



## Module 3: Consuming Web Services

### Goal

The screenshot displays the MuleSoft Studio interface with two HTTP endpoints and their associated flows.

**Top Endpoint: localhost:8081/united**

This endpoint is configured with a GET method and a path of `/united`. The flow is named `getUnitedFlightsFlow`. It includes an `HTTP` connector for the request and an `HTTP` connector for the response. The response body is a JSON array of flight objects. The flow also includes an `Error handling` section.

```

{
  "flights": [
    {
      "airlineName": "United",
      "price": 400,
      "departureDate": "2015/03/20",
      "planeType": "Boeing 737",
      "origin": "MDA",
      "code": "E838sd",
      "emptySeats": 0,
      "destination": "SFO"
    },
    {
      "airlineName": "United",
      "price": 345.99,
      "departureDate": "2015/02/11",
      "planeType": "737",
      "origin": "MDA",
      "code": "E845LE",
      "emptySeats": 12,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 346,
      "departureDate": "2015/04/11",
      "planeType": "737",
      "origin": "MDA",
      "code": "E845jd",
      "emptySeats": 12,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 423,
      "departureDate": "2015/04/11",
      "planeType": "737",
      "origin": "MDA",
      "code": "E89945",
      "emptySeats": 0,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 845,
      "departureDate": "2015/07/11",
      "planeType": "727",
      "origin": "MDA",
      "code": "E89fjo",
      "emptySeats": 12,
      "destination": "CLE"
    },
    {
      "airlineName": "United",
      "price": 245,
      "departureDate": "2015/08/11",
      "planeType": "747",
      "origin": "MDA",
      "code": "E83KSD",
      "emptySeats": 13,
      "destination": "CLE"
    }
  ]
}

```

**Bottom Endpoint: localhost:8081/delta**

This endpoint is configured with a GET method and a path of `/delta`. The flow is named `getDeltaFlightsFlow`. It includes an `HTTP` connector for the request and a `Delta SOAP Request` connector for the response. The response body is an XML document. The flow also includes an `Error handling` section.

```

<?xml version='1.0'?>
<root>
  <airlineName>Delta</airlineName>
  <code>E838sd</code>
  <departureDate>2015/03/20</departureDate>
  <emptySeats>0</emptySeats>
  <origin>MDA</origin>
  <planeType>Boeing 737</planeType>
  <price>400.00</price>
  <destination>SFO</destination>
  </root>

```

The screenshot also shows a `Logger` connector in the `getDeltaFlightsFlow` flow, which is used to log the response data.

2 | All contents Copyright © 2015, MuleSoft, Inc.

## Objectives

- In this module, you will learn:
  - About RESTful and SOAP based web services
  - What RAML is and how it can be used
  - To consume RESTful web services with and without RAML definitions
  - To consume SOAP web services

## Understanding web services

## SOAP web services

- Traditional way to expose web services
  - Can bridge protocols, application platforms, programming languages and hardware architectures
- Use SOAP protocol to define a message architecture and message formats
  - Simple Object Access Protocol (SOAP)
- Self-descriptive
  - WSDL (Web Service Description Language)

## SOAP protocol

- Based on XML (SOAP envelope uses XML)
- Defines operations, arguments, data types, and more
- Requires tooling to publish and consume
- Usually over HTTP, but can use any protocol
- SOAP request is sent as the body of a HTTP POST

## SOAP message example

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap=
"http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
</soap:Header>
  <soap:Body>
    <m:GetStockPrice
      xmlns:m="http://www.fooCo.com/stockPrice">
      <m:StockName>FOOC</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

7 | All contents Copyright © 2015, MuleSoft Inc.



## RESTful web services

- Second generation web services
- REST stands for Representational State Transfer
  - An architectural style where clients and servers exchange representations of resources using standardized HTTP protocol
  - The resources are acted upon by using a set of simple, well-defined operations: PUT, GET, POST, DELETE
- Lightweight without a lot of extra XML markup
- Human readable results (usually JSON or XML)
- Easy to build, no toolkits required

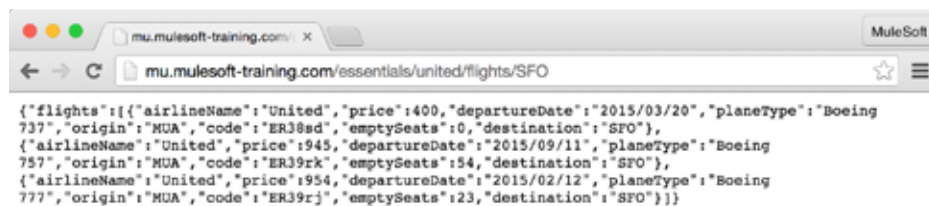
8 | All contents Copyright © 2015, MuleSoft Inc.



## RESTful web services

- Data and resources are accessed using URIs
- Resources are manipulated using a fixed set of operations
  - GET retrieves the current state of a resource in some representation (usually JSON or XML)
  - POST creates a new resource
  - PUT transfers a new state onto a resource
  - DELETE deletes a resource

## RESTful web service example



```
{
  "flights": [
    {
      "airlineName": "United",
      "price": 400,
      "departureDate": "2015/03/20",
      "planeType": "Boeing 737",
      "origin": "MUA",
      "code": "ER38sd",
      "emptySeats": 0,
      "destination": "SFO"
    },
    {
      "airlineName": "United",
      "price": 945,
      "departureDate": "2015/09/11",
      "planeType": "Boeing 757",
      "origin": "MUA",
      "code": "ER39rk",
      "emptySeats": 54,
      "destination": "SFO"
    },
    {
      "airlineName": "United",
      "price": 954,
      "departureDate": "2015/02/12",
      "planeType": "Boeing 777",
      "origin": "MUA",
      "code": "ER39rj",
      "emptySeats": 23,
      "destination": "SFO"
    }
  ]
}
```

# Consuming RESTful web services

11 | All contents Copyright © 2015, MuleSoft Inc.

## Consuming RESTful web services

- First check and see if there is an existing Anypoint Connector to connect to the service provider
- If there is not, use the HTTP Request connector
  - For the connector
    - Specify host, port, and optionally, a base path
  - For the endpoint
    - Specify path and method



12 | All contents C



## Walkthrough 3-1: Consume a RESTful web service

- Create a second flow and rename flows
- Add an HTTP Listener connector endpoint to receive requests at <http://localhost:8081/united>
- Add an HTTP Request connector endpoint to consume a RESTful web service for United flight data

The screenshot shows a web browser window with the address bar at `localhost:8081/united`. The page displays a JSON array of flight data. To the right, a MuleSoft flow diagram titled `getUnitedFlightsFlow` is shown. It features an `HTTP` connector (listener) and an `HTTP` connector (request) labeled `United REST Request`. The flow is part of an `Error handling` section.

```

{"flights":[{"airlineName":"United","price":460,"departureDate":"2015/03/20","planeType":"Boeing 737","origin":"MUA","code":"ER38ed","emptySeats":0,"destination":"STO"}, {"airlineName":"United","price":345.99,"departureDate":"2015/02/11","planeType":"737","origin":"MUA","code":"ER45if","emptySeats":52,"destination":"LAX"}, {"airlineName":"United","price":346,"departureDate":"2015/04/11","planeType":"777","origin":"MUA","code":"ER45jd","emptySeats":12,"destination":"LAX"}, {"airlineName":"United","price":423,"departureDate":"2015/06/11","planeType":"787","origin":"MUA","code":"ER945","emptySeats":6,"destination":"LAX"}, {"airlineName":"United","price":845,"departureDate":"2015/07/11","planeType":"727","origin":"MUA","code":"ER9fje","emptySeats":32,"destination":"CLE"}, {"airlineName":"United","price":245,"departureDate":"2015/08/11","planeType":"747","origin":"MUA","code":"ER3xfs","emptySeats":13,"destination":"CLE"}]}
  
```

13 | All

## Passing data to a RESTful web service

- For an HTTP Request endpoint, you can add parameters
  - URI parameters
  - Query parameters
  - Headers

The screenshot shows the `Parameters` configuration window in MuleSoft. It has a table with columns for `uri-param`, `Name`, and `Value`. One parameter is defined: `customer_id` with a value of `2`. There is an `Add Parameter` button at the bottom.

uri-param	Name	Value
	customer_id	2

- Send form parameters with a request by setting them in the payload

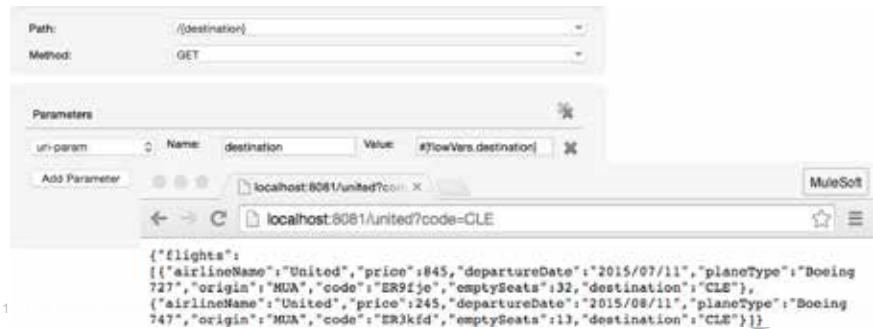


- Include attachments by adding an Attachment transformer to your flow



## Walkthrough 3-2: Pass arguments to a RESTful web service

- Modify the HTTP Request connector endpoint to use a URI parameter for the destination
- Set the destination to a static value
- Set the destination to a dynamic query parameter value
- Create a variable to set the destination



## Introducing RAML



## Approaches to API design



### HAND CODING

- Time-consuming
- Maintenance nightmare
- Difficult to read/ consume
- Disconnected from consumer



### SWAGGER

- Verbose
- Primarily for doc generation
- Limited reuse
- JSON based



### BLUEPRINT

- Markdown
- Design focused
- Limited reuse
- Tooling is proprietary



### RAML

- Simple and succinct
- Intuitive
- Open, non-proprietary
- Based on standards

17 | All contents Copyright © 2015, MuleSoft Inc.



## RAML: RESTful API Modeling Language



- A simple and succinct way of describing RESTful APIs
  - Resources, schema, parameters, responses, and more
- Developed to help out the current API ecosystem
  - Encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices
- A non-proprietary, vendor-neutral open spec
- Built on broadly-used standards such as YAML and JSON
  - YAML Ain't a Markup Language
    - A human-readable data serialization format where data structure hierarchy is maintained by outline indentation
- <http://raml.org>

18 | All contents Copyright © 2015, MuleSoft Inc.



## RAML example

```
1 #%RAML 0.8
2
3 title: World Music API
4 baseUri: http://example.api.com/{version}
5 version: v1
6 traits:
7   - paged:
8     queryParams:
9       pages:
10        description: The number of pages to return
11        type: number
12   - secured: !include http://raml-example.com/secured.yml
13 /songs:
14   is: [ paged, secured ]
15   get:
16     queryParams:
17       genre:
18        description: filter the songs by genre
19   post:
20     /{songId}:
21       get:
22         responses:
23           200:
24             body:
25               application/json:
26                 schema: !
```



## Consuming RESTful web services with RAML definitions

## Consuming RESTful web services with RAML definitions

- A RAML location (local file or external URI) can be specified for an HTTP Request connector
- After you specify a RAML location
  - All of the other fields will be automatically populated based on what's specified in the RAML

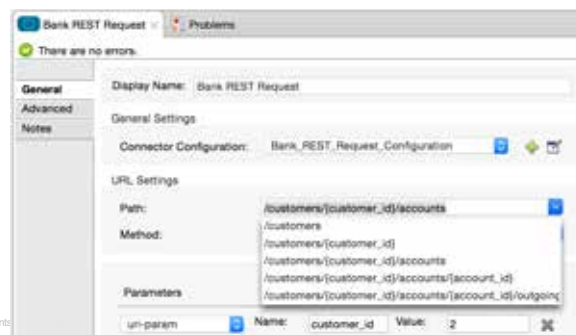


21 | All contents Copy



## RAML metadata

- The RAML metadata
  - Will be obtained every time you open the project in Studio
  - Kept in cache until you close the project
  - Can be refreshed/reloaded if the RAML changes
- Is used by Studio to offer intelligent suggestions

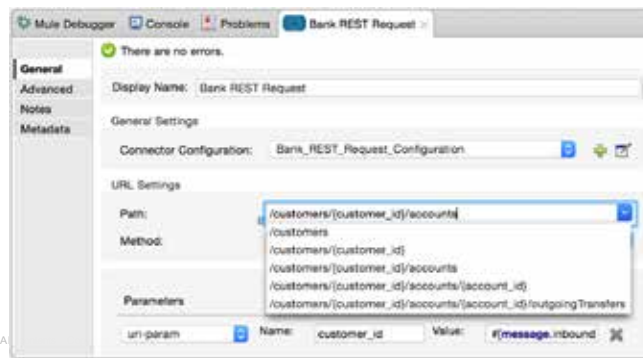


22 | All contents



### Walkthrough 3-3: Consume a RESTful web service that has a RAML definition

- Add a third flow to the application
- Add an endpoint to receive requests at <http://localhost:8081/bank>
- Add an HTTP Request endpoint to consume a RESTful web service defined with a RAML file



## Consuming SOAP web services

## Apache CXF

- Mule's SOAP support is based on Apache CXF
  - A web services framework in Java for SOAP messaging
  - Handles all serialization and deserialization
  - Handles all SOAP envelope and namespace processing
  - Developer sees only POJOs, etc. - not SOAP XML

1 All contents Copyright © 2015, MuleSoft Inc.



## Consuming SOAP web services

- First check and see if there is an existing Anypoint Connector to connect to the service provider
- If there is not, use the Web Service Consumer connector
  - Provide the location of the WSDL
  - The rest will be configured for you: host, port, address, available operations
- If you need more features, use the CXF component
  - Also used to expose an endpoint as a SOAP service



26 1 All contents Copyright © 2015, MuleSoft Inc.



## Walkthrough 3-4: Consume a SOAP Web Service

- Create a fourth flow with an endpoint to receive requests at <http://localhost:8081/delta>
- Add a Web Service Consumer connector to consume a SOAP web service for Delta flight data
- Use the DOM to XML transformer to display the SOAP response



## Passing data to a SOAP web service

- Using the Web Service Consumer
  - Use DataWeave (or the DataMapper transformer) to add and map an input argument
    - You will do this in module 5

# Summary

29 | All contents Copyright © 2015, MuleSoft Inc.

## Summary

- In this module, you learned to consume web services
- Use the Web Consumer connector to consume SOAP web services
- Use the HTTP Request connector to consume REST web services
  - With or without URI parameters and query parameters
  - With or without a RAML definition
- RAML is the Restful API Modeling Language
  - A simple, succinct, open-spec standard based way to describe RESTful APIs

30 | All contents Copyright © 2015, MuleSoft Inc.

