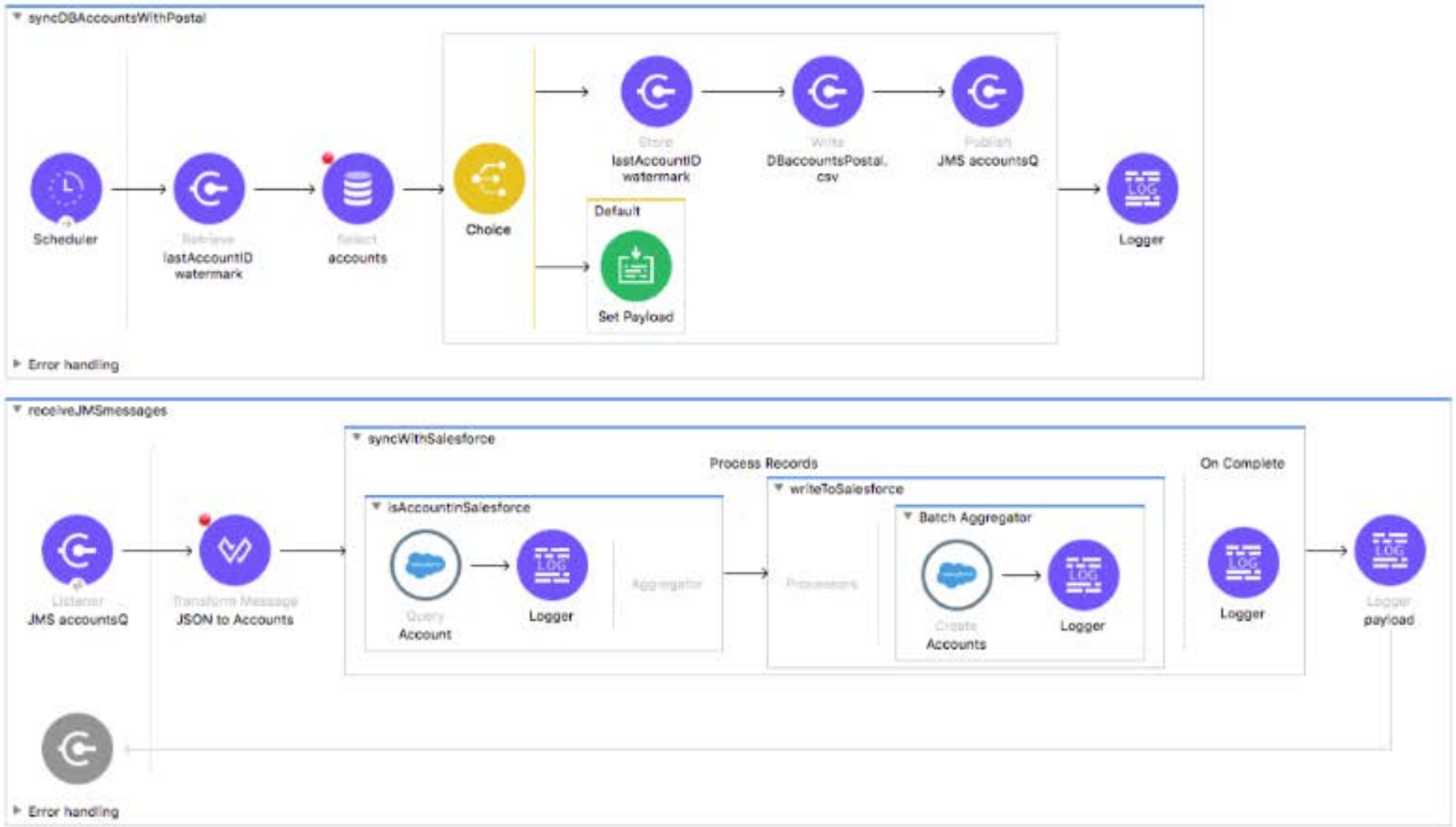

MuleSoft
PART 3: Building Applications to Synchronize Data

Goal



At the end of this part, you should be able to

- Trigger flows when files or database records are added or updated
- Schedule flows
- Persist and share data across flow executions
- Publish and consume JMS messages
- Process items in a collection sequentially
- Process records asynchronously in batches

Module 12: Triggering Flows

Goal

How have we initiated flows so far?

In this module, we will learn new ways

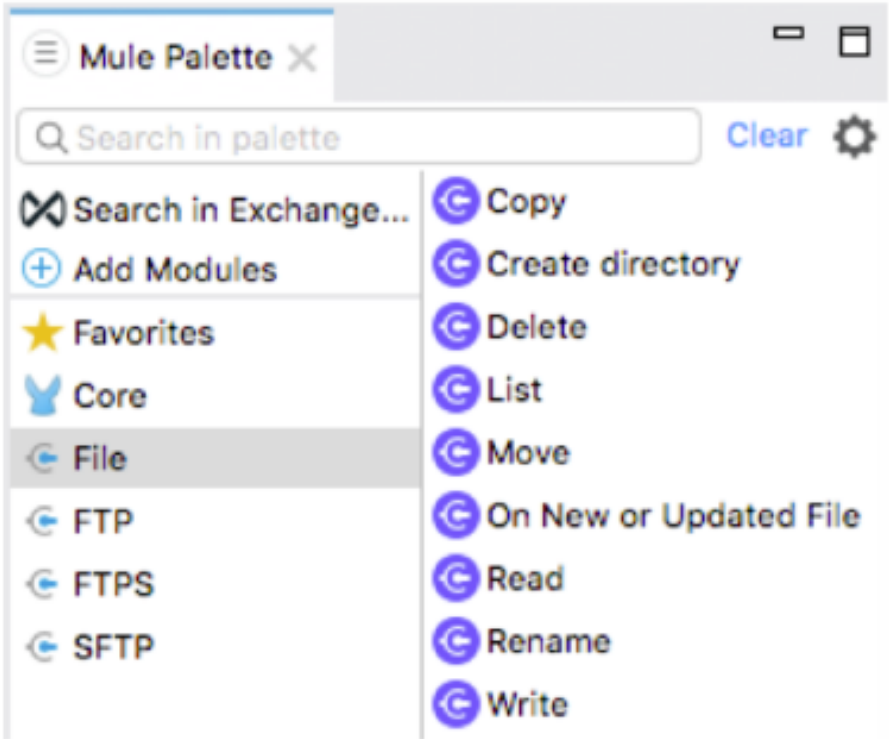
At the end of this module, you should be able to

- Read and write files
- Trigger flows when files are added, created, or updated
- Trigger flows when new records are added to a database table
- Schedule flows to run at a certain time or frequency
- Persist and share data in flows using the Object Store
- Publish and consume JMS messages

Reading and writing files

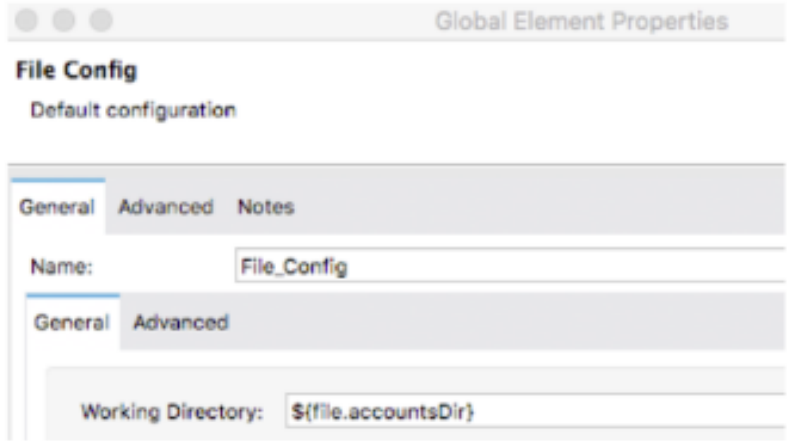
Reading and writing files

- There are 4 connectors for working with files and folders
  - File (for locally mounted file system)
  - FTP
  - FTPS
  - SFTP
- All have the same set of operations and they behave almost identically
- Support for
  - File matching functionality
  - Locking files
  - Overwriting, appending, and generating new files



Using the File connector

- Add the File module to the project
- Create a global element configuration
  - Not required but a best practice
  - Set the **working directory** that will be the root for every path used with the connector

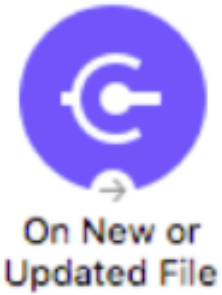


- Use one of the connector operations and specify its properties

- On **CloudHub**, the connector can only be used with the /tmp folder
- On **Customer-hosted** Mule runtimes, the account running Mule must have read and/or write permissions on the specified directories
- *Be careful not to permanently delete or overwrite files*
  - Move or rename them after processing

Trigger a flow when a new file is created or updated

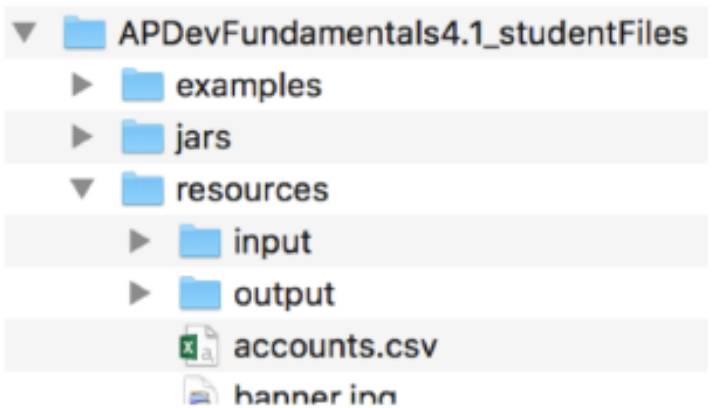
- Use the **On New or Updated File** listener
  - Polls a directory for files that have been created or updated
  - One message is generated for each file that is found



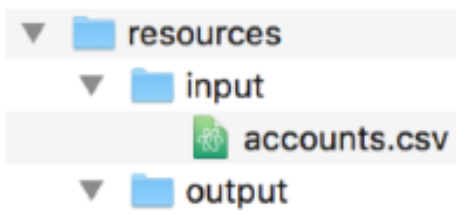
- Multiple ways to ensure a file is new
  - Delete each file after it has been processed so all files in the next poll will be new
  - Move each file to a different directory after it has been processed
  - Rename a file after it has been processed and filter the files to be processed
  - Save and compare the file creation or modification times

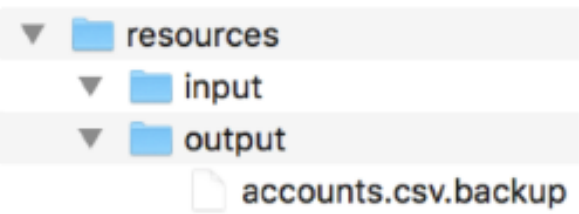
Walkthrough 12-1: Trigger a flow when a new file is added to a directory

- Add and configure a File listener to watch an input directory
- Restrict the type of file read
- Rename and move the processed files




```
111 Boulevard Hausmann,Paris,France,,Dog Park Industries,75008
400 South St,San Francisco,USA,CA,Iguana Park Industries,91156
777 North St,San Francisco,USA,CA,Cat Park Industries,91156
```

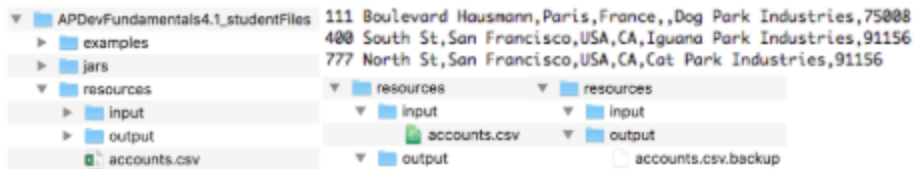




Walkthrough 12-1: Trigger a flow when a new file is added to a directory

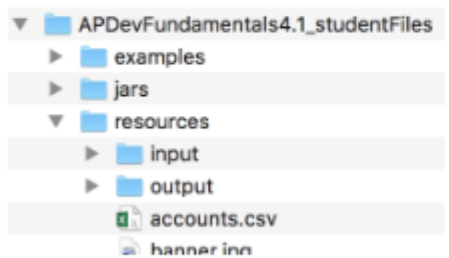
In this walkthrough, you load data from a local CSV file when a new file is added to a directory. You will:

- Add and configure a File listener to watch an input directory.
- Restrict the type of file read.
- Rename and move the processed files.



Locate files and folders

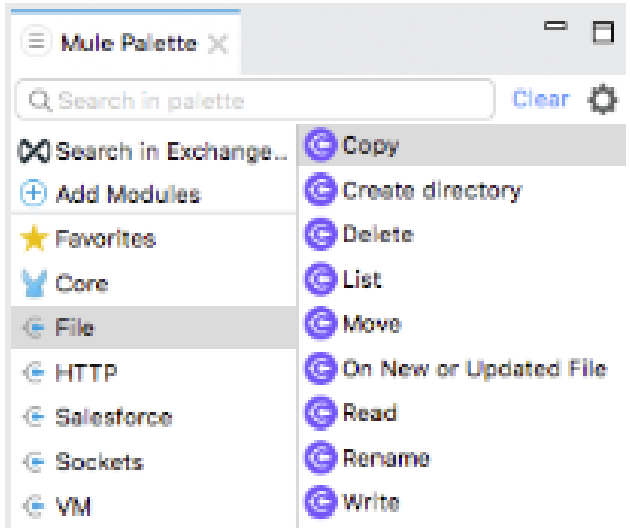
1. In your computer's file browser, return to the student files for the course.
2. Open the resources folder and locate the accounts.csv file and the input and output folders.



3. Leave this folder open.

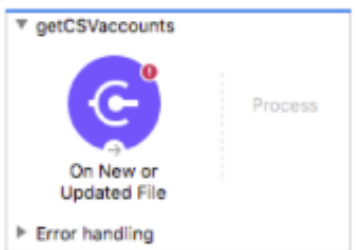
Add the File module to the project

4. Return to Anypoint Studio and open the apdev-examples project.
5. Open accounts.xml.
6. In the Mule Palette, select Add Modules.
7. Select the File connector in the right side of the Mule Palette and drag and drop it into the left side.
8. If you get a Select module version dialog box, select the latest version and click Add.



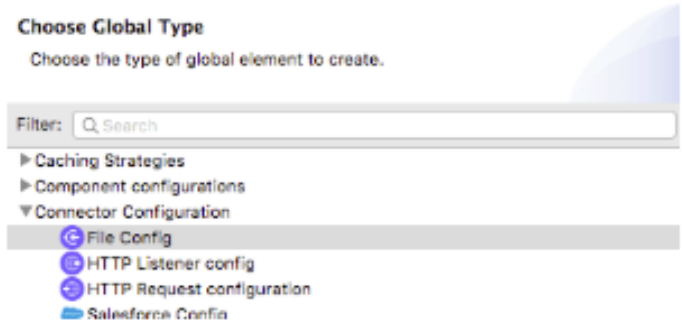
Create a flow that monitors a location for new files

9. Locate the On New or Updated File operation for the File connector in the right side of the Mule Palette and drag and drop it at the top of the canvas to create a new flow.
10. Rename the flow to getCSVaccounts.

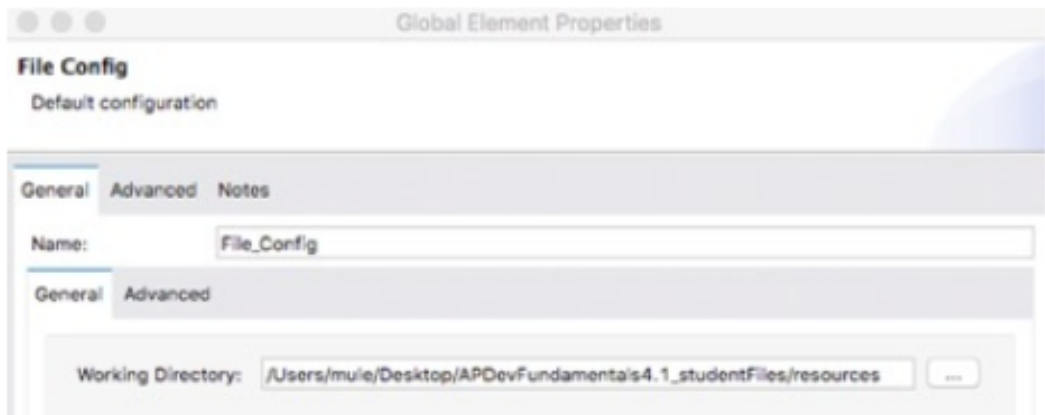


Configure the File connector

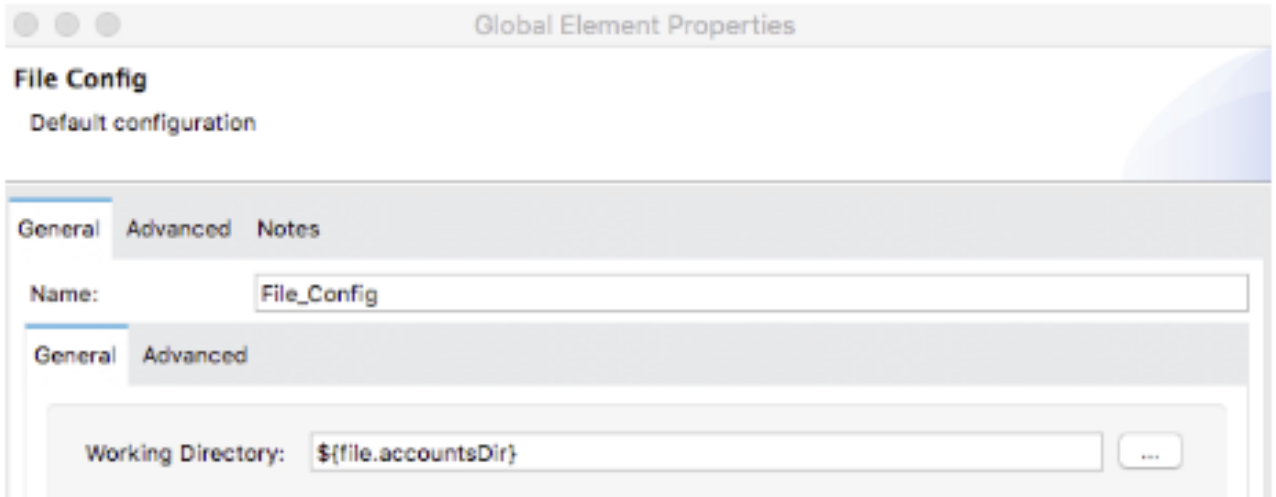
11. Open global.xml and switch to the Global Elements view.
12. Click Create.
13. In the Choose Global Type dialog box, select Connector Configuration > File Config and click OK.



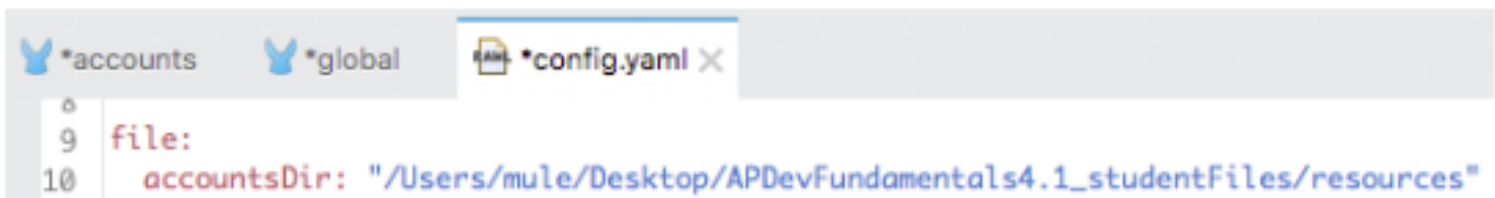
14. In the Global Element Properties dialog box, click the browse button next to working directory.
15. Browse to and select the student files folder for the course.
16. Select the resources folder and click Open.



17. Select and cut the value of the working directory that got populated and replace it with a property ${file.accountsDir}.

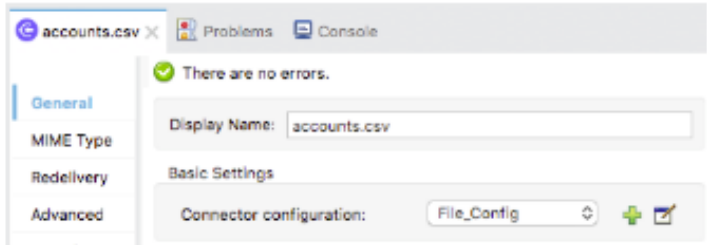


18. In the Global Element Properties dialog box, click OK.
19. Open config.yaml in src/main/resources.
20. Create a new property file.accountsDir and set it equal to the value you copied.

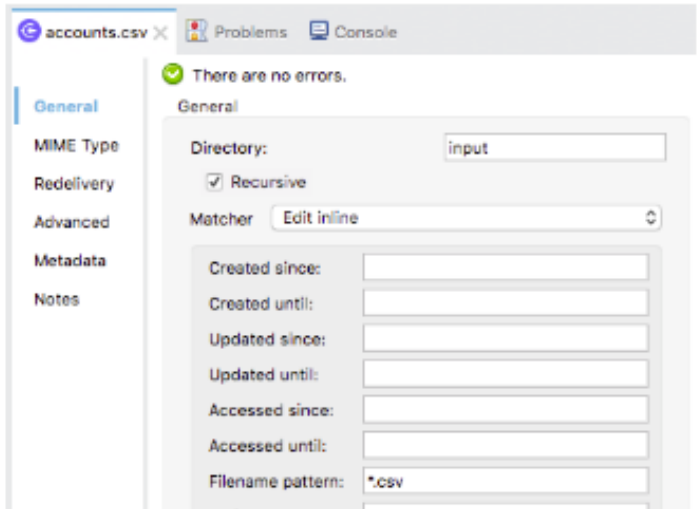


Configure the On New or Updated File operation

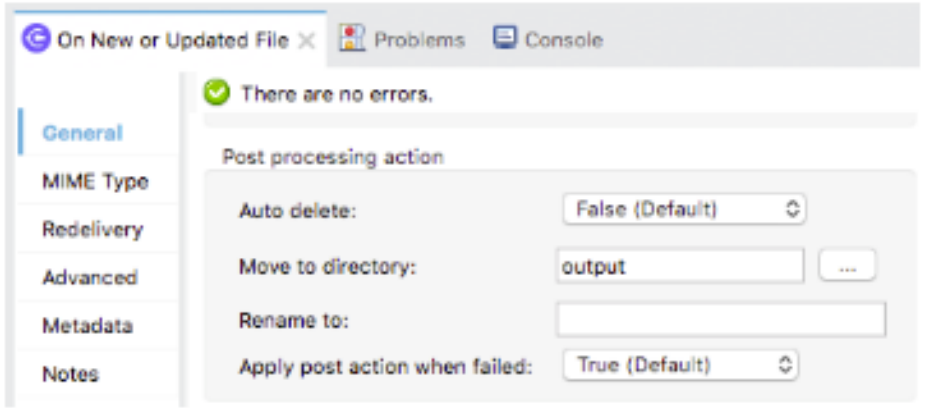
21. Return to accounts.xml.
22. In the On New or Updated File properties view, change the display name to accounts.csv.
23. Set the connector configuration to the existing File_Config.



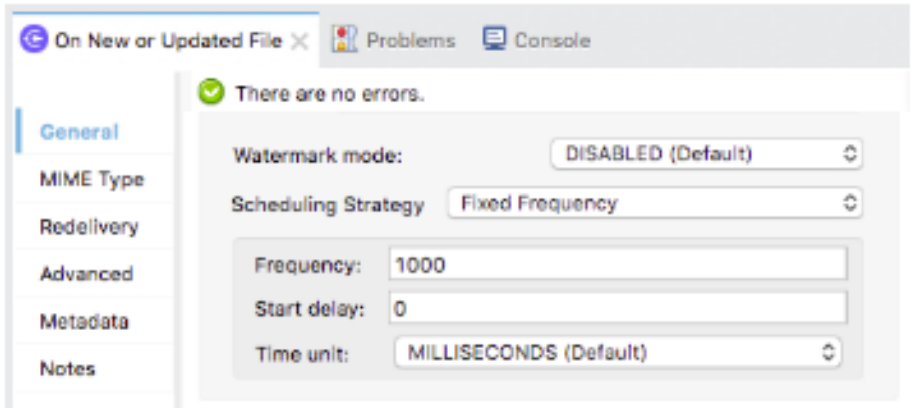
24. In the General section, set the directory to input.
25. Set the matcher to Edit inline.
26. Set filename pattern to *.csv.



27. In the post processing action section, set the move to directory to output.

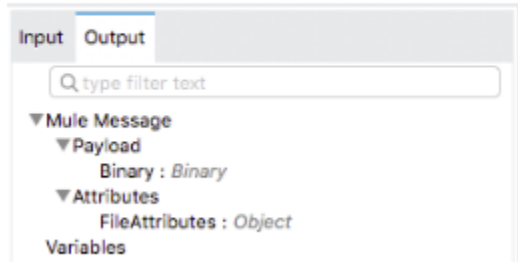


28. In the General section, review the default scheduling information; the endpoint checks for new files every 1000 milliseconds.



Review event metadata in the DataSense Explorer

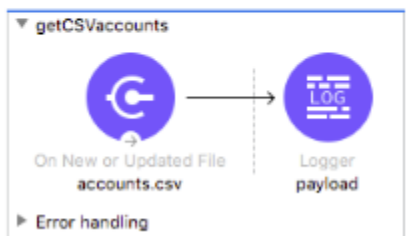
29. Select the Output tab in the DataSense Explorer; you should see no metadata about the structure of the CSV file.



Display the file contents

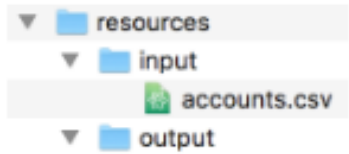
30. Add a Logger to the flow.
31. In the Logger properties view, set the display name to payload and the message to display the payload.

```
#[payload]
```



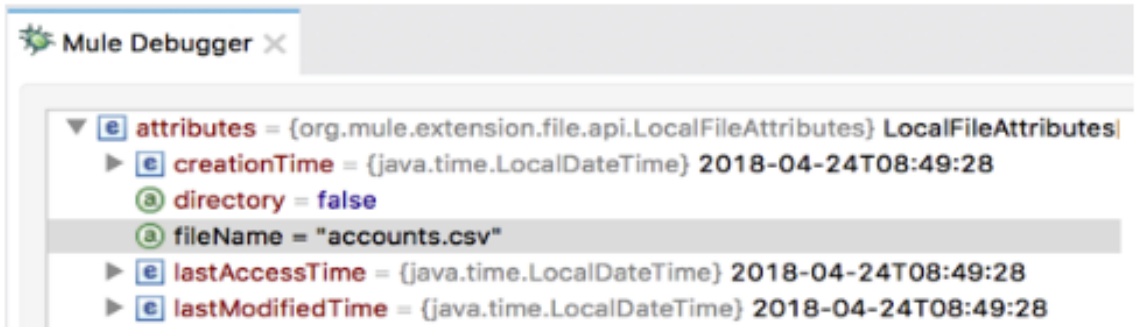
Debug the application

32. Add a breakpoint to the Logger.
33. Debug the project.
34. In your computer's file browser, return to the student files for the course.
35. Move the accounts.csv file to the input folder.

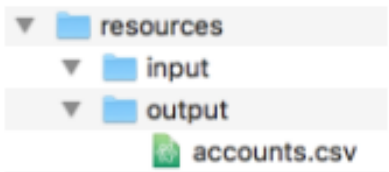


36. Return to the Mule Debugger and look at the payload.

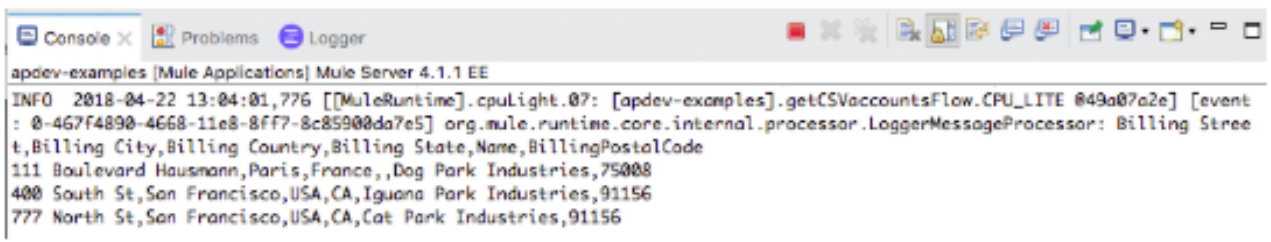
37. Expand Attributes and locate the fileName and path attributes.



38. Step to the end of the application.
39. In your computer's file browser, return to the student files for the course; you should see accounts.csv has been moved to the output folder.



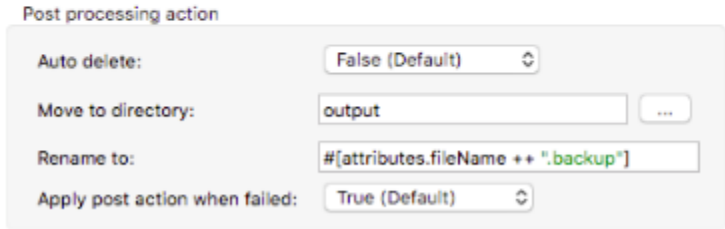
40. Return to Anypoint Studio and look at the console; you should see the file contents displayed.



Rename the file

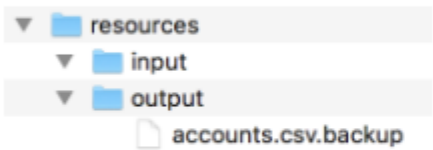
41. Return to the File properties view and in the Post processing action section, set the rename to property to an expression that appends .backup to the original filename.

```
#[attributes.fileName ++ ".backup"]
```

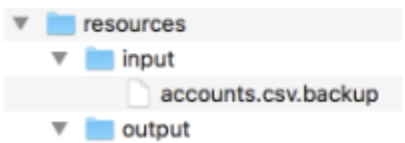


Test the application

42. Save the file to redeploy the project.
43. Remove the breakpoint from the Logger.
44. In your computer's file explorer, move the accounts.csv file from the output folder back to the input folder; you should see it appear in the output folder with its new name.



45. Move the accounts.csv.backup file from the output folder back to the input folder; it should not be processed and should stay in the input folder.



46. Return to Anypoint Studio and switch perspectives.
47. Stop the project.

Synchronizing data with watermarks

Synchronizing data from one system to another

- The general process
  - The first time, you need to sync all the data
  - After that, you only need to sync the new data

- How do you determine what is new and needs to be synced?
  - On the first sync, store the latest timestamp for any item in the data set
  - On later syncs, retrieve that timestamp and compare the timestamp of each item and see if it is later

- The timestamp is often a
  - Creation timestamp
  - Modification timestamp
  - Record ID

Introducing watermarks

- The timestamp that is stored each sync and then retrieved and compared against in the next sync is called a **watermark**

- Where did the name come from?
  - After a flood, one might record how high the water got by marking the level on a wall
  - Similarly, for data, we want to look at the last value - how "high" it was in the last sync

Types of watermarking in Mule

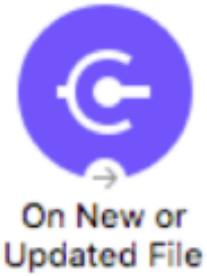
- **Automatic**
  - The saving, retrieving, and comparing is automatically handled for you
  - Available for several connector listeners
    - On New or Updated File
    - On Table Row
  - Restricted in how you can specify what items/records are retrieved

- **Manual**
  - You handle saving, retrieving, and comparing the watermark
  - More flexible in that you specify exactly what records you want retrieved

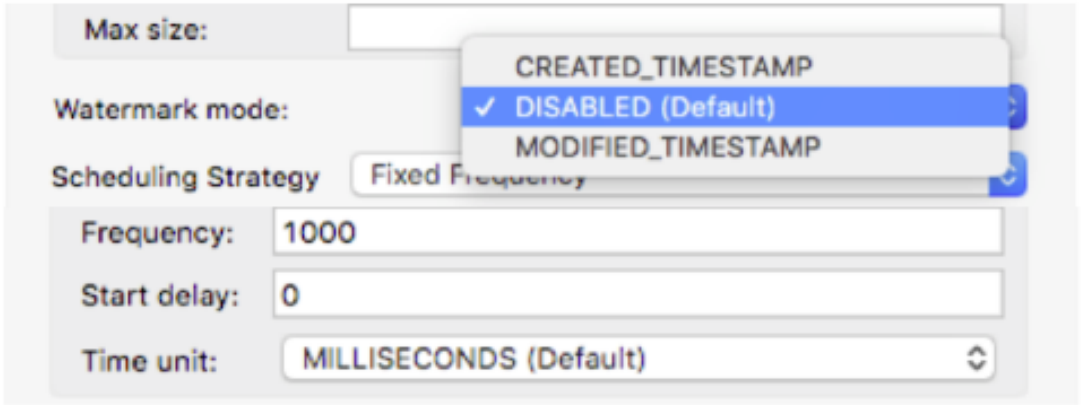
Using listeners with automatic watermarking

Using automatic watermarking with files

- There is a watermarking option for the **On New and Updated File** operation for the family of file connectors



- There are two watermarking modes
  - CREATION_TIMESTAMP
  - MODIFIED_TIMESTAMP

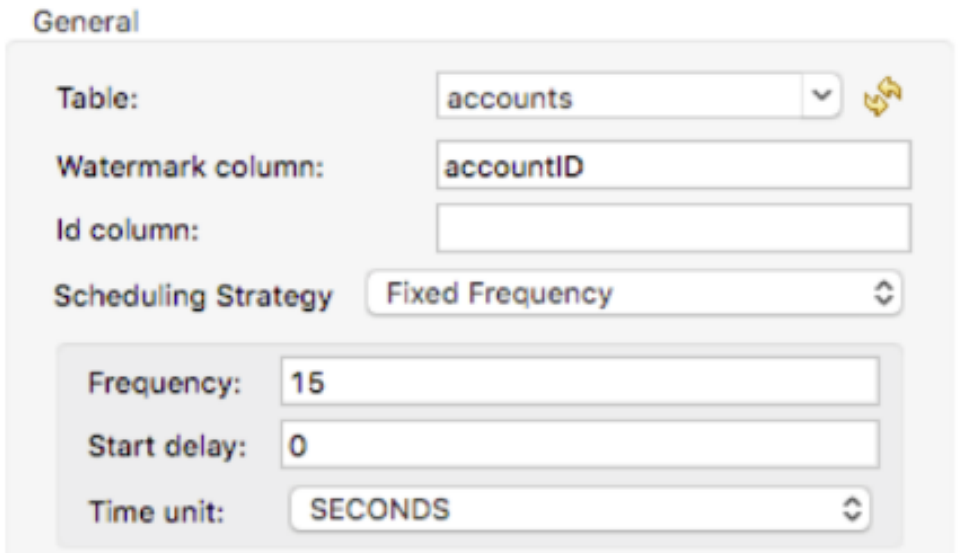


- This can be used for one of the ways introduced last section to ensure a file is new
  - Other options: Delete, move, filter
  - **Save and compare the file creation or modification times**

Triggering a flow for each row in a database table

- The Database connector has an **On Table Row** operation that is triggered for every row in a table

- The operation can handle
  - Generating the query
  - Watermarking
  - Idempotency across concurrent requests
- You can specify one, both, or neither of
  - Watermark column
  - Idempotency column

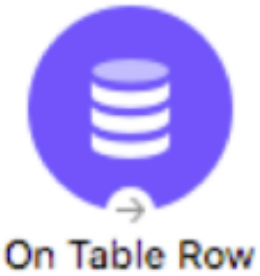


Using automatic watermarking with database tables

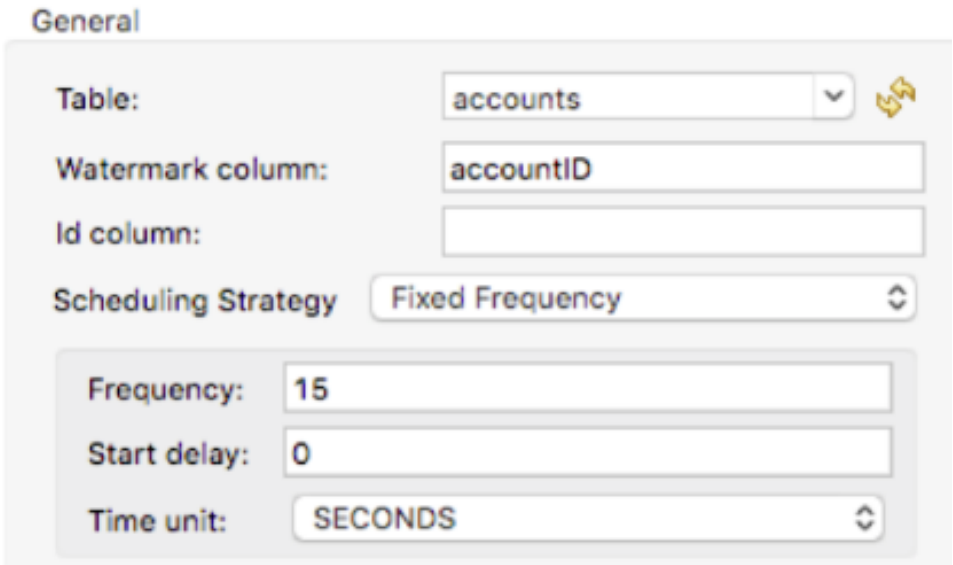
MuleSoft

- When a watermark column is specified, this query is automatically generated and used

```
SELECT * FROM table
WHERE TIMESTAMP > :watermark
```



- On each poll, the component will go through all the retrieved rows and store the maximum value obtained

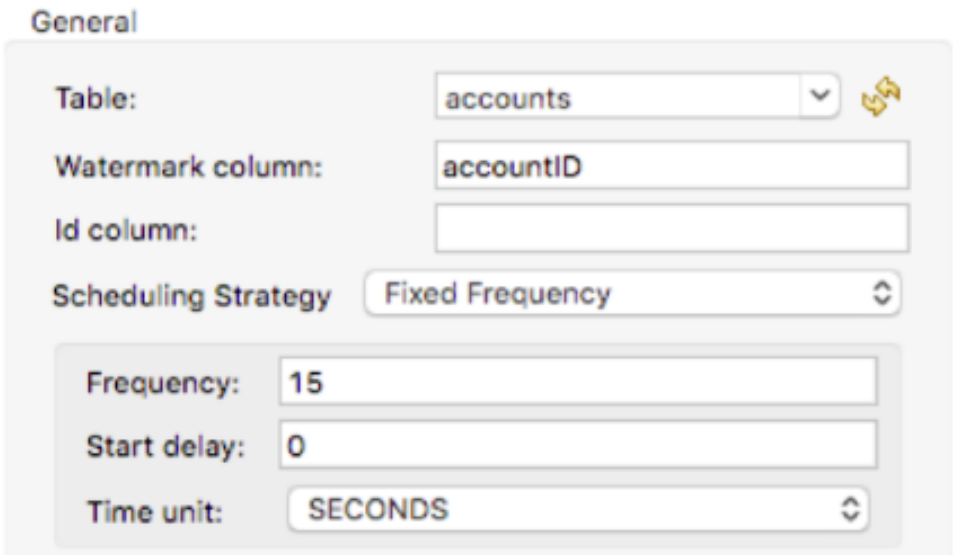


Handling idempotency across concurrent requests

MuleSoft

- **A new poll can be executed before the watermark is updated if**
  - The poll interval is small
  - The amount of rows is big
  - Processing one single row takes too much time

- **To avoid a record being processed more than once**
  - Specify an ID column
    - A unique identifier for the row
  - The listener will make sure the row is not processed again if
    - It has already been retrieved and
    - Processing of it hasn't finished yet



Walkthrough 12-2: Trigger a flow when a record is added to a database and use automatic watermarking

- Add and configure a Database listener to check a table on a set frequency for new records
- Use the listener's automatic watermarking to track the ID of the latest record retrieved and trigger the flow whenever a new record is added
- Output new records to a CSV file
- Use a form to add a new account to the table and see the CSV updated

MUA Accounts

Create More Accounts

accountID	name	street	city	state
6706	Max Mule	77 Geary Street	San Francisco	California
6705	Minnie Mule	77 Geary Street	San Francisco	California

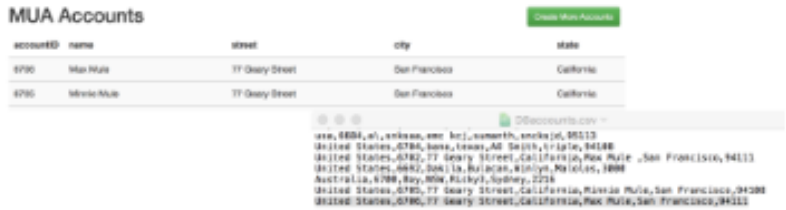
DBaccounts.csv

```
usa,6684,a\,snksaa,emc kcj,sumanth,sncksjd,95113
United States,6704,bana,texas,AO Smith,triple,94108
United States,6702,77 Geary Street,California,Max Mule ,San Francisco,94111
United States,6692,Dakila,Bulacan,Winlyn,Malolos,3000
Australia,6700,Bay,NSW,Ricky3,Sydney,2216
United States,6705,77 Geary Street,California,Minnie Mule,San Francisco,94108
United States,6706,77 Geary Street,California,Max Mule,San Francisco,94111
```

Walkthrough 12-2: Trigger a flow when a new record is added to a database and use automatic watermarking

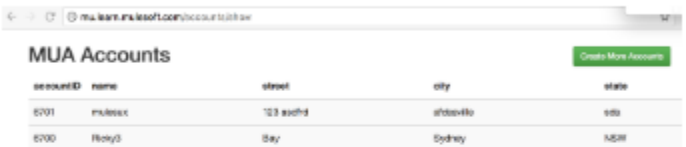
In this walkthrough, you work with the accounts table in the training database. You will:

- Add and configure a Database listener to check a table on a set frequency for new records.
- Use the listener's automatic watermarking to track the ID of the latest record retrieved and trigger the flow whenever a new record is added.
- Output new records to a CSV file.
- Use a form to add a new account to the table and see the CSV file updated.



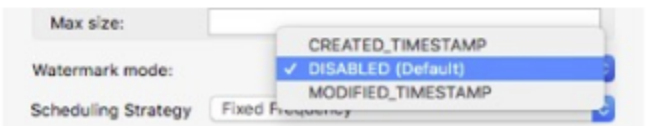
View accounts data

1. In a web browser, navigate to http://mu.mulesoft-training.com/accounts/show.
2. Review the data and the names of the columns—which match those of database table.



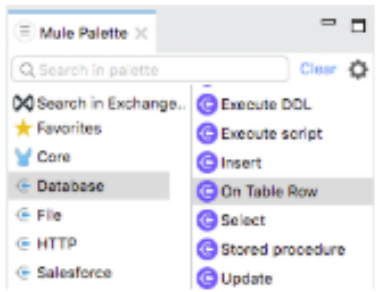
Look at the File listener settings for watermarking

3. Return to accounts.xml in Anypoint Studio.
4. Return to the On New or Updated File properties view for the listener in getCSVaccounts.
5. Locate the watermark mode setting in the General section and look at its possible values.



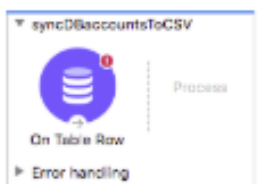
Add the Database module to the project

6. In the Mule Palette, select Add Modules.
7. Select the Database connector in the right side of the Mule Palette and drag and drop it into the left side.
8. If you get a Select module version dialog box, select the latest version and click Add.



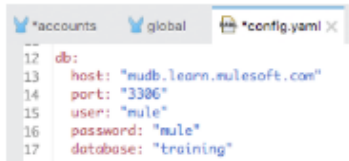
Create a flow to monitor a database

9. Locate the On Table Row operation for the Database connector in the right side of the Mule Palette and drag and drop it at the top of the canvas to create a new flow.
10. Rename the flow to syncDBaccountsToCSV.



Configure a Database connector

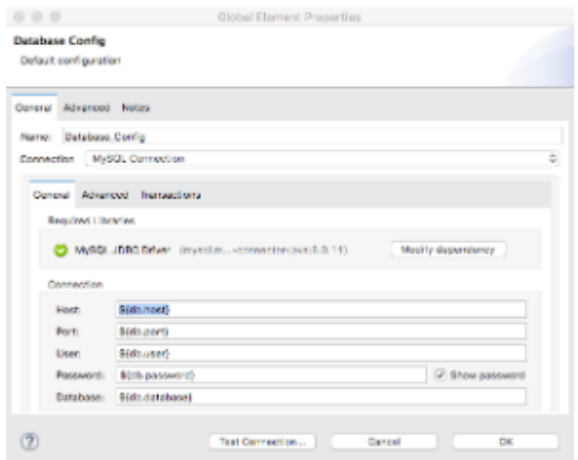
11. Return to the course.snippets.txt file.
12. Locate and copy the MySQL (or Derby) database properties in the Module 4 section.
13. Return to config.yaml and paste the properties.



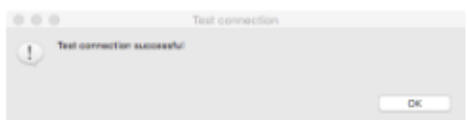
14. Save the file.
15. Return to global.xml and create a new Database Config that uses these properties.

    Note: If necessary, refer to the detailed steps in Walkthrough 4-2.

16. Add the MySQL (or Derby) JDBC driver.



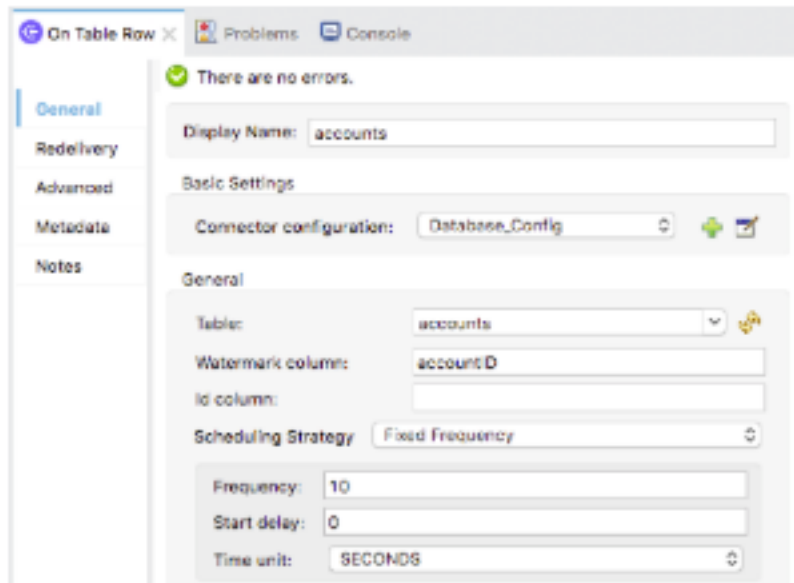
17. Test the connection and make sure it is successful.



Configure the Database listener

18. Return to accounts.csv.
19. In the On Table Row properties view, set the following values:

    - Display Name: accounts
    - Connector configuration: Database_Config
    - Table: accounts
    - Watermark column: accountID
    - ID column: accountID
    - Frequency: 10
    - Time unit: SECONDS

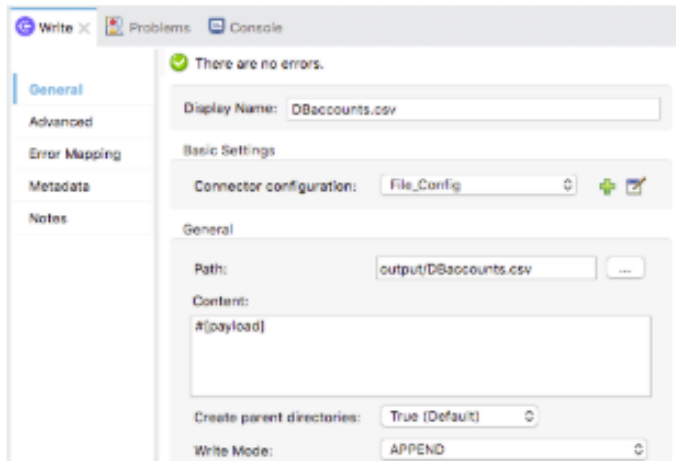


Write new records to a CSV file

20. Add a File Write operation to the flow.

21. In the Write properties view, set the following values:

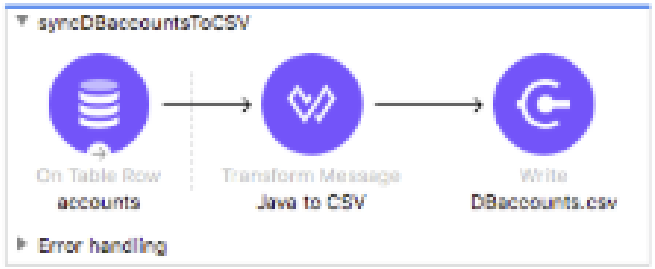
- Display Name: DBaccounts.csv
- Connector configuration: File_Config
- Path: output/DBaccounts.csv
- Write mode: APPEND



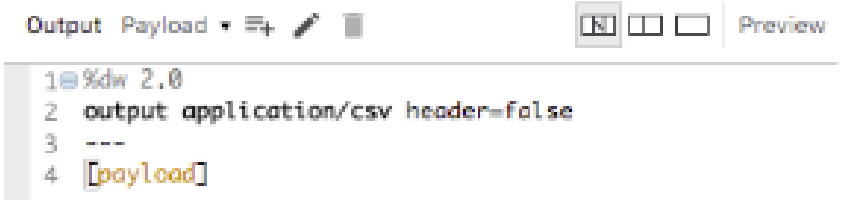
22. Select the input tab in the DataSense Explorer.
23. Expand Payload; you should see the operation will get an Array of Objects from the Database operation.

Transform the records to CSV

24. Add a Transform Message component before the Write operation.
25. Change the display name to Java to CSV.

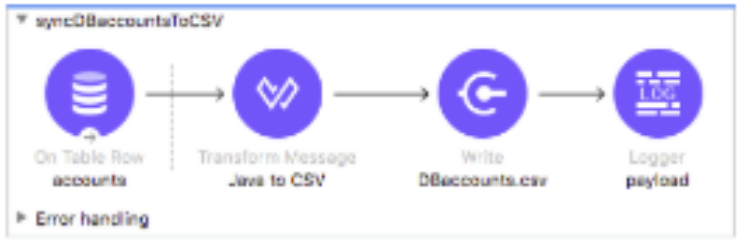


26. In the Transform Message properties view, change the output type to application/csv and add a header property equal to false.
27. Set the body expression to [payload].



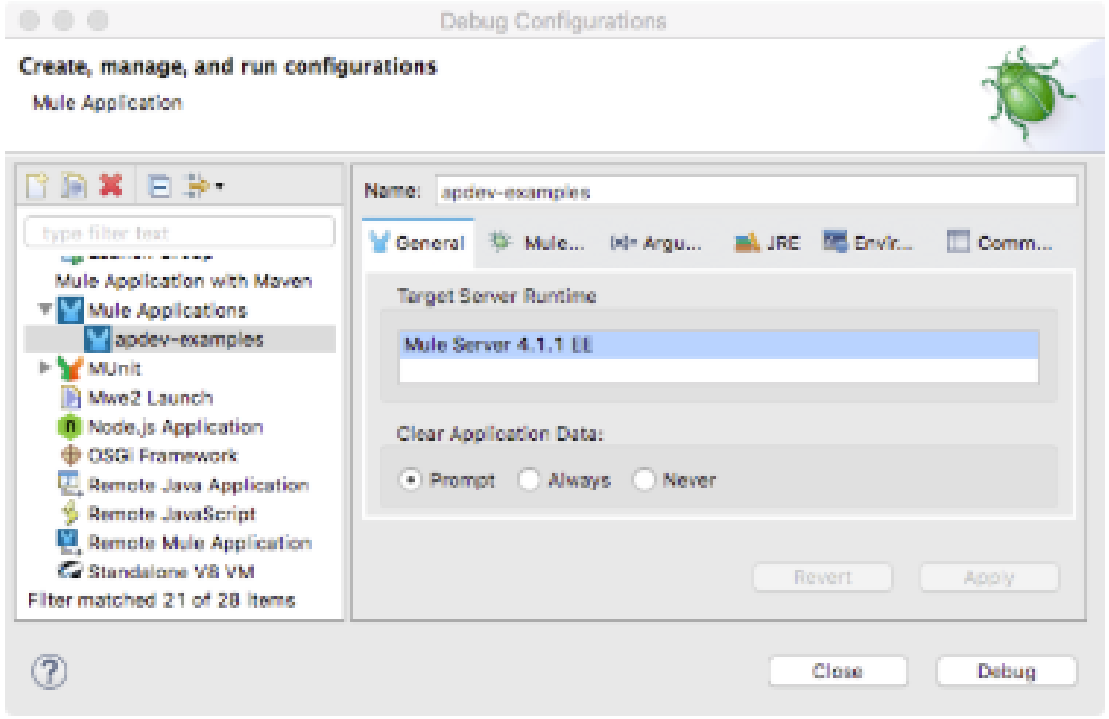
Add a Logger to display the records

28. Add a Logger to the flow.
29. Change the display name to payload.
30. Set the message to display the payload.



Set the application to prompt to clear application data when it starts

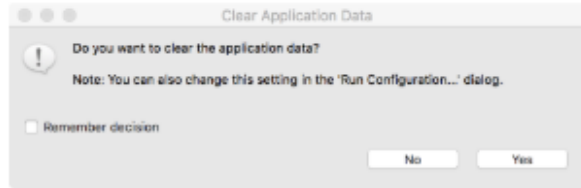
31. Add a breakpoint to the Transform Message component.
32. In the main menu, select Run > Debug Configurations.
33. In the Debug Configurations dialog box, locate the Clear Application Data section in the General tab and change it to Prompt.



Note: You need to rerun or debug an application to get the prompt; you cannot just save to redeploy.

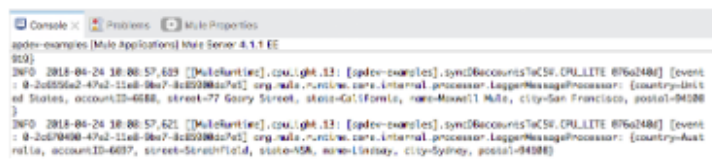
Debug the project

34. Click Debug.
35. In the Clear Application dialog box, select Yes.



36. In the Mule Debugger, expand Payload.

37. Click Resume and step through several of the records.
38. Look at the console, you should records displayed.

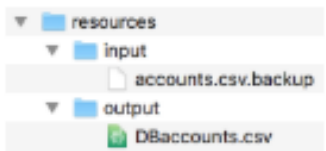


39. Stop the project and switch perspectives.

Clear application data and test the application

40. Run the project.
41. In the Clear Application dialog box, select Yes.
42. Look at the console; you should see many records.

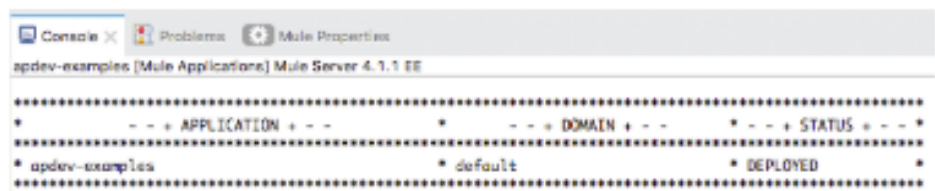
43. Stop the project.
44. Return to the resource folder in your computer's file explorer; you should see a new DBaccounts.csv file appear in the output folder.



45. Open the DBaccounts.csv file with a text editor; you should see the records.

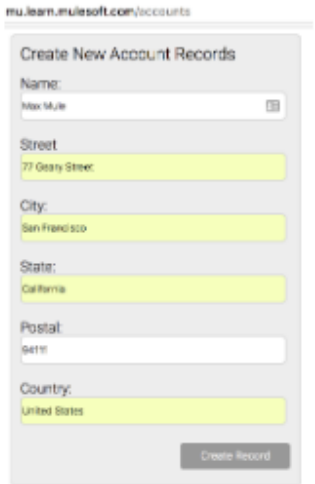
Test the application again without clearing application data

46. Return to Anypoint Studio and run the project.
47. In the Clear Application dialog box, select No.
48. Look at the console; you should see no new records output.



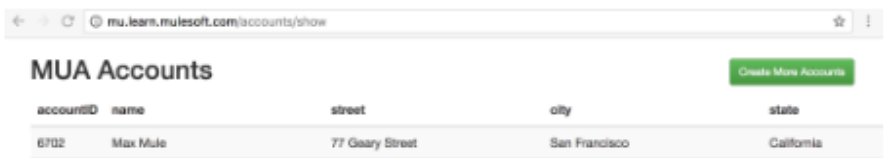
Add a new account to the database

49. Return to the account data in the web browser at http://mu.mulesoft-training.com/accounts/show.
50. Click Create More Accounts.
51. Fill out the form with data and click Create Record.

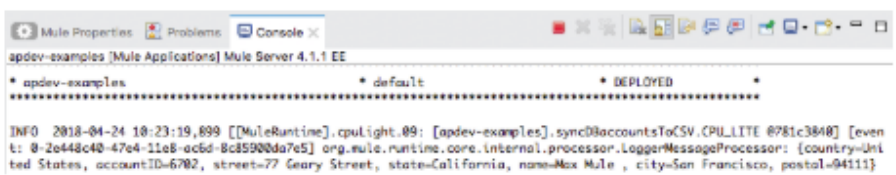


Note: Set the postal code to a specific value. In the next walkthrough, you will retrieve only new records with this specific postal code, so you do not get all of the records.

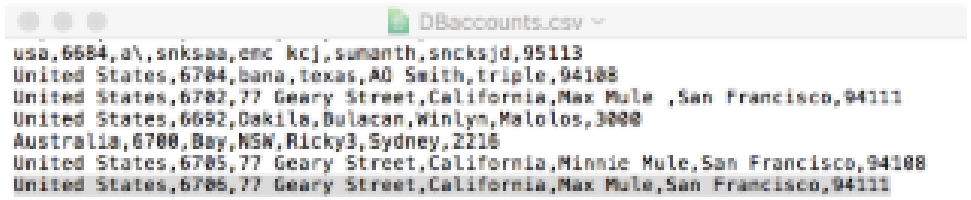
52. Click View Existing Accounts; you should see your new account.



53. Return to the console in Anypoint Studio; you should see your new record.



54. Stop the project.
55. Return to your computer's file explorer; the modified date on the DBAccounts.csv file should have been updated.
56. Open DBaccounts.csv and locate your new record at the end of the file.



Using manual watermarking and scheduling flows

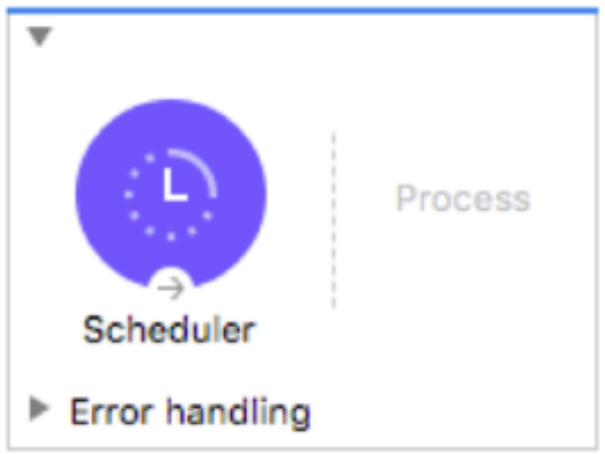
Handling watermarking manually

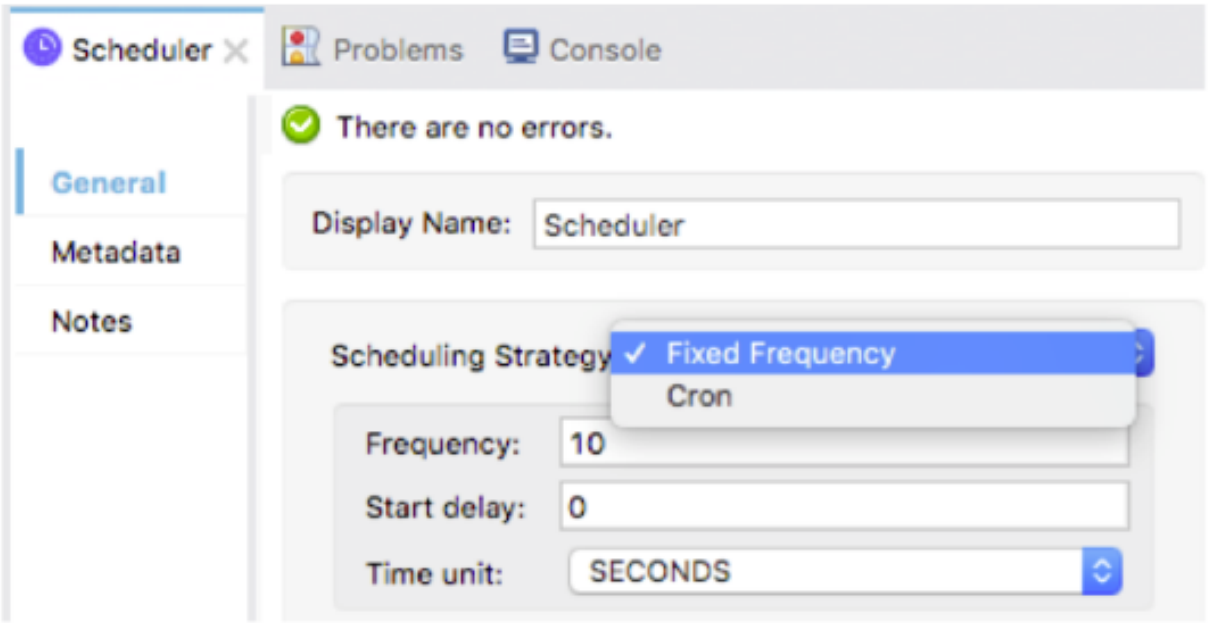
- The general process
  - Schedule when a flow should be executed
  - Give the watermark a default value
  - On the first sync
    - Determine a new watermark value
    - Store the watermark value so it available in the future to other flow executions
  - On later syncs
    - Retrieve the watermark from storage
    - Check if each item in the data set should be retrieved based on the watermark value

Triggering flows at a certain time or frequency

- Some connector operations use a scheduling strategy to trigger a flow
  - Like On New or Updated File and On Table Row

- To trigger **any** flow at **any** time, use the **Scheduler** component





Two types of scheduling strategies

- **Fixed frequency**
  - The default is to poll every 1000 milliseconds

- **Cron**
  - A standard for describing time and date information
  - Can specify either
    - An event to occur just once at a certain time
    - A recurring event on some frequency

0 15 10 ? * *	Poll at 10:15am every day
0 15 10 * * ? 2018	Poll at 10:15pm every day in 2018
1 1 1 1,6 *	Poll the first day of January and June every year in the first second of the first minute of the first hour

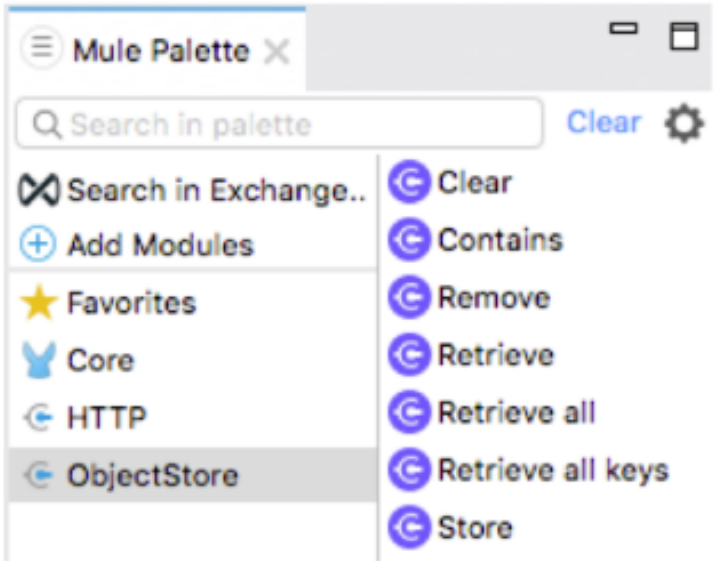
Persisting data across executions of flows

- Use the **Object Store** component to store simple key-value pairs
  - The component was designed to store
    - Synchronization information like watermarks
    - Temporal information like access tokens
    - User information
  - The values are accessible as event variables

- Each Mule application has an Object Store that is
  - Available without any setup or configuration
  - Persistent
    - Saved to file for embedded Mule and standalone Mule runtime
    - Saved to data storage for CloudHub
    - Saved to shared distributed memory for clustered Mule runtimes

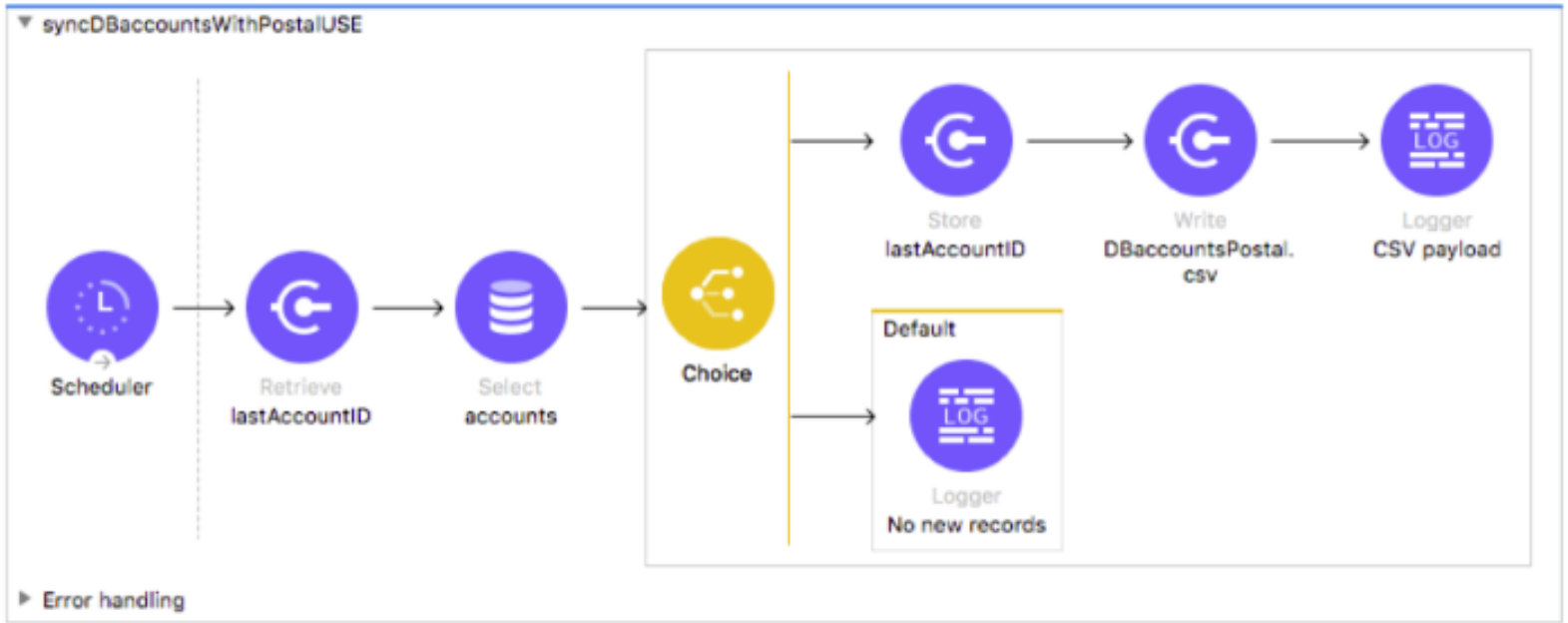
Using the Object Store connector for watermarking

- Add the ObjectStore module to the project
- Use the **Retrieve** operation to retrieve a watermark value and to assign a default value for the first poll
- Use the watermark value in a processor to retrieve the desired items
  - Like in a database query for records in a table
- Use the **Store** operation to determine and store a watermark value



Walkthrough 12-3: Schedule a flow and use manual watermarking

- Use the Scheduler component to create a new flow that executes at a specific frequency
- Retrieve accounts with a specific postal code from the accounts table
- Use the Object Store component to store the ID of the latest record and then use it to only retrieve new records



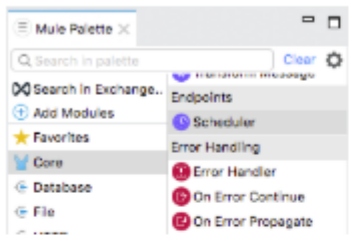
Walkthrough 12-3: Schedule a flow and use manual watermarking

In this walkthrough, you continue to work with the accounts table in the training database. You will:

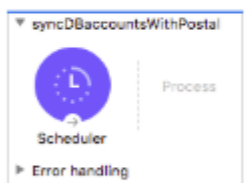
- Use the Scheduler component to create a new flow that executes at a specific frequency.
- Retrieve accounts with a specific postal code from the accounts table.
- Use the Object Store component to store the ID of the latest record and then use it to only retrieve new records.

Create a flow that executes at a specific frequency

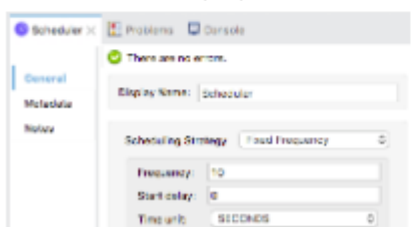
1. Return to accounts.xml in Anypoint Studio.
2. Locate the Scheduler component in the Core section of the Mule Palette.



3. Drag a Scheduler component from the Mule Palette and drop it at the top of the canvas to create a new flow.
4. Change the name of the flow to syncDBaccountsWithPostal.



5. In the Scheduler properties view, set the frequency to 10 seconds.



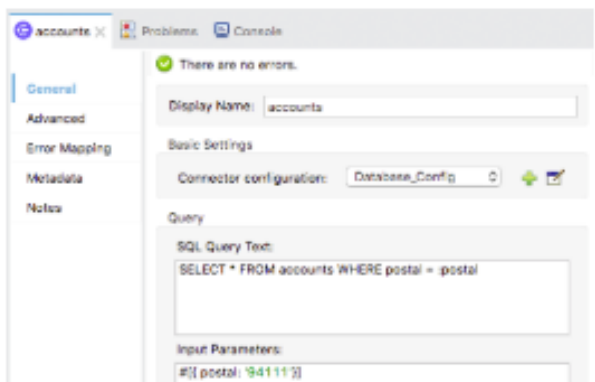
Retrieve records with a specific postal code from the database

6. From the Mule Palette, drag a Database Select operation and drop it in the flow.

7. In the Select properties view, set the following:

   - Display Name: accounts
   - Connector configuration: Database _Config
   - SQL query text: SELECT * FROM accounts WHERE postal = :postal
   - Input parameters: #[{ postal: *'yourPostalValue'*}]

   Note: If you want, you can store the postal code as a property in config.yaml and then reference it here in the DataWeave expression as #[{ postal: p('propertyName')}].

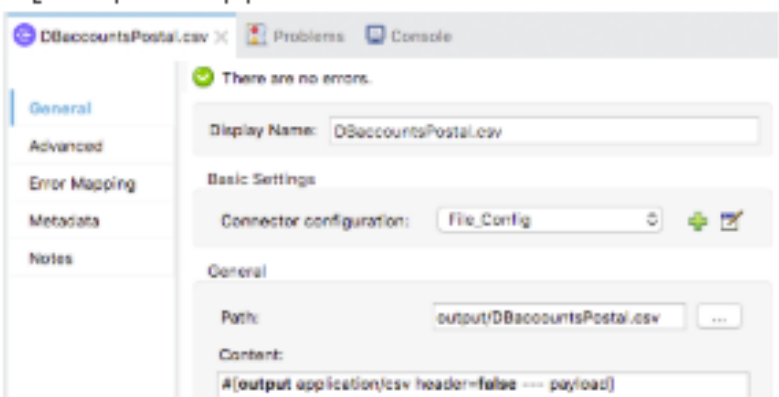


Output the records to a CSV file

8. Copy the Write operation in syncDBaccountsToCSV and paste it at the end of syncDBaccountsWithPostal.

9. In the properties view for the Write operation, set the following values:
   - Display Name: DBaccountsPostal.csv
   - Path: output/DBaccountsPostal.csv
10. Change the content to output the payload as application/csv with a header property equal to false.

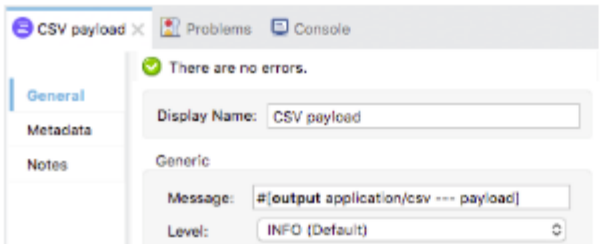
```
#[output application/csv header=false --- payload]
```



Log the payload

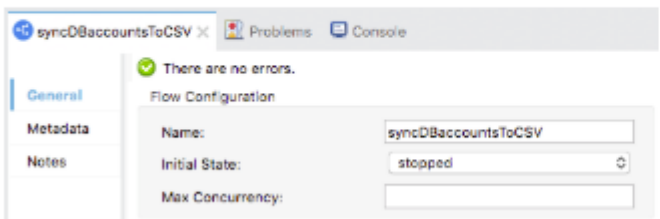
11. Add a Logger at the end of the flow and change the display name to CSV payload.

12. Set the message to the payload as type application/csv.



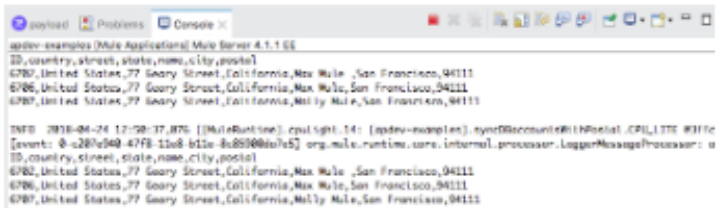
Stop the other syncDBaccountsToCSV flow so it does not run

13. In the properties view for the syncDBaccountsToCSV flow, set the initial state to stopped.



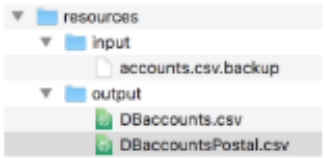
Test the application

14. Run the project.
15. In the Clear Application dialog box, select Yes.
16. Watch the console, you should see the same records displayed every 10 seconds.



Note: Right now, all records with matching postal code are retrieved – over and over again. Next, you will modify this so only new records with the matching postal code are retrieved.

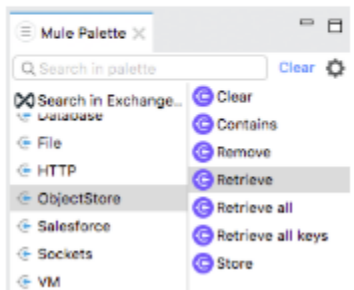
17. Stop the project.
18. Return to the resources folder in your computer's file browser; you should see the new DBaccountsPostal.csv file.



19. Open the file and review the contents.
20. Delete the file.

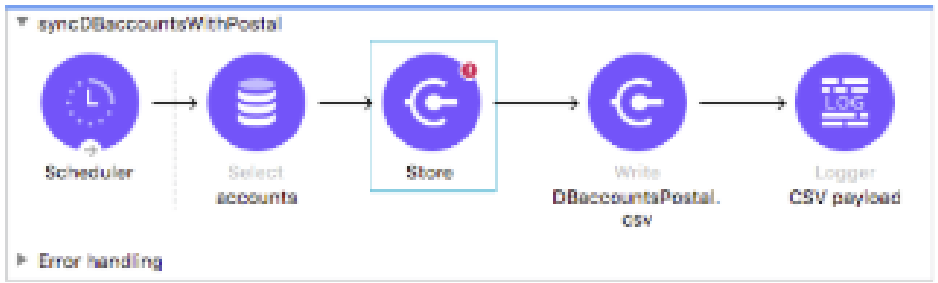
Add the ObjectStore module to the project

21. In the Mule Palette, select Add Modules.
22. Select the ObjectStore connector in the right side of the Mule Palette and drag and drop it into the left side.
23. If you get a Select module version dialog box, select the latest version and click Add.



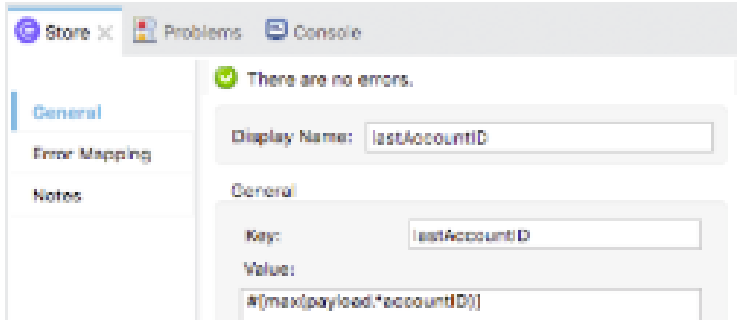
Store the ID of the last record retrieved

24. Drag the Store operation for the ObjectStore connector from the Mule Palette and drop it after the Database Select operation.



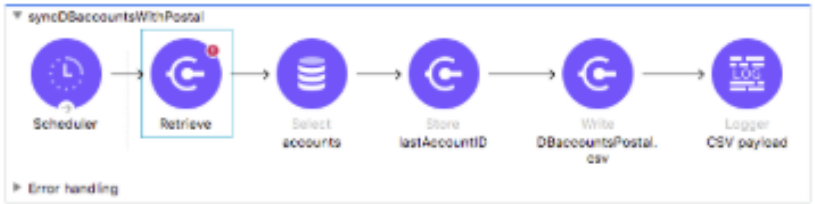
25. In the Store properties view, set the display name and key to lastAccountID.
26. Set the value to an expression for the maximum value of lastAccountID in the payload, the retrieved records.

```
#[max(payload.*accountID)]
```



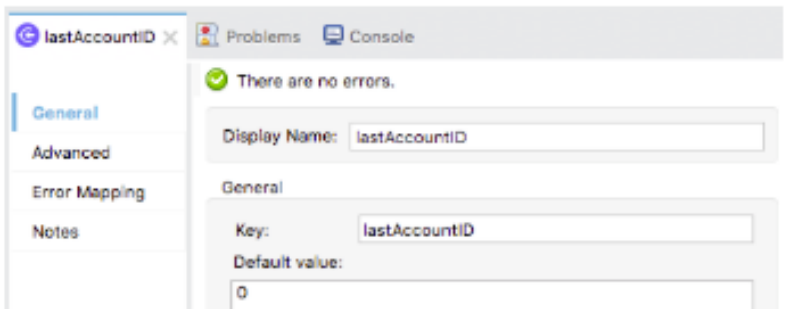
Before the query, retrieve the ID of the last record retrieved

27. Drag the Retrieve operation for the ObjectStore connector from the Mule Palette and drop it before the Database Select operation.

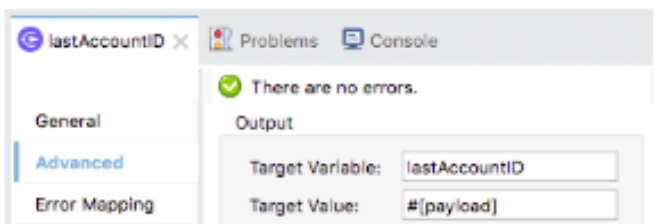


28. In the Retrieve properties view, set the following values:

- Display Name: lastAccountID
- Key: lastAccountID
- Default value: 0

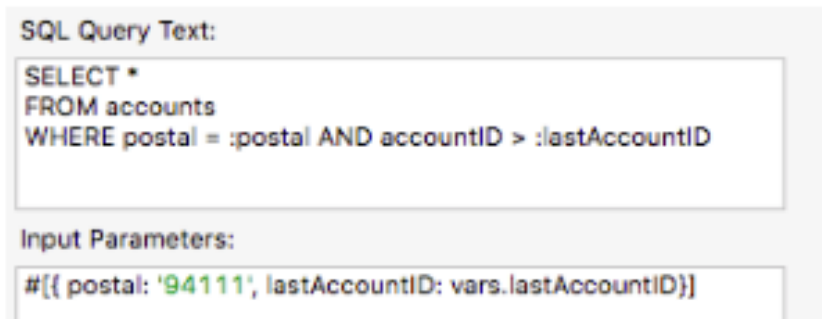


29. Select the Advanced tab and set the target variable to lastAccountID so the value is stored in a variable called lastAccountID.



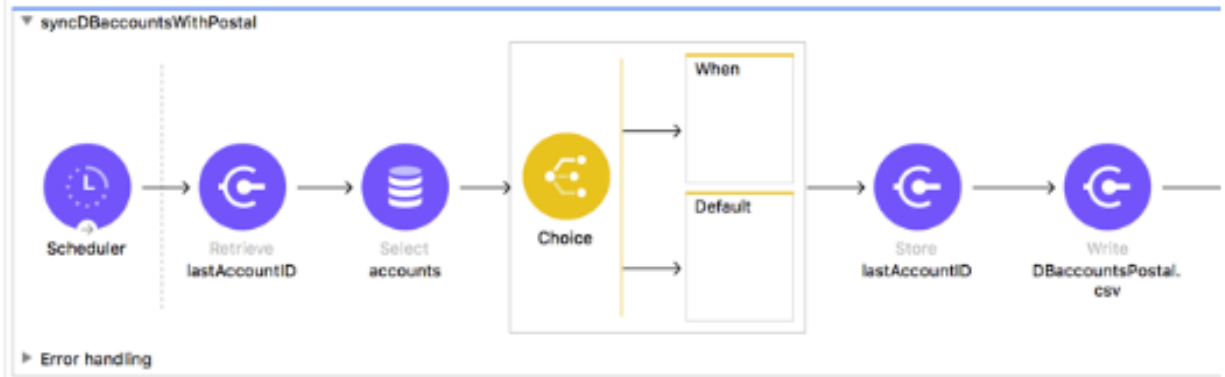
Modify the query to only retrieve new records with a specific postal code

30. In the accounts Select properties view, add a second query input parameter called lastAccountID that is equal to the watermark value.
    `#[{postal: 'yourValue', lastAccountID: vars.lastAccountID}]`
31. Modify the SQL query text to use this parameter to only retrieve new records.
    `SELECT * FROM accounts WHERE postal = :postal AND accountID > :lastAccountID`

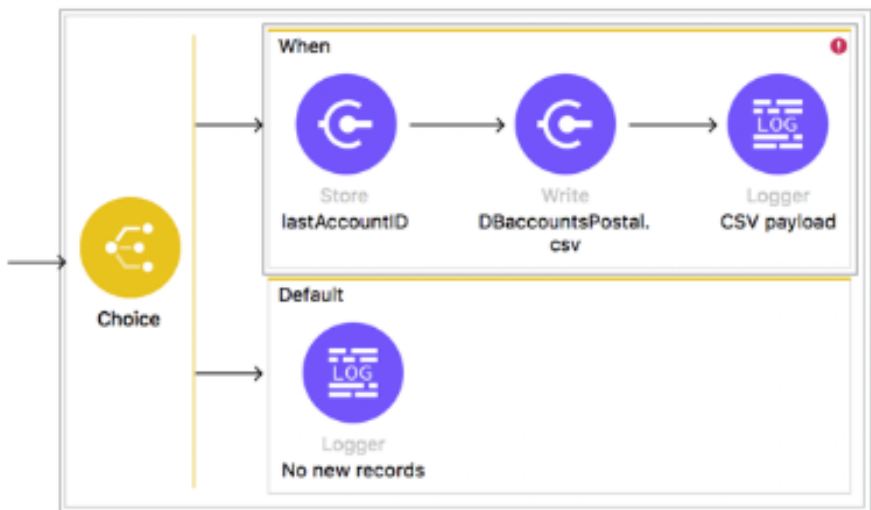


Only store the value if new records were retrieved

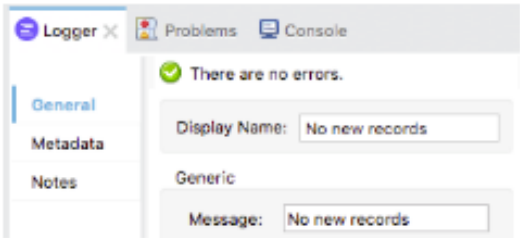
32. Add a Choice router after the Select operation.



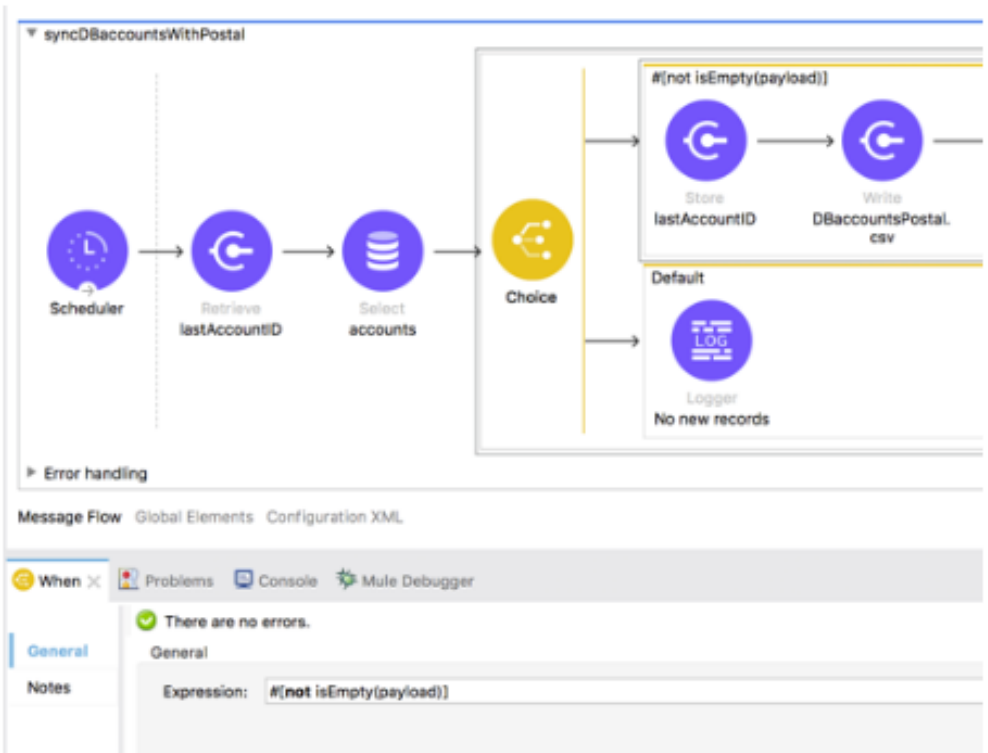
33. Move the Store, Write, and Logger operations into the when branch of the router.
34. Add a Logger to the default branch.



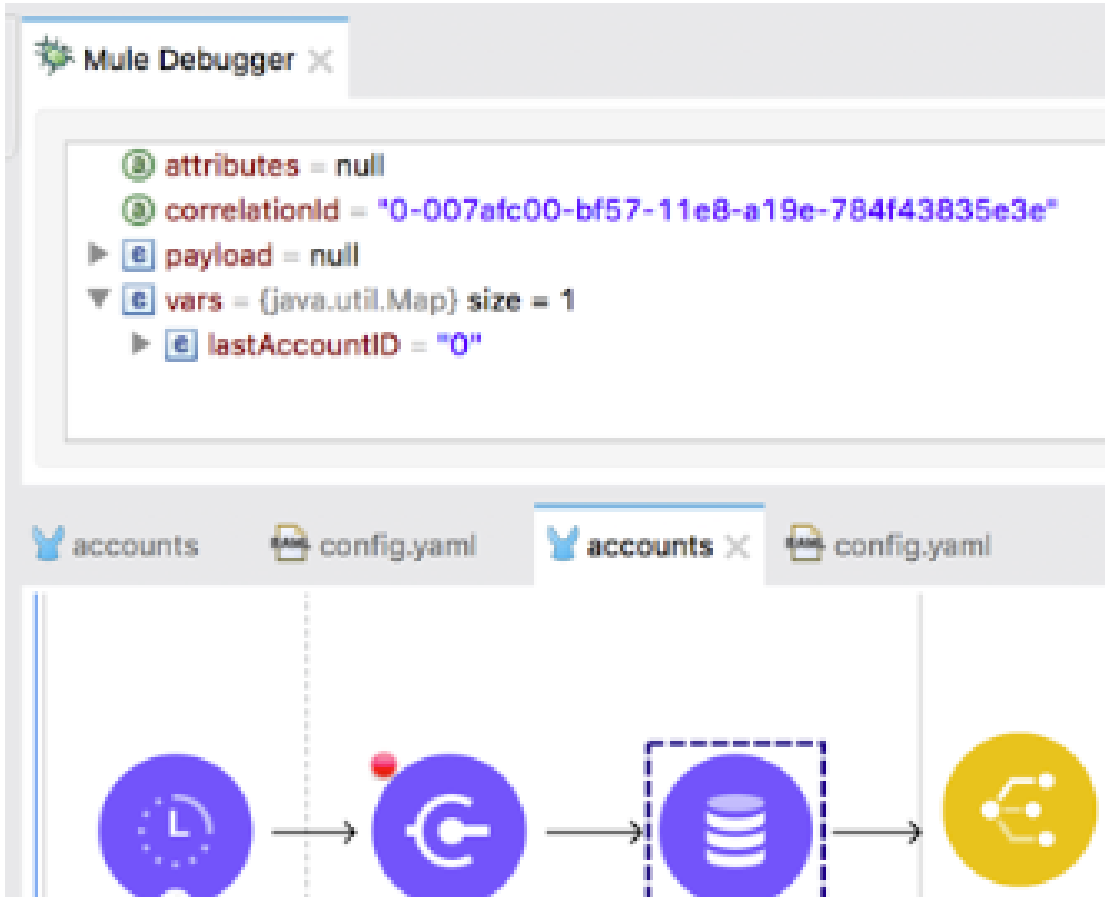
35. In the Logger properties view, set the display name and value to No new records.



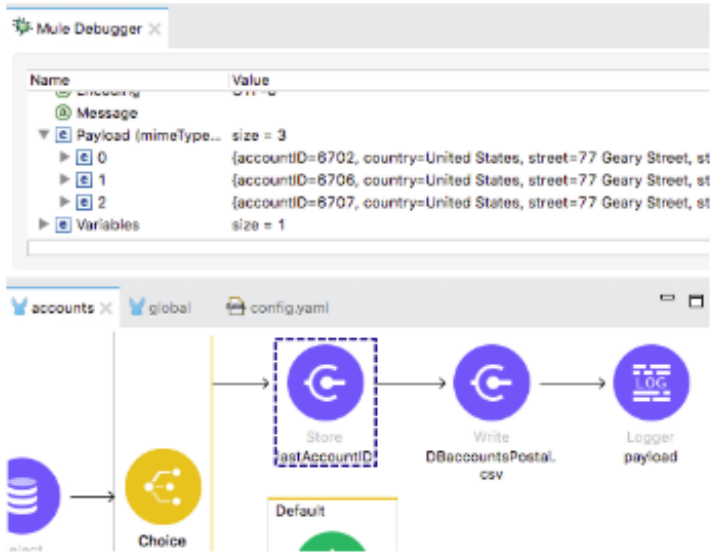
36. In the Choice router, click the when scope and in the properties view, add an expression to route the event to the first branch when the payload is not empty.
`#[not isEmpty(payload)]`



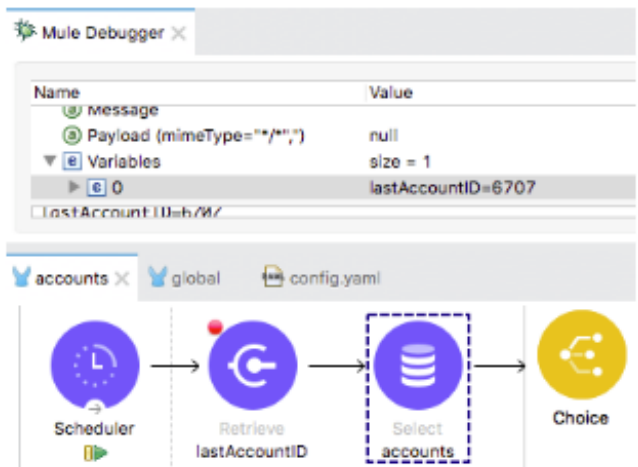
37. Add a breakpoint to the Retrieve operation.
38. Debug the project.
39. In the Clear Application dialog box, select Yes.
40. In the Mule Debugger, step to the Select operation; you should see a lastAccountID variable with a value of 0.



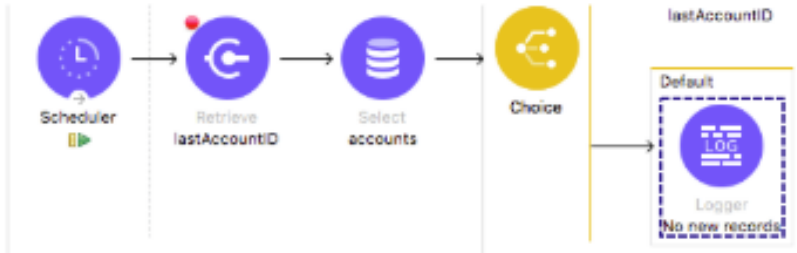
41. Step to the Choice router; you should see record(s) were retrieved from the database.



42. Step through the rest of the application and click resume; the flow should be executed again, and execution should be stopped back at the beginning of the flow.
43. Step to the Select operation; you should see the lastAccountID variable now has a non-zero value.



44. Step into the Choice router; you should see no records were retrieved from the database (unless someone else just added one with the same postal code!).

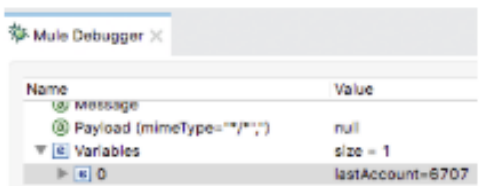


45. Stop the project and switch perspectives.
46. Return to your computer's file explorer and locate and open the new DBaccountsPostal.csv file.

    Note: If you see the same records more than once, it was because during debugging, the flow was executed again before the watermark was stored.

Debug the application and do not clear application data

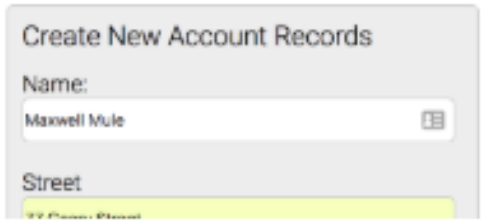
47. Return to Anypoint Studio and debug the project.
48. In the Clear Application dialog box, select No.
49. In the Mule Debugger, step to the Select operation; you should see a lastAccountID variable with the same non-zero value.



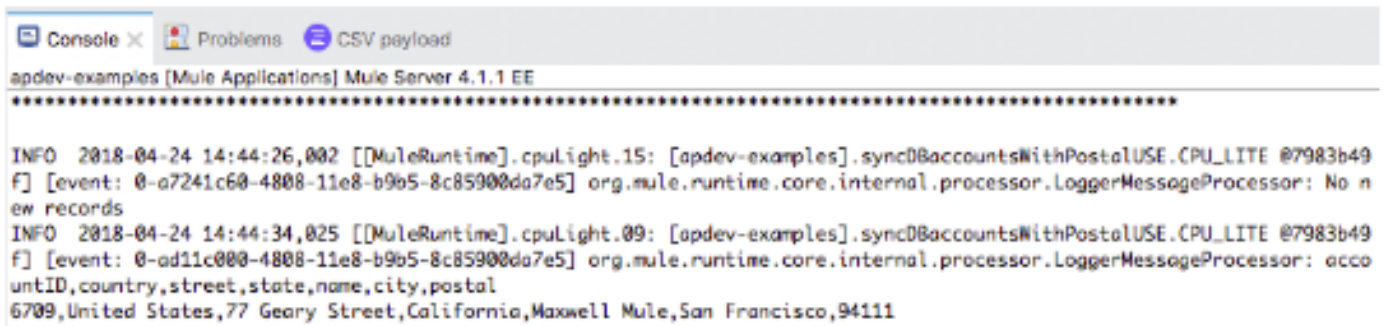
50. Step to the Choice router; no new records should have been returned.
51. Remove the breakpoint from the Retrieve operation.
52. Click Resume until application execution is no longer stopped anywhere.
53. Make sure there are no other breakpoints in the flow.

Add a new account to the database

54. Return to the account data in the web browser at http://mu.mulesoft-training.com/accounts/show.
55. Click Create More Accounts.
56. Fill out the form with data and use the same postal code.



57. Click Create Record.
58. Click View Existing Accounts; you should see your new account.
59. Return to Anypoint Studio; you should see your new record in the console.



60. Return to your computer's file explorer; the modified date on the DBAccountsPostal.csv file should have been updated.
61. Open DBaccountsPostal.csv and locate your new record at the end of the file.
62. Return to Anypoint Studio and switch perspectives.
63. Stop the project.

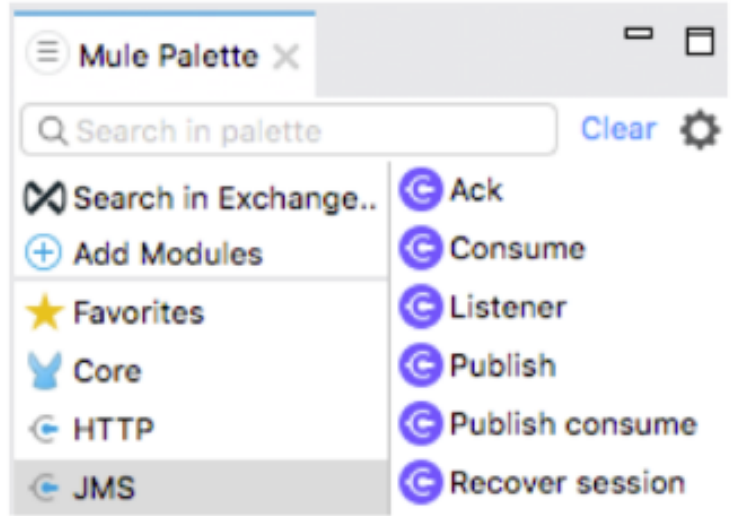
Publish and consume JMS messages

Java Messaging Service (JMS)

- Is a widely-used API for enabling applications to communicate through the exchange of messages
- Simplifies application development by providing a standard interface for creating, sending, and receiving messages
- Supports two messaging models
  - **Queues**: PTP (point-to-point or 1:1)
    - A sender sends messages to a queue & a single receiver pulls the message off the queue
    - The receiver does not need to be listening to the queue at the time the message is sent
  - **Topics**: Pub-Sub (publish/subscribe or 1:many)
    - A publisher sends a message to a topic & all active subscribers receive the message
    - Subscribers not actively listening will miss the published message (unless messages are made durable)

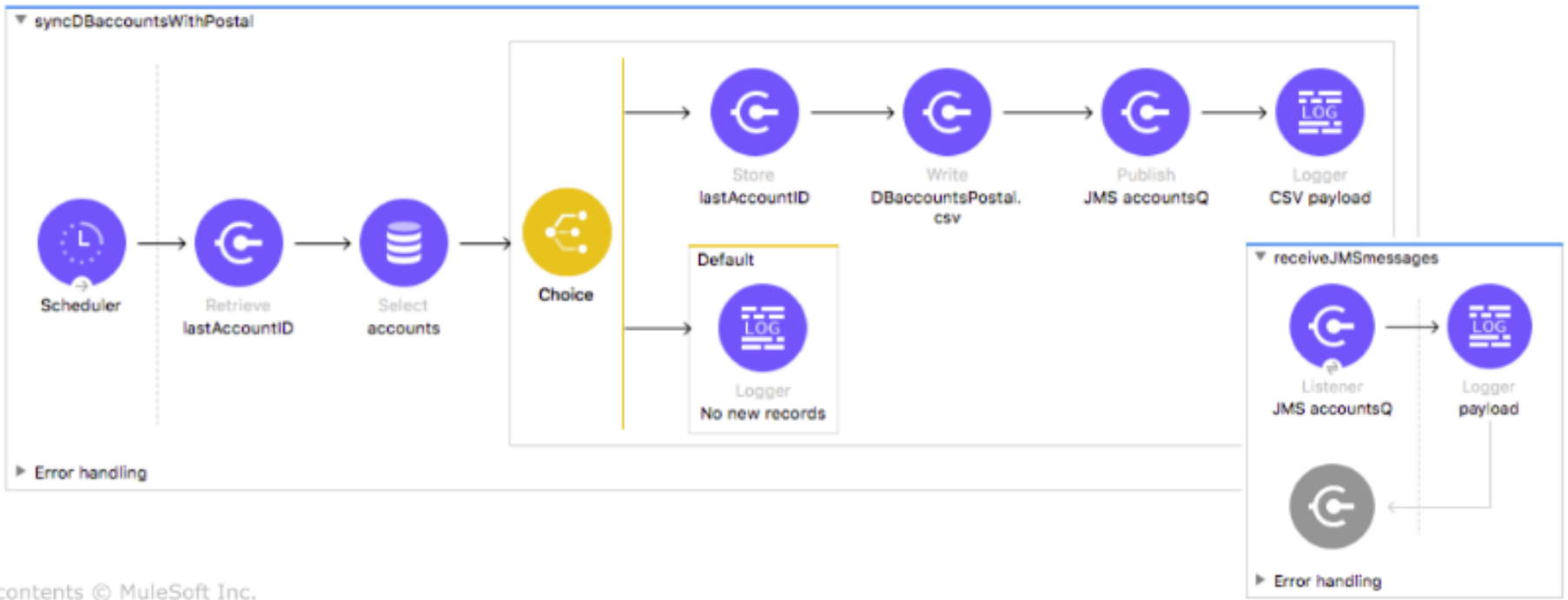
Using the JMS connector

- Add the JMS module to the project
- Configure a global element configuration
  - By default, it is set up with a finely tuned set of values for both for publishing and consuming messages
  - Typically, you just need to configure which connection should be used
- Use operations to publish and/or consume messages to destinations



Walkthrough 12-4: Publish and listen for JMS messages

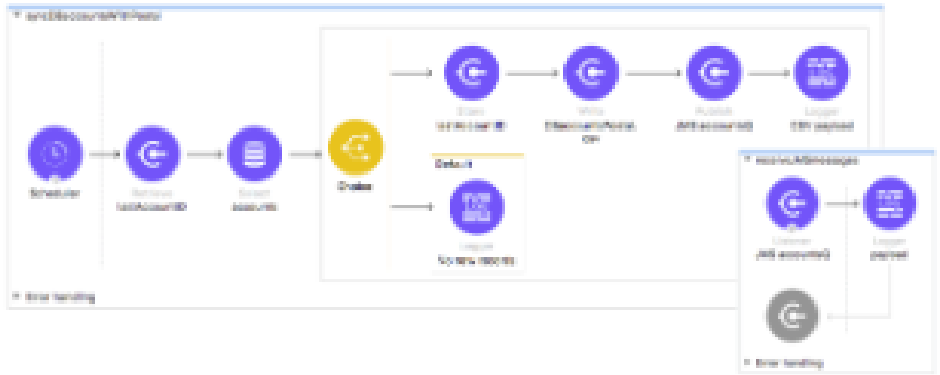
- Add and configure a JMS connector for ActiveMQ (that uses an in-memory broker)
- Send messages to a JMS queue
- Listener for and process messages from a JMS queue



Walkthrough 12-4: Publish and listen for JMS messages

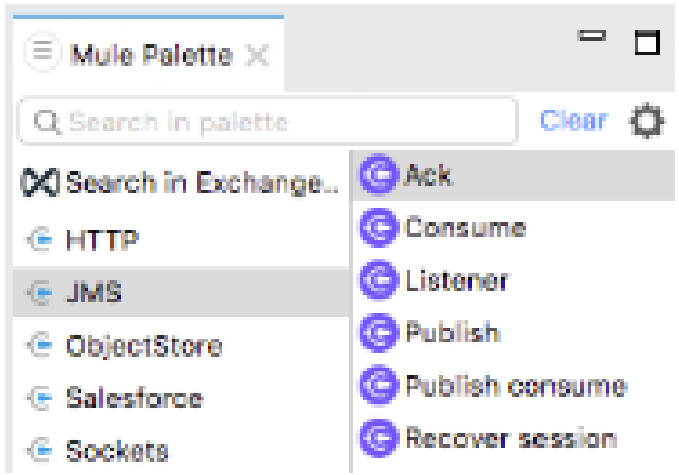
In this walkthrough, you send a JMS message for each new database record so it can be processed asynchronously. You will:

- Add and configure a JMS connector for ActiveMQ (that uses an in-memory broker).
- Send messages to a JMS queue.
- Listener for and process messages from a JMS queue.



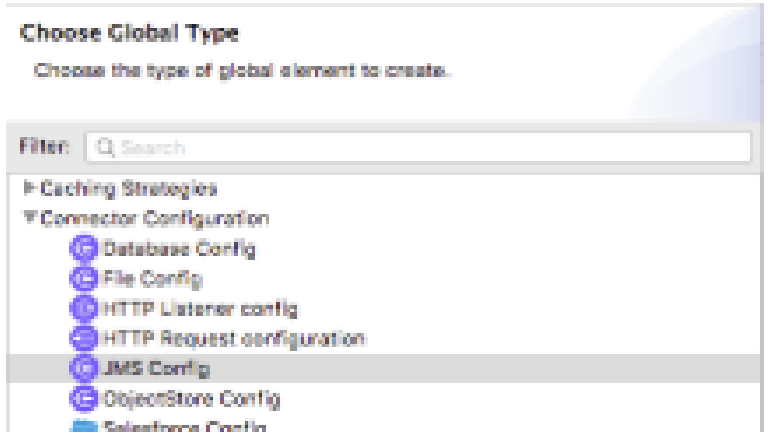
Add the JMS module to the project

1. Return to accounts.xml.
2. In the Mule Palette, select Add Modules.
3. Select the JMS connector in the right side of the Mule Palette and drag and drop it into the left side.
4. If you get a Select module version dialog box, select the latest version and click Add.



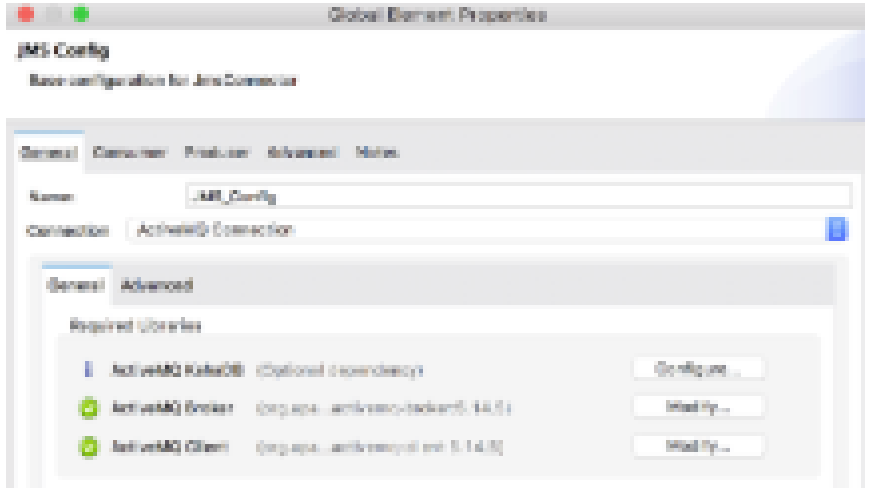
Configure the JMS connector

5. Return to the Global Elements view of global.xml.
6. Click Create.
7. In the Choose Global Type dialog box, select Connector Configuration > JMS Config and click OK.

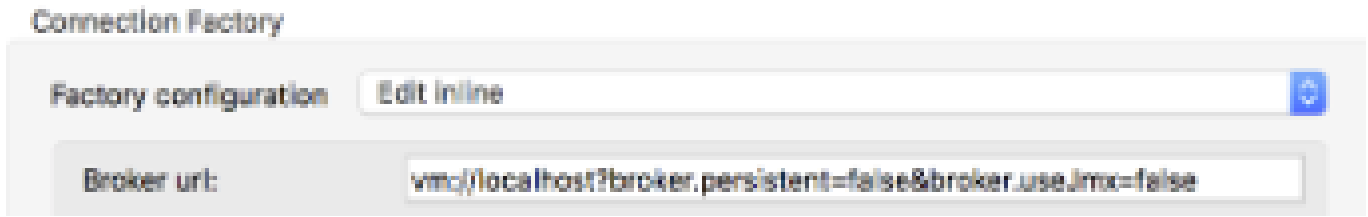


8. In the Global Element Properties dialog box, make sure the connection is set to ActiveMQ Connection.
9. Click the Configure > Add Maven dependency option for the ActiveMQ Broker.
10. In the Maven dependency dialog box, enter activemq-broker in the Search Maven Central field.

11. Select the org.apache.activemq:activemq-broker entry and select Finish.



12. Scroll down and locate the Connection Factory section.
13. Select Factory configuration, select Edit inline; the Broker url should already be populated.



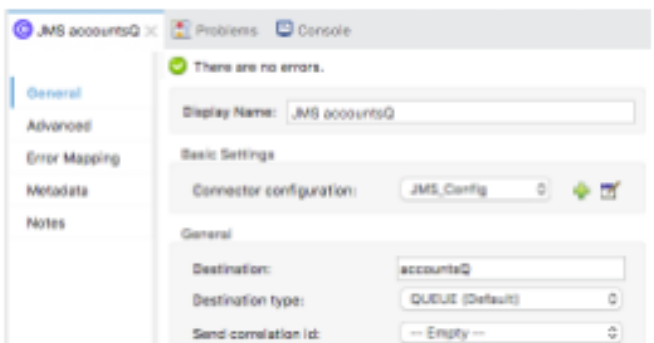
14. Click OK.

Use the JMS Publish operation to send a message to a JMS queue

15. Return to accounts.xml.
16. Drag out a JMS Publish operation from the Mule Palette and drop it after the Write operation in syncDBaccountsWithPostal.

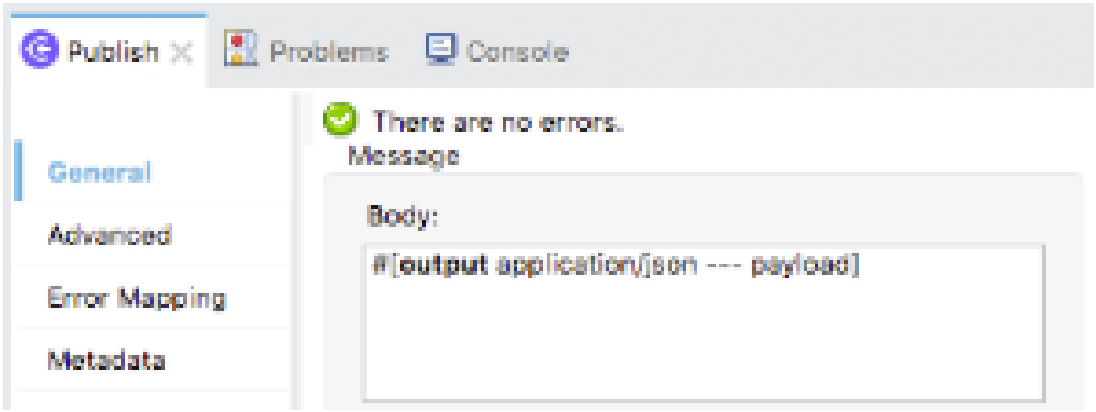
17. In the Publish properties view, set the following values:

- Display Name: JMS accountsQ
- Connector configuration: JMS_Config
- Destination: accountsQ
- Destination type: QUEUE



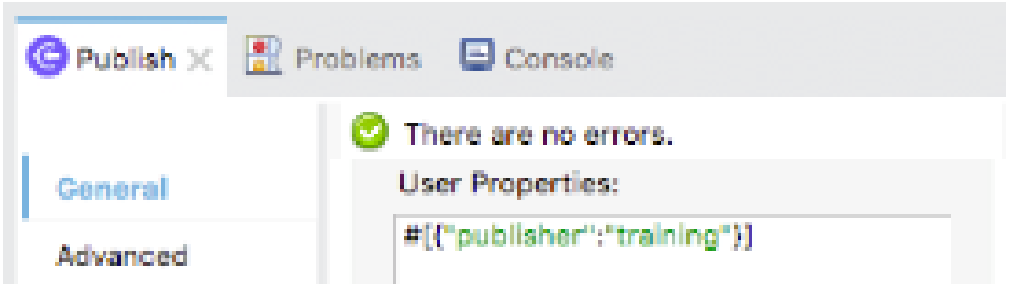
18. Modify the message body to be of type application/json.

```
#[output application/json --- payload]
```



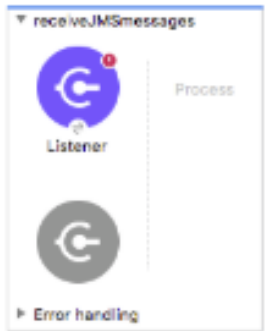
19. Scroll down and locate the User Properties field.
20. Set it equal to an object a key called publisher with a value of training.

```
#[{"publisher":"training"]
```



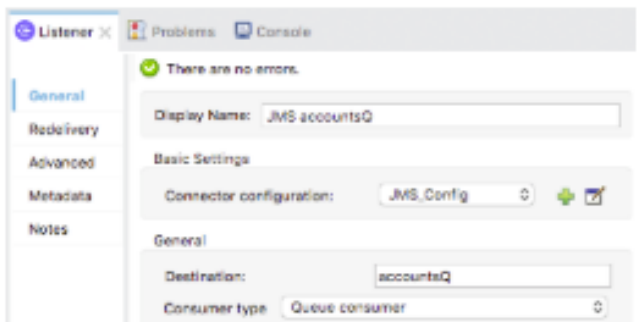
Create a flow to listen to a JMS topic

21. Drag out the JMS Listener operation from the Mule Palette and drop it in the canvas to create a new flow.
22. Give the flow a new name of receiveJMSMessages.



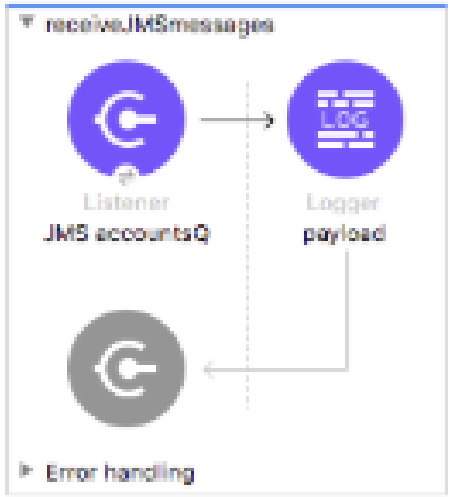
23. In the Listener properties view, set the following values:

    - Display Name: JMS accountsQ
    - Connector configuration: JMS_Config
    - Destination: accountsQ
    - Consumer type: Queue consumer



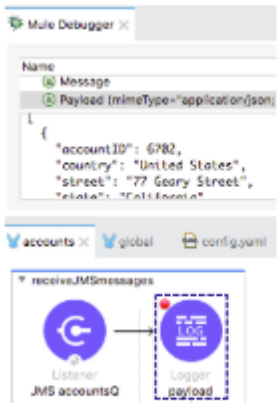
24. Add a Logger to the flow.

25. Set its display name to payload and its message to #[payl

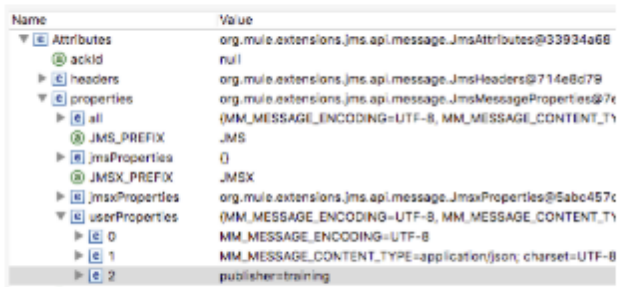


Clear application data and debug the application

26. Add a breakpoint to the Logger in receiveJMSmessages.
27. Debug the project.
28. In the Clear Application Data dialog box, select Yes.
29. Wait until the project deploys; application execution should stop in receiveJMSMessages.
30. In the Mule Debugger view, look at the value of the payload.



31. Expand the Attributes and locate the publisher property in the userProperties.



32. Step through the rest of the application; you should see the JSON message in the console.

    Note: If you want to test further, return to the account data in the web browser at http://mu.mulesoft-training.com/accounts/show and add a new record with the same postal code.

33. Stop the project and switch perspectives.

Summary

Summary

- Use **watermarks** to synchronize data across data stores
  - Use either **manual** or the **automatic** watermarking available for some connectors
- Use the family of **File**, **FTP**, **FTPS**, and **SFTP** connectors to work with files and folders
- Use the **On New or Updated File** listener to trigger flows when files are added, created, or updated
  - Use the connector's **automatic watermarking** to determine if a file is new based on a creation or modification timestamp
- Use the **On Table Row** listener when new records are added to a database table
  - Use the connector's **automatic watermarking** to determine if the record is new

Summary

- Use the **Scheduler** component to schedule flows to run at a certain time or frequency
  - Use a watermark to keep a persistent variable between scheduling events
- Use the **Object Store** connector to persist and share a watermark (or other data) across flow executions
- Use the **JMS** connector to publish and consume messages
  - Connect to any JMS messaging service that implements the JMS spec

DIY Exercise 12-1: Read and write files using the File, FTP, and Database connectors

Time estimate: 3 hours

Objectives

In this exercise, you connect Mule applications and other systems together using various connectors. You will:

- Insert files into a database using the Database connector.
- Write files using the FTP connector.
- Write files using the File connector.

Scenario

You are assigned to a project involving the company's banking branch, Mule Bank. The bank has partnered with several retail partners to distribute gift cards that are funded by the bank but handled and processed by Visa. The bank has decided to use tightly coupled solutions for their partners who are not yet supporting REST or SOAP.

Gift card numbers are generated from your credit card processing partner Visa and are then uploaded in bulk as CSV files to the Mule bank's middleware server. Each partner's new Visa gift card numbers need to be stored in a separate CSV file for that partner. Your task is to develop a network of Mule applications that will retrieve these Visa gift card CSV files and then route and process them to the correct partner's receiving servers.

Each partner will use a different technology to receive their lists of Visa gift cards:

- The Food-N-Savings retail partner has set up a database table named Giftcards to receive gift cards. In order to successfully insert each row in the CSV file into the database, you will need to property map each Visa gift card row in the CSV file to the correct database columns.
- The Meals-N-Go retail partner has set up an FTP server in their DMZ (demilitarized zone) to receive gift cards. They have an existing processing system that expects gift cards in a specific CSV format that is different from the Visa gift card CSV format, so you will have to map from the Visa gift card CSV file format to the Meals-N-Go CSV format.
- The Online Retail Plus partner is a little more modern. They already have an Enterprise Service Bus implementation with a JMS messaging bus. They also already have a JMS message format set up to receive and process gift cards, so you will have to map from the Visa gift card CSV format to the JMS message format.

Note: For ease of development and testing, you will first create all the flows for all Mule applications in one Mule application. This helps you rapidly create and test the Process API logic and data mappings. In the next exercise, you will refactor these flows into separate Mule applications to be distributed to different partners, which can communicate with each other using an Enterprise messaging service like JMS.

Import the starting project

Import /files/module12/connectors-mod12-various-systems-starter.jar in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio and name it connect-various-systems.

Install the mock-servers dependency located in MUFundamentals_DIYexercises.zip into your local m2 repository to allow the project to run the mock FTP and mock Database servers that you will use for this project. For help, follow the step-by-step Install a Mule application as a dependency walkthrough in module 6.

Create a flow to read in gift cards

Create a new Mule configuration file named process-visa-gift-cards. Create a flow named processVisaGiftCards to retrieve CSV files from a folder that Visa will use to transfer CSV files to this Mule application.

Use property placeholders to configure the file reader, so properties including the input and output folder can be changed when the Mule application is deployed to a different environment, such as to a production environment.

Note: Your operations team will set up a folder in a server where Visa can upload files. The server will host this Mule application in a customer-hosted Mule runtime. To learn how to deploy Mule applications to customer-hosted Mule runtimes, take the Anypoint Platform Operations: Customer-Hosted Runtimes course.

Answer the following questions

- Instead of listening to new files being written to a server, a database table will constantly get updated with new gift cards. What operation and configuration would you need to replace the On New File operation with to ensure new gift card records are processed only once?

Create a flow to insert files into the Food-N-Savings database

The Food-N-Savings partner has set up a Postgres database to consume gift cards. The table is named Giftcards and has fields named number, sourceID, balance, and createdOn. The sourceID is the Mule Bank's ID, which is MULEBANK-0949. Access to the Postgres database has not yet been allowed by the Food-N-Savings security team. To help you focus on just developing the Mule application, the starter project embeds an in-memory Java Derby database that automatically launches when you run the project. This database uses the same schema as the Postgres database, so it can be used instead of starting up a local Postgres instance during development.

Note: The configuration to run this in-memory Java Derby database also shows you how to use Spring beans in a Mule 4 application to run Java code when a Mule application starts and stops.

Create a new Mule configuration file named foodnsavings.xml to send gift cards to the Food-N-Savings partner. A sample Visa CSV file for Food-N-Savings is included in the starter project's src/test/resources folder. The CSV file has the following format:
card_no,processor_id,amount_granted,partner

703530278.884400.150.Food and Savings

Add an environment variable to your Mule application and then add logic to the Food-N-Savings Mule configuration file's flow to use the Java Derby database if the environment is not set to prod.

Note: Remember that each flow that sends files to a partner must ultimately be an individual Mule application that can be deployed in the same region as the partner's server.

Add logic to the processVisaGiftCards flow to test the CSV file's partner column. If the partner value is "Food N Savings", route the event to the processing flow in foodnsavings.xml using a Flow Reference component.

Note: In the next exercise, you will replace the Flow Reference with a JMS Publish operation to a JMS queue and add a corresponding JMS Listener to the foodnsavings.xml file's flow.

In the foodnsavings.xml flow, use the following information and insert each CSV file row into the Giftcards table in the database:

Visa CSV file	**Food-N-Savings Giftcard database table**	**Values**
card_no	number	The Visa gift card number
amount_granted	balance	The current Visa gift card balance
	sourceID	The Mule Bank ID "MULEBANK-0949"
	createdOn	The current datetime when this mapping occurs

Note: To perform the database operations safely (to avoid SQL injection), be sure to use parameterized queries to insert data into the table.

Create a flow to write files to the Meals-N-Go partner's FTP server

The Meals-N-Go partner has set up an FTP server to receive the gift cards. To help you focus on just developing the Mule application, the starter project automatically launches when you run the project.

Note: The configuration to run this in-memory FTP server also shows you how to use Spring beans in a Mule 4 application to run Java code when a Mu

Create a new Mule configuration file named mealsngo.xml. Create a flow to process the Visa gift card CSV file by transforming the CSV file to the Meal transformed file to a Meals-N-Go managed FTP server.

Add logic to the processVisaGiftCards flow to test the CSV file's partner column. If the partner value is "Meals n Go", route the event to the processing Reference component.

Note: In the next exercise, you will replace the Flow Reference with a JMS Publish operation to a JMS queue and add a corresponding JMS Listener to

In the mealsngo.xml flow, use the following information and insert each row of the CSV file into the FTP server:

Visa CSV file	Meals-N-Go CSV file	Values
card_no	gc_card_number	The Visa gift card number
amount_granted	gc_balance	The current Visa gift card balance
	origin	The Mule Bank ID "MULEBANK-0949"
	card_type	"VISA"
	expiration	3 months after the current datetime when this mapping occurs, in milliseconds since Unix Epoch

The uploaded files must follow the naming pattern: MULEBANK-gc-{datetime}.csv, where {datetime} is the current datetime when the file is received by and must be in milliseconds since 1970 January 1 (Unix Epoch).

Create a flow to send a payload to the Online Retail Plus partner

Create a new Mule configuration file named oretailplus.xml. Create a flow to process the Visa gift card CSV file by transforming payload into a CSV file.

Add logic to the processVisaGiftCards flow to test the CSV file's partner column. If the partner value is "Online Retail Plus", route the event to the processing flow in oretailplus.xml using a Flow Reference component.

Note: In the next exercise, you will replace the Flow Reference with a JMS Publish operation to a JMS queue and add a corresponding JMS Listener to the oretailplus.xml file's flow.

In the oretailplus.xml flow, use the following information and convert the entire Visa CSV file with the following mapping:

Visa CSV file	Online Retail Plus message	Values
card_no	cardNo	The Visa gift card number
amount_granted	amount	The current Visa gift card balance
	bankOriginationID	The Mule Bank ID "MULEBANK-0949"

Finally, write out a copy of the payload to a local CSV file. The CSV files should be named MULEBANK-{partner3.name}-{datetime}-{count}.csv, where {count} is the record count for the CSV file and {datetime} is the current date and time in milliseconds since Unix Epoch.

Handle invalid partners

Write out a report to a file if there is a partner that is not Food-N-Savings, Meals-N-Go, or Online Retail Plus.

Verify your solution

Import the solution /files/module12/connectors-mod12-various-systems-part1-solution.jar deployable archive file (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

DIY Exercise 12-2: Route messages using JMS queues

Time estimate: 2 hours

Objectives

In this exercise, you continue to build the Visa gift card distribution Mule applications. You will:

- Conditionally process and route Mule events using different JMS queues.

Import the starting project

Use your solution from exercise 12-1 or import /files/module12/connectors-mod12-various-systems-part1-solution.jar (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio.

If you have not yet installed the mock-servers dependency, install the dependency located in MUFundamentals_DIYexercises.zip into your local m2 repository to allow the project to run the mock FTP and mock database server you will use for this project. For help, follow the step-by-step Install a Mule application as a dependency walkthrough.

Route the messages using JMS queues

Each partner should use a different JMS queue. Replace all the Flow Reference components in the processVisaGiftCards flow with different JMS message queues, one for each partner type: Food-N-Savings, Meals-N-Go, Online Retail Plus.

For testing purposes, use the default virtual JMS server that will be created when the Mule application is deployed. Set each queue name using a property placeholder, so it can be changed later by the Operations team without having to re-write the Mule application(s).

Answer the following questions

- Is it preferable to send one message per file or one message per record of a file?
- How do you specify the payload format of the JMS messages?
- What would you do if the file is too big to fit into memory?

Write out a summary report in the process-visa-gift-cards Mule application

Write out a report of the number of gift cards processed for each partner and which transfers succeeded. Each transfer needs to send a success or fail message back to the process-visa-gift-cards Mule application.

The process-visa-gift-cards Mule application will take the response messages from each of the other Mule applications and write out a summary report text file reporting the number of gift cards processed for each partner and the success or fail status of each transfer Mule application.

Answer the following questions

- What sort of operation do you need to use to retrieve status messages from each partner flow?

Verify your solution

Import the solution /files/mpdule12/connectors-mod12-various-systems-part2-solution.jar deployable archive file (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

Going further: Refactor the Mule application into separate Mule applications

After testing that all the JMS queues are working correctly, refactor the Mule applications into separate Mule projects, one for each retail partner, and a separate process-visa-gift-cards Mule application for the processVisaGiftCards flow. For each retail partner, debug the process-visa-gift-cards and the retail partner's Mule application at the same time.

Going further: Prototype a Mule domain project

Months have passed, and the bank would like to improve performance. Modify the solution to combine all the Mule applications into one Mule domain that will run inside the Mule Bank network, instead of collocated at each partner's network.

In this new refactoring, the distribution of Visa gift card CSV files over JMS queues will occur in the local data center, and then each partner Mule application will distribute the processed gift cards over remote connections:

- The Food-N-Savings Mule application will make a remote connection to the Food-N-Savings database.
- The Meals-N-Go Mule application will make a remote connection to the Meals-N-Go FTP server.
- The Online Retail Plus Mule application will output the final JMS message over a remote TCP connection to the Online Retail Plus Mule application.

Answer the following questions

- What connector could you use instead of JMS to try to increase system performance for Mule applications in the same domain?
- What features might be missing with this new connector?
- Aside from replacing the JMS connector with this new connector, would you need to modify anything else for the entire system to work?
- What are the advantages and disadvantages of using this other connector?

Going further: Experiment with other messaging connections

Replace the JMS connections with other messaging connections such as VM, Anypoint MQ, or Active MQ and compare behaviors and performance.

Answer the following questions

- Can different Mule applications in the default Mule domain share a VM queue if they are running on different Mule runtimes?
- Can different Mule applications in a non-default Mule domain share a VM queue if they are running on different Mule runtimes?
- What are the limitations and advantages of a VM queue vs. a JMS queue?

Note: You can learn more about scaling VM queues to multiple Mule runtimes in a cluster in the Anypoint Platform Operations: Customer-Hosted Runtimes course and the Anypoint Platform Operations: CloudHub course.

Going further: Use an API-led approach

Refactor your solution to be API-led by creating System APIs and Process APIs to augment and control the lower-level protocols. Write an Experience API for a consumer to receive a gift card on their mobile client.

- What are the advantages or disadvantages of using an API-led approach?
- How does IT handle and manage the development lifecycle of all of these projects without wrapping them in APIs or using an API-led approach?
- With an API-led approach, does IT need to be directly involved in the actual development and delivery of each project?
- How could IT encourage line of business teams to consume and reuse these APIs, as well as encourage them to share and distribute their own APIs with other teams?
- Without an API-led approach, how might you separate out business logic for who gets particular gift cards and what type of gift cards?
- What is the impact on your projects to change gift card distribution business logic versus if you had separate System, Process, and Experience APIs?
- How many projects and flows need to be changed to integrate all these gift card Mule applications with a new mobile app project, and how does this effort compare to what is needed if you have separate System, Process, and Experience APIs?