)

Step-By-Step

1. Create a new channel named *Local Folder to FTP*
2. Configure the source connector as a File Reader that reads from a network share
   a. On the Source tab, select File Reader in the Connector Type drop-down list
   b. In the first Directory field, type `/Mirth Training/inbox`
   c. In the Move-to Directory field, type `/Mirth Training/processed`
   d. For the Move-to File Name field, define the file name as the name of the file read by the File Reader, prefixed with a system timestamp
      i. In the box to the right of the three "Move-to" fields, select SYSTIME and drag-and-drop it into the Move-to File Name field to insert the ${SYSTIME} placeholder
      ii. After the ${SYSTIME} placeholder, type an underscore character (_)
      iii. Drag ORIGINALNAME and drop it into the Move-to File Name field after the underscore. The Move-to File Name field should now contain: ${SYSTIME}_${ORIGINALNAME}
3. Configure the destination connector as a File Writer that writes to an FTP server
   a. On the Destinations tab, rename the default File Writer destination to `FTP Server`
   b. In the Method drop-down list, select ftp
   c. In the first ftp:// field, type the given IP address for the instructor's computer (which is hosting the FTP server) 192.168.1.101
   d. In the second ftp:// field, type `outbox`
   e. For the File Name field, drag-and-drop Original File Name from the Destination Mappings list to insert the ${ORIGINALNAME} placeholder
   f. For the Anonymous setting, select No
   g. In the Username field, replace the default contents with `sX`, where X is your given student number
   h. In the Password field, replace the default contents with `abc12345`

i. Click the Test Read button (to the right of the Method drop-down list). After a few seconds, a pop-up should appear indicating that you are able to successfully connect.
j. For the Template field, drag-and-drop Encoded Data from the Destination Mappings list

4. Save the channel
5. Deploy the channel
6. View the current contents of your student folder on the FTP server (optional)
   a. Start the FileZilla FTP Client application installed on your student computer
   b. Beneath the menu bar and tool bar you should see fields in which to enter values for Host, Username, Password and Port. Enter the following values for these fields:
      i. Host: The given IP address for the instructor's computer
      ii. Username: `sX`, where X is your given student number
      iii. Password: abc12345
      iv. Port: [leave blank]
   c. Click the Quickconnect button to the right of the Port field to connect
   d. In the Remote Site pane, you should see three folders: inbox, outbox and processed. Each of these folders should be empty.
7. Copy one or more HL7 message files from the /Mirth Training/Messages/HL7 folder to the /Mirth Training/inbox folder.

Results

Within a few seconds of copying the HL7 messages files to the inbox folder, the channel will process them. The Dashboard statistics for the channel in the Mirth Administrator should show Received and Sent counts equal to the number of HL7 message files copied to the inbox.

Using Windows Explorer to view the /Mirth Training/ folder, you should see that the inbox folder is now empty, and the processed folder now contains the four files processed, each prefixed with a timestamp in long integer format.

Next, using the FileZilla FTP Client to view the outbox folder, you should see the four HL7 files read by the channel, each written using the original file name, prefixed with a unique ID.

Lab 8 Create a Channel with a TCP Listener and LLP Sender

Objectives

In this lab, you will create a channel that listens for messages using a TCP Listener, and sends the message to an LLP server on the instructor's computer using the LLP Sender. You will also use a Message Builder transformer step to change the Sending Application field (MSH.3.1) to your student number. Additionally, you will use the Respond From feature of the TCP Listener to send the response received by the LLP Sender (i.e., and ACK message) as its response.

To send to the channel's TCP Listener, you will use HL7 Inspector. Even though HL7 Inspector uses the LLP protocol, because LLP is built on top of TCP, it can also be used to send to the TCP Listener.

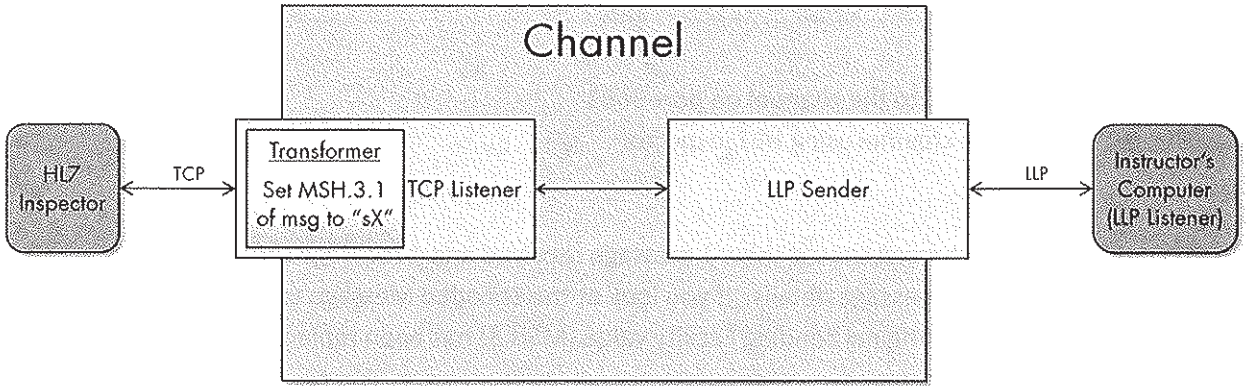


Figure 8-1 TCP/LLP Lab Processing Flow

Estimated Time to Complete

10 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector

Step-By-Step

1. Create a new channel named *TCP to LLP*
2. Configure the source connector as a TCP Listener
   a. On the Source tab, select TCP Listener in the Connector Type drop-down list
   b. In the Listener Port field, change the value to 9663
3. Add transformer step on source connector to change the Sending Application field to your student number
   a. Click Edit Transformer in the Channel Tasks menu
   b. Click Add New Step in the Transformer Tasks menu
   c. In the new entry added to the transformer steps list, double-click the cell in the Type column
   d. Select Message Builder from the drop-down list

e. In the Message Segment field, type the following, which is the code to access the Sending Application field in the inbound message:
   `msg['MSH']['MSH.3']['MSH.3.1']`
f. In the Mapping field, type `"sX"` (including quotes), where X is your given student number
g. Click Back to Channel in the Mirth Views menu

4. Configure the destination port as an LLP Sender that will send to the instructor's computer
   a. On the Destinations tab, rename the default connector to `LLP Server`
   b. Select LLP Listener in the Connector Type drop-down list
   c. In the Host Address field, type the IP address for the instructor's computer
   d. In the Host Port field, type `9100`
5. Update the source connector to respond from the LLP Sender destination connector
   a. On the Source tab, select LLP Server from the Respond From drop-down list
6. Save the channel
7. Deploy the channel
8. Configure HL7 Inspector to send to the channel on port 9663
9. Send any HL7 message to the channel using HL7 Inspector

Results

The Dashboard statistics for the channel should show 1 message received and 1 message sent. Switch to Channel Messages view to see the message details. If you look at the Encoded Message tab for the source connector, you should see that the original Sending Facility value (MSH.3) has been replaced by your student number.

In the HL7 Inspector, you should see the ACK message sent back from your channel. In the MSA segment, you should see an acknowledgement code of AA in MSA.1 and the following message in MSA.3:

"Successfully received message from Student X."

Where X is your student number.

Lab 9 Create a Channel with a Database Reader

Objectives

In this lab, you will create a channel that uses a Database Reader to read from a table of new patients, and uses Message Builder transformer steps to create an outbound ADT message from that data.

Estimated Time to Complete

15-20 minutes

Tools Used

- Mirth Connect Administrator
- SQuirreL SQL Client (optional)
- Windows Explorer

Step-By-Step

1. Create a new channel named *Database Reader*
2. Configure the basic Database Reader source connector settings
   a. On the Source tab, select Database Reader in the Connector Type drop-down list
   b. In the Driver drop-down list, select PostgreSQL
   c. In the URL field, type:
   `jdbc:postgresql://localhost:5432/mirthtraining`
   d. In the Username field, type `postgres`
   e. In the Password field, type `abc12345`
3. Generate the SQL query to select unprocessed rows from the NewPatients table
   a. Click the Select button above the SQL editor on the right-hand side
   b. In the SQL Creation dialog, select all of the columns, except Processed, in the NewPatients table by checking the box to the left of the column name
   c. Click the Generate button
   d. Edit the inserted SELECT statement to select on rows marked as unprocessed by deleting the semicolon at the end of the statement and adding the following line:
   `WHERE processed = false;`
   The SELECT statement should now look like the following:

```
SELECT newpatients.newpatientid AS newpatients_newpatientid,
newpatients.lastname AS newpatients_lastname, newpatients.firstname AS
newpatients_firstname, newpatients.middlename AS newpatients_middlename,
newpatients.gender AS newpatients_gender, newpatients.dateofbirth AS
newpatients_dateofbirth, newpatients.ssn AS newpatients_ssn,
newpatients.facility AS newpatients_facility, newpatients.eventtype AS
newpatients_eventtype, newpatients.eventtime AS newpatients_eventtime
FROM newpatients
WHERE processed = false;
```

4. Generate the SQL statement to update selected rows to mark them as processed
   a. Change the Run On-Update Statement setting to Yes

b. Click the Update button above and to the right of the On-Update SQL editor
c. In the SQL Creation dialog, check the box next to the Processed column in the NewPatients table
d. Click the Generate button
e. Edit the inserted UPDATE statement so that it looks like the following:

```
UPDATE newpatients
SET processed = true
WHERE newpatientid = ;
```

f. Insert a placeholder for the patient ID from the message by selecting newpatients_newpatientid in the box to the right of the On-Update SQL editor and dragging and dropping it into the WHERE clause after the equals sign. The UPDATE statement should now look like the following:

```
UPDATE newpatients
SET processed = true
WHERE newpatientid = ${newpatients_newpatientid};
```

5. Generate the outbound ADT message using Message Builder transformer steps
   a. Click Edit Transformer in the Channel Tasks menu
   b. On the Message Templates tab, change the Data Type setting to HL7 v2.x
   c. Set the outbound template to the message in /Mirth Training/Messages/HL7/Templates/ADT Template.hl7
   d. On the Message Trees tab, create Message Builder steps by selecting a node from the Inbound Message Template Tree and dragging and dropping it onto the corresponding node in the Outbound Message Template Tree, as indicated in the table below.

Inbound Message Template Field	Outbound Message Template Field
newpatients_newpatientid	PID.2.1 (ID)
newpatients_lastname	PID.5.1 (Patient Name – Family Name)
newpatients_firstname	PID.5.2 (Patient Name – Given Name)
newpatients_middlename	PID.5.3 (Patient Name – Second and Further Given Names or Initials Thereof)
newpatients_gender	PID.8.1 (Administrative Sex)
newpatients_dateofbirth	PID.7.1 (Date/Time of Birth)
newpatients_ssn	PID.19.1 (SSN Number – Patient)
newpatients_facility	MSH.4.1 (Sending Facility)
newpatients_eventtype	MSH.9.2 (Trigger Event)
newpatients_eventtime	MSH.7.1 (Time of an Event)

6. Configure the destination connector as a File Writer to write the generated ADT message
   a. On the Destinations tab, rename the default File Writer destination to `Write HL7 File`
   b. In the Directory field, type `/Mirth Training/outbox`
   c. For the File Name field, define the name as the message ID followed by a .hl7 extension
      i. Drag-and-drop the Message ID mapping from the Destination Mappings list
      ii. Type `.hl7` after the ${message.id} placeholder
   d. For the Template field, drag-and-drop the Encoded Data mapping from the Destination Mappings list
7. Save the channel
8. View the current state of the data in the NewPatients table (optional)

a. Open the SQuirreL SQL Client application
b. In the Alias window, select Mirth Training Database
c. Right-click, and select Connect from the context menu
d. In the "Connect to" dialog that appears, click the Connect button to connect to the database and open the session window
e. In the text area of the window, type the following SQL query:

```
SELECT * FROM NewPatients;
```

f. With the cursor still on the line that you just typed, press Ctrl-Enter to execute the query (or press the button on the toolbar that looks like a person running)
g. In the results pane of the window, you should see four rows of patient data. Note that the last column, Processed, shows a value of false for each row.

9. Deploy the channel

Results

Upon deployment, the channel will almost immediately read the four rows in the database table, generate the outbound ADT messages from the data and write the messages out to files. The Dashboard statistics for the channel in the Mirth Administrator should show 4 messages received and 4 messages sent. In the Channel Messages view, you can view the data read from the database for each row in XML format, as well as the HL7 messages created from the data.

If you execute the query of the NewPatients table in SQuirreL SQL Client once again, you should now see that the Processed column for each row is now set to true.

Finally, you should see four new files in the /Mirth Training/outbox folder. Each of these files should contain an HL7 ADT message with the appropriate fields set to the corresponding values read from the database. The remaining fields (that is, the fields that weren't specifically set with a Message Builder step) should remain as they were in the ADT outbound message template.

Lab 10 Install a New Database Driver for Use with the Database Reader/Writer Connectors

Objectives

In this lab, you will install database driver for Advantage Database Server 8.1 for use with the Database Reader and Database Writer connectors. This will involve placing the driver file in the correct location in the Mirth installation directory, as well as updating the Mirth Connect database driver configuration file.

Estimated Time to Complete

5 minutes

Tools Used

- Mirth Connect Administrator
- Mirth Connect Server Manager
- Windows Explorer
- Any text editor, such as Notepad++

Step-By-Step

1. Copy the file adsjdbc.jar from the /Mirth Training/jars folder to the lib/custom folder of the Mirth Connect installation folder (usually /Program Files/Mirth or /Program Files (x86)/Mirth)
2. Add an entry for the new database driver to the dbdrivers.xml configuration file
   a. Open the file dbdrivers.xml in the conf/custom folder of the Mirth Connect installation folder for editing (your computer is installed with the Notepad++ editor, or you can use Notepad or WordPad)
   b. Add the following line after the last driver element in the XML document:
   ```
   <driver class="com.extendedsystems.jdbc.advantage.ADSDriver"
   name="Advantage Database Server 8.1" />
   ```
   c. Save your changes
3. If the Mirth Connect Administrator is currently running, close the application
   a. Click Logout from the Other menu
   b. In the Login window, click the Exit button
4. Restart the Mirth Connect Server
   a. Double-click the Mirth icon in the Windows notification area (system tray), as shown in Figure 10-1 below, to open the Mirth Connect Server Manager

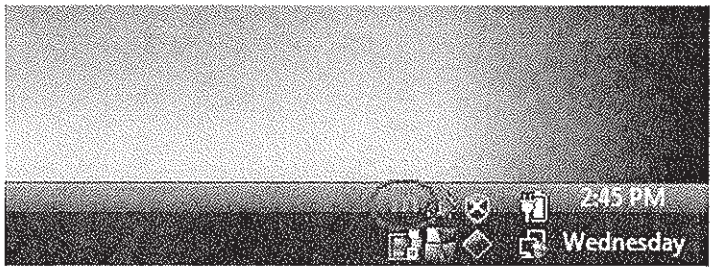


Figure 10-1 Mirth Connect Server Manager Icon in Notification Area

b. In the Mirth Connect Server Manager, click the Restart button
c. After a short while, a pop-up will appear indicating that the server has successfully restarted. Click the OK button to proceed

5. In the Mirth Connect Server Manager, click the Administrator button to re-launch the Mirth Connect Administrator
6. Verify that Advantage Database Server 8.1 now appears as an option in the Driver drop-down list for both the Database Reader and Database Writer
   a. Create a new channel
   b. On the Source tab, change the connector type to Database Reader
   c. In the Driver drop-down list, verify that Advantage Database Server 8.1 appears at the end of the list
   d. On the Destinations tab, change the connector type to Database Writer
   e. In the Driver drop-down list, verify that Advantage Database Server 8.1 appears at the end of the list

Lab 11 Pass an HL7 Message between Two Channels Using HTTP

Objectives

In this lab, you will create two channels: one with an HTTP Sender and another with an HTTP Listener. The channel with HTTP Sender will send an HL7 message to the channel with the HTTP Listener. This will demonstrate how messages can be sent over an HTTP connection, as well as the extra steps that are required to send HL7 messages over HTTP, such as encoding and decoding the request content to avoid problems with special characters such as the ampersand and less-than and greater-than characters.

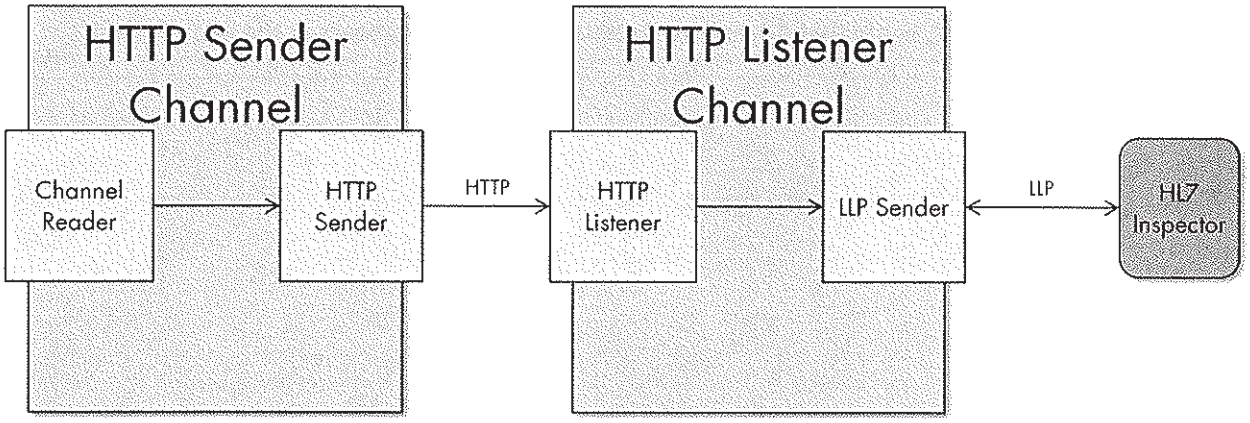


Figure 11-1 HTTP Lab Processing Flow

Estimated Time to Complete

15 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector

Step-By-Step

1. Create a new channel named *HTTP Listener*
2. Configure the source connector as an HTTP Listener
   a. Change the Connector Type to HTTP Listener
   b. Change the Listener Port to `9091`
   c. For the Append Payload Variable setting, click Yes
   d. From the Payload URL Encoding drop-down list, select Encode
3. Add a transformer step to the source connector to decode the request payload and store it in a mapper variable
   a. Click Edit Transformer
   b. Click Add New Step in the Transformer Tasks menu
   c. In the Variable field, type `hl7Message`

d. In the Mapping field, type the following:

```
Packages.java.net.URLDecoder.decode(msg['payload'].toString())
```

e. Click Back to Channel in the Mirth Views menu to return to channel view

4. Configure the destination connector as an LLP Sender
   a. Rename the destination to `LLP Destination`
   b. Change the Connector Type to LLP Sender
   c. Change the Host Port setting to `9670`
   d. In the Template field, delete the default value of ${message.encodedData}
   e. Drag-and-drop the hl7Message destination mapping into the Template field
5. Save the channel
6. Create a second new channel named HTTP Sender
7. Configure the source connector as a Channel Reader
8. Configure the destination connector as an HTTP Sender
   a. Change the Connector Type to HTTP Sender
   b. For the URL field, type the following URL:
      `http://localhost:9091`
   c. In the Request Variables list, double-click the Value cell for the $payload variable to make it editable
   d. Drag-and-drop the Encoded Data destination mapping into the cell and press the Enter key to accept the change
9. Save the channel
10. Deploy the channels
11. Configure HL7 Inspector to listen on the port specified by the LLP Sender
    a. Open HL7 Inspector if not already open
    b. From the Tools menu, select Receive Messages to open the Message Receiver tab (or click the button on the toolbar with the downward-pointing green arrow)
    c. In the Message Receiver tab, click the Setup network button (second button from the right)
    d. In the Receive Network Setup dialog that appears, type the port specified in the LLP Sender (9670), and check the Reuse box
    e. Click OK to accept the changes
    f. Click the Start receiving button (the button with the blue triangle/"play") to start the receiver listening on port 9670
12. Send a message to the HTTP Sender channel using the Send Message feature
    a. In the Mirth Connect Administrator's Dashboard, select the HTTP Sender channel
    b. In the Status Tasks menu, click Send Message
    c. In the Message window that appears, click the Open File button
    d. Browse to and open any HL7 message in the /Mirth Training/Messages/HL7 folder
    e. Click the Process Message button to send the message

Results

For both the HTTP Sender and HTTP Listener channels, the statistics in the Dashboard should show that 1 message was received and 1 message was sent. If you view the Raw Message for either connector in the Channel Messages view, you can see that the XML message contains a payload element whose contents are the

HL7 message after URL encoding. On the Mappings tab, you can see a Channel Map variable called payload, which contains the HL7 message after URL decoding.

Finally, in the HL7 Inspector, the Message Receiver tab should show the HL7 message that it received from the LLP Sender connector in the HTTP Listener 2 channel.

Lab 12 Create a Channel with a SOAP Listener

Objectives

In this lab, you will create a channel with a SOAP Listener source connector. You will then use a third-party tool called soapUI to read the SOAP Listener's WSDL, create a request envelope containing an HL7 message, and send the request to the channel.

Estimated Time to Complete

10 minutes

Tools Used

- Mirth Connect Administrator
- soapUI 3.0

Step-By-Step

1. Create a new channel named *SOAP Listener*
2. Configure the source connector as a SOAP Listener
   a. Change the Connector Type to SOAP Listener
   b. Change the Port value to 9081
3. Configure the destination connector as a "null" Channel Writer
4. Save the channel
5. Deploy the channel
6. Configure soapUI to send an HL7 message to the channel
   a. Open soapUI 3.0
   b. Create a new soapUI project by selecting New soapUI Project from the File menu (or type Ctrl-N)
   c. For the Project Name field, type a name for your project
   d. For the Initial WSDL/WADL field, type the following URL, or copy and paste it from the SOAP Listener WSDL URL field:
      `http://localhost:9081/services/Mirth?wsdl`
   e. Click the OK button
   f. You should now have a soapUI project with a default request for the acceptMessage method. Expand the tree under your project until you see Request 1
   g. Double-click Request 1 to open the Request Editor for the request
   h. In the request envelope, delete the question mark in the body of the in0 element
   i. Open any HL7 message in a text editor (such as Notepad++), copy it and paste it into the body of the in0 element of the request envelope (where the question mark was)
   j. At the beginning of the body of the in0 element, just before the start of the HL7 message, type the following:
      `<![CDATA[`
   k. At the end of the HL7 message, just before the closing tag of the in0 element, type the following:
      `]]>`
7. Send the request to the channel by clicking the green arrow button in the Request Editor's toolbar

Results

Once you send the request using soapUI, a new pane should appear in the Request Editor containing the response envelope returned by the channel's SOAP Listener. The body of the acceptMessageReturn element should contain a message in the following format:
Message: [message id] has been successfully received

In the Mirth Connect Administrator, the statistics in the Dashboard should show that 1 message was received and 1 message was sent for the SOAP Listener channel. Switching to the Channel Messages view, the Raw Message tab for both the source and destination connector entries should show the HL7 message that was sent as the in0 parameter in the request envelope.

Lab 13 Create a Channel with a SOAP Sender

Objectives

In this lab, you will create a channel with a SOAP Sender destination connector. The SOAP Sender will send an HL7 message to the SOAP Listener source connector in the channel created in the last lab. This will require configuring the SOAP Sender to load the WSDL for the SOAP Listener and editing the request envelope as needed using the configuration tools for the SOAP Sender connector.

Estimated Time to Complete

5 minutes

Tools Used

- Mirth Connect Administrator

Step-By-Step

1. Create a new channel named *SOAP Sender*
2. Configure the source connector as a Channel Reader
3. Configure the destination connector as a SOAP Sender to send to the channel created in Lab 12.
   a. Change the Connector Type to SOAP Sender
   b. In the WSDL Path field, type the following URL, or copy and paste it from the SOAP Listener created in Lab 12:
      `http://localhost:9081/services/Mirth?wsdl`
   c. Click the Get Methods button to the right of the WSDL Path field to load the WSDL and automatically generate a request envelope
   d. In the tree structure below the Method drop-down list, select the "string in0 = " node.
   e. In the box to the right of the tree structure, double-click the cell to the right of the "value" cell to make it editable
   f. Type the following:
      `<![CDATA[`
   g. After the CDATA code, drag-and-drop the Encoded Data destination mapping
   h. After the Encoded Data placeholder, type the following:
      `]]>`
      The value in that cell should now look like:
      `<![CDATA[${message.encodedData}]]>`
4. Save the channel
5. Deploy the channel
6. Use the Dashboard's Send Message feature to send any HL7 message to the SOAP Sender channel

Results

In the Mirth Connect Administrator, the statistics in the Dashboard should show that 1 message was received and 1 message was sent for the SOAP Sender channel, and the received and sent numbers increased by one for the SOAP Listener channel. In the Channel Messages view for the SOAP Sender channel, select the entry for the

destination and click the Mappings tab. Here, you can see the response envelope returned by the SOAP Listener channel in the Response Map.

Lab 14 Create JMS Writer and JMS Reader Channels to Write to and Read from a JMS Queue

Objectives

In this lab, you will create two different channels to demonstrate the ability to write to and read from a JMS queue. The first channel will use a JMS Writer to write messages to a queue hosted on the instructor's computer. The second channel will use a JMS Reader to read the messages from that queue.

The JMS queue implementation used for this lab is Apache's ActiveMQ 5.1. As part of the lab, you will also use ActiveMQ's admin web interface you view the contents and statistics for you queue.

Estimated Time to Complete

15-20 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer
- Any web browser, such as Internet Explorer or Firefox

Step-By-Step

1. Configure Mirth Connect to make the Apache ActiveMQ core library available to the JMS connectors
   a. Copy the file activemq-core-5.2.0.jar, located in the /Mirth Training/jars folder, to the lib/custom folder in the Mirth Connect installation folder
   b. Exit Mirth Connect Administrator
   c. Restart the Mirth Connect Server using the Mirth Connect Server Manager
   d. Start the Mirth Connect Administrator
2. Create a new channel named *JMS Writer*
3. Configure the source connector as a Channel Reader
4. Configure the destination connector as a JMS Writer
   a. Rename the destination to `JMS Queue`
   b. Change the Connector Type to JMS Writer
   c. In the Connection Factory Class field, type:
      `org.apache.activemq.ActiveMQConnectionFactory`
   d. In the Destination field, type `MirthTraining_sX`, where X is your given student number
   e. Add the brokerURL property required by ActiveMQ to define the URL for the JMS Queue
      i. Click the New button to the right of the Additional Properties list
      ii. Double-click the cell in the Property column for the new entry to make it editable, and replace the default value with `brokerURL`
      iii. Double-click the cell in the Value column for the new entry to make it editable, and type the following:
      `tcp://host:61616?keepAlive=true`
      where *host* is the IP address of the instructor's computer

5. Save the channel
6. Create a second new channel named *JMS Reader*
7. On the Summary tab, set the Initial State to Stopped
8. Configure the source connector as a JMS Reader
   a. In the Connection Factory Class field, type:
      `org.apache.activemq.ActiveMQConnectionFactory`
   b. In the Destination field, type `MirthTraining_sX`, where X is your given student number
   c. Add the brokerURL property required by ActiveMQ to define the URL for the JMS Queue
      i. Click the New button to the right of the Additional Properties list
      ii. Double-click the cell in the Property column for the new entry to make it editable, and replace the default value with `brokerURL`
      iii. Double-click the cell in the Value column for the new entry to make it editable, and type the following:
      `tcp://host:61616?keepAlive=true`
      where *host* is the IP address of the instructor's computer
9. Configure the destination connector as a "null" Channel Writer
10. Save the channel
11. Deploy the channels
12. Use the Dashboard's Send Message feature to send one or more messages (any HL7 message) to the JMS Writer channel
13. Use ActiveMQ's admin web interface to view the messages in the queue
    a. Open a web browser such as Internet Explorer or Firefox
    b. In the address field, type `http://host:8161/admin`, where *host* is IP address of the instructor's computer
    c. On the web page that appears, click Queues in the menu bar just below the ActiveMQ header at the top of the page
    d. In the Queues list, find the queue pertaining to your student number. The values in the Number of Pending Messages and Messages Sent columns should reflect the number of messages that you sent to your channel
    e. Click the link for your queue. Here, you can see more details about the messages in the queue.
    f. Click the link for one of the messages in your queue. This page will show even more detail about the message, including the actual HL7 message text
14. Start the JMS Reader channel to read the messages out of your queue
    a. In the Dashboard, select the JMS Reader channel
    b. Click Start All Channels in the Status Tasks menu to start the channel
15. Go back to the Queues page in the ActiveMQ admin web page to view the current statistics
    a. Click Queues in the menu bar
    b. Once again find the queue that corresponds to your student number. The value in the Number of Pending Messages column should now be 0, and the value in the Messages Received column should equal the number of messages that you initially sent.
    c. Click the link for your queue. You should now see that there are no messages in the queue.

Results

In the Administrator's Dashboard, the received and sent statistics for both the JMS Reader and JMS Writer channels should be equal to the number of messages sent to the JMS Writer channel. If you go to the Channel Messages view for the JMS Reader channel, you should see the same messages that you sent to the JMS Writer channel.

Lab 15 Create a Channel that Generates a Patient Report PDF Document

Objectives

In this lab, you will create a channel that uses the Document Writer connector to create a patient report PDF document from patient data in the inbound message. As in previous labs, you will use Mapper transformer steps to extract the patient data from the message.

Estimated Time to Complete

10 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer
- Any text editor, such as Notepad++
- Adobe Acrobat Reader

Step-By-Step

1. Create a new channel named *PDF Writer*
2. Configure the source connector as a Channel Reader
3. Configure the destination connector as a PDF document writer
   a. Rename the destination to `Generate PDF`
   b. Change the Connector Type to Document Writer
   c. Set the Directory field to `/Mirth Training/outbox`
   d. Open the file "PDF Patient Report Template.html" from the /Mirth Training/Channels/Miscellaneous folder in a text editor, and copy and paste it into the Template field
4. Extract the patient data required for the report from the inbound message using Mapper transformer steps
   a. Click Edit Transformer in the Channel Tasks menu
   b. Set the inbound message template to any ADT message
   c. For each of the following PID fields, create a Mapper variable:
      - PID.2.1 (ID)
      - PID.5.1 (Patient Name – Family Name)
      - PID.5.2 (Patient Name – Given Name)
      - PID.5.3 (Patient Name – Second and Further Given Names or Initials Thereof)
      - PID.7.1 (Date/Time of Birth)
      - PID.8.1 (Administrative Sex)
      - PID.13.1 (Phone Number – Home)
      - PID.19.1 (SSN Number – Patient)

   d. Format the date of birth and gender (sex) variables using the same methods used in Lab 6 (optional)
   e. Click Back to Channel in the Mirth Views menu to return to the Destinations tab
5. Finish configuring the PDF Writer destination

a. Set the File Name to the patient ID followed by a .pdf extension: ${patientIdentification_patientId_id}.pdf
b. For each of the fields in the template, drag the corresponding destination mapping and drop it into the body of the empty <td></td> tag

6. Save the channel
7. Deploy the channel
8. Use the Dashboard's Send Message feature to send one or more ADT, ORU or OUL messages (or any other message with a PID segment)

Results

For each message sent to the channel, you should see the received and sent counts increased by one in the Dashboard. In the /Mirth Training/outbox folder, you should see a new PDF file for each message sent, with the name of each file being the patient ID value from PID.2.1. In each file, you should see the Mirth Connect logo at the top of the document, with a formatted table containing the patient data beneath it. Figure 15-1, below, shows an example of what the contents of your PDF file should look like.

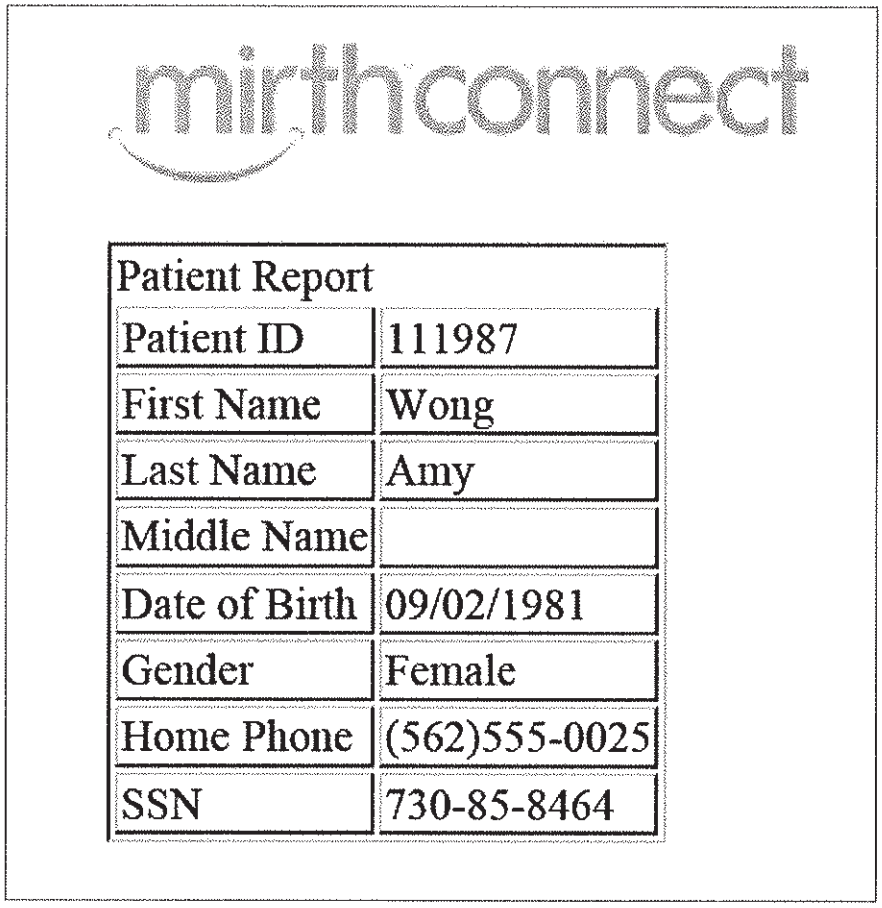


Figure 15-1 Example Output from PDF Writer

Lab 16 Create a Channel with an Email Sender

Objectives

In this lab, you will create a channel that uses an Email Sender to send an email that contains patient data extracted from the inbound message, and has a plain text attachment that contains the message itself.

Estimated Time to Complete

10-15 minutes

Tools Used

- Mirth Connect Administrator
- Any web browser, such as Internet Explorer or Firefox, or whichever application you use to access your email

Step-By-Step

1. Create a new channel named *Email Sender*
2. Configure the source connector as a Channel Reader
3. Configure the destination connector as an Email Sender
   a. Rename the destination to `Send Email`
   b. For the SMTP Server Host and SMTP Server Port settings, use the IP address and port, respectively, provided to you on your Student Information sheet
   c. In the To field, enter your own email address. If you don't have access to your email, you can use the Gmail account provided to you.
   d. In the From field, type `email@mirthtraining.com`
   e. In the Subject field, type `Message Received`
4. Create Mapper transformer steps for data needed in email
   a. Click Edit Transformer in the Channel Tasks menu
   b. Set the inbound message template to any ADT message
   c. Create Mapper steps for patient ID, (PID.2.1), last name (PID.5.1) and first name (PID.5.2)
   d. Create a Mapper step for the current time
      i. Click Add New Step from the Transformer Tasks menu
      ii. In the Variable field, type `currentTime`
      iii. In the Mapping field, type `DateUtil.getCurrentDate("HH:mm:ss")`
   e. Create a Mapper step for the current date
      i. Click Add New Step from the Transformer Tasks menu
      ii. In the Variable field, type `currentDate`
      iii. In the Mapping field, type `DateUtil.getCurrentDate("MM/dd/yyyy")`
   f. Click Back to Channel in the Mirth Views menu
5. Finish configuring the Send Email destination
   a. In the Body field, type the following text:

```
Received message for FIRST LAST (ID: PID) at TIME on DATE.
Message is attached.
```

b. For each of the following in the Body text, drag over the corresponding destination mapping and replace the field to which it corresponds

FIRST	patientIdentification_patientName_givenName
LAST	patientIdentification_patientName_familyName
PID	patientIdentification_patientId_id
TIME	currentTime
DATE	currentDate

6. Add the message as a plain text attachment with its name in the format LAST_FIRST.txt
   a. Click the New button to the right of the Attachments list
   b. Double-click the cell in the Name column for the new entry to make it editable
   c. Drag-and-drop the destination mapping for the patient's last name
   d. Type an underscore character
   e. Drag-and-drop the destination mapping for the patient's first name
   f. Type `.txt` and press the Enter button to accept your changes
   g. Double-click the cell in the Content column for the new entry to make it editable
   h. Drag-and-drop the Encoded Data destination mapping and press the Enter key to accept your changes
   i. Double-click the cell in the MIME Type column for the new entry to make it editable
   j. Type `text/plain` and press the Enter key to accept your changes
7. Save the channel
8. Deploy the channel
9. Use the Dashboard's Send Message feature to send any ADT message

Results

The statistics in the Dashboard for the channel should show that 1 message was received and 1 message was sent. Check the email account which you specified as the To address in the Email Sender. Within a short amount of time, you should receive the email generated by the Email Sender with a subject of "Message Received". In the body of the message, you should have the text specified in the Email Sender's Body field, with the placeholders replaced by the corresponding data from the message. For example:

Received message for Amy Wong (ID: 111987) at 10:09:00 on 06/15/2009.
Message is attached.

Also in the email should be an attachment containing the actual HL7 message sent. Open the attachment to verify it can be viewed.

Lab 17 Create a Channel with a JavaScript Filter

Objectives

In this lab, you will create a channel that uses a JavaScript filter to accept or reject channels based on a certain set of conditions. Those conditions are as follows:

- The sending facility must be one of the following:
  - Acme Labs
  - ABC Diagnostics
  - General Hospital
  - Sacred Heart
- If the sending facility is a lab (i.e., Acme Labs or ABC Diagnostics), the message type and trigger event must be one of:
  - ORU-R01
  - OUL-R21
- If the sending facility is a hospital (i.e., General Hospital or Sacred Heart), the message type must be ADT

There are several ways the JavaScript code to accomplish this could be written. The Step-By-Step instructions below merely show you one way.

Estimated Time to Complete

20 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector

Step-By-Step

1. Create a new channel named *JavaScript Filters*
2. Configure the source connector as an LLP Listener
   a. Change the Listener Port to `9610`
3. Create the JavaScript filter that accepts or rejects messages based on the rules described above
   a. Click Edit ~~Transformer~~ FILTER in the Channel Tasks menu
   b. Set the inbound message template to any HL7 message
   c. Click Add New Rule in the Filter Tasks menu
   d. For the new entry added to the filters list, double-click the cell in the Type column and select JavaScript from the drop-down list
   e. We will use a Boolean variable to keep track of whether or not the message should be accepted, which will be returned at the end of the code. Start off your code with the following line:

      ```
      var acceptMessage = false;
      ```

   f. Next, check to see if the message is from "Acme Labs" or "ABC Diagnostics"
      i. From the Inbound Message Template Tree, drag over the Sending Facility in MSH.4.1

ii. Write an if statement block that compares the value in MSH.4.1 to Acme Labs" or "ABC Diagnostics"

This part of the code will look like this (comments optional):

```
// If message is from an approved partner lab...
if (msg['MSH']['MSH.4']['MSH.4.1'].toString() == "Acme Labs" ||
    msg['MSH']['MSH.4']['MSH.4.1'].toString() == "ABC Diagnostics")
{
}
```

g. Within the body of the if statement that you just wrote, add code to check the message type and trigger event is either ORU-R01 or OUL-R21

i. From the Inbound Message Template Tree, drag over the Message Type in MSH.9.1 and the Trigger Event in MSH.9.2

ii. Write if/else if statement blocks that check the values in MSH.9.1 and MSH.9.2 and, if together they match ORU-R01 or OUL-R21, the body of the statements should set the `acceptMessage` variable to true

This part of the code will look like this (comments optional):

```
// If the message type/event is ORU-R01...
if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "ORU" &&
    msg['MSH']['MSH.9']['MSH.9.2'].toString() == "R01")
{
      // Accept message
      acceptMessage = true;
}
// Or if the message type/event is OUL-R21...
else if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "OUL" &&
       msg['MSH']['MSH.9']['MSH.9.2'].toString() == "R21")
{
      // Accept message
      acceptMessage = true;
}
```

h. Next, check to see if the message is from "General Hospital" or "Sacred Heart"

i. From the Inbound Message Template Tree, drag over the Sending Facility in MSH.4.1

ii. Write an if statement block that compares the value in MSH.4.1 to "Acme Labs" or "ABC Diagnostics"

This part of the code will look like this (comments optional):

```
// If message is from an approved partner hospital...
if (msg['MSH']['MSH.4']['MSH.4.1'].toString() == "General Hospital" ||
    msg['MSH']['MSH.4']['MSH.4.1'].toString() == "Sacred Heart")
{
}
```

i. Within the body of the if statement that you just wrote, add an if statement block to check the message type equals "ADT"

i. From the Inbound Message Template Tree, drag over the Message Type in MSH.9.1

ii. Write an if statement block to check if MSH.9.1 is equal to "ADT". If the MSH.9.1 value equals "ADT", set the `acceptMessage` variable to true

This part of the code will look like this (comments optional):

```
// If the message type is ADT...
if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "ADT")
{
      // Accept message
      acceptMessage = true;
}
```

j. Finally, add a line at the end of your code to return the `acceptMessage` variable:

```
return acceptMessage;
```

k. Your final code should look something like this (comments optional):

```
var acceptMessage = false;

// If message is from an approved partner lab...
if (msg['MSH']['MSH.4']['MSH.4.1'].toString() == "Acme Labs" ||
    msg['MSH']['MSH.4']['MSH.4.1'].toString() == "ABC Diagnostics")
{
      // If the message type/event is ORU-R01...
      if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "ORU" &&
          msg['MSH']['MSH.9']['MSH.9.2'].toString() == "R01")
      {
            // Accept message
            acceptMessage = true;
      }
      // Or if the message type/event is OUL-R21...
      else if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "OUL" &&
               msg['MSH']['MSH.9']['MSH.9.2'].toString() == "R21")
      {
            // Accept message
            acceptMessage = true;
      }
}
// Otherwise, if the message is from an approved partner hospital...
else if (msg['MSH']['MSH.4']['MSH.4.1'].toString() == "General Hospital" ||
         msg['MSH']['MSH.4']['MSH.4.1'].toString() == "Sacred Heart")
{
      // If the message type is ADT...
      if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "ADT")
      {
            // Accept message
            acceptMessage = true;
      }
}

// Return the acceptMessage flag
return acceptMessage;
```

l. Click Back to Channel in the Mirth Views menu

4. Configure the destination connector as "null" Channel Writer
5. Save the channel
6. Deploy the channel
7. Open and configure HL7 Inspector to send to the LLP Listener on port 9610
8. Send the following messages to the channel using HL7 Inspector:
   - ADT-A01 - Jane MotherToBe - General Hospital.hl7
   - ADT-A03 - Zapp Brannigan - LAMH.hl7
   - ADT-A08 - John Panucci - Sacred Heart.hl7
   - ORU-R01 - Hattie McDoogal - Acme Labs.hl7
   - ORU-R01 - Hermes Conrad - Hospital A.hl7
   - OUL-R21 - Firstname Lastname - Acme Labs.hl7
   - OUL-R21 - Leo Wong - ABC Diagnostics.hl7

Results

After sending all seven of the above HL7 message files, the Dashboard statistics for the channel should be 5 Received, 2 Filtered and 5 Sent. In HL7 Inspector, for the messages filtered, you should see an ACK response with a code of AR (application rejected). All messages that were accepted should have an ACK response with a code of AA (application accepted).

The following table shows which messages were accepted or rejected, and the reason for the acceptance or rejection.

Message File Name	Source Connector Reject/Accept	Reject /Accept Reason
ADT-A01 - Jane MotherToBe - General Hospital.hl7	Accepted	Message type (ADT) and Sending Facility (General Hospital) requirements met
ADT-A03 - Zapp Brannigan - LAMH.hl7	Rejected	Sending Facility (LAMH) requirement not met
ADT-A08 - John Panucci - Sacred Heart.hl7	Accepted	Message type (ADT) and Sending Facility (Sacred Heart) requirements met
ORU-R01 - Hattie McDoogal - Acme Labs.hl7	Accepted	Message type/Trigger Event (ORU-R01) and Sending Facility (Acme Labs) requirements met
ORU-R01 - Hermes Conrad - Hospital A.hl7	Rejected	Sending Facility (Hospital A) requirement not met
OUL-R21 - Firstname Lastname - Acme Labs.hl7	Accepted	Message type/Trigger Event (OUL-R21) and Sending Facility (Acme Labs) requirements met
OUL-R21 - Leo Wong - ABC Diagnostics.hl7	Accepted	Message type/Trigger Event (OUL-R21) and Sending Facility (ABC Diagnostics) requirements met

Lab 18 Create a Channel that uses to JavaScript Transformers to Iterate over Message Segments and Manipulate the Message

Objectives

In this lab, you will create a new channel with two destination connectors. The first destination connector is a File Writer that uses a JavaScript transformer step to iterate over the segments of a lab results message and build a lab results report from some of the values in the OBR and OBX segments.

The second destination is an LLP Sender that uses a JavaScript transformer step to once again iterate over the message's segments, this time looking for any Z segments. If any Z segments are found, they should be deleted from the message. Additionally, this transformer step will add a new NTE (Note) segment after the existing PID segment.

These JavaScript transformers will demonstrate the concepts of iterating over segments in a message, deleting segments from a message and adding new segments to a message.

Estimated Time to Complete

20-25 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector
- Windows Explorer
- Any text editor, such as Notepad++

Step-By-Step

1. Create a new channel named *JavaScript Transformers*
2. Configure the source connector as a Channel Reader
3. Configure the default destination as a File Writer to write the lab report
   a. Rename the destination to `Lab Report File`
   b. Set the Directory field value to `/Mirth Training/outbox`
   c. Set the File Name field to the Unique ID destination mapping, followed by a .txt extension: `${UUID}.txt`
4. Create a JavaScript transformer step to build the text for the lab report file
   a. Click Edit Transformer in the Channel Tasks menu
   b. Click Add New Step in the Transformer Tasks menu
   c. In the new row added to the transformer steps list, double-click the cell in the Type column and select JavaScript from the drop-down menu
   d. Double-click the cell in the Name column to make it editable, and type `Build Lab Report`
   e. In the JavaScript editor, we will need a variable to hold the text for the lab report string. Add the following line of code:

```
var labReport = "";
```

f. Next, add code to loop over each segment in the message:

```
for each (seg in msg.children())
{
}
```

g. Within the loop, your code will need to check to see if the segment in the current iteration is an OBR, and if so, add a line to the report to show the order number:

```
if (seg.name() == "OBR")
{
      labReport += "Results for Order " +
      seg['OBR.2']['OBR.2.1'].toString() + "\n";
}
```

h. Also within the loop, if the segment in the current iteration is not an OBR, add code to check to see if it is an OBX, and if so, add a line to the lab report for the observation name (OBX.3.2), observation value (OBX.5.1) and observation units (OBX.6.1):

```
else if (seg.name() == "OBX")
{
      labReport += seg['OBX.3']['OBX.3.2'].toString() + ": " +
                   seg['OBX.5']['OBX.5.1'].toString() + " " +
                   seg['OBX.6']['OBX.6.1'].toString() + "\n";
}
```

i. Finally, after having iterated through all of the message's segments, you will need a line of code to add the generated lab report string to the channel map so that it is available to the destination connector:

```
channelMap.put("labReport", labReport);
```

j. The final code for the JavaScript transformer step should look like this (comments optional):

```
// String to hold lab report text.
var labReport = "";

// Loop through all segments
for each (seg in msg.children())
{
      // If OBR, output line with order number
      if (seg.name() == "OBR")
      {
            labReport += "Results for Order " +
            seg['OBR.2']['OBR.2.1'].toString() + "\n";
      }
      // If OBX, output line with observation name, value and units
      else if (seg.name() == "OBX")
      {
            labReport += seg['OBX.3']['OBX.3.2'].toString() + ": " +
                         seg['OBX.5']['OBX.5.1'].toString() + " " +
                         seg['OBX.6']['OBX.6.1'].toString() + "\n";
      }
```

```
}

// Put the report string in the channel map for use in the destination
template.
channelMap.put("labReport", labReport);
```

k. Click Back to Channel in the Mirth Views menu

5. Set the output Template value for the Lab Report File destination to the labReport channel map variable created in the previous step
6. Create and configure a second destination connector as an LLP Sender that will strip out any Z segments from the message and add a Note (NTE) segment
   a. Click New Destination in the Channel Tasks menu
   b. Rename the new destination connector to `Alter and Forward Message`
   c. Change the Connector Type to LLP Sender
   d. Change the Host Port field value to `9660`
7. Create a JavaScript transformer step to strip out any Z segments from the message and add an NTE segment
   a. Click Edit Transformer in the Channel Tasks menu
   b. Click Add New Step in the Transformer Tasks menu
   c. In the new row added to the transformer steps list, double-click the cell in the Type column and select JavaScript from the drop-down menu
   d. Double-click the cell in the Name column to make it editable, and type `Delete Z segments and insert NTE`
   e. In the JavaScript editor, to look for and delete any Z segments in the message, add the code to loop over each segment:

```
for each (seg in msg.children())
{
}
```

   f. Within the loop, add code to check if the segment is a Z segment (by looking at the first character of the segment's name) and, if so, delete it:

```
var name = seg.name().toString();
if (name.substr(0, 1) == "Z")
{
      delete msg[name];
}
```

   g. Next, after the loop, add code to add an NTE segment after the message's PID segment:

```
var noteSegment = createSegment('NTE');
noteSegment['NTE.1']['NTE.1.1'] = 1;
noteSegment['NTE.3']['NTE.3.1'] = "This is a note about the patient";
msg['PID'] += noteSegment;
```

   h. The final code for the JavaScript transformer step should look like this (comments optional):

```
// Loop through all segments
for each (seg in msg.children())
{
      // If the segment is a Z segment, delete it.
```

```
        var name = seg.name().toString();
        if (name.substr(0, 1) == "Z")
        {
              delete msg[name];
        }
}

// Create a NTE (note) segment, populate, and insert after PID segment.
var noteSegment = createSegment('NTE');
noteSegment['NTE.1']['NTE.1.1'] = 1;
noteSegment['NTE.3']['NTE.3.1'] = "This is a note about the patient";
msg['PID'] += noteSegment;
```

i. Click Back To Channel in the Mirth Views menu

8. Save the channel
9. Deploy the channel
10. Configure HL7 Inspector to listen on port 9660
11. Use the Dashboard's Send Message feature to send the following message (which contains lab order results as well as two Z segments) to the channel:
    OUL-R21 - Leo Wong - ABC Diagnostics.hl7

Results

The Dashboard statistics for the channel should show 1 received and 2 sent. In the /Mirth Training/outbox folder, you should see a new file with a unique number and .txt extension. This is the lab report file created by the first destination connector, Lab Report File. The contents of the file should be as follows:

```
Results for Order 1035052
White blood cells: 33.0 *10E 9/L
TSH: * mU/L
Albumin: * g/L
Vit. B12: * pmol/L
```

In the HL7 Inspector's Message Receiver window, you should see the message sent by the second destination connector, Alter and Forward Message. Notice that after the PID segment there is now an NTE segment:

```
NTE|1||This is a note about the patient
```

Also notice that the two Z segments in the original message, ZAA and ZBB, have been removed.

Lab 19 Create a Channel with a Database Writer that Uses JavaScript

Objectives

In this lab, you will create a channel that uses JavaScript in a Database Writer to conditionally insert a row in the database table, depending upon whether a row for the patient currently exists or not. This demonstrates how using JavaScript in a database connector allows for more complex logic, as well as multiple SQL statement executions per message processed.

This lab also has an optional second part. If you finish early, you can expand your JavaScript code for the Database Writer to update any existing rows in the database with data from the HL7 messages when a match is found.

If you wish to attempt this lab in your own local environment, the scripts to create the database table and insert the rows required by this lab are contained in the file "Patients Table.sql", located in the SQL subfolder of both your training materials CD and the Mirth Training folder on your student computer.

Estimated Time to Complete

15 minutes, plus an additional 10 minutes for the optional second part

Tools Used

- Mirth Connect Administrator
- SQuirreL SQL Client

Step-By-Step

1. Create a new channel named *Database Writer With JavaScript*
2. Configure the source connector as a Channel Reader
3. Configure the destination connector as a Database Writer with JavaScript
   a. Change the Connector Type to Database Writer
   b. In the Driver drop-down list, select PostgreSQL
   c. In the URL field, type `jdbc:postgresql://localhost:5432/mirthtraining`
   d. In the Username field, type `postgres`
   e. In the Password field, type `abc12345`
   f. For the Use JavaScript setting, select Yes
4. Create Mapper transformer steps to extract patient data from message
   a. Click Edit Transformer in Channel Tasks menu
   b. Set inbound message template to any ADT message
   c. For each of the PID segment fields specified in the table below, create a Channel Map variable with the given name

Inbound Message Template Field	Channel Map Variable Name
PID.2.1 (ID)	patientId
PID.5.1 (Patient Name – Family Name)	lastName
PID.5.2 (Patient Name – Given Name)	firstName
PID.5.3 (Patient Name – Second and Further Given Names or Initials Thereof)	middleName
PID.7.1 (Date/Time of Birth)	dateOfBirth
PID.8.1 (Administrative Sex)	gender
PID.11.1 (Patient Address – Street Address)	address1
PID.11.2 (Patient Address – Other Designation)	address2
PID.11.3 (Patient Address – City)	city
PID.11.4 (Patient Address – State or Province)	state
PID.11.5 (Patient Address – Zip or Postal Code)	postalCode

d. Click Back to Channel in the Mirth Views menu

5. Complete the JavaScript code to perform either an update or an insert depending upon if a row for the patient specified in the message already exists in the database
   a. After the line that creates the database connection, add the following two lines that query the Patients table for the count of existing rows with a given patient ID:

```
var sql = "SELECT Count(*) FROM Patients WHERE PatientID = '" +  + "'";
var result = dbConn.executeCachedQuery(sql);
```

   b. Drag the patientId variable from the Destination Mappings and drop it in-between the two plus signs in the statement above
   c. Next, add the following line of code, which creates an ArrayList object to hold the parameters for the INSERT statement:

```
var params = new Packages.java.util.ArrayList();
```

   d. Next, add the following if statement block:

```
if (result.next() && result.getInt(1) == 0)
{
      sql = "INSERT INTO patients (patientid, lastname, firstname, " +
            "middlename, gender, dateofbirth, " +
            "address1, address2, city, state, postalcode) " +
            "VALUES (?, ?, ?, ?, ?, To_Date(?, 'yyyyMMdd'), ?, ?, ?, ?, ?)";

      params.add();
      params.add();
      params.add();
      params.add();
      params.add();
      params.add();
      params.add();
      params.add();
      params.add();
      params.add();
      params.add();
}
```

This code checks the result from the executed query, and if a row for the given patient does not already exist, it creates the SQL statement to insert the row with the data from the message, and populates the parameter array with that data.

e. From the Destination Mappings list, drag each of the following variables (in order) and drop them into the parentheses of the params.add() call:
   - patientId
   - lastName
   - firstName
   - middleName
   - gender
   - dateOfBirth
   - address1
   - address2
   - city
   - state
   - postalCode

f. Finally, within the if block and after the last statement that adds a parameter to the array, add the following line of code to execute the INSERT statement:

```
dbConn.executeUpdate(sql, params);
```

g. At this point, your JavaScript code should look like this (comments optional):

```
var dbConn =
DatabaseConnectionFactory.createDatabaseConnection('org.postgresql.Driver',
'jdbc:postgresql://localhost:5432','postgres','abc12345');

var sql = "SELECT Count(*) FROM Patients WHERE PatientID = '" +
$('patientId') + "'";
var result = dbConn.executeCachedQuery(sql);

var params = new Packages.java.util.ArrayList();

// If no row exists, insert a new row.
if (result.next() && result.getInt(1) == 0)
{
      sql = "INSERT INTO patients (patientid, lastname, firstname, " +
            "middlename, gender, dateofbirth, " +
            "address1, address2, city, state, postalcode) " +
            "VALUES (?, ?, ?, ?, ?, To_Date(?, 'yyyyMMdd'), ?, ?, ?, ?, ?)";

      params.add($('patientId'));
      params.add($('lastName'));
      params.add($('firstName'));
      params.add($('middleName'));
      params.add($('gender'));
      params.add($('dateOfBirth'));
      params.add($('address1'));
```

```
        params.add($('address2'));
        params.add($('city'));
        params.add($('state'));
        params.add($('postalCode'));

        dbConn.executeUpdate(sql, params);
    }


    dbConn.close();
```

6. Save the channel
7. Deploy the channel
8. View the current state of the data in the Patients table (optional)
   a. Open the SQuirreL SQL Client application
   b. In the Alias window, select Mirth Training Database
   c. Right-click, and select Connect from the context menu
   d. In the "Connect to" dialog that appears, click the Connect button to connect to the database and open the session window
   e. In the text area of the window, type the following SQL query:
      `SELECT * FROM Patients;`
   f. With the cursor still on the line that you just typed, press Ctrl-Enter to execute the query (or press the button on the toolbar that looks like a person running)
   g. In the results pane of the window, you should see two rows of patient data, for patients Amy Wong and Zapp Brannigan. Note the current address information for these patients.
9. Use the Dashboard's Send Message feature to send the following messages to the channel:
   - ADT-A03 - Zapp Brannigan - LAMH.hl7
   - ADT-A08 - John Panucci - Sacred Heart.hl7
   - ADT-A08 - Lars Fillmore - Hospital B.hl7
   - ADT-A60 - Amy Wong - LAMH.hl7

Results

After you have sent all four messages, the statistics in the Dashboard for the channel should show 4 received and 4 sent. In the Squirrel SQL Client, if you run the SELECT statement once again, you should now see four rows. Note that for the two previously existing patients, Zapp Brannigan and Amy Wong, their rows in the table remain unaltered. In addition, the other two patients, John Panucci and Lars Fillmore, have been added to the table with the data from their respective messages.

Step-By-Step, Part 2

10. In the Database Writer's JavaScript code, add code to update an existing row with data from the HL7 message if a match has been found.
    a. Add the following else statement block after the existing if block:

```
else
{
      sql = "UPDATE Patients " +
            "SET Address1 = ?, Address2 = ?, City = ?, State = ?, PostalCode
= ?  " +
            "WHERE PatientID = ? ";

      params.add();
      params.add();
      params.add();
      params.add();
      params.add();
      params.add();

      dbConn.executeUpdate(sql, params);
}
```

b. From the Destination Mappings list, drag each of the following variables (in order) and drop them into the parentheses of the params.add() call:
   - address1
   - address2
   - city
   - state
   - postalCode

c. At this point, your JavaScript code should look like this (comments optional):

```
var dbConn =
DatabaseConnectionFactory.createDatabaseConnection('org.postgresql.Driver',
'jdbc:postgresql://localhost:5432/mirthtraining','postgres','abc12345');

var sql = "SELECT Count(*) FROM Patients WHERE PatientID = '" +
$('patientId') + "'";
var result = dbConn.executeCachedQuery(sql);

// Create an ArrayList to hold the parameters for the subsequent SQL
statement.
var params = new Packages.java.util.ArrayList();

// If no row exists, insert a new row.
if (result.next() && result.getInt(1) == 0)
{
      sql = "INSERT INTO patients (patientid, lastname, firstname, " +
            "middlename, gender, dateofbirth, " +
            "address1, address2, city, state, postalcode) " +
            "VALUES (?, ?, ?, ?, ?, To_Date(?, 'yyyyMMdd'), ?, ?, ?, ?, ?)";

      params.add($('patientId'));
      params.add($('lastName'));
      params.add($('firstName'));
      params.add($('middleName'));
```

```
        params.add($('gender'));
        params.add($('dateOfBirth'));
        params.add($('address1'));
        params.add($('address2'));
        params.add($('city'));
        params.add($('state'));
        params.add($('postalCode'));

        dbConn.executeUpdate(sql, params);
    }
    // Otherwise, update the existing row
    else
    {
        sql = "UPDATE Patients " +
              "SET Address1 = ?, Address2 = ?, City = ?, State = ?, PostalCode
    = ?  " +
              "WHERE PatientID = ? ";

        params.add($('address1'));
        params.add($('address2'));
        params.add($('city'));
        params.add($('state'));
        params.add($('postalCode'));
        params.add($('patientId'));

        dbConn.executeUpdate(sql, params);
    }

    dbConn.close();
```

11. Save the channel
12. Deploy the channel
13. Use the Dashboard's Send Message feature to send the following messages to the channel:
    - ADT-A03 - Zapp Brannigan - LAMH.hl7
    - ADT-A60 - Amy Wong - LAMH.hl7

Results, Part 2

After you have sent all four messages, the statistics in the Dashboard for the channel should now show 6 Received and 6 Sent. In the Squirrel SQL Client, if you run the SELECT statement once again, you should still see four rows. Note that for the two previously existing patients, their address information has now been updated with the address data from the PID segment of the message.

Lab 20 Create a Channel with a JavaScript Reader to Poll a Java Application for Message

Objectives

In this lab, you will create a channel that uses a JavaScript Reader source connector to poll a Java class for messages. This simulates a situation where a Java library already exists that is used to connect to some system to retrieve messages.

The Java class that will be called by the JavaScript Reader, MessageFactory, randomly returns an ArrayList of anywhere from 0 to 4 HL7 messages each time its getMessages() function is called. The class exists in a JAR file, MessageFactory.jar in the /Mirth Training/jars folder. As part of this lab, you will need to install this file for use by Mirth Connect by copying it to the correct folder and restarting the Mirth Connect Server.

Note: because the channel created in this lab will continuously poll for messages, and usually receive at least one message each time the polling occurs, it is recommended that you stop and disable the channel upon completion of the lab.

Estimated Time to Complete

7-8 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer

Step-By-Step

1. Copy the MessageFactory.jar file, located in /Mirth Training/jars, to Mirth Connect's lib/custom folder
2. Close Mirth Connect Administrator if it is currently open
3. Restart the Mirth Connect Server
4. Restart the Mirth Connect Administrator
5. Create a new channel named *JavaScript Reader*
6. Configure the source connector as a JavaScript Reader
   a. On the Source tab, select JavaScript Reader from the Connector Type drop-down list
   b. Change the value for the Polling Frequency (ms) field to `20000` (20 seconds)
   c. In the JavaScript editor, first add the line to call the MessageFactory class to get the list of messages:

```
var messages =
Packages.com.mirthcorp.mirthconnect.training.MessageFactory.getMessages();
```

   d. Next, add code to log the total number of messages received in the list, as well as the contents of each message in the list:

```
logger.info("Received " + messages.size() + " messages.");
for (i = 0; i < messages.size(); i++)
{
      logger.info("message[" + i + "] = " + messages.get(i));
}
```

e. Finally, add code to return the list of messages:

```
return messages;
```

f. The final JavaScript Reader code should look like this (comments optional):

```
// Get the message(s) from the message factory.
var messages =
Packages.com.mirthcorp.mirthconnect.training.MessageFactory.getMessages();

// Log the message info.
logger.info("Received " + messages.size() + " messages.");
for (i = 0; i < messages.size(); i++)
{
      logger.info("message[" + i + "] = " + messages.get(i));
}

// Return the list of messages.
return messages;
```

7. Configure the destination connector as a File Writer
   a. Set the Directory field to `/Mirth Training/outbox`
   b. Set the File Name field to the Unique ID destination mapping, followed by a .txt extension: `${UUID}.txt`
   c. Set the Template field to the Raw Data destination mapping
8. Save the channel
9. Deploy the channel

Results

Immediately after deployment, you should see the channel processing the first time. Because the interval value was set to 20 seconds, after 20 seconds you should see the channel processing once again, as well as every 20 seconds after that until you stop or disable the channel.

Each time the channel performs its processing (i.e., each time the JavaScript Reader polls for data by executing its code), you should see entries in the Server Log tab indicating the number of messages received during that execution, as well as each message received. In the Dashboard statistics for the channel, you should see the Received and Sent counts incremented by the number of messages received from the MessageFactory, as indicated in the logging output.

Finally, in the /Mirth Training/outbox folder, you should see a file created for each message received by the JavaScript Reader.

Lab 21 Create Global Deploy Script to Load Configuration Properties from Database

Objectives

In this lab, you will edit the global Deploy script to load configuration property/value pairs from a database table. You will then load all of the properties into the Global Map. These configuration properties will be used later to demonstrate dynamic configuration in Lab 26.

Estimated Time to Complete

10 minutes

Tools Used

- Mirth Connect Administrator

Step-By-Step

1. Go to the global Deploy script
   a. If not already in Channels view, click Channels in the Mirth Connect menu
   b. Click Edit Global Scripts in the Channel Tasks menu
   c. In the Script drop-down menu at the top of the pane, select Deploy
2. Delete any existing code
3. Insert the code to execute the query to get the configuration properties
   a. From the reference list at the right, drag over Perform Database Query and drop it into the JavaScript editor
   b. In the first line of the inserted text, delete the first parameter, `'driver'` (including single-quotes)
   c. In its place, drag over Postgres Driver from the reference list
   d. Replace the `'address'` parameter with `'jdbc:postgresql://localhost:5432/mirthtraining'`
   e. Replace the `'username'` parameter with `'postgres'`
   f. Replace the `'password'` parameter with `'abc12345'`
   g. In the second line of code, replace the `'expression'` parameter with `'SELECT * FROM Configuration'`
4. Insert the code to loop over the results and add the property/value pairs to the Global Map
   a. After the line with the call to executeCachedQuery(), add the following code to loop over the results:

```
while (result.next())
{
}
```

   b. Within the while loop, drag over the Put Global Variable Map template to add the code to add a global variable
   c. In the call to `globalMap.put()`, replace the first parameter, `key`, with `result.getString(1)`
   d. In the call to `globalMap.put()`, replace the second parameter, `''`, with `result.getString(2)`
   e. Add a semicolon to the end of the call to `globalMap.put()`

f. After the while loop, add the following line to log the contents of the global map:

```
logger.info("Loaded properties into global map: " +
            globalMap.globalVariableMap);
```

g. The Deploy script code should now look like this:

```
var dbConn =
DatabaseConnectionFactory.createDatabaseConnection("org.postgresql.Driver",
'jdbc:postgresql://localhost:5432/mirthtraining', 'postgres', 'abc12345');
var result = dbConn.executeCachedQuery('SELECT * FROM Configuration');
while (result.next())
{
      globalMap.put(result.getString(1), result.getString(2));
}
logger.info("Loaded properties into global map: " +
            globalMap.globalVariableMap);
dbConn.close();
```

5. Save the script by clicking Save Scripts in the Script Tasks menu
6. Deploy all channels

Results

After the enable channels have been deployed, you should see an entry on the Server Log tab in the Dashboard that displays the contents of the Global Map. In these contents, you should see the nine property/value pairs loaded from the Configuration table.

Lab 22 Write a Preprocessor Script that Strips Z Segments from the Message

Objectives

In this lab, you will create a new channel with a preprocessor script that strips any Z segments from all messages that it processes. To accomplish this, the code in the script will:

- Use the JavaScript String object's split() function to split the message into its individual segments
- Create a new variable to hold the "fixed" message
- Iterate over the elements of the array that holds the message segments, and if the segment is not a Z segment, add it to the new message string

Estimated Time to Complete

10-15 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector

Step-By-Step

1. Create a new channel named *Preprocessor Script Strip Z Segments*
2. Configure the source connector as an LLP Listener
   a. Change the Listener Port to `9650`
3. Configure the destination connector as an LLP Sender
   a. Change the Host Port to `9651`
4. Create a preprocessor script to strip Z segments from the message
   a. Click the Scripts tab
   b. In the Script drop-down list, select Preprocessor
   c. In the JavaScript editor, first write the code to split the message into segments by splitting over the carriage return character:

```
// Split the message into its segments.
var segments = message.split("\r");
```

   d. Next, create a variable that will be used to build the "fixed" version of the message:

```
var newMessage = "";
```

   e. Next, write the loop statement to iterate over each segment and, if it is not a Z segment, add the segment string to the `newMessage` variable. To check if a string is a Z segment, look at the first character in the string:

```
// For each segment in the message...
for (i = 0; i < segments.length; i++)
{
      // If this is not a Z segment...
      if (segments[i].substr(0, 1) != "Z")
```

```
        {
                // Add the segment to the new message.
                newMessage += segments[i] + "\r";
        }
}
```

f. Finally, return the newMessage string:

```
return newMessage;
```

g. The final code, in its entirety, should look like:

```
// Split the message into its segments.
var segments = message.split("\r");

var newMessage = "";

// For each segment in the message...
for (i = 0; i < segments.length; i++)
{
        // If this is not a Z segment...
        if (segments[i].substr(0, 1) != "Z")
        {
                // Add the segment to the new message.
                newMessage += segments[i] + "\r";
        }
}

return newMessage;
```

5. Save the channel
6. Deploy the channel
7. Configure HL7 Inspector to send to port 9650
8. Configure HL7 Inspector to receive on port 9651
9. Use HL7 Inspector to send one or more of the following messages (which all have Z segments) to the channel:
   - ADT-A01 - Jane MotherToBe - General Hospital.hl7
   - DFT-P03 - Betsy Cardiacpatient.hl7
   - OUL-R21 - Leo Wong - ABC Diagnostics.hl7

Results

For each message that you send to the channel, you should see the received and sent statistics in the Dashboard increase by one. In the Channel Messages view, if you look at the Raw Message tab for the source connector, you should see that the message's Z segments have been removed. As well, looking in the Message Receiver tab of HL7 Inspector, you should see the messages received there have also had their Z segments removed.

Lab 23 Create a Channel with a Postprocessor Script that Generates a Custom ACK Response

Objectives

In this lab, you will create a channel that uses transformer steps and a postprocessor script to create a custom ACK response message. The destination connector uses an outbound template and transformer steps to build an ACK message from data in the inbound message. It then sends a request to a web server using the HTTP Writer. Based on the return values in the response returned by the web server, the postprocessor script will populate the ACK message with the proper acknowledgement code and message and create the correct response type. The ACK response will then be returned to the original caller by the LLP Listener source connector.

Whether the response from the HTTP server has a status of SUCCESS or FAILURE depends upon the data given to it. For the request to be a success, the following parameters must be present in the request with one of the indicated values:

- patientId – values 1000 to 1005
- messageType – ADT
- messageEvent – A01, A02 or A03

Estimated Time to Complete

25-30 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector

Step-By-Step

1. Create a new channel named *Custom Response*
2. Configure the source connector as an LLP Listener, listening on port 9662
3. Configure the destination connector as an HTTP Sender, sending to a web application running on the instructor's computer
   a. Rename the default destination to `HTTP Server`
   b. Change the Connector Type to HTTP Sender
   c. Set the URL to `http://host:8081/mirthtraining/HttpServer`, where *host* is the IP address of the instructor's computer
4. Set the inbound and outbound message templates for the destination connector
   a. Click Edit Transformer in the Channel Tasks menu
   b. On the Message Templates tab, set the Inbound Message Template to any ADT message, or any other message with a PID segment
   c. Set the Outbound Message Template to "ACK Template.hl7" in the /Mirth Training/Messages/HL7/Templates folder

5. Create the transformer steps to populate the ACK message (outbound template) with data from the inbound message
   a. Click the Message Trees tab
   b. For each of the mappings in the table below, use the drag-and-drop method from the Inbound Message Template Tree to the Outbound Message Template Tree to generate Message Builder steps

Inbound Message Field	Outbound Message Field
MSH.3.1	MSH.5.1
MSH.4.1	MSH.6.1
MSH.10.1	MSH.10.1
MSA.2.1	MSH.10.1
MSH.11.1	MSH.11.1
MSH.12.1	MSH.12.1

   c. Populate the message date/time field (MSH.7.1) in the outbound message with the current date time
      i. In the Outbound Message Template Tree, select the MSH.7.1 node, right click and select Map Segment from the context menu
      ii. In the Mapping field, type the following code:

```
DateUtil.getCurrentDate('yyyyMMddHHmmss')
```

   d. Add a JavaScript transformer step that puts the outbound message template variable, tmp, into the Channel Map for later use by the Postprocessor script
      i. Click Add New Step in the Transformer Tasks menu
      ii. Double-click the cell in the Type column for the new step, and select JavaScript from the drop-down list
      iii. In the JavaScript editor, type the following line of code:

```
channelMap.put('ackMessage', tmp);
```

6. Create Mapper transformer steps to extract the data needed to send to the HTTP server
   a. Drag-and-drop the following fields from the Inbound Message Template tree to create Channel Map variables:
      - MSH.9.1 (Message Type)
      - MSH.9.2 (Trigger Event)
      - PID.2.1 (Patient ID)
   b. Click Back to Channel in the Mirth Views menu
7. Add the following request variables to the HTTP request:

Variable Name	Value (from destination mappings)
messageType	${messageHeader_messageType_messageType}
messageEvent	${messageHeader_messageType_triggerEvent}
patientId	${patientIdentification_patientId_id}

8. Create the Preprocessor script that will get the response from the HTTP Server destination, set the acknowledgement code and message based on the response status, and put the ACK message into the Response Map
   a. Get the response from the HTTP Server destination:

```
var httpResponse = responseMap.get('HTTP Server');
```

b. Get the ACK message created placed into the Channel Map in the transformer:

```
var ackMessage = channelMap.get('ackMessage');
```

c. Get the response body, which is the last line in the list of headers returned as the response message:

```
var headers = httpResponse.getMessage().split("\n");
responseBody = headers[headers.length - 1];
```

d. Set the acknowledgement code and message as needed:

```
if (httpResponse.getStatus() == "FAILURE")
{
      ackMessage['MSA']['MSA.1']['MSA.1.1'] = "AE";
}
ackMessage['MSA']['MSA.3']['MSA.3.1'] = responseBody;
```

e. The ACK message, stored and updated in XML format, must now be converted back to HL7:

```
ackMessage =
SerializerFactory.getHL7Serializer(false,false,true).fromXML(ackMessage.toString());
```

You can get the code to serialize the XML into HL7 by using the Convert XML to HL7 code template in the reference list

f. Finally, create a response an put it into the Response Map variable "ACK Message" based on the response status:

```
if (httpResponse.getStatus() == "SUCCESS")
{
      responseMap.put('ACK Message',
                  ResponseFactory.getSuccessResponse(ackMessage));
}
else
{
      responseMap.put('ACK Message',
                  ResponseFactory.getFailureResponse(ackMessage));
}
```

g. The final Postprocessor script code should look like this (comments optional):

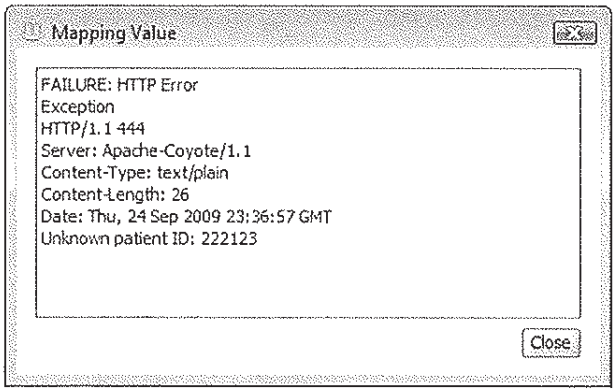
```
// Get the response string.
var httpResponse = responseMap.get('HTTP Server');

// Get the acknowledgement message that we built in the ACK Message
destination.
var ackMessage = channelMap.get('ackMessage');

// Get each individual header line.  The last line is the response body.
var headers = httpResponse.getMessage().split("\n");
responseBody = headers[headers.length - 1];

// If the HTTP connection failed...
if (httpResponse.getStatus() == "FAILURE")
```

```
{
      ackMessage['MSA']['MSA.1']['MSA.1.1'] = "AE";
}
ackMessage['MSA']['MSA.3']['MSA.3.1'] = responseBody;

// Convert the ACK message to HL7.
ackMessage =
SerializerFactory.getHL7Serializer(false,false,true).fromXML(ackMessage.toStr
ing());

// If the message was successfully processed...
if (httpResponse.getStatus() == "SUCCESS")
{
      // Put the success ACK message in the response map.
      responseMap.put('ACK Message',
                  ResponseFactory.getSuccessResponse(ackMessage));
}
// Otherwise...
else
{
      // Put the failure ACK message in the response map.
      responseMap.put('ACK Message',
                  ResponseFactory.getFailureResponse(ackMessage));
}
```

9. Update the LLP Listener source connector to respond using the ACK response stored in the "ACK Message" Response Map variable
   a. On the Source tab, for the Send ACK setting, select Respond from
   b. In the drop-down list to the right of the Respond from radio button, select ACK Message
10. Save the channel
11. Deploy the channel
12. Configure HL7 Inspector to send to the channel on port 9662
13. Send the following messages to the channel using HL7 Inspector:
    - ADT-A01 - Testfirst Testlast - SHM.hl7
    - ADT-A03 - Zapp Brannigan - LAMH.hl7
    - ADT-A08 - Lars Fillmore - Hospital B.hl7
    - ORU-R01 - Hermes Conrad - Hospital A.hl7

Results

The Dashboard statistics for the channel should show 4 Received, 1 Sent, and 3 Errored. Go to the Channel Messages view to see the details for the messages. Click the Mappings tab. Select each row for the HTTP Server destination connector. As you select each row, you can see the response from the HTTP server in the Response map. Double-click the Response row to view the Mapping Value.



23-1 Error Response from HTTP Server

The number value after HTTP/1.1 is the response code (200=OK, 444=error). The last line response body returned from the HTTP server. This will indicate either that the message was successfully processed, or give the reason why it returned an error code. The following table shows the expected responses for each of the messages sent:

HL7 Message	Response Code	Message
ADT-A01 - Testfirst Testlast - SHM.hl7	200	Message successfully processed
ADT-A03 - Zapp Brannigan - LAMH.hl7	444	Unknown patient ID: 222123
ADT-A08 - Lars Fillmore - Hospital B.hl7	444	Unsupported message type/event: ADT-A08
ORU-R01 - Hermes Conrad - Hospital A.hl7	444	Unsupported message type/event: ORU-R01

Finally, in HL7 Inspector, you should see the custom ACK message responses created by the channel and sent back by the LLP Listener. For each ACK, the MSA segment should contain the code corresponding to the response code in the table above (200 = AA, 444 = AE) in MSA.1. In MSA.3.1 should contain the message from the body of the HTTP response.

Lab 24 Create Code Template Functions for Normalizing and De-normalizing Social Security Numbers

Objectives

In this lab, you will create two code template functions that will be available for use in any JavaScript context. The first function, normalizeSSN(), takes a Social Security Number and removes any dashes from it. The second function, denormalizeSSN(), takes a normalized Social Security Number and inserts dashes in the appropriate locations.

Estimated Time to Complete

7-8 minutes

Tools Used

- Mirth Connect Administrator

Step-By-Step

1. In the Mirth Connect Administrator, click Channels in the Mirth Connect menu
2. Click Edit Code Templates in the Channel Tasks menu
3. Create the normalizeSSN() code template function
   a. Click New Code Template in the Code Template Tasks menu
   b. Double-click the cell in the Name column for the new entry in the code templates list to make it editable
   c. Replace the default name with `normalizeSSN` and press the Enter key
   d. In the Context drop-down list, select Global
   e. In the Description field, type:
      `Removes all dashes from a given SSN`
   f. In the Function JavaScript editor, type the following code to define the function:

```
function normalizeSSN(ssn)
{
      return ssn.replace(/-/g, "");
}
```

4. Create the denormalizeSSN() code template function
   a. Click New Code Template in the Code Template Tasks menu
   b. Double-click the cell in the Name column for the new entry in the code templates list to make it editable
   c. Replace the default name with `denormalizeSSN` and press the Enter key
   d. In the Context drop-down list, select Global
   e. In the Description field, type:
      `Inserts dashes at the appropriate locations into a normalized SSN`
   f. In the Function JavaScript editor, type the following code to define the function:

```
function denormalizeSSN(ssn)
{
	// First, make sure the number is normalized.
	ssn = normalizeSSN(ssn);

	return ssn.substr(0, 3) + "-" + ssn.substr(3, 2) + "-" + ssn.substr(5);
}
```

5. Click Save Code Templates in the Code Template Tasks menu

Results

Now that you have created and save the two code template functions, they should now be available in your reference list in all JavaScript contexts. To test, create a new channel, and go to any of channel scripts on the Scripts tab. In the Filter drop-down list for the reference, select User Defined Functions. You should see the two functions that you created. If you hover your mouse above the function in the list, you should see a tool tip with the description of the function. You should also be able to drag either of the functions into the JavaScript editor.

Lab 25 Create a Channel that Routes Messages Using Channel Writer, VMRouter

Objectives

In this lab, you will create a channel that has two destination connectors. The first destination is a File Writer with a JavaScript transformer step that uses the VMRouter to conditionally route the message to another channel based on the message type. If the message type is ADT, the message is routed to the PDF Writer channel. If the message type is RDE, the message is routed to the Email Sender channel.

The second destination is a Channel Writer, which unconditionally routes the message to the JavaScript Transformers channel.

Estimated Time to Complete

15 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector
- Windows Explorer

Step-By-Step

1. Create a new channel named *VMRouter and Channel Writer*
2. Configure the source connector as an LLP Listener
   a. Change the Listener Port to `9655`
3. Configure the default destination connector as a File Writer
   a. Rename the destination to Write to File
   b. Set Directory to /Mirth Training/outbox
   c. Set File Name to the Message ID destination mapping followed by a .txt extension: `${message.id}.txt`
   d. Set the Template to the Encoded Data destination mapping
4. Add a JavaScript transformer step to conditionally route the message to other channels
   a. Click Edit Transformer in the Channel Tasks menu
   b. Set the Inbound Message Template to any HL7 message
   c. Click Add New Step in the Transformer Tasks menu
   d. Double-click the cell in the Name column for the new step, and type `Routing with VMRouter`
   e. Double-click the cell in the Type column, and select JavaScript from the drop-down list
   f. In the JavaScript editor, add the if statement that will route the message to the PDF Writer channel if the message type is ADT
      i. Add an empty if block:

```
if ()
{
}
```

ii. In the condition part of the statement, drag over the code for MSH.9.1 from the Inbound Message Template Tree
iii. Complete the if condition by adding `== "ADT"`
iv. In the body of the if statement, drag over Route Message to Channel from the Reference tab (Utility Functions)
v. In the call to `router.routeMessage()`, replace the `channelName` parameter with `'PDF Writer'`
vi. Delete the `'message'` parameter
vii. In its place, drag over Convert XML to HL7 from the Reference tab (Conversion Functions)
viii. In the call to `SerializerFactory.getHL7Serializer()`, replace each of the three default parameters with `false`
ix. In the call to `fromXML()` on the same line, replace message with `msg`
x. Remove the semicolon after the call to `fromXML()`
xi. The if statement code should now look as follows:

```
if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "ADT")
{
  router.routeMessage('PDF Writer',
  SerializerFactory.getHL7Serializer(false, false, false).fromXML(msg));
}
```

g. After the if statement block, create an else-if statement that will route the message to the Email Sender channel if the message type is RDE
xii. Follow the sub-steps step f above, changing the following:
- Use an else-if instead of an if statement
- The if condition should compare to "RDE" instead of "ADT"
- Route to the Email Sender channel instead of PDF Writer

h. The final code should look like this:

```
if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "ADT")
{
      router.routeMessage('PDF Writer',
      SerializerFactory.getHL7Serializer(false, false, false).fromXML(msg));
}
else if (msg['MSH']['MSH.9']['MSH.9.1'].toString() == "RDE")
{
      router.routeMessage('Email Sender',
      SerializerFactory.getHL7Serializer(false, false, false).fromXML(msg));
}
```

i. Click Back to Channel in the Mirth Views menu

5. Create and configure a second destination connector as a Channel Writer
a. Click New Destination in the Channel Tasks menu
b. Rename the destination to `Forward to SOAP Listener Channel`
c. Select Channel Writer from the Connector Type drop-down list
d. In the Channel Name drop-down list, select SOAP Listener
e. Set Wait for Channel Response to Yes

6. Save the channel
7. Make sure that the Email Sender, PDF Writer and SOAP Listener channels are all enabled
8. Deploy the channels
9. Configure HL7 Inspector to send to the channel on port 9655
10. Send the following three messages to the VMRouter and Channel Writer channel using HL7 Inspector:
    - ADT-A01 - Jane MotherToBe - General Hospital.hl7
    - DFT-P03 - Betsy Cardiacpatient.hl7
    - RDE-O11 - Ada Clare.hl7

Results

In the Dashboard's statistics, the VMRouter and Channel Writer channel should show 3 Received and 6 Sent. The SOAP Listener channel's statistics should have increased by 3 for both Received and Sent (all three messages were sent to this channel unconditionally by the Channel Writer destination). The Email Sender channel's statistics should have increased by 1 for both Received and Sent (the RDE-O11 message was routed to it using the VMRouter). The PDF Writer channel's statistics should have also increased by 1 for both Received and Sent (the ADT-A01 message was routed to it using the VMRouter).

In the /Mirth Training/outbox folder, you should see four new files: three text files created by the Write to File destination, and one PDF file created by the Write PDF channel.

Lab 26 Create a Channel that Uses Dynamic Configuration to Route Message

Objectives

In this lab, you will create a channel that demonstrates the use of dynamic configuration to route messages to certain destination locations. The channel will have an LLP Listener source connector, a File Writer destination connector and an LLP destination connector. Each of these connectors will use the configuration properties read in from the Configuration database table and stored in the global map in Lab 21. Also, the File Writer will write to a subfolder based on the value in the MSH.4.1 (Sending Facility) field.

Estimated Time to Complete

10-15 minutes

Tools Used

- Mirth Connect Administrator
- HL7 Inspector
- Windows Explorer

Step-By-Step

1. Create a new channel named *Dynamic Configuration*
2. Configure the source connector as an LLP Listener, using the property names from the Configuration table as placeholders (Velocity template variables) as listed in the following table:

LLP Listener Field	Configuration Property Name
Listener Address	llplistener_ipaddress
Listener Port	llplistener_port
Successful ACK Code	llplistener_ackmsgsuccess
Error ACK Code	llplistener_ackmsgerror
Rejected ACK Code	llplistener_ackmsgrejected

3. Configure the default destination connector as a File Writer
   a. Rename the destination to `File`
   b. Set the Template to the Encoded Data destination mapping
4. Add a Mapper transformer step to get the sending facility value from MSH.4.1 and store it in a Channel Map variable named sendingFacility
5. Finish configuring the File Writer destination
   a. Set the Directory to a subfolder based on the sending facility value in the inbound message, in the folder specified by the file_outbox property:
   `${file_outbox}/${sendingFacility}`
   b. Set File Name to the value of the file_prefix property, followed by the Unique ID destination mapping, followed by a .txt extension:
   `${file_prefix}${UUID}.txt`
6. Create and configure a second destination connector as an LLP Sender
   a. Name the destination `LLP`

b. Change the Connector Type to LLP Sender
c. Set Host Address to the configuration property llpsender_ipaddress
d. Set Host Port to the configuration property llpsender_port

7. Save the channel
8. Deploy the channel
9. Configure HL7 Inspector to send to port 9680, and listen on port 9681
10. Send any message to the channel using HL7 Inspector

Results

The Dashboard's statistics for the channel should show 1 Received and 2 Sent. In the /Mirth Training/outbox, you should see a folder with the same name as the value specified in MSH.4.1 (Sending Facility) of the sent message. Within that folder should be a file whose name starts with "training_", followed by a unique ID. Finally, in HL7 Inspector's Message Receiver window, you should see that HL7 Inspector successfully received the sent message.

Lab 27 Create Channels that Use Base64 Encoding/Decoding

Objectives

In this lab, you will create two channels to demonstrate the encoding and decoding of binary data in Mirth Connect. The first channel will use a File Reader source connector to read a binary PDF file and a File Writer destination connector to encode the binary PDF data into Base64, insert it into the OBX segment of an outbound HL7 message template, and write that message out to a file as ASCII data. The channel will use a preprocessor script to place the PDF data into an XML structure that is usable by the source connector.

The second channel will use a File Reader to read the ASCII HL7 file created by the first channel, while filtering out any messages that don't contain PDF data in the OBX segment. A File Writer destination connector will then take the Base64 data from the OBX segment for the PDF and write it out as a binary file, making it once again readable as a PDF file.

Estimated Time to Complete

15-20 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer
- Adobe Acrobat Reader

Step-By-Step

1. Create a new channel named *Encode PDF in OBX*
2. On the Summary tab, change the Incoming Data type to XML
3. Create a preprocessor script to put the message data into an XML structure
   a. Click the Scripts tab
   b. Select Preprocessor from the Script drop-down list
   c. Replace the default preprocessor script code with the following code to put the message data into an XML document:

```
return "<pdf><data>" + message + "</data></pdf>";
```

4. Configure the source connector as a File Writer that will read binary PDF files
   a. On the Source tab, select File Reader from the Connector Type drop-down list
   b. Set Directory to `/Mirth Training/inbox`
   c. Set Filename Filter Pattern to `*.pdf`
   d. Set Delete File After Read to Yes
   e. Set Check File Age to Yes
   f. Set File Age (ms) to `3000`
   g. Set File Type to Binary
5. Configure the destination connector as a File Writer to write an ASCII HL7 file
   a. Rename the default destination connector to `Write HL7 to File`
   b. Set Directory to `/Mirth Training/outbox`

c. Set the File Name field to the Unique ID destination mapping, followed by a .txt extension: `${UUID}.txt`
d. Set Template to the Encoded Data destination mapping

6. Set the inbound and outbound message templates needed to create the outbound message
   a. Click Edit Transformer in the Channel Tasks menu
   b. On the Message Templates tab, in the field for the Inbound Message Template, type the following XML:

      ```
      <pdf><data>DATA</data></pdf>
      ```

   c. Set the Outbound Message Template to "MDM-T02 Template.hl7" in the /Mirth Training/Messages/HL7/Templates folder
7. Create the Message Builder transformer steps required to set the OBX fields in the outbound message for the PDF data
   a. On the Message Trees tab, for the Outbound Message Template Tree, expand the OBX branch by selecting OBX, right-clicking, and selecting Expand from the context menu
   b. Set the OBX.3.1 (Observation Identifier) value to a unique identifier
      i. Select the OBX.3.1 node, right-click and select Map Segment from the context menu to create a new Message Builder step for the field
      ii. In the Mapping field for the new step type the following code:

          ```
          UUIDGenerator.getUUID()
          ```

      iii. Note that you can also get this code from the Generate Unique ID code template in the Reference tab.
   c. Set the OBX.5.2 value to `"PDF"` using the same method in step b above
   d. Also set the OBX.5.3 value to `"PDF"` using the same method in step b above
   e. Set the OBX.5.5 value to the value in the data element of the inbound XML message by dragging the node in the data element in the Inbound Message Template Tree to the OBX.5.5 node in the Outbound Message Template Tree
8. Save the channel
9. Create a second new channel named *Decode PDF in OBX*
10. Configure the source connector as a File Reader to read ASCII HL7 messages
    a. On the Source tab, select File Reader from the Connector Type drop-down menu
    b. Set Directory to `/Mirth Training/inbox`
    c. Set Filename Filter Pattern to `*.txt`
    d. Set Delete File After Read to Yes
11. Create filter rules for the source connector to only accept messages that are of type MDM-T02 and have a value of "PDF" in the OBX.5.3 field (optional)
    a. Click Edit Filter in the Channel Tasks menu
    b. Set the Inbound Message Template to "MDM-T02 Template.hl7" in the /Mirth Training/Messages/HL7/Templates folder
    c. Using the drag-and-drop method from the Inbound Message Template Tree, create Rule Builder filter rules for the following three conditions:
       - MSH.9.1 equals "MDM"
       - MSH.9.2 equals "T02"

- OBX.5.3 equals "PDF"

12. Configure the destination connector as a File Writer that will create a binary PDF file from the Base64 PDF data in OBX.5.5
    a. Rename the default destination connector to `Write PDF File`
    b. Set Directory to `/Mirth Training/outbox`
    c. Set the File Name field to the Unique ID destination mapping, followed by a .pdf extension: `${UUID}.pdf`
    d. Set File Type to Binary
13. Create a Channel Map variable named pdfData from the value in OBX.5.5 of the inbound message
14. For the Template field of the destination connector, set it to the pdfData destination mapping created in the previous step
15. For each of the four message templates (inbound and outbound on source and destination connectors), change the properties to disable the automatic conversion of line feed (LF) to carriage return (CR) characters
    a. On the source connector, click Edit Transformer in the Channel Tasks menu
    b. Select the Message Templates tab in the right-hand pane
    c. For the Inbound Message Template, click the Properties button
    d. In the Properties dialog that appears, uncheck the Convert LF to CR option

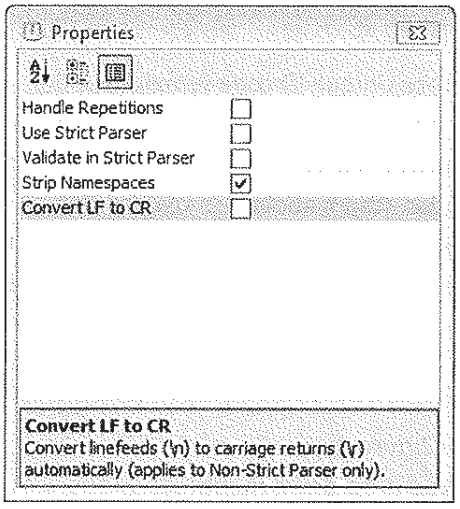


Figure 27-1 Properties dialog for HL7 template

    e. Click the close button (the X button in the top-right corner) to close the dialog
    f. Repeat steps c through 0 above for the Outbound Message Template properties
    g. Repeat steps a through f above to disable the Convert LF to CR option for the templates on the destination connector
16. Save the channel
17. Deploy the channels
18. Copy a PDF file to the /Mirth Training/inbox folder. You can copy a file such as /Mirth Training/References/ Mirth Connect Programming Reference.pdf, or one of the PDF files created in Lab 15

19. Once the Encode PDF to OBX channel has processed the PDF file and generated an HL7 file in /Mirth Training/outbox, copy that HL7 file back to /Mirth Training/inbox to be processed by the Decode PDF in OBX channel

Results

For both channels, the Dashboard statistics should show 1 Received and 1 Sent. In the /Mirth Training/outbox folder, you should now see the files generated by the two channels: an HL7 text file with a unique ID for its name and a PDF file, also with a unique ID for its name. If you view the HL7 file, you can see that the OBX.3.1 field has been populated with a unique ID, OBX.5.2 and OBX.5.3 have been populated with the string "PDF", and OBX.5.5 now contains the Base64 data for the PDF file read by the channel.

If you view the PDF file that was generated by the Decode PDF in OBX channel, you should be able to view the original document as a PDF.

Lab 28 Add Destination to PDF Writer Channel to Encode PDF to Z Segment

Objectives

In this lab, you will clone and modify the existing PDF Writer channel created in Lab 15 to add a File Writer destination connector. This new connector will read the PDF file created in the existing PDF Writer destination, encode it into Base64, and insert it into the inbound message as a new Z segment. It will then write out the message to a file. This demonstrates how you can use different destinations in the same channel to both create a PDF file and encode its contents into an HL7 message.

Estimated Time to Complete

15 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer

Step-By-Step

1. Clone the existing channel PDF Writer (created in Lab 15), and rename it to *PDF Writer Encoder*
   a. In Channels view, select the PDF Writer channel
   b. Click Clone Channel in the Channel Tasks menu
   c. An input window will pop up prompting you for a new name for the cloned channel. Type `PDF Writer Encoder` and click OK
2. Create and configure a new File Writer destination connector that will create an HL7 file containing the Base64-encoded data for the PDF file created in the Generate PDF destination
   a. Click New Destination in the Channel Tasks menu
   b. Rename the new destination to `HL7 File with Encoded PDF`
   c. Set Directory to `/Mirth Training/outbox`
   d. Set the File Name field to the patient ID extracted in the Generate PDF destination (and stored in variable patientIdentification_patientId_id), followed by a .hl7 extension: `${patientIdentification_patientId_id}.hl7`
   e. Set the Template to the Encoded Data destination mapping
3. Add a JavaScript transformer step to the new destination to read the PDF file created in the Generate PDF destination, create a new Z segment in the message and store the Base64-encoded PDF data in the first field of the Z segment
   a. Click Edit Transformer in the Channel Tasks menu
   b. Click Add New Step in the Transformer Tasks menu
   c. Double-click the cell in the Name column for the new entry to make it editable. Type `Encode PDF to ZPR.1` and press the Enter key
   d. Double-click the cell in the Type column, and select JavaScript from the drop-down list

e. The JavaScript code will first need to read the PDF file created in the Generate PDF destination. On the Reference tab, select Utility Functions in the Filter drop-down list and drag Read File As Bytes into the JavaScript editor
f. Next, you will need to edit the call to FileUtil.readBytes() to specify the name of the PDF file to read. Replace the default `'filename'` string with the following:

```
'/Mirth Training/outbox/' + $('patientIdentification_patientId_id') + '.pdf'
```

The placeholder for the patient ID can be dragged over from the Available Variables list in the Reference tab
g. Add the code to create a new segment named ZPR. On the Reference tab, select Message Functions in the Filter drop-down list and drag Create Segment (in message) into the JavaScript editor
h. Replace the default `'segmentName'` parameter with `'ZPR'`
i. Finally, add the code to encode the contents of the file into Base64 and insert into ZPR.1.1
    i. Add the following line of code to set the ZPR.1.1 field:

```
msg['ZPR']['ZPR.1']['ZPR.1.1'] =
```

    ii. On the Reference tab, select Utility Functions in the Filter drop-down list and drag BASE-64 Encode Data into the JavaScript editor after the equals sign
    iii. Replace the default data parameter with contents
j. The final code for the transformer step should look like this:

```
var contents = FileUtil.readBytes('/Mirth Training/outbox/' +
$('patientIdentification_patientId_id') + '.pdf');

createSegment('ZPR', msg);
msg['ZPR']['ZPR.1']['ZPR.1.1'] = FileUtil.encode(contents);
```

k. Click Back to Channel in the Mirth Views menu

4. Save the channel
5. Deploy the channel
6. Use the Dashboard's Send Message feature to send any ADT message to the channel

Results

The Dashboard's statistics for the channel should show 1 Received and 2 Sent. In the /Mirth Training/outbox folder, you should see two new files: a PDF file and an HL7 text file. The name for each file should be the patient's ID, as specified in PID.2.1 of the message. If you view the HL7 file, you should see the original HL7 message with an additional ZPR segment at the end. In the ZPR segment's only field is the Base64 data for the PDF file.

Lab 29 Create Channel that Stores Base64 PDF Data as an Attachment

Objectives

In the lab, you will create a channel that reads an HL7 file containing PDF data stored as Base64 and uses a preprocessor script to remove the Base64 data and store it as a message attachment. A File Writer destination connector will then write the message back out to a file, reinserting the Base64 data back into the message. This demonstrates how message attachments can be used to temporarily remove large amounts of Base64 data from the message to make message transformation much more efficient.

Estimated Time to Complete

15 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer

Step-By-Step

1. Create a new channel named *Attach Base64 PDF*
2. Write a preprocessor script to remove the Base64 PDF data from the ZPR segment, store it as a message attachment, and replace data with the attachment ID
   a. Replace the default preprocessor code with the following code (comments optional):

```
// Look for ZPR segment.
var pdfDataStart = message.indexOf("ZPR|") + 4;

// If found...
if (pdfDataStart > 4)
{
      // Add the attachment as a PDF
      var attachment =
            addAttachment(message.substring(pdfDataStart),
                  "application/pdf");

      // Replace the Base64 with the attachment ID
      message = message.substring(0, pdfDataStart) +
                  attachment.getAttachmentId() + "\r";
}

return message;
```

   Note that you can get the code template for the `addAttachment` function from the Utility Functions in the Reference list
3. Configure the source connector as a File Reader
   a. On the Source tab, select File Reader in the Connector Type drop-down list
   b. Set Directory to `/Mirth Training/inbox`

c. Set Filename Filter Pattern to `*.hl7`
d. Set Delete File After Read to Yes
e. Set Check File Age to Yes
f. Set File Age (ms) to `3000`

4. Configure the destination connector as a File Writer that will write the original file contents with the message attachment included
   a. Rename the default destination connector to Write HL7 with Attachments to File
   b. Set Directory to /Mirth Training/outbox
   c. Set the File Name field to the Message ID destination mapping, followed by a .hl7 extension: `${message.id}.hl7`
   d. Set Append to file to No
   e. For the Template field, drag over the Message with Attachment Data destination mapping
5. Save the channel
6. Deploy the channel
7. Copy the HL7 text file created by the channel in the previous lab (Lab 28) from /Mirth Training/outbox to /Mirth Training/inbox

Results

The Dashboard statistics for the channel should show 1 Received and 1 Sent. Switch to Channel Messages view to view the message processed. In the Raw Message for the source connector, notice that the Base64 data in the ZPR segment has been replaced with an attachment ID. Also notice that there is now an Attachments tab available (the rightmost tab). Click on this tab, and you should see a listing for an attachment with the same ID as was inserted in the ZPR segment in place of the Base64 data.

Finally, in the /Mirth Training/outbox folder, you should see a new HL7 text file that contains the HL7 message sent to the channel, with the Base64 data reinserted into the ZPR segment.

Lab 30 Create Channel that Transforms CDA and CCD XML Documents into HTML

Objectives

In this lab, you will create a channel that uses an XSLT Step transformer step to convert inbound CDA and CCD XML document files into HTML files using a given XSLT template.

Estimated Time to Complete

5-10 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer
- A web browser, such as Internet Explorer or Firefox
- A text file editor, such as Notepad++

Step-By-Step

1. Create a new channel named *CDA and CCD to HTML*
2. On the Summary tab, change the Incoming Data type to XML
3. Configure the source connector as a File Reader
   a. Set the Directory field to `/Mirth Training/inbox`
   b. Set the Filename Filter Pattern to `*.xml`
   c. Set the Move-to Directory to `/Mirth Training/processed`
   d. Set the Move-to File Name to a system timestamp, followed by an underscore, followed by the original inbound file name
      i. From the box to the right of the Move-to fields, drag over the SYSTIME placeholder
      ii. After the SYSTIME placeholder, type an underscore
      iii. Drag over the ORIGINALNAME placeholder

      The Move-to File Name field should now contain `${SYSTIME}_${ORIGINALNAME}`
4. Add a transformer step to build the output file name from the inbound file name
   a. Click Edit Transformer in the Channel Tasks menu
   b. Click Add New Step in the Transformer Tasks menu
   c. In the Variable field, type `outputFilename`
   d. In the Mapping field, type the following code, which gets the file name for the inbound message file, and replaces the .xml extension with a .html extension:

```
channelMap.get("originalFilename").replaceAll("xml", "html");
```

5. Add an XSLT Step transformer step to convert the inbound CDA/CCD XML document into HTML
   a. Click Add New Step in the Transformer Tasks menu
   b. For the new entry added to the transformer steps list, double-click the cell in the Type column and select XSLT Step from the drop-down list
   c. Double-click the cell in the Name column to make it editable, and type `CDA and CCD to HTML`

d. Set the Source XML String field to `msg`
e. Set the Result field to `xsltTransformedMsg`
f. Open the file "CDA to HTML.xslt" in the /Mirth Training/Messages/XML/Templates folder in a text editor, such as Notepad++
g. Copy the contents of the "CDA to HTML.xslt" file to the XSLT Template field

6. Configure the destination connector as a file writer that will write out the results of the XSLT transformation
   a. Rename the default destination connector to `Write HTML File`
   b. Set the Directory field to `/Mirth Training/outbox`
   c. For the File Name field, drag over the outputFilename destination mapping
   d. Change the Append to file setting to No
   e. For the Template field, drag over the xsltTransformedMsg destination mapping
7. Save the channel
8. Deploy the channel
9. Copy the following two files from the /Mirth Training/Messages/XML folder to the /Mirth Training/inbox folder:
   - SampleCCDDocument.xml
   - SampleCDADocument.xml

Results

Within a few seconds of copying the above two files to the inbox folder, your channel should process them. In the Dashboard's statistics for the channel, you should see 2 received and 2 sent. In the /Mirth Training/processed folder, you should see the two XML files, each prefixed with a numerical timestamp.

In the /Mirth Training/outbox folder, you should see the two HTML files generated from the two inbound XML files. Open each file in a web browser to view. You should see a nicely formatted web page containing the patient data from the inbound XML document.

Lab 31 Configure SMTP Host Settings

Objectives

In this lab, you will configure the SMTP host settings for Mirth Connect.

Estimated Time to Complete

2-3 minutes

Tools Used

- Mirth Connect Administrator

Step-By-Step

1. Click Settings in the Mirth Connect menu to view the Settings page
2. In the Email section of the Settings page, enter the SMTP Host settings from the Student Information sheet provided to you at the beginning of the course.
3. Click Save Settings in the Settings Tasks menu

Results

The SMTP settings should now match the information provided to you on your Student Information sheet.

Lab 32 Create an Alert

Objectives

In this lab, you will create an alert that will send an email to the address (or addresses) that you specify. To trigger the alert, you will also create a channel with a File Writer that attempts to write to a non-existent directory.

Estimated Time to Complete

10 minutes

Tools Used

- Mirth Connect Administrator
- Any web browser, such as Internet Explorer or Firefox, or whichever application you use to access your email

Step-By-Step

1. Create a new channel named *Alert Test*
   a. Source connector is a Channel Reader
   b. Destination connector is a File Writer
   c. Set File Writer's Directory value to some non-existent directory (for example, Z:\Mirth Training\outbox)
   d. Set other required File Writer fields to valid values
   e. Save and deploy the channel
2. Click Alerts in the Mirth Connect menu
3. Click New Alert in Alert Tasks menu
4. Configure the new alert
   a. Change the name of the alert to `File Exception Alert`
   b. In the Apply to Channels list, find the Alert Test channel, and check the box in the Applied column
   c. Click the Add button to the right of the Emails to Receive Alerts list
   d. Double-click the new entry in the Emails to Receive Alerts list to make it editable, and enter the email address at which you wish to receive the alert email. Press the Enter key to accept.
   e. Repeat steps c and d to add any additional email or SMS addresses at which you want to receive the alert email
   f. In the Error Codes list, select 403: File connector, and drag-and-drop it into the Error Matching Field box
   g. In the Error Template field, type the following:

```
An error occurred!
Channel: ${channelName}
Date: ${date}
Systime: ${SYSTIME}
Error: ${error}
```

5. Save the alert by clicking Save Alerts in the Alert Tasks menu
6. Enable the alert by selecting the alert in the alerts list and clicking Enable Alert in the Alert Tasks menu
7. Send a message to the Alert Test channel using the Dashboard's Send Message feature to trigger the alert

Results

Within a short while, you should receive an email (and/or SMS message) at the address (or addresses) that you specified. The subject of the email is Mirth Alert. The body of the email should look similar to the following:

```
An error occurred!
Channel: Alert Test
Date: Jun 25, 2009 11:06:56 AM
Systime: 1253902016946
Error: ERROR-403: File connector error
ERROR MESSAGE:      Error writing file
java.io.FileNotFoundException: Z:\Mirth Training\Test.txt (The system cannot find the path
specified)
       at java.io.FileOutputStream.openAppend(Native Method)
       at java.io.FileOutputStream.<init>(Unknown Source)
       at
com.webreach.mirth.connectors.file.filesystems.FileConnection.writeFile(FileConnection.java:1
65)
       at
com.webreach.mirth.connectors.file.FileMessageDispatcher.doDispatch(FileMessageDispatcher.jav
a:106)
       at
com.webreach.mirth.connectors.file.FileMessageDispatcher.doSend(FileMessageDispatcher.java:14
4)
       at
org.mule.providers.AbstractMessageDispatcher.send(AbstractMessageDispatcher.java:167)
       at org.mule.impl.MuleSession.sendEvent(MuleSession.java:191)
       at org.mule.impl.MuleSession.sendEvent(MuleSession.java:130)
       at
org.mule.routing.outbound.AbstractOutboundRouter.send(AbstractOutboundRouter.java:85)
       at
org.mule.routing.outbound.FilteringMulticastingRouter.route(FilteringMulticastingRouter.java:
45)
       at
org.mule.routing.outbound.OutboundMessageRouter$1.doInTransaction(OutboundMessageRouter.java:
78)
       at org.mule.transaction.TransactionTemplate.execute(TransactionTemplate.java:48)
       at
org.mule.routing.outbound.OutboundMessageRouter.route(OutboundMessageRouter.java:82)
       at org.mule.impl.model.DefaultMuleProxy.onCall(DefaultMuleProxy.java:247)
       at org.mule.impl.model.seda.SedaComponent.doSend(SedaComponent.java:209)
       at org.mule.impl.model.AbstractComponent.sendEvent(AbstractComponent.java:277)
       at org.mule.impl.MuleSession.sendEvent(MuleSession.java:201)
       at org.mule.routing.inbound.InboundMessageRouter.send(InboundMessageRouter.java:180)
       at org.mule.routing.inbound.InboundMessageRouter.route(InboundMessageRouter.java:147)
       at
org.mule.providers.AbstractMessageReceiver$DefaultInternalMessageListener.onMessage(AbstractM
essageReceiver.java:493)
       at
org.mule.providers.AbstractMessageReceiver.routeMessage(AbstractMessageReceiver.java:272)
       at
org.mule.providers.AbstractMessageReceiver.routeMessage(AbstractMessageReceiver.java:231)
       at
com.webreach.mirth.connectors.vm.VMMessageReceiver.getMessages(VMMessageReceiver.java:178)
```

```
        at
org.mule.providers.TransactedPollingMessageReceiver.poll(TransactedPollingMessageReceiver.jav
a:108)
        at org.mule.providers.PollingMessageReceiver.run(PollingMessageReceiver.java:90)
        at org.mule.impl.work.WorkerContext.run(WorkerContext.java:290)
        at
edu.emory.mathcs.backport.java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolEx
ecutor.java:650)
        at
edu.emory.mathcs.backport.java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecut
or.java:675)
        at java.lang.Thread.run(Unknown Source)
```

Lab 33 Configure Message Pruner

Objectives

In this lab, you will configure the Message Pruner to run on an hourly basis. To demonstrate the Message Pruner, you will import a sample channel that has its Prune Messages setting set to prune after 7 days. You will then also import messages for that channel that are older than 7 days.

Estimated Time to Complete

7-8 minutes

Tools Used

- Mirth Connect Administrator

Step-By-Step

1. Import a sample channel with old messages
   a. While in Channels view, click Import Channel in the Channel Tasks menu
   b. Browse to and open "Sample Channel for Message Pruner.xml" in the /Mirth Training/Channels folder
   c. Save the channel
   d. Deploy the channel
   e. Go to the Channel Messages view for the channel by selecting the channel and clicking View Messages in the Status Tasks menu
   f. Click Import Messages in the Message Tasks menu
   g. Browse to and open "Sample Channel for Message Pruner Messages.xml" in the /Mirth Training/Channels/Miscellaneous folder
   h. If you wish to view the messages, change the filter to start on August 1, 2009. You should see a total of 1000 messages
2. Click Plugins in the Mirth Connect menu
3. Click the Message Pruner tab
4. In the When to prune area, select Hourly
5. Set Enable Batch Pruning to Yes
6. Set Pruning Block Size to 1000
7. Click Save in the Pruner Tasks menu to begin pruning

Results

After about a minute or two, you should see a new entry in the Activity Log. The values in this row should be as follows:

- Channel – Batch Pruning: messages older than 7 days
- Time Pruned – The date/time of the pruning
- Messages Pruned – 1000

Lab 34 Create a Server Configuration Backup File

Objectives

In this lab, you will create a Server Configuration backup file. This backup file will be used in Lab 35.

Estimated Time to Complete

2-3 minutes

Tools Used

- Mirth Connect Administrator
- Windows Explorer

Step-By-Step

1. Click Settings in the Mirth Connect menu
2. In the Server Configuration section of the Settings page, click the Backup button
3. In the Save dialog, navigate to the /Mirth Training/Backup directory, and save the file with its default backup name
4. You will see a dialog appear indicating that the server configuration file was created. Click the OK button

Results

The Server Configuration section of the Settings page should now show the date and time of the backup that you just performed. In the /Mirth Training/Backup folder, you should now see the XML backup file with the name "YYYY-MM-DD Mirth Backup.xml".

Lab 35 Change the Mirth Connect Database to PostgreSQL

Objectives

In this lab, you will configure the Mirth Connect Server to use PostgreSQL as its internal database. You will also restore the server configuration backup file created in the previous lab on the new database schema. Finally, you will change the server configuration back to use the original Derby database.

Estimated Time to Complete

10 minutes

Tools Used

- Mirth Connect Administrator
- Mirth Connect Server Manager

Step-By-Step

1. Exit the Mirth Connect Administrator if it is open
2. Change the Mirth Connect Server configuration to use the PostgreSQL database
   a. Open the Mirth Connect Server Manager
   b. Click the Database tab
   c. In the Type drop-down list, select postgres
   d. Change the URL value to the following:
      `jdbc:postgresql://localhost:5432/mirthdb`
   e. In the Username field, type `postgres`
   f. In the Password field, type `abc12345`
   g. Click the Service tab
   h. Click the Restart button to restart the Mirth Connect Server
   i. A dialog will appear asking if you would like to save your changes before continuing. Click Yes
3. Once the service has started, click the Administrator button to open the Mirth Connect Administrator
4. Once logged into the Administrator, notice that there are no channels, users, alerts, etc.
5. Restore the server configuration (created in Lab 34) to the new database
   a. Click Settings in the Mirth Connect menu
   b. In the Server Configuration section, click the Restore button
   c. Browse to and open the server configuration backup file created in Lab 34
   d. You will see a message confirming that you want to import the configuration and overwrite all current channels, etc. Click Yes
   e. When the import is complete, a dialog will appear informing you that the configuration was successfully restored. Click OK
6. Explore the various views. You should now see all of the channels, users, etc. that you had created from the original Derby database. Notice, however, that no messages were imported.
7. Exit the Mirth Connect Administrator
8. Change the Mirth Connect Server configuration back to using the Derby database
   a. Open the Mirth Connect Server Manager
   b. Click the Database tab

c. In the Type drop-down list, select derby
d. Click the Service tab
e. Click the Restart button to restart the Mirth Connect Server
f. A dialog will appear asking if you would like to save your changes before continuing. Click Yes

9. Once the service has started, click the Administrator button to open the Mirth Connect Administrator

Results

Once you have completed all of the above steps, you should be back to where you started. In other words, you should have all of your channels, etc., *including* messages.

Lab 36 Connect to Mirth Connect Database

Objectives

In this lab, you will connect to Mirth Connect's internal Apache Derby database using the SQuirreL SQL Client. This will allow you to see how Mirth Connect components, such as channels, messages, alerts, etc., are stored internally.

Estimated Time to Complete

5 minutes

Tools Used

- Mirth Connect Server Manager
- SQuirreL SQL Client

Step-By-Step

1. Exit Mirth Connect Administrator if it is open
2. Open the Mirth Connect Server Manager
3. On the Service tab, click the Stop button to stop the Mirth Connect Server
4. Open SQuirreL SQL Client
5. In the Aliases window, select Mirth Connect Derby DB and click the connect button (first button from left)
6. In the Connect dialog that appears, click the Connect button
7. In the session window that opens, click the Objects tab
8. In the Object tree, expand APP, and then expand TABLE. This will display all of Mirth Connect's internal tables
9. To view the content of any of the tables, select the table name that you wish to view, then click the Content tab in the pane on the right-hand side
10. Once you are finished exploring the Mirth Connect database, close the Mirth Connect Derby DB window to end the session
11. Restart the Mirth Connect Server
12. Restart the Mirth Connect Server Administrator

Lab 37 Launch Shell Client, Execute Commands

Objectives

In this lab, you will use Mirth Connect's Shell Client to issue commands to the Mirth Connect Server, such as to start and stop deployed channels.

Estimated Time to Complete

5 minutes

Tools Used

- Windows Command Prompt
- Mirth Connect Administrator

Step-By-Step

1. Open the Windows Command Prompt window
2. Change to the Mirth Connect installation directory (usually \Program Files\Mirth for 32-bit computers or \Program Files (x86) for 64-bit computers)
3. Type `shell` to start the shell client
4. At the shell client's command prompt ($), type help to list available commands
5. Type `status` to show the status of all deployed channels
6. Type `stop` to stop all deployed channels
7. In the Mirth Connect Administrator, notice that all channels are now stopped
8. Type `start` to restart the deployed channels
9. In the Mirth Connect Administrator, notice that all channels are now started
10. Issue whatever other commands you would like
11. When finished, typed `quit` to exit the shell client

Lab 38 Create a Script File and Run the Shell Client in Script Mode

Objectives

In this lab, you will create a batch file and edit the Shell.bat file to execute the Shell Client in script mode.

Estimated Time to Complete

5-10 minutes

Tools Used

- Any text editor, such as Notepad++
- Windows Command Prompt
- Windows Explorer

Step-By-Step

1. In a text editor such as Notepad++, create a new file
2. In the file, type the following:

```
stop
exportcfg "\Mirth Training\Backup\ServerConfig.xml"
dump stats "\Mirth Training\Backup\Stats.txt"
resetstats
start
```

3. Save the file as Script.txt in the Mirth Connect installation directory (e.g., /Program Files (x86)/Mirth)
4. Make a copy of the file Shell.bat in the Mirth Connect installation directory, naming it ShellBatch.bat
5. Open ShellBatch.bat in a text editor
6. At the end of the line that begins with "java", type the following, including a leading space character:
   `-s Script.txt`
7. Save and close ShellBatch.bat
8. In the Windows Command Prompt window, type `ShellBatch`

Results

In the Command Prompt window, you should see the following output from the execution of the batch:

```
Connected to Mirth server @ https://127.0.0.1:8443 (1.8.1.4211)
Executing statement: stop
Channel Alert Test Stopped
Channel Sample Channel for Message Pruner Stopped
Channel Custom Response Stopped
Channel File Test Stopped
Executing statement: exportcfg "\Mirth Training\Backup\ServerConfig.xml"
Exporting Configuration
Configuration Export Complete.
Executing statement: dump stats "\Mirth Training\Backup\Stats.txt"
```

```
Stats written to \Mirth Training\Backup\Stats.txt
Executing statement: resetstats
Executing statement: start
Channel Alert Test Started
Channel Sample Channel for Message Pruner Started
Channel Custom Response Started
Channel File Test Started
Disconnected from server.
```

In the Mirth Connect Administrator, you should see that all Dashboard statistics (Received, Sent, etc.) have been reset to 0. In the /Mirth Training/Backup folder, you should see two new files: ServerConfig.xml and Stats.txt. ServerConfig.xml is a server configuration backup file, the same as was created in Lab 34. Stats.txt is a comma-separated value (CSV) file that contains the statistics (Received, Sent, etc.) for all channels.