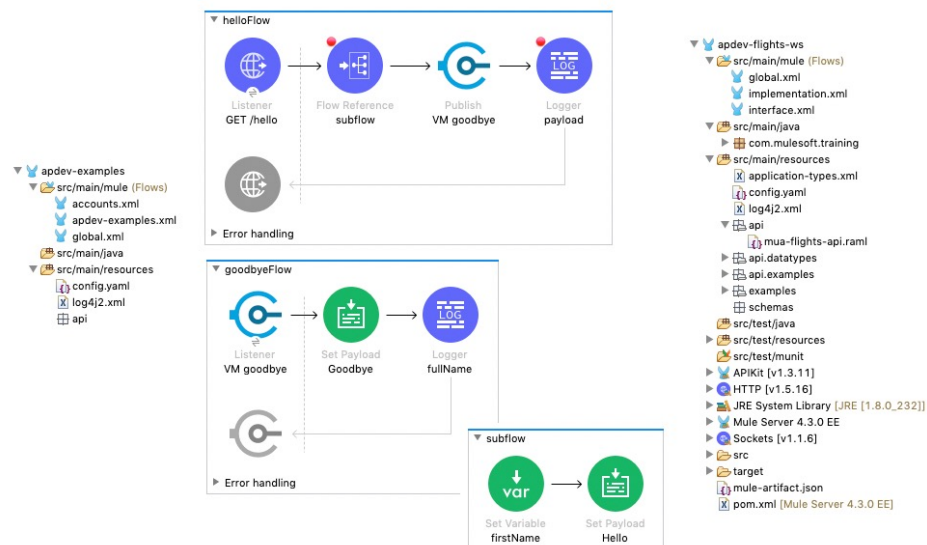




Module 7: Structuring Mule applications

1

Goal



2

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



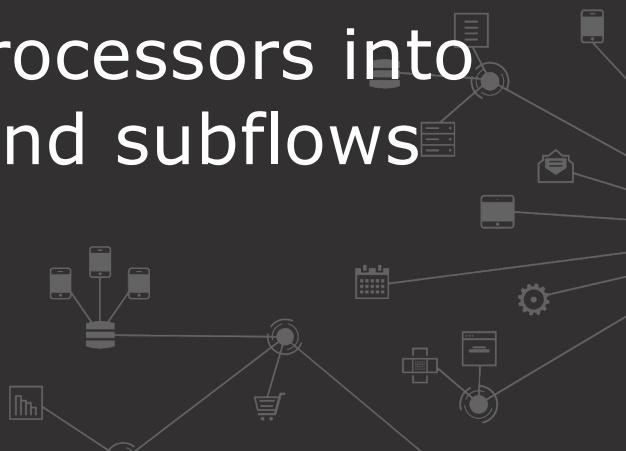
- Create applications composed of multiple flows and subflows
- Pass events between flows using asynchronous queues
- Encapsulate global elements in separate configuration files
- Specify application properties in a separate properties file and use them in the application
- Describe the purpose of each file and folder in a Mule project
- Define and manage application metadata

All contents © MuleSoft Inc.

3

3

Encapsulating processors into separate flows and subflows



4

Break up flows into separate flows and subflows



- Makes the graphical view more intuitive
 - You don't want long flows that go off the screen
- Makes XML code easier to read
- Enables code reuse
- Provides separation between an interface and implementation
 - We already saw this
- Makes them easier to test

All contents © MuleSoft Inc.

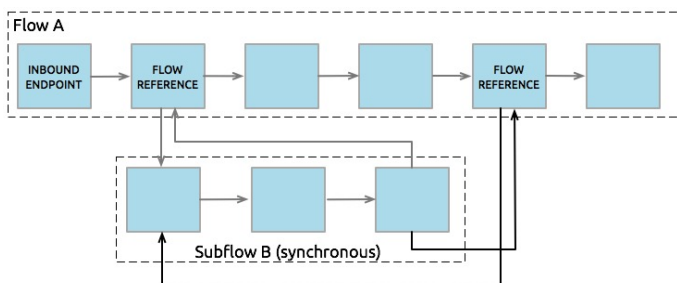
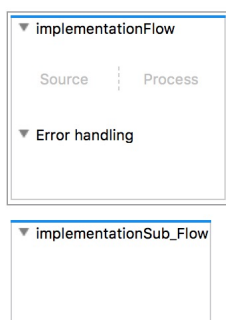
5

5

Flows vs subflows



- **Flows** can have their own error handling strategy, **subflows** cannot
- Flows without event sources are sometimes called **private** flows
- Subflows are executed exactly as if the processors were still in the calling flow



All contents © MuleSoft Inc.

6

6

Creating flows and subflows



- Several methods
 - Add a new scope: Flow or Sub Flow
 - Drag any event processor to the canvas – creates a flow
 - Right-click processor(s) in canvas and select Extract to
- Use Flow Reference component to pass events to other flows or subflows
 - Metadata is automatically propagated and can be customized
- Variables persist through all flows unless the event crosses a transport boundary
 - We saw this in the last module

Scopes

- All
- Any
- Async
- Cache
- Flow
- For Each
- Sub Flow
- Try
- Until Successful

Components

- Custom Business Event
- Dynamic Evaluate
- Flow Reference
- Idempotent Message Validator
- Logger
- Parse Template
- Set Transaction Id
- Transform Message

All contents © MuleSoft Inc.

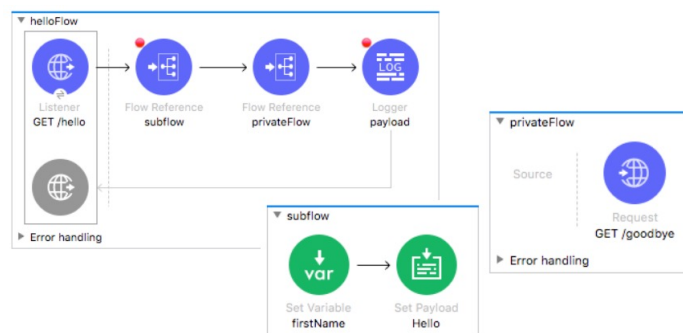
7

7

Walkthrough 7-1: Create and reference subflows and private flows



- Extract processors into separate subflows and private flows
- Use the Flow Reference component to reference other flows
- Explore event data persistence through subflows and private flows

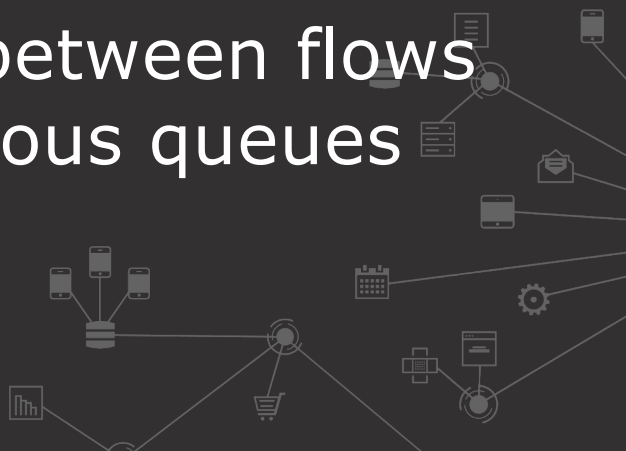


All contents © MuleSoft Inc.

8

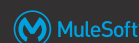
8

Passing events between flows using asynchronous queues



9

Passing events between flows using asynchronous queues



- When using Flow Reference, events are passed synchronously between flows
- You may want to pass events asynchronously between flows to
 - Achieve higher levels of parallelism in specific stages of processing
 - Allow for more-specific tuning of areas within a flow's architecture
 - Distribute work across a cluster
 - Communicate with another application running in the same Mule domain
 - Domains will be explained later this module
 - Implement simple queueing that does not justify a full JMS broker
 - JMS is covered in Module 12
- This can be accomplished using the **VM connector**

All contents © MuleSoft Inc.

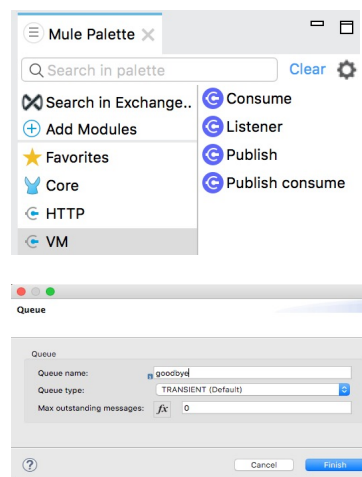
10

10

Using the VM connector



- Use the connector for intra and inter application communication through asynchronous queues
- Add the VM module to the project
- Configure a global element configuration
 - Specify a queue name and type
 - Queues can be transient or persistent
 - By default, the connector uses in-memory queues
 - **Transient** queues are faster, but are lost in the case of a system crash
 - **Persistent** queues are slower but reliable
- Use operations to publish and/or consume events



All contents © MuleSoft Inc.

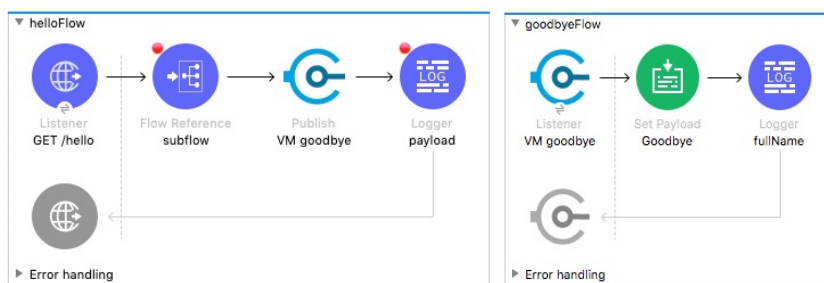
11

11

Walkthrough 7-2: Pass events between flows using the VM connector



- Pass events between flows using the VM connector
- Explore variable persistence with VM communication
- Publish content to a VM queue and then wait for a response
- Publish content to a VM queue without waiting for a response



All contents © MuleSoft Inc.

12

12

Organizing Mule application files



13

Separating apps into multiple configuration files



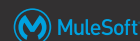
- Just as we separated flows into multiple flows, we also want to separate configuration files into multiple configuration files
- Monolithic files are difficult to read and maintain
- Separating an application into multiple configuration files makes code
 - Easier to read
 - Easier to work with
 - Easier to test
 - More maintainable

All contents © MuleSoft Inc.

14

14

Encapsulating global elements in a configuration file



- If you reference global elements in one file that are defined in various, unrelated files
 - It can be confusing
 - It makes it hard to find them
- A good solution is to put most global elements in one config file
 - All the rest of the files reference them
 - If a global element is specific to and only used in one file, it can make sense to keep it in that file

All contents © MuleSoft Inc.

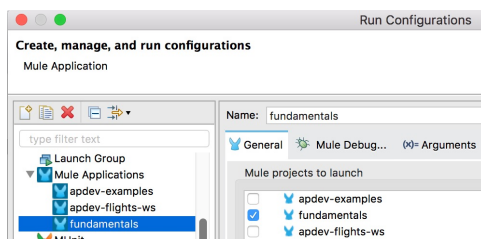
15

15

Creating multiple applications



- You are also not going to want all your flows in one application/project
- Separate functionality into multiple applications to
 - Allow managing and monitoring of them as separate entities
 - Use different, incompatible JAR files
- Run more than one application at a time in Anypoint Studio by creating a run configuration

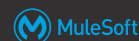


All contents © MuleSoft Inc.

16

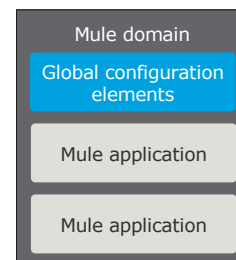
16

Sharing global elements between applications



- **A domain project** can be used to share global configuration elements between applications, which lets you

- Ensure consistency between applications upon any changes, as the configuration is only set in one place
- Expose multiple services within the domain on the same port
- Share the connection to persistent storage (Module 12)
- Call flows in other applications using the VM connector



- Only available for customer-hosted Mule runtimes, not on CloudHub
- The general process
 - Create a Mule Domain Project and associate Mule applications with a domain
 - Add global element configurations to the domain project

All contents © MuleSoft Inc.

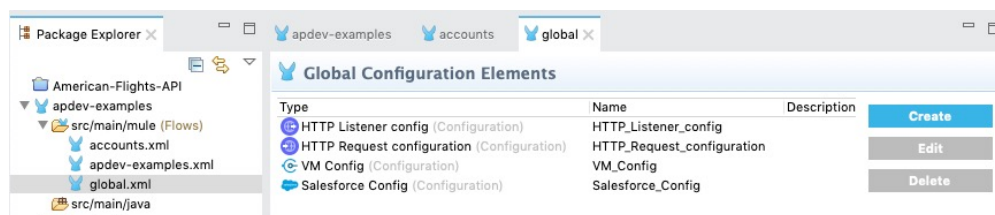
17

17

Walkthrough 7-3: Encapsulate global elements in a separate configuration file



- Create a new configuration file with an endpoint that uses an existing global element
- Create a configuration file global.xml for just global elements
- Move the existing global elements to global.xml
- Create a new global element in global.xml and configure a new connector to use it



All contents © MuleSoft Inc.

18

18

Organizing and parameterizing application properties



19

Application properties



- Provide an easier way to manage connector properties, credentials, and other configurations
- Replace static values
- Are defined in a configuration file
 - Either in a .yaml file or a .properties file
- Are implemented using property placeholders
- Can be encrypted
- Can be overridden by system properties when deploying to different environments

All contents © MuleSoft Inc.

20

20

Defining application properties



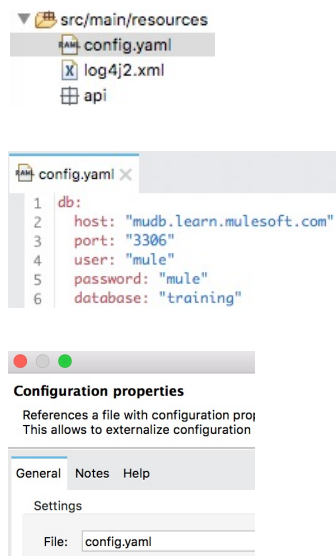
- Create a YAML properties file in the src/main/resources folder

`config.yaml`

- Define properties in the hierarchical YAML file

```
db:
  port: "3306"
  user: "mule"
```

- Create a Configuration properties global element



All contents © MuleSoft Inc.

21

21

Using application properties

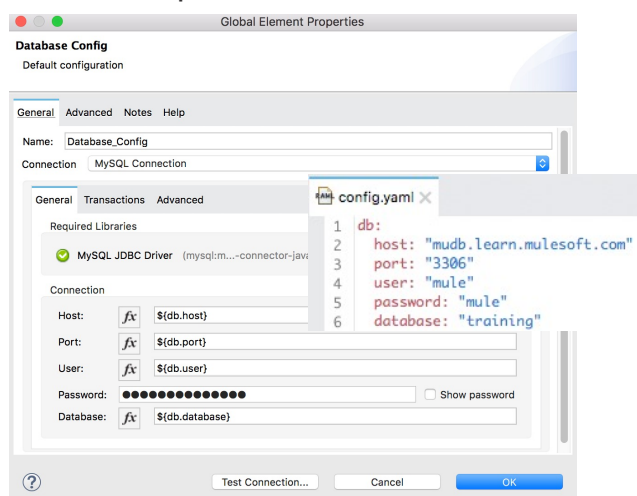


- In global element configurations and event processors

`${db.port}`

- In DataWeave expressions

`{port: Mule::p('db.port')}`



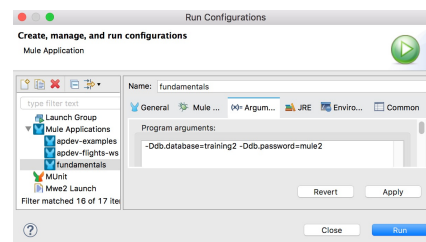
All contents © MuleSoft Inc.

22

22

Overriding property values in different environments

- Use **system properties** to override property values when deploying an application to a different environment (like dev, qa, production),
- Set system properties (JVM parameters) from
 - Anypoint Studio
in Run > Run Configurations > Arguments
 - The command line for a standalone Mule instance
`mule -M-Ddb.database=training2 -M-Ddb.password=mule2`



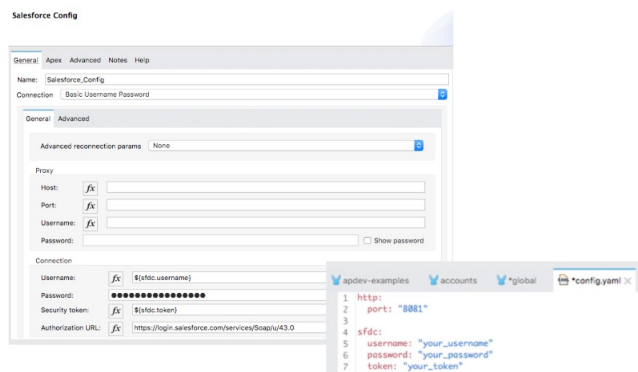
All contents © MuleSoft Inc.

23

23

Walkthrough 7-4: Use property placeholders in connectors

- Create a YAML properties file for an application
- Configure an application to use a properties file
- Define and use HTTP and Salesforce connector properties



All contents © MuleSoft Inc.

24

24

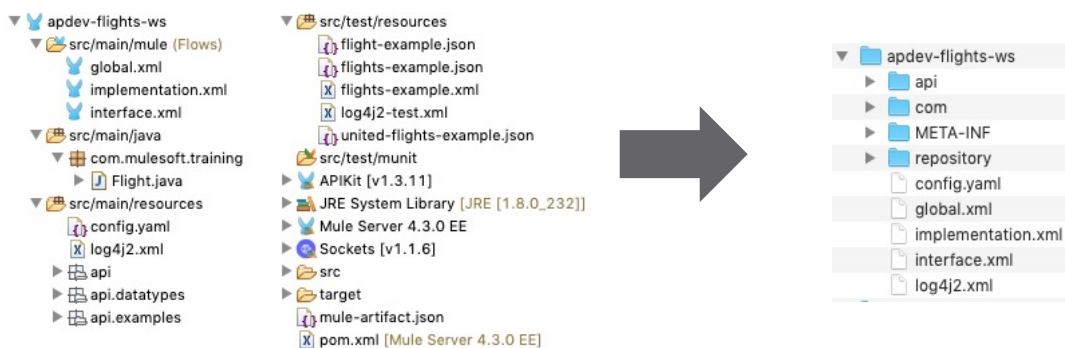
Organizing Mule project files

25

Examining the folder structure for a Mule project



- The names of folders indicate what they should contain
- src/test folders should contain files only needed at development time
 - Like schema and example files for metadata types, sample data for transformations
 - They are not included in the application JAR when it is packaged



All contents © MuleSoft Inc.

26

26

In Mule 4, Mule applications are Maven projects



- **Maven** is a tool for building and managing any Java-based project that provides
 - A standard way to build projects
 - A clear definition of what a project consists of
 - An easy way to publish project information
 - A way to share JARs across several projects
- Maven manages a project's build, reporting, and documentation from a central piece of information – the **project object model (POM)**
- A Maven build produces one or more **artifacts**, like a compiled JAR
 - Each artifact has a group ID (usually a reversed domain name, like com.example.foo), an artifact ID (just a name), and a version string



Apache Maven Project

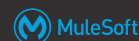
<http://maven.apache.org/>

All contents © MuleSoft Inc.

27

27

The POM (Project Object Model)



- Is an XML file that contains info about the project and configuration details used by Maven to build the project including
 - Project info like its version, description, developers, and more
 - Project dependencies
 - The plugins or goals that can be executed

```

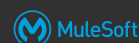
1 <?xml version="1.0" encoding="UTF-8" stand
2 <project xmlns="http://maven.apache.org/POM
3   <modelVersion>4.0.0</modelVersion>
4
5   <groupId>com.mycompany</groupId>
6   <artifactId>apdev-flights-ws</artifactId>
7   <version>1.0.0-SNAPSHOT</version>
8   <packaging>mule-application</packaging>
9
10  <name>apdev-flights-ws</name>
11
12  <properties>
13    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
14    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
15
16    <app.runtime>4.3.0</app.runtime>
17    <mule.maven.plugin.version>3.3.5</mule.maven.plugin.version>
  
```

All contents © MuleSoft Inc.

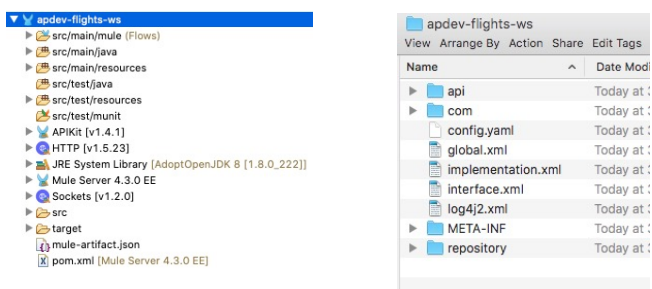
28

28

Walkthrough 7-5: Create a well-organized Mule project



- Create a project based on a new API in Design Center
- Review the project's configuration and properties files
- Create an application properties file and a global configuration file
- Add Java files and test resource files to the project
- Create and examine the contents of a deployable archive for the project



All contents © MuleSoft Inc.

29

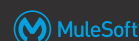
29

Managing metadata for a project

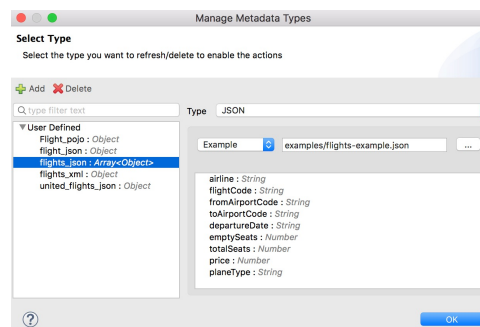


30

Defining metadata



- It is often beneficial to define metadata for an application
 - For the output structures required for transformations
 - You did this in Module 4 when transforming database output to JSON defined in the API
 - For the output of operations that can connect to data sources of different structures
 - Like the HTTP Request connector
 - For the output of connectors that are not DataSense enabled
 - And do not automatically provide metadata about the expected input and output



All contents © MuleSoft Inc.

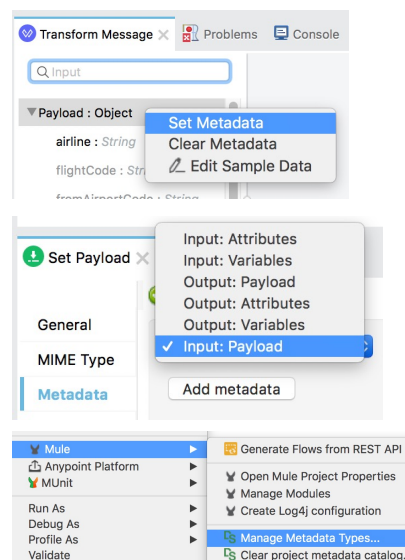
31

31

Ways to access the Metadata Editor



- From the Transform Message component
 - Set Metadata
 - Clear Metadata
 - Edit Sample Data
- From the Metadata tab in the properties view for most event processors
 - Input: Attributes
 - Input: Variables
 - Output: Payload
 - Output: Attributes
 - Output: Variables
 - Input: Payload
 - Add metadata
- From a project's menu in the Package Explorer
 - Generate Flows from REST API
 - Open Mule Project Properties
 - Manage Modules
 - Create Log4j configuration
 - Manage Metadata Types...
 - Clear project metadata catalog...

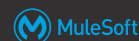


All contents © MuleSoft Inc.

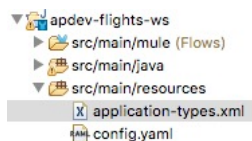
32

32

Where is metadata stored?



- In application-types.xml in src/main/resources



```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
3   <types:catalog>
4     <types:type name="flights_json" format="json">
5       <types:shape format="weave" example="examples/flights-example.json"><![CDATA[
6
7   type flights_json = Array<{ | airline: String, flightCode: String, fromAirpor
8
9   ]]></types:shape>
10  </types:type>
11  <types:type name="flight_json" format="json">
12    <types:shape format="weave" example="examples/flight-example.json"><![CDATA[
13
14  type flight_json = { |
15    airline: String,
16    flightCode: String,
17    fromAirportCode: String,
18    toAirportCode: String,
19    departureDate: String,
20    emptySeats: Number {"typeId": "int"},
21    totalSeats: Number {"typeId": "int"},
22    price: Number {"typeId": "int"},
23    planeType: String
24  }

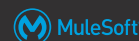
```

All contents © MuleSoft Inc.

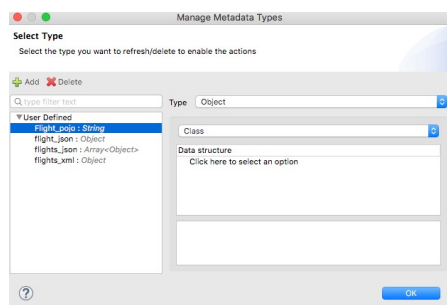
33

33

Walkthrough 7-6: Manage metadata for a project



- Review existing metadata for the training4-american-ws project
- Define new metadata to be used in transformations in the new apdev-flights-ws project
- Manage metadata



```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
3   <types:catalog>
4     <types:type name="flights_json" format="json">
5       <types:shape format="weave" example="examples/flights-example.json"><![CDATA[%dw 2.0
6
7   type flights_json = Array<{ | airline: String, flightCode: String, fromAirportCode: Str
8
9   ]]></types:shape>
10  </types:type>
11  <types:type name="flight_json" format="json">
12    <types:shape format="weave" example="examples/flight-example.json"><![CDATA[%dw 2.0
13
14  type flight_json = { |
15    airline: String,
16    flightCode: String,
17    fromAirportCode: String,
18    toAirportCode: String,
19    departureDate: String,
20    emptySeats: Number {"typeId": "int"},
21    totalSeats: Number {"typeId": "int"},
22    price: Number {"typeId": "int"},
23    planeType: String
24  }

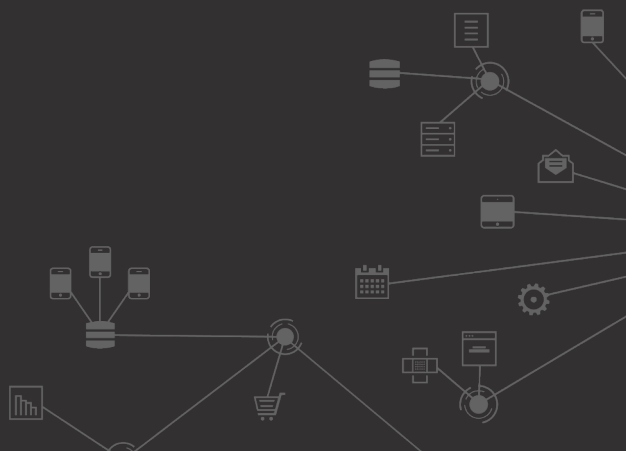
```

All contents © MuleSoft Inc.

34

34

Summary



35

Summary



- Separate functionality into **multiple applications** to allow managing and monitoring of them as separate entities
- Mule applications are **Maven** projects
 - A project's **POM** is used by Maven to build, report upon, and document a project
 - Maven builds an artifact (a Mule deployable archive JAR) from multiple dependencies (module JARs)
- Separate application functionality into **multiple configuration files** for easier development and maintenance
 - Encapsulate **global elements** into their own separate configuration file
- Share resources between applications by creating a **shared domain**

All contents © MuleSoft Inc.

36

36

Summary



- Define **application properties** in a YAML file and reference them as `${prop}`
- Application **metadata** is stored in `application-types.xml`
- Create applications composed of multiple **flows** and **subflows** for better readability, maintenance, and reusability
- Use a **Flow Reference** to call flows synchronously
- Use the **VM connector** to pass events between flows using asynchronous queues