MuleSoft.U Development Fundamentals

DIY Exercises

Mule runtime 3.9
March 7, 2018

Table of Contents

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 3-1: Create an API specification with RAML

Time estimate: 1 hour

In this exercise, you create an API specification using RAML. You will:

- Appropriately specify GET and POST operations.
- Appropriately define URI parameters, query parameters, and headers.
- Restrict possible values using an enumeration.

Scenario

Your company needs to expose a customer accounts database as a system API for the rest of the organization. The first step is to create the RAML specification and post it to your company's private Exchange, so all stakeholders can review and provide feedback.

Create the API specification

In Design Center, create a RAML specification called Accounts API with the following requirements:

- The API has a resource called accounts.
- All client requests must include the required header Requester-ID. This header helps identify the person in the organization making account requests.
- The /accounts resource has a GET method with a required queryParameter named type, where type must be either personal or business. The type query parameter is used to return records that match that account's type.
- The GET method for the /accounts resource has two optional string queryParameters named name and country. The name query parameter value is used to filter and return all account records for a particular account owner. The country query parameter value is used to filter and return records matching the account owner's country of residence.
- The data returned from the GET method is an array of Account objects. To represent the response data, define and use an Account data type that contain the following fields: id, firstName, lastName, address, postal, country, miles, creationDate, and type. The creationDate field is of type datetime and miles is of type integer.
- The specification defines and uses a data type for the Account object that is stored in a separate RAML file. This data type has an example that can be specified in the main API RAML file or in the data type RAML file.

- The /accounts resource has a POST method that accepts an array of Account objects (with no id and no creationDate field) in JSON format and returns a JSON message: {"message": "Account created (but not really)"}. This Account object is represented as a new data type that does not have id or creationDate fields and has an example.
- Each method should have a description.
- Each method should have a custom 400 error status message example in JSON format.

Publish the API to Anypoint Exchange

Publish the API to Anypoint Exchange, test the mocked endpoint using the API console, and create a public portal for the API.

Verify your solution

Load the solution /files/module03/accounts-mod03-api-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 4-1: Implement a REST API using APIkit

Time estimate: 2 hours

In this exercise, you implement a REST API that has a RAML specification. You will:

- Use APIkit to create implementation flows.
- Add logic to APIkit auto-generated flows.
- Enhance the API and regenerate the flows using APIkit.

Scenario

A RAML specification for an Accounts API was published to Anypoint Exchange. You need to implement this API using Anypoint Studio. Account data should be retrieved from the flights_customers database and then transformed to match the Account data type defined in the Accounts API.

After you finish implementing the API, the requirements change, and you will have to extend the existing RAML specification and modify the API implementation.

Use Anypoint Studio to create an API implementation from an API

Create a new Anypoint Studio project that uses APIkit to generate an API implementation for your API solution from exercise 3-1 or use /files/module03/accounts-mod03-api.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

Retrieve account data

Add logic to return all customer data from the flights_customers database for GET /accounts using the following MySQL database credentials:

- Host: mudb.mulesoft-training.com
- Port: 3306
- User: mule
- Password: mule
- Database: training
- Table: flights_customers

Add logic to only return accounts that match a particular type (business or personal).

Note: For GET /accounts, you DO NOT need to implement the optional query parameters for name and country. However, if you want to implement these query parameters, you can use the Choice router to implement different query scenarios (which is covered in module 10 in the Fundamentals course). You also do not need to implement anything for the Request-ID header.

Answer the following question

- What is the data type and format of the database results?

Map results from the database to Account objects

Transform the database results to match the Account schema specified in the accounts-api.raml file. Here is an example of the desired transformed output:

```
[
  {
    "id": "100"
    "firstName": "Alice",
    "lastName": "Green",
    "address": "77 Geary St., San Francisco"
    "postal": "94108"
    "country": "USA",
    "creationDate": "2018-10-01T23:57:59Z+0:00",
    "accountType": "business",
    "miles": 1000
  }
]
```

Hint: *The DataWeave functions splitBy and joinBy can help you split the name into a first name and a last name. Assume that any word after the first space in the name is the last name.*

Enhance the API specification

Even though you have already started implementing the API implementation, requirements suddenly change; your key stakeholders now want the API to manage specific accounts, with the ability to retrieve and modify one account based on the account id.

Modify the Accounts API RAML specification with the following new requirements:

- The API has new nested resource /accounts/{id}, where id is a URI parameter that should match the accountID column of the flights_customers database. Unlike the /accounts resource, make the type query parameter optional for this resource.
- The /accounts/{id} resource has a GET method that returns an Account object in JSON format.

  Note: Don't duplicate the definition of the Account RAML type in /accounts/id.
- The /accounts/{id} resource has a PUT method that returns a JSON message: {"message": "account replaced (but not really)"}
- The /accounts/{id} resource has a PATCH method that returns a JSON message: {"message": "account modified (but not really)"}
- The /accounts/{id} resource has a DELETE method that returns a JSON message: {"message": "account deleted (but not really)"}
- Each method should have a description.
- Each method should have a custom 400 message error.

Update the API in Anypoint Exchange

Publish the new version of the API to Anypoint Exchange and update the public portal.

Regenerate the API implementation

Use APIkit to regenerate the implementation for your updated RAML file.

Note: There was a bug for certain versions of Anypoint Studio that caused duplicate flows to be created when an API was reimported from Design Center. If this happens to you, undo the step and then directly import the RAML files into Anypoint Studio and do not import them from Design Center.

Answer the following question

- What happens to the existing APIkit-generated private flows?

Implement the /accounts/{id} GET endpoint

After you regenerate the flows, add logic to implement the /accounts/{id} GET endpoint. In the flow, get all rows from the flights_accounts database where the accountID matches the supplied id URI parameter, then transform the result to an Accounts JSON object (as specified in the project's datatypes/account.raml file).

Verify your solution

Load the solution /files/module04/accounts-mod04-api-implementation-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 5-1: Deploy and secure an API

Time estimate: 2 hours

In this exercise, you deploy and secure an API implementation. You will:

- Add property placeholders to the Mule application.
- Deploy the Mule application to CloudHub.
- Create a proxy for the Mule application and apply an SLA based rate limiting policy.

Scenario

The Accounts API has been implemented and you are now ready to deploy it to Anypoint Platform. Before deploying, parameterize application properties (like the database credentials) to make it easier to deploy it to multiple environments. After deploying, secure it by creating an API proxy for it and applying an SLA based rate limiting policy.

Import the starting project

Use your solution from exercise 4-1 or import /files/module04/accounts-mod04-api-implementation-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio. This application requires values for the query parameter type and the header Requester-ID to be passed for the Mule application to work.

Add and use property placeholders

Change the TCP port specified for the HTTP Listener to use an http.port property set to 9091 (for the dev environment). Also add property placeholders for the database credentials.

Create API client test flows

Create test flows that proxy and test the accounts-api.raml endpoints, so that each API endpoint can be tested by making simple GET requests. For each flow, set example values for any required parameters and headers according to the accounts-api.raml specification in the project's src/main/api folder. In order to trigger each test flow, use one shared HTTP Listener configuration, listening on port 9091.

Note: This is an alternate way to test your API without using the mocking service or a separate REST client.

Deploy the Mule application to CloudHub and test the API

Deploy the Mule application to your Anypoint Platform account.
Test the flow endpoints (URIs) by making requests to http://appname.cloudhub.io:9091/{yourTestEndpoint}.

Answer the following questions

- What happens when you try to access your test flow endpoints on port 9091?
- What additional steps might you need to take to make your test endpoints available in CloudHub?
- What happens when you try to access your test flow endpoints at http://appname.cloudhub.io/{yourTestEndpoint}?
- What happens when you try to access your test flow endpoints at http://appname.cloudhub.io:8081/{yourTestEndpoint}?

Test the API directly

Try to access the various API endpoints (including the correct URI parameters, query parameters, and headers) at various baseUris including the following:

- http://appname.cloudhub.io/api
- http://appname.cloudhub.io:9090/api
- http://mule-worker-appname.cloudhub.io/api
- http://mule-worker-appname.cloudhub.io:8081/api
- http://mule-worker-appname.cloudhub.io:9090/api

Answer the following questions

- What happens when you try to access the API through the baseUrl http://appname.cloudhub.io?
- What happens when you try to access the API through the baseUrl at http://appname.cloudhub.io:9090/api?
- What happens when you access your API through the baseUrl http://mule-worker-appname.cloudhub.io/api?
- What happens when you access your API through the baseUrl http://mule-worker-appname.cloudhub.io:8081/api?
- What happens when you access your API through the baseUrl http://mule-worker-appname.cloudhub.io:9090/api?

Note: To learn more about using CloudHub, take the Anypoint Platform Operations: CloudHub course.

Apply a rate-limiting SLA based policy to your API implementation

Now that the Mule application is deployed and tested, secure the API by creating an API proxy for it and then apply a rate limiting SLA based policy with the following SLA levels:

- Free (with automatic approval)
- Professional (with manual approval)
- Enterprise (with manual approval)

Verify your solution

Load the solution /files/module05/accounts-mod05-deploy-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

Going further: Secure property values

Secure the database password by either making it a safely hidden property for CloudHub deployments or by using a secure properties placeholder for deployment to a customer-hosted Mule runtime.

Note: You can learn more about hidden and secure properties in the Anypoint Platform Operations: CloudHub course.

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 6-1: Troubleshoot and refactor a Mule application

Time estimate: 2 hours

In this exercise, you troubleshoot and refactor a Mule application. You will:

- Diagnose and repair common Mule message-related, RAML-related, and connector-related issues.
- Organize and refactor Mule flows.
- Use flow variables to enhance a Mule application.

Scenario

Your colleagues are in need of assistance; they have modified the Accounts API specification to include a new salesID field and broken the Accounts API implementation when trying to update it. Review their changes and then troubleshoot and fix the API implementation.

Import the starting project

Import /files/module06/accounts-mod06-debugging-starter.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio. This project has existing errors that you need to diagnose and fix.

Note: This Mule project file has errors specifically designed to help you learn to diagnose common errors. Do not use your previous solution.

Review the modified API in the starter project

Review the updated API located in the accounts-mod06-debugging-starter project.

Review the existing (broken) implementation

Review implementation.xml in the accounts-mod06-debugging-starter project.

Troubleshoot and fix the Mule application

Debug the Mule application and fix all the errors that prevent it from starting up.

Here is the expected output from the application where salesId is a concatenation of id, first name, last name, postal, and creationDate:

```
[
  {
    "salesId": "100Alice Green941082018-10-01T23:57:59Z+0:00"
    "id": "100"
    "firstName": "Alice",
    "lastName": "Green",
    "address": "77 Geary St., San Francisco"
    "postal": "94108"
    "country": "USA",
    "creationDate": "2018-10-01T23:57:59Z+0:00",
    "accountType": "business",
    "miles": 1000
  }
]
```

Make sure the APIkit Consoles view is able to load without any errors and that you can use it to submit a GET request to /accounts.

Refactor the application to use flow variables

Refactor the application so that the SQL statements refer to flow variables instead of directly to the query parameters in the database components.

Verify your solution

Load the solution /files/module06/accounts-mod06-debugging-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 7-1: Create a Mule domain project

Time estimate: 1.5 hours

In this exercise, you use a domain project to share connectors between Mule applications. You will:

- Create a Mule domain project.
- Move global configuration elements to be shared to the domain project.
- Add applications to the domain.

Scenario

Your operations team has decided to deploy the accounts and flights Mule applications to the same host, and all the HTTP Listeners must use port 8081. Different teams are working on the applications and it has been decided to combine these projects into one Mule domain instead of combining them into a single application. This will enable the different teams to share some of their global configurations elements, including that for the HTTP Listener. This setup will also allow you to add additional applications, such as other proxy applications, to the same domain without having to restart the domain or other Mule applications already running.

Note: For a video on how to create a domain project, see: https://blogs.mulesoft.com/dev/training-talks/how-to-create-a-mule-domain-project/.

Import the starting projects

Import /files/module07/flights-mod07.zip zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio and rename the project to flights. Import /files/module05/mod05-deploy-solution.zip and rename this project to accounts.

Run both applications

Create a Run configuration to enable you to run both Mule applications at the same time. Submit HTTP requests to both Mule applications and verify that only one Mule application is working correctly. Identify the type of TCP bind errors that occur.

Note: For documentation on how to deploy multiple Mule applications at the same time from Anypoint Studio, see https://docs.mulesoft.com/anypoint-studio/v/6/deploy-more-than-one-application.

Create a domain project and add both applications to this domain

Create a new domain project named reservations. Move the HTTP Listener global configuration elements from both the accounts and flights applications into the reservations domain project. If you want, you can combine other global configuration elements as well.

Test the applications

Run both applications and verify that there are no TCP bind errors and that both Mule applications are working.

Verify your solution

Load the solution /files/module07/reservations-domain-mod07-accounts-and-flights-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

Going further: Explore the relationship between applications in a domain

Create a third Mule application and add it to the reservations domain. Configure this third Mule application to use the shared HTTP Listener configuration. Deploy this Mule application and verify there are no bind errors and that the third Mule application works correctly.

Continue exploring with your domain project and Mule applications to answer the following questions:

- Can one Mule application call another flow that exists in another Mule application belonging to the same Mule domain?
- Do flow variables get transferred from one flow to another flow via a flow reference component?
- Do flow variables get transferred from one transport to another (HTTP, JMS, VM)?
- What is another method to transfer variables from one transport to another?

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 8-1: Orchestrate web services

Time estimate: 3 hours

In this exercise, you pull data from several different sources and combine them using DataWeave. You will:

- Retrieve data from a REST service.
- Retrieve data from a SOAP service.
- Use DataWeave to transform both data sources into a new data structure.
- Use a REST service to upload transformed data.

Scenario

The Finance department is using two APIs: Accounts and Transactions. Currently, their analysts have to pull information from both APIs separately and then combine them using a spreadsheet. The Finance department is requesting a process API that can combine accounts information with transaction information to reduce the manual work needed to combine the data in a spreadsheet.

To implement this process API, create a new Mule application that retrieves account information using the Accounts system API and then uses the Transactions system API to retrieve transaction data for each account. Each account can have zero or more transactions. Next, transfer these records to several different Finance directors using an existing experience API. To do this, you need to join the transaction data for each account with the other account record details to create a new data structure compatible with the existing experience API. The expected behavior of the new application is specified in the Transfer Accounts API: /files/module08/accounts-transactions-mod08-api.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

Review the Accounts API

Review the Accounts API you created in exercise 3-1 or the solution /files/module03/accounts-mod03-api.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

The Accounts API:

- Has an endpoint http://apdev-accounts-ws.cloudhub.io/api/accounts to request all accounts of a particular type (personal or business).
- Requires a query parameter named type set to personal or business.

- Requires a Requester-ID header.
- Has two other optional query parameters named country and name that can be used to filter the accounts to just a particular account owner's name or to the country of residence of the account holder.
- Returns a JSON array of objects.

Create a new application to retrieve data from the Accounts API

In Anypoint Studio, create a new Mule application that uses the Accounts API to retrieve accounts with the required and optional parameters specified in the Accounts API. The application should accept the type, name, and country query parameters. Set the Requester-ID header to a static value.

Hint: *You can use a conditional statement in MEL to help you dynamically evaluate variables:*

```
#[(value == empty)? "Value is empty" : "Value is not empty" ]
```

Retrieve data from the transaction web service

After you have the array of JSON objects returned from the Accounts system API, pass this list of account IDs to the GetTransactionsforCustomers operations of the Transaction SOAP web service: http://apdev-accounts-ws.cloudhub.io/api/transactions?wsdl. The web service expects the body to contain an XML request where the customerID should correspond to each accountID returned from the Accounts system API:

```
<?xml version='1.0' encoding='UTF-8'?>
<ns0:GetTransactionsforCustomers xmlns:ns0="http://training.mulesoft.com/">
  <customerID>4402</customerID>
  <customerID>4404</customerID>
  <customerID>4464</customerID>
</ns0:GetTransactionsforCustomers>
```

Hint: *You can use the following DataWeave script to return an array of account IDs from the payload resulting from a call to the Accounts system API. The code: default [] returns an array by default if payload.*id evaluates to null.*

```
{customerIDs: payload.*id default []}
```

Submit a request to the Mule application and verify transactions are returned from the SOAP query.

Here are a few example transactions results:

```
    {
      "amount": "9717.0",
      "customerRef": "4412",
      "flightID": "UNITEDER09452015-06-11",
      "region": "EU",
      "transactionID": "181948488"
    },
    {
      "amount": "1030.0",
      "customerRef": "4985",
      "flightID": "DELTAA1B3D42015-02-12",
      "region": "US",
      "transactionID": "181948625"
    }
```

Combine accounts and transactions data

After retrieving data from accounts and transactions, combine them together in the simplest way:

```
{
  "accounts": Array of Account Objects
  "transactions": Array of all Transaction Objects for all accounts
}
```

Note: In a later exercise, you will create a more complex and normalized data structure.

Hint: *A Transform Message component can create several different transformations at a time and output the results to flow variables as well as properties. To add a new output target for the Transform Message component, click the plus icon to the left of the pencil icon in the Script Editor.*

Transform accounts-transaction data

Use the following DataWeave code to normalize the accounts-transaction data structure you just created to join the transaction details to each account. You can also find this code in the text file /files/moule08/dwTransform.txt (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

```
%dw 1.0
%output application/java
%var directorIDs = ["JKLS483S","FJSA48JD","NMA4FJ9K"]

//Combines Accounts and Transactions by the Account ID. Assigns each account //to a
director
%function consolidateAccountsTrans (payload)
  payload.accounts map ( (account, index) ->
  using (id = account.id as :string)
  (
    account ++
    {
      transactions: payload.transactions filter ($.customerRef == id)
    } ++
    {
      assignedDirector: directorIDs[index mod sizeOf directorIDs]
    }
  )
)
---
using (data = consolidateAccountsTrans(payload))
(data groupBy $.assignedDirector)
```

Transfer assigned accounts to a Finance Director

An experience API has been created that can be used to transfer records to the Finance directors. It can be accessed at the following endpoint:
http://apdev-accounts-ws.cloudhub.io/api/accounts_transactions

Hint: *This web service is slow and takes more than 10 seconds to return a response, which causes the HTTP request to timeout with an error. For information on how to fix this error, see (https://docs.mulesoft.com/mule-user-guide/v/3.9/http-connector-reference.*

Return the number of records processed

Modify the Mule application to return a final payload with a message confirming the number of account records processed.

Verify your solution

Load the solution /files/module08/accounts-transactions-mod08-joining-data-solution.zip zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 9-1: Add exception strategies

Time estimate: 1 hour

In this exercise, you handle errors in a Mule application. You will:

- Extend the Mapping Exception Strategy for APIkit.
- Add global default exception strategies and local exception strategies.

Scenario

Make the account-transactions solution from exercise 8-1 more robust by adding exception handling and investigate how exceptions are handled when they are thrown in referenced flows and subflows.

Import the starting project

Use your own solution from exercise 8-1 or import /files/module08/accounts-transactions-mod08-joining-data-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio.

Review the flows in the starting project

Review the structure of the transactions_accounts.xml file:

- The Mule application has only one endpoint and expects a query parameter called type with a value of either personal or business and a header called Requester-ID that can have any value.
- Requests are received by the processAccountsTransactionsFlow.
- This flow has a flow reference to the getAccounts flow.
- The getAccounts flow has a flow reference to the getTransactionsForGivenAccountsFlow subflow.

Add a new exception to APIkit's Mapping Exception Strategy

Extend APIkit's exception handling capabilities by adding a new exception mapping to the Mapping Exception Strategy. Configure the Exception Mapping to map any exception that extends java.lang.Exception to the 500 status code.

Note: Because this is at the bottom of the Mapping Exception Strategy, this new Exception Mapping will only match any exception types not already caught by the earlier exception mapping types.

Create a global Catch Exception Strategy

Create a global.xml file and add a global Catch Exception Strategy to handle some common exception types including:

- DataWeave errors
- Java OutofMemoryError
- Java IO errors
- Java InterruptedException
- All other exceptions

Add an exception handling strategy to a flow

In transactions-accounts.xml, add an exception handling strategy to processAccountsTransactionsFlow to handle:

- Any DataWeave generated exceptions of type com.mulesoft.weave.*
- All other exceptions

Add a Validator component and test

Add a Validator component that throws a new Exception object at the beginning of getAccounts flow, or explicitly throw an exception with a Groovy component.

Note: Validator components are covered in Module 10.

Debug and step through the flows and observe how the exception is handled.

Answer the following question

- Which exception strategy handles the exception?

Handle exceptions thrown in a subflow instead of a flow

Copy the exception strategy in the processAccountsTransactionsFlow and add it to the getAccounts flow. Move the Validation component to the beginning of the getTransactionsForGivenAccounts subflow. Run or debug the Mule application again and observe how the exception is handled.

Answer the following questions

- Which exception strategy processes the error thrown by getAccounts flow?
- How does this behavior compare to when the exception was thrown in a flow instead of a subflow?

Verify your solution

Load the solution /files/module09/accounts-transactions-mod09-exceptions-part1-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

Exercise 9-2: Add context specific exception handling

Time estimate: 1 hour

In this exercise, you handle additional errors in a Mule application. You will add reference context specific global exception strategies for different connector types.

Scenario

Continue to make the account-transactions solution from exercise 8-1 more robust by adding more specific global exception handling strategies that can ultimately be reused across projects.

Import the starting project

Use your own solution from exercise 9-1 or import the file /files/module09/accounts-transactions-mod09-exceptions-part1-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio.

Add exception handling to handle common REST related exceptions

In the global.xml file, add a Choice Exception Strategy to handle REST generated exceptions including:

- org.mule.module.http.*
- java.net.UnknownHostException
- org.mule.api.*

Use the Reference Exception Strategy

For the processAccountsTransactionsFlow and getAccounts flows, replace the last catch all Choice Exception Strategy with a Reference Exception Strategy that references the global REST exception strategy you just created.

Add exception handling to handle common SOAP related exceptions

In the global.xml file, add a Choice Exception Strategy for SOAP generated exceptions caused by the Web Service Consumer including:

- org.mule.module.http.*
- org.mule.module.ws.consumer.*
- org.apache.cxf.*

Experiment with how exceptions are handled in different parts of an application

Move the exception throwing component (a Validator component or a Groovy script) to different parts of your Mule application. For each change, debug the Mule application and step through the Mule application and notice how exceptions are handled in child and parent flows and subflows, and how exceptions are handled if they are not handled in the flow or subflow that throws the exception.

Answer the following questions

- When a child's flow exception handler does not catch the exception, which exception handler then catches the exception: the parent flow, the referenced exception strategy, or the default exception strategy?
- When a referenced exception strategy does not handle the exception, is the default exception strategy called?

Verify your solution

Load the solution /files/module09/accounts-transactions-mod09-exceptions-part2-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

Going further: Add additional exception handling

If you want more practice, add exception handling to the flights application: apdev-flights-ws-WT10.4.zip from the *Fundamentals* course.

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 10-1: Use filters to stop message processing

Time estimate: 2 hours

In this exercise, you control message flow using filters. You will:

- Filter out favicon requests.
- Filter out invalid HTTP Request methods.
- Combine filters.

Scenario

Enhance the flights Mule application created in the Fundamentals class by adding filters to filter unwanted requests to invalid endpoints and to filter out favicon requests.

Import the starting project

Import the flights application /files/module10/apdev-flights-ws-WT10.4.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

Add a filter to filter out favicon requests

Add a filter to the mua-flights-api-main flow in interface.xml to filter out favicon requests. You can use a message-property-filter:

```
<not-filter name="not-favicon-request" doc:name="Not">
   <message-property-filter
      pattern="http.request.path=/api/favicon.ico" caseSensitive="true"
      scope="inbound"/>
</not-filter>
```

or an expression-filter:

```
<expression-filter
expression="#[message.inboundProperties.'http.request.uri' != '/favicon.ico']"
doc:name="Filter favicon"/>
```

Test the application

Test your solution by making a request to http://localhost:8081/api/flights?airline=american&code=LAX.

Log filtered messages

Move the favicon filter to the global.xml file to make it a global filter. Call this global filter from a Message filter in the interface.xml file's mua-flights-api-main flow before the APIkit router. In the Message filter, configure and code a compensating flow. A compensating flow is defined as a flow that a filter calls when the filter condition fails. In the compensating flow, set the http.status to 404, set a response payload, and log the issue.

Note: This technique allows you to reuse the favicon.ico filter for multiple flows.

Filter out invalid HTTP Request methods

Create a global Message filter to test if the http.method is GET. Call this global filter from a Message filter in the interface.xml file's mua-flights-api-main flow. If the filter fails, call a compensation flow to set the http.status to 404, set a response payload, and log the issue.

For the Logger components in all the filter compensation flows, set a log category named com.mulesoft.training.flights.api and the log level to WARN to allow these log messages to be ignored in production.

Combine filters

In the interface.xml file's mua-flights-api-main flow, add a Message filter to test if the message.inboundProperites.'http.uri.params'.flights resource is not empty. If the filter fails, call a compensation flow named invalidRestResource. The compensation flow should set the http.status to 404, set a response payload, and log the issue.

After testing this filter is working, use an And filter to combine the method type filter and this new non-empty resource filter together. In the interface.xml file, combine the two filters into one Message filter using this new global And filter.

Answer the following question

- Which Filter types can throw exceptions, and when can you specify the type of exception that is thrown?

Going further: Log and handle empty flights results

Modify the getAllAirlineFlightsFlow's Scatter-Gather filters so they each use a Message filter to call a compensation flow if the payload is not an ArrayList type. Have this compensating flow set the payload to an error JSON object to indicate which airline is not returning any results. Then modify the final

getFlightsFlow's final Transform Message component's DataWeave expression to create a more complex response JSON object that separates out all flights with available seats and the error responses.

For example, create and populate a new schema such as:

```
"flightsWithSeatsResponse": {
    "flights": {
      flight1: { … }
    },
    "errors": {
      "error1": {
        "error": "no flights were found for Delta"
      }
    }
  }
}
```

Verify your solution

Load the solution /files/module10/apdev-flights-mod10-filters-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

Exercise 10-2: Use validators to validate response schemas

Time estimate: 1.5 hours

In this exercise, you control message flow using validators. You will:

- Validate REST response schemas.
- Validate a SOAP response schema.

Scenario

Further enhance the flights Mule application by adding logic to validate the responses from each of the airline web services including JSON responses from the United and American REST services and XML responses from the Delta SOAP service.

Import the starting project

Use your own solution from exercise 10-1 or import /files/module10/apdev-flights-mod10-filters-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio.

Validate the United REST response schema

In the implementation.xml file's getUnitedFlightsFlow flow, add a Validate JSON Schema message processor to validate the REST response from the United REST Request HTTP requester. Configure the Validate JSON Schema message processor with a JSON schema that matches the JSON response from the United REST service.

Hint: *You can easily build JSON schemas from examples using https://jsonschema.net/#/. You may also need to add a Byte Array to String transformer to convert the response stream to a text string.*

The Validate JSON Schema message processor serializes the JSON payload to a String, so you need to modify the last Transform Message processor to properly read the JSON String:

```
%dw 1.0
%output application/java
%var validInput = read(payload, "application/json")
---
validInput map ((payload01, indexOfPayload01) ->
  //Do rest of transformations
)
```

Test the application

Test your solution by making a request to either of these URLs:
http://localhost:8081/flights?airline=united&code=SFO
http://localhost:8081/api/flights?airline=united&code=SFO

Note: In the current solution from Fundamentals, the interface.xml file does not yet call the implementation.xml file's flow. You can either use the direct URL's to the implementation.xml file's HTTP listeners http://localhost:8081/flights, or you can replace the Set Payload transformer in the interface.xml file's get:/flights:mua-flights-api-config with a flow-reference to getFlightsFlow, so you can test requests through the APIkit router by making GET requests to http://localhost:8081/api/flights.

Change the validation rule so the response fails validation

After you have the Validate JSON Schema message processor working correctly, edit the American XSD file to introduce a validation rule that is not met by the United REST API's response. For example, change the JSON Schema file to make at least one key required, and change the key name to something different than what is actually returned by the United REST service response:

```
      },
       "required": [
         "code",
         "price-BAD",
         "origin",
         "destination",
         "departureDate",
         "planeType",
         "airlineName",
         "emptySeats"
       ]
     }
   }
},
```

After you verify the schema is being validated, change the JSON Schema back to the correct JSON Schema file, so that United responses are correctly validated.

Validate the American REST response schema

Create and add a JSON schema file to match the response from the getAmericanFlightsFlow's American REST Request HTTP request. Use your JSON schema in a Validate JSON Schema message processor.

As before, the Validate JSON Schema message processor serializes the JSON payload to a String, so you need to modify the last Transform Message processor to properly read the JSON String into the DataWeave transformation expression:

```
%dw 1.0
%output application/java
%var validInput = read(payload, "application/json")
---
validInput.flights map ((flight , indexOfFlight) -> {
      //Do rest of transformations
)
```

Change the validation rule so the response fails validation

After you have the Validate JSON Schema message processor working correctly, edit the American XSD file to introduce a validation rule that is not met by the American REST API's response.

For example, change the JSON Schema file to make at least one key required, and change the key name to something different than what is actually returned by the American REST service response:

```
   },
    "required": [
      "ID",
      "code",
      "price-BAD",
      "departureDate",
      "origin",
      "destination",
      "emptySeats",
      "plane"
    ]
  }
}
```

Verify that the JSON Schema message processor throws an exception indicating the REST response does not validate against the XSD file. Test your solution by making a request to either of these URLs:
http://localhost:8081/flights?airline=american&code=LAX
http://localhost:8081/api/flights?airline=american&code=LAX

Validate the Delta SOAP response schema

Add a Schema Validation Filter in the getDeltaFlightsFlow flow to validate the response XML payload from the Delta SOAP Web Service Consumer. To do this, add an XSD file to the project using a Delta XML response from a breakpoint just after the Delta response, or copy the <xs:schema> element from the Delta SOAP service WSDL file, which is available at http://mu.mulesoft-training.com/essentials/delta?wsdl.

Hint: *Wrap the filter in a Message filter so you can call a compensation flow and/or throw an exception.*

Test the application

Test your solution by sending a request to either of these URLs:
http://localhost:8081/flights?airline=delta&code=LAX
http://localhost:8081/api/flights?airline=delta&code=LAX

After the solution tests correctly, modify the XSD file by changing the name of one of the required elements. For example, change price to price-BAD.

After you verify the schema is being validated, change the JSON Schema back to the correct JSON Schema file, so that United responses are correctly validated.

Answer the following question

- Which Validator types can throw exceptions, and when can you specify the type of exception that is thrown?

Verify your solution

Load the solution /files/module10/apdev-flights-mod10-validations-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

Going further: Validate the Delta SOAP request schema

Add another Schema Validation Filter before the Delta Web Consumer message processor. Wrap this filter in a Message filter so it can call a compensation flow to handle the filtered message. Replace the Delta schema file with one that will cause the schema validation to fail with a response from the Delta web service. Modify the Message filter compensation flows to set http.status to 400 and set an error message.

Test that requests sent with the query parameter airline=delta return status 400 with an error message string.

Test that requests made with the query parameter airline=all also return the Delta error message string, which also fails the filter in the Scatter-Gather component's Delta flights branch. Modify this branch to set a flowVar with the airline name for each branch, so that the common compensation flow for each branch of the Scatter-Gather component sets the payload to an error string with the current calling airline name.

Note: This allows you to reuse the same flow to return different error strings in the result array from the Scatter-Gather component.

In the getFlightsFlow flow's final Transform Message component, modify the DataWeave expression to gather up the successful flight results with available seats, plus any error responses from other airlines.

If you want more practice, add in filters and validators for the Accounts Transactions application: /files/module08/account-transactions-mod08-joining-data-solution.zip zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

MuleSOft.U Development Fundamentals - DIY Exercises

Exercise 11-1: Write DataWeave transformations

Time estimate: 2.5 hours

In this exercise, you combine multiple data sources using DataWeave. You will:

- Use Map operator to iterate through arrays.
- Use when/otherwise constructs to change output based on conditions.
- Format numbers and dates.
- Use sort and filter operators to modify data structures.
- Use the read operator to translate MIME types.

Scenario

You need to combine several different data sources and return a list of flights along with the transactions and account details related for each flight. Start with the results from the three online airline flights data sources for American, Delta, and United, and then join that data with results from two additional online data sources:

- A Transactions web service that returns a list of all transactions related to a flight: http://apdev-accounts-ws.cloudhub.io/api/transactions?wsdl
- An Accounts API that returns a list of account details associated with flight transactions: http://apdev-accounts-ws.cloudhub.io/api/accounts

The application should return a list of flight information along with transaction information and account details for each flight, organized into a collection of structured objects. A sample final output can be found in the file: /files/module11/sample_output.xml (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

Import the starting project

Import /files/module11/flights-mod11-dataweave-starter.zip zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio. This is an enhanced version of the flights Mule application solution from Module 11 of the *Fundamentals* course and has an updated RAML file.

Retrieve transactions

Create a new flow that makes a call to the Transactions web service. The web service takes a list of flight IDs in the SOAP request, then returns a list of transactions associated with each flight id. Here is the configuration information for the web service:

```
webservice.wsdl = http://apdev-accounts-ws.cloudhub.io/api/transactions?wsdl
webservice.service = TransactionServiceService
webservice.port = TransactionServicePort
webservice.address = http://apdev-accounts-ws.cloudhub.io/api/transactions
webservice.operation = GetTransactionsforFlights
```

Here is a sample request to the web service where each flightID in the request XML is composed by combining the airlineName, flightCode, and departureDate keys from each flight response:

```
<?xml version='1.0' encoding='UTF-8'?>
<ns0:GetTransactionsforFlights xmlns:ns0="http://training.mulesoft.com/">
  <flightID>UNITEDER38SD2015-03-20</flightID>
  <flightID>UNITEDER39RK2015-09-11</flightID>
  <flightID>UNITEDER39RJ2015-02-12</flightID>
</ns0:GetTransactionsforFlights>
```

Here is a sample response from the web service:

```
<?xml version='1.0' encoding='UTF-8'?>
<ns2:GetTransactionsforFlightsResponse xmlns:ns2="http://training.mulesoft.com/">
  <transaction>
    <amount>3177.0</amount>
    <customerRef>4456</customerRef>
    <flightID>UNITEDER39RK2015-09-11</flightID>
    <region>US</region>
    <transactionID>181948797</transactionID>
  </transaction>
  <transaction>
    <amount>6032.0</amount>
    <customerRef>4968</customerRef>
    <flightID>UNITEDER38SD2015-03-20</flightID>
    <region>US</region>
    <transactionID>181948717</transactionID>
  </transaction>
</ns2:GetTransactionsforFlightsResponse>
```

Store the transactions results in a flow variable so that the Transactions web service response can be combined with the flights response.

Note: For an additional optional challenge, use Message Enrichers to set this variable without modifying the current message payload.

Retrieve account data for each flight transaction

Create a flow to make a request to the Accounts API to retrieve account details for each transaction record. The API has a POST api/accountList endpoint that expects a JSON-formatted array of account ids formatted as strings and returns a string that represents an array of Account objects in JSON format.

Here is the configuration information for the API:

```
accounts.host = apdev-accounts-ws.cloudhub.io
accounts.port = 80
accounts.basePath = /api
```

To make a request to the Accounts API, you need to provide a valid input to the POST operation and set the Requester-API header. The customerRef fields from the previous transactions SOAP API provide valid accountID values for the Accounts API requests. For example, you can write DataWeave expressions to extract these customerRef fields from the previous transactions response example:

```
[
  "4456",
  "4968"
]
```

Note: For ease of implementation, the Accounts API is not RESTful due to the POST api/accountList endpoint. The POST method is used to make it easy to include the array of account ids in the request rather than asking you to serialize the array with escaped characters into a GET request.

After the Accounts API returns a valid response, save the response payload to a flow variable so it can be joined with the flights and transactions responses.

***Hint**: To transform a string into a target object, you can use the DataWeave function:*

```
read(data, "application/json")
```

Note: For an additional optional challenge, use Message Enrichers to set this variable without modifying the current message payload.

Join all data into a nested object structure

The final step is to combine flights, transactions, and accounts data. The application contains starter code for the required DataWeave transformation you need to write. The key transformation is located in the Combine all payloads and convert to XML Transform Message component in the getFlightsFlow.

Complete the join in two steps. First, join each flight object with the transaction with the same flightID value, so it creates a nested object:

```
airlineName: "UNITED",
flightCode: "ER39RK",
departureDate:" 2015-Mar-20",
...
transactions:
{
  idCustomer" 4564,
  transID: "181948544",
  amount: "735.0",
  ...
}
```

Next, join the accounts details (account name and account type) with each transaction:

```
airlineName: "UNITED",
flightCode: "ER39RK",
departureDate:" 2015-Mar-20",
...
  transactions:
    {
      idCustomer" 4564,
      transID: "181948544",
      amount: "735.0",
      nameCustomer: "Max Plank",
      type: "business"
      ...
    }
```

Output combined data as XML data

Transform the data to valid XML and add a flow reference to getFlightsFlow inside the get:/flightstrans:mua-flights-transactions-api-config flow. Move or add some of the response data fields as XML attributes instead of XML elements. Here is an example final XML response:

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
    <flight flightCode="ER38sd">
        <airline>United</airline>
        <departureDate>2015-Mar-20</departureDate>
        <emptySeats>0</emptySeats>
        <fromAirportCode>MUA</fromAirportCode>
        <planeType>Boeing 737</planeType>
        <price>400.0</price>
        <toAirportCode>SFO</toAirportCode>
        <transactions>
            <transaction transID="181948544" amount="735.0">
                <idCustomer>4564</idCustomer>
                <nameCustomer>Max Plack</nameCustomer>
                <type>business</type>
                <amountUSFormatted>735.00</amountUSFormatted>
            </transaction>
            <transaction transID="181948717" amount="6032.0">
                <idCustomer>4968</idCustomer>
                <nameCustomer>Nick The Shoemaker</nameCustomer>
                <type>personal</type>
                <amountUSFormatted>6,032.00</amountUSFormatted>
            </transaction>
        </transactions>
    </flight>
</flights>
```

Verify your solution

Load the solution /files/module11/flights-mod11-dataweave-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

MuleSoft.U Development: Fundamentals - DIY Exercises

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

Time estimate: 3 hours

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 brought on to a special 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 will be stored in a separate CSV file for that partner. You are tasked with developing a network of 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 different technology to receive their Visa gift card lists:

- 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 layer 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, such as JMS.

Import the starting project

Import /files/module12/connectors-mod12-various-systems-starter.zip in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio and name it connect-various-systems. This starter project contains all the libraries and helper files you need to build your application.

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 applications to customer-hosted Mule runtimes, take the Anypoint Platform Operations: Customer-Hosted Runtimes class.

Create a flow to insert files into 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: 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.

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
830804083,243796,70,Food and Savings
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 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 message to the processing flow in foodnsavings.xml using a Flow Reference component.

Note: In the next exercise you will refactor each Mule configuration file by replacing each flow reference with a JMS messaging queue.

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

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 embeds an in-memory FTP server that automatically launches when you run the project.

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 Meals-N-Go CSV format and then writing the 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 message to the processing flow in mealsngo.xml using a Flow Reference component.

Note: In the next exercise you will refactor each Mule configuration file by replacing each flow reference with a JMS messaging queue.

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 the process-visa-giftcards application and must be in milliseconds since 1970 January 1 (Unix Epoch).

Create a flow to send JMS messages to the Online Retail Plus partner

The Online Retail Plus partner has set up an JMS server to receive gift card information. You can use their server at tcp://jms.mulesoft-training.com:61616 or you can use an in-memory server at vm://localhost.

Create a new Mule configuration file named oretailplus.xml. Create a flow to process the Visa gift card CSV file by transforming the CSV file to the JMS message format supported by the Online Retail Plus JMS queue.

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

Note: In the next exercise you will refactor each Mule configuration file by replacing each flow reference with a JMS messaging queue.

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

Visa CSV file	Online Retail Plus JMS Map 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"

Send the processed payload as a byte array to the JMS server to the onlineretailplus.output queue.

Finally write out a copy of the JMS message 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.

Verify your solution

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

MuleSoft.U Development Fundamentals - DIY Exercises

Exercise 12-2: Route messages using JMS queues

Time estimate: 2 hours

In this exercise, you continue to build the Visa gift card distribution applications. You will conditionally process and route Mule messages 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.zip zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio. This project contains all the libraries and helper files you need to build your application.

Route the messages using JMS queues

Each partner should use a different JMS queue. Replace all the flow-references in the processVisaGiftCards flow with different JMS message queues, one for each partner type: foodnsavings.visa.input, mealsngo.visa.input, and onlineretailplus.visa.input.

Note: Set each queue name using a property placeholder, so it can be changed later by ops staff 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 report in the process-visa-gift-cards application

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

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

Verify your solution

Load the solution /files/mpdule12/connectors-mod12-various-systems-part2-solution.zip (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 application will distribute the processed gift cards over remote connections:

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

Answer the following questions:

- What connector could you use instead of JMS to try to increase system performance for 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 transports

Replace the JMS transports with other messaging transports such as VM 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 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, vs. if you had separate system layer, process layer, and experience layer APIs?
- How many projects and flows need to be changed to integrate all these gift card applications with a new mobile app project, and how does this effort compare to what is needed if you have separate system layer, process layer, and experience layer APIs?

Exercise 13-1: Create a batch job to process records

Time estimate: 2 hours

In this exercise, you create a batch system. You will:

- Control batch-step processing using batch filters.
- Exchange data between batch-steps using record variables.
- Trigger batch processing using a Poll scope.
- Use watermarks to avoid duplicate processing.
- Improve performance by using batch commits.
- Improve performance and share data with other Mule applications using VM queues.

Scenario

The Finance department needs to audit certain transactions and needs a Mule application to consistently pull data from a database and write out these transactions as CSV files to a server.

To meet compliance standards, a CSV file can have no more than 50 records and the Mule application must be deployed on a private server where the Mule application will share the same Mule domain with other financial compliance Mule applications. You do not, however, need to create a new Mule domain project yourself; another developer will be responsible for deploying your project under an existing Mule domain.

Create a new project that retrieves transactions from the database using batch

Create a new Mule application that retrieves data from the flights_transactions table in the database using the following information:

- Host: mudb.mulesoft-training.com
- Port: 3306
- User: mule
- Password: mule
- Database: training
- Table: flights_transactions

Schedule the main flow to automatically run every 5 seconds and to use the watermark feature of the Poll scope to only retrieve new database records based on the value of the primary key field transactionID.

Send each transaction record to a VM queue for conditional testing

Send each transaction record to a VM queue for the other financial compliance Mule applications in the Mule domain to process.

Filter out only transactions that need auditing

There is another Mule application already created (you do not have to create it) listening on the VM queue named validate. It expects one record and returns the value true or false, where true indicates that the record needs to be audited.

Use batch filtering and record variables to keep track of your records throughout each batch step. Set the Source and Target of this batch step to track the response (true or false) from the VM queue.

***Hint**: For test development, you can create a flow that listens on the validate queue path and arbitrarily return true or false for each record processed.*

Write out transactions as CSV files

In a second batch step, use an Accept Expression to only process this second batch step if the previous VM queue response was true. Inside this batch step, transform the database results to a CSV file and save this CSV file to this Mule application's file system. Use a property placeholder for the file location so the file location can be modified by Ops staff at deployment time. Add a Batch Commit scope so no more than 50 records at a time are written to each CSV file.

Test your solution

Debug your solution. Step through several polling steps and verify some queries return true from the JMS queue and are processed by the second batch step, but other database queries return false and skip the second batch step. Also verify the output CSV files contain at most 50 records each.

Verify your solution

Load the solution /files/module13/audit-mod13-file-write-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.

Going further: Handle exceptions

Add logic to the first batch step to throw exceptions.

- Call a flow at the beginning of the first batch step and add message processors to this flow that sometimes throws an exception, but not for every record.
- Experiment with what happens when you handled the exception in the flow vs. if you don't handle the exception in the flow.
- Add a third batch step with an ONLY_FAILURES Accept Policy to report failed messages to a dead letter JMS queue for failed batch steps.
- In the Batch scope's general configurations, change the Max Failed Records to 1 and observe the behavior of subsequent batch records after a record throws an exception. Change this value to 2 and observe any changes in behavior.
- In the Batch scope's general configurations, change the Scheduling Strategies options to ROUND_ROBIN and observe the behavior, and compare it with the default ORDERED_SEQUENTIAL option's behavior.
- Look at the logs for the On Complete phase to see how many times the same exception is reported for each record of the batch job.
- In the first batch step's referenced flow, add a Choice router and a sleep timer that sleeps for a minute. Add logic to the Choice router to only call the sleep timer if the transactionID ends with 6. Observe if later records in the batch job can skip ahead while some records are paused by the sleep timer.

Note: For info about handling batch exceptions, see: https://blogs.mulesoft.com/dev/mule-dev/handle-errors-batch-job/