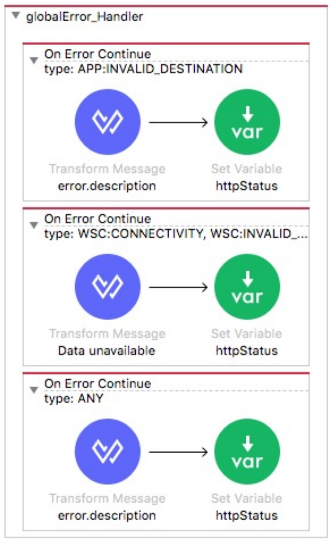
MuleSoft

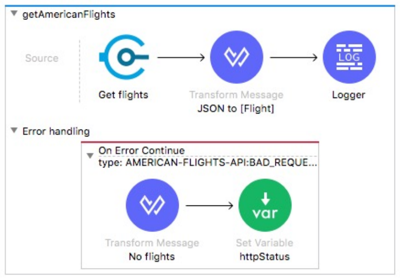
Module 10: Handling errors

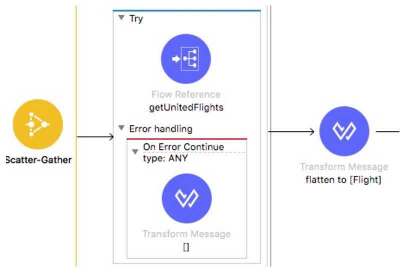
1

Goal

MuleSoft

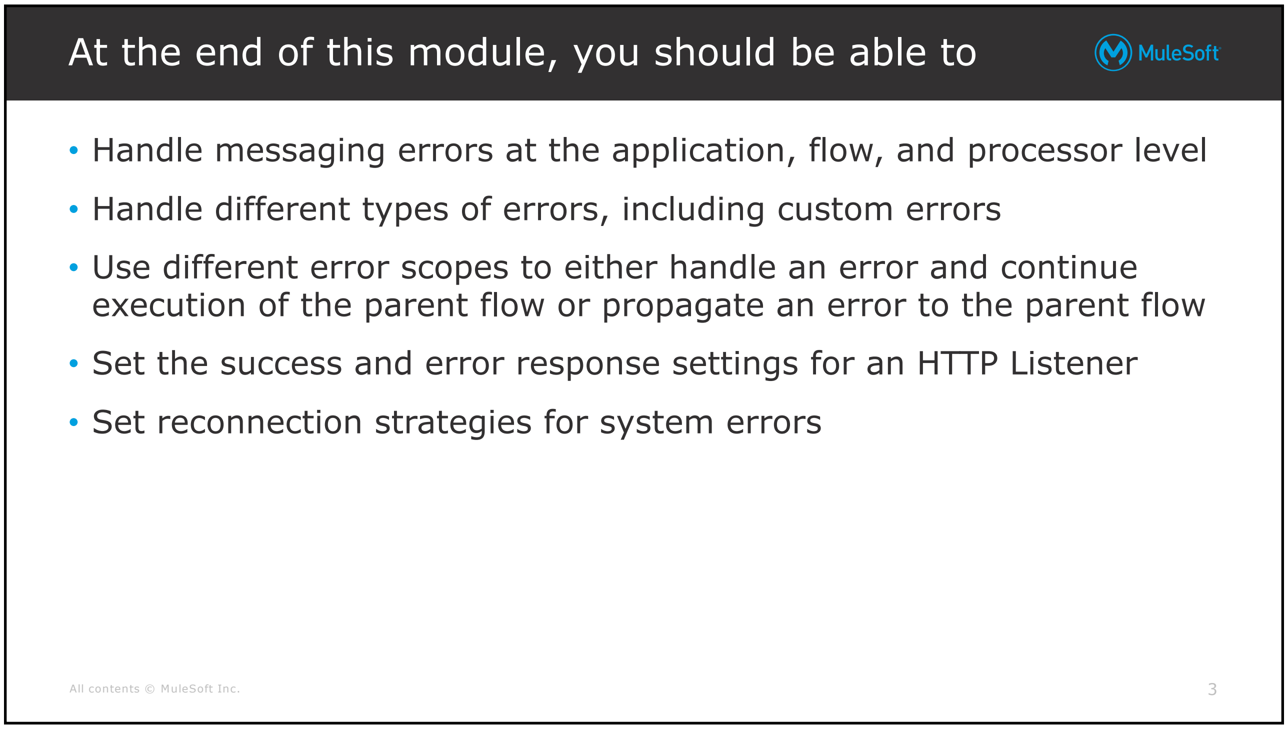






2

2



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

MuleSoft

- Handle messaging errors at the application, flow, and processor level
- Handle different types of errors, including custom errors
- Use different error scopes to either handle an error and continue execution of the parent flow or propagate an error to the parent flow
- Set the success and error response settings for an HTTP Listener
- Set reconnection strategies for system errors

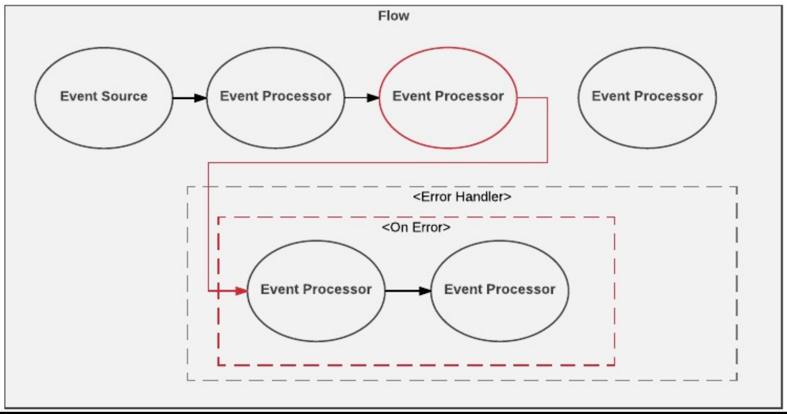
3

Reviewing the default handling of messaging errors

4

Handling messaging errors

- When an event is being processed through a Mule flow that throws an error
  - Normal flow execution stops
  - The event is passed to the first processor in an error handler

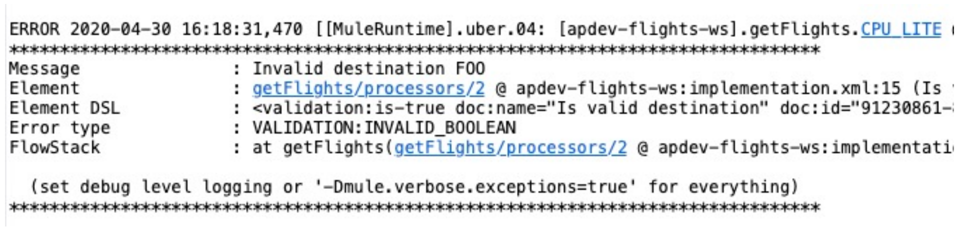


5

5

Default error handler behavior

- If there is no error handler defined, a **Mule default error handler** is used
  - Implicitly and globally handles all messaging errors thrown in Mule applications
  - Stops execution of the flow and logs information about the error
  - Cannot be configured



```
ERROR 2020-04-30 16:18:31,470 [[MuleRuntime].uber.04: [apdev-flights-ws].getFlights.CPU_LITE
********************************************************************************
Message                 : Invalid destination FOO
Element                 : getFlights/processors/2 @ apdev-flights-ws:implementation.xml:15 (Is
Element DSL             : <validation:is-true doc:name="Is valid destination" doc:id="91230861-
Error type              : VALIDATION:INVALID_BOOLEAN
FlowStack               : at getFlights(getFlights/processors/2 @ apdev-flights-ws:implementati

  (set debug level logging or '-Dmule.verbose.exceptions=true' for everything)
********************************************************************************
```

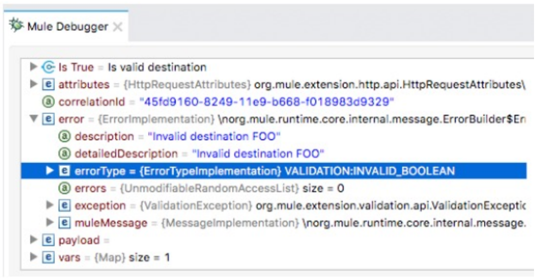
6

6

Information about the error

- When an error is thrown, an **error** object is created

- Two of its properties include
  - error.**description** – a string
  - error.**errorType** – an object



Mule Debugger

```
▶ Is True = Is valid destination
▶ attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes\
  correlationId = "45fd9160-8249-11e9-b668-f018983d9329"
▼ error = {ErrorImplementation} \norg.mule.runtime.core.internal.message.ErrorBuilder$En
    description = "Invalid destination FOO"
    detailedDescription = "Invalid destination FOO"
  ▶ errorType = {ErrorTypeImplementation} VALIDATION:INVALID_BOOLEAN
    errors = {UnmodifiableRandomAccessList} size = 0
  ▶ exception = {ValidationException} org.mule.extension.validation.api.ValidationExceptic
  ▶ muleMessage = {MessageImplementation} \norg.mule.runtime.core.internal.message.
▶ payload =
▶ vars = {Map} size = 1
```

- Error types are identified by a namespace and an identifier
  - $\underbrace{\text{HTTP}}_{\text{namespace}}:\underbrace{\text{UNAUTHORIZED}}_{\text{identifier}}$, HTTP:CONNECTIVITY, VALIDATION:INVALID_BOOLEAN

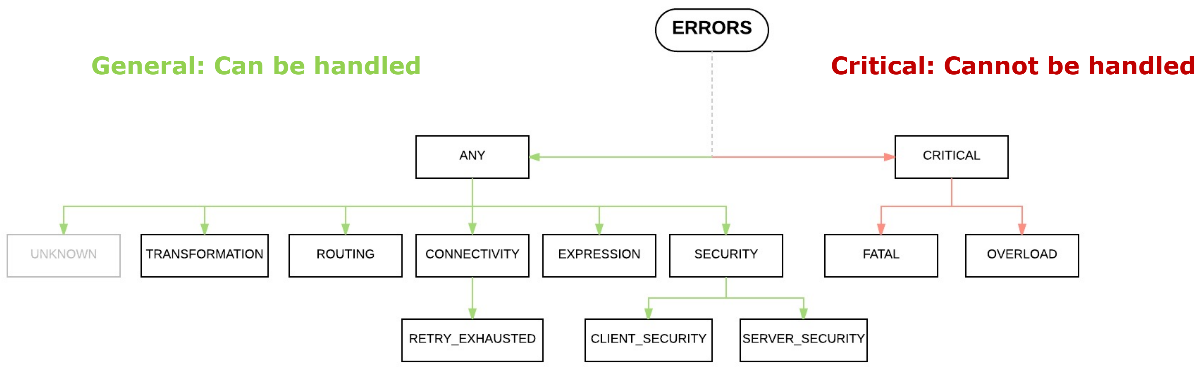
Error types follow a hierarchy

- Each error type has a parent
  - HTTP:UNAUTHORIZED has MULE:CLIENT_SECURITY as the parent, which has MULE:SECURITY as the parent
  - VALIDATION:INVALID_BOOLEAN has VALIDATION:VALIDATION as the parent, which has MULE:VALIDATION as the parent

- The error type ANY is the most general parent

```
{
  "identifier": "INVALID_BOOLEAN",
  "parentErrorType": {
    "identifier": "VALIDATION",
    "parentErrorType": {
      "identifier": "VALIDATION",
      "parentErrorType": {
        "identifier": "ANY",
        "parentErrorType": null,
        "namespace": "MULE"
      },
      "namespace": "MULE"
    },
    "namespace": "VALIDATION"
  },
  "namespace": "VALIDATION"
}
```

Error type hierarchy reference

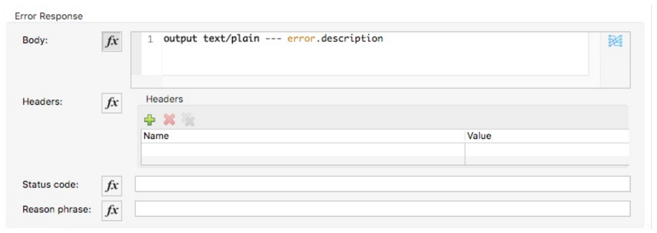


9

9

Information returned from HTTP Listeners

- By default, for a **success response**
  - The payload
  - A status code of 200
- By default, for an **error response**
  - The error description
  - A status code of 500
- You can override these values for an HTTP Listener



Error Response

Body:

```
output text/plain --- error.description
```

Headers:

Headers

Name	Value

Status code:

Reason phrase:

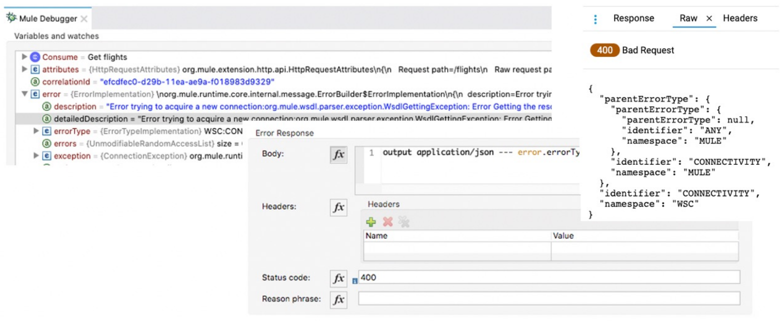
10

10

Walkthrough 10-1: Explore default error handling

MuleSoft

- Explore information about different types of errors in the Mule Debugger and the console
- Review the default error handling behavior
- Review and modify the error response settings for an HTTP Listener



```
Consume = Get flights
attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes\n{\n  Request path=/flights\n  Raw request pa
correlationId = "efcdfec0-d29b-11ea-ae9a-f018983d9329"
error = {ErrorImplementation} \norg.mule.runtime.core.internal.message.ErrorBuilder$ErrorImplementation\n{\n  description=Error tryir
  description = "Error trying to acquire a new connection:org.mule.wsdl.parser.exception.WsdlGettingException: Error Getting the res
  detailedDescription = "Error trying to acquire a new connection:org.mule.wsdl.parser.exception.WsdlGettingException: Error Getting
  errorType = {ErrorTypeImplementation} WSC:CON
  errors = {UnmodifiableRandomAccessList} size =
  exception = {ConnectionException} org.mule.runti
```

Body: `output application/json --- error.errorTy`

Headers:

Status code: 400

Reason phrase:

400 Bad Request

```
{
  "parentErrorType": {
    "parentErrorType": {
      "parentErrorType": null,
      "identifier": "ANY",
      "namespace": "MULE"
    },
    "identifier": "CONNECTIVITY",
    "namespace": "MULE"
  },
  "identifier": "CONNECTIVITY",
  "namespace": "WSC"
}
```

11

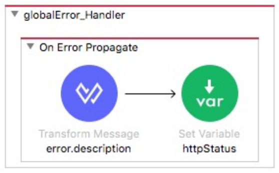
11

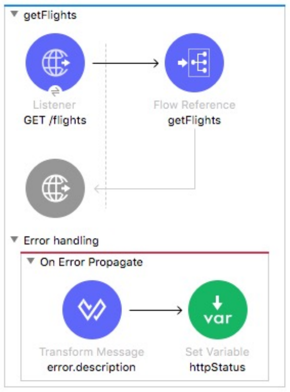
Creating error handlers

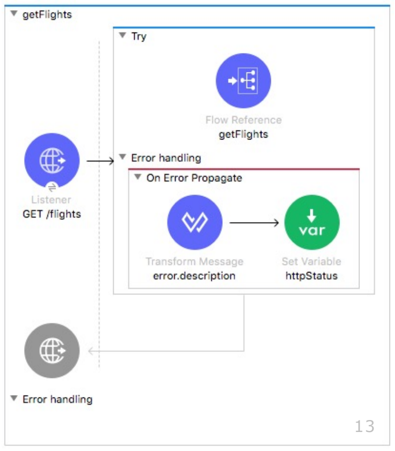
12

Creating error handlers

- Error handlers can be added to
  - An application (outside of any flows)
  - A flow
  - A selection of one or more event processors





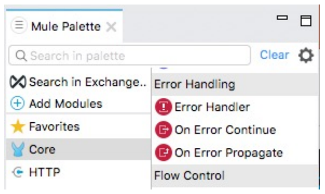


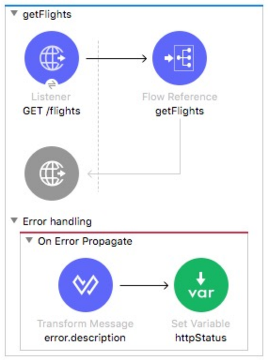
13

13

Adding error handler scopes

- Each error handler can contain one or more error handler scopes
  - On Error Continue
  - On Error Propagate
- Each error scope can contain any number of event processors





14

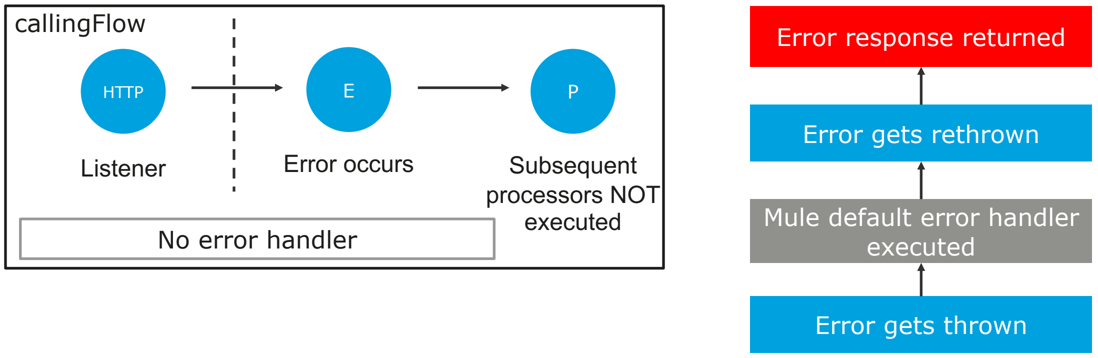
14

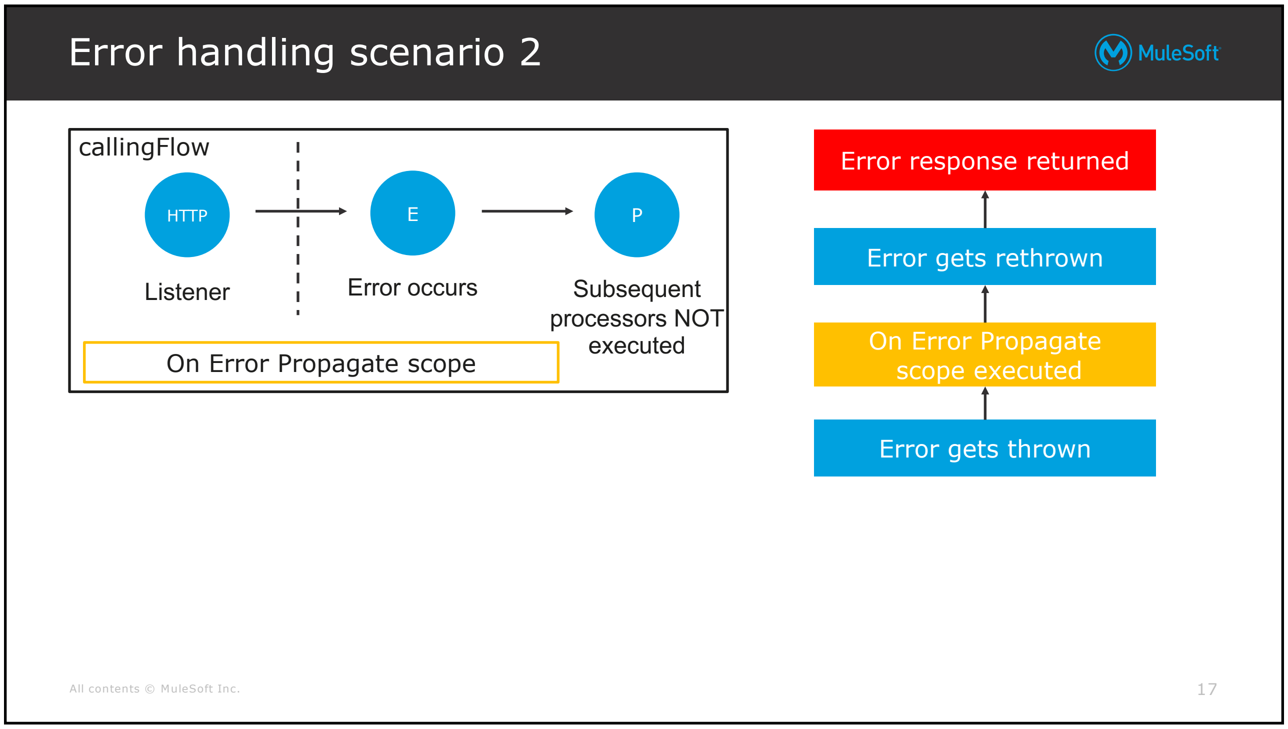
Two types of error handling scopes

- **On Error Propagate**
  - All processors in the error handling scope are executed
  - At the end of the scope
    - The rest of the flow that threw the error is not executed
    - *The error is rethrown up to the next level and handled there*
  - An HTTP Listener returns an ***error*** response

- **On Error Continue**
  - All processors in the error handling scope are executed
  - At the end of the scope
    - The rest of the flow that threw the error is not executed
    - *The event is passed up to the next level as if the flow execution had completed successfully*
  - An HTTP Listener returns a ***successful*** response

Error handling scenario 1

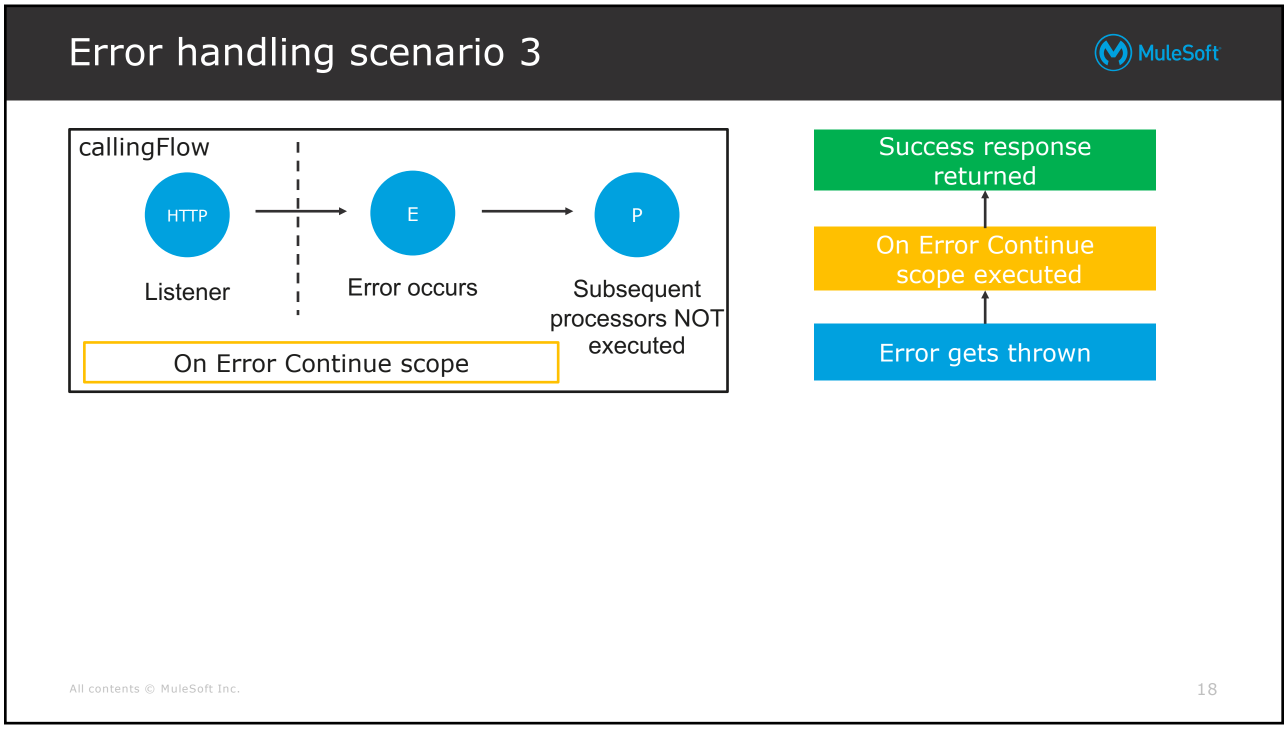




Error handling scenario 2

17

17

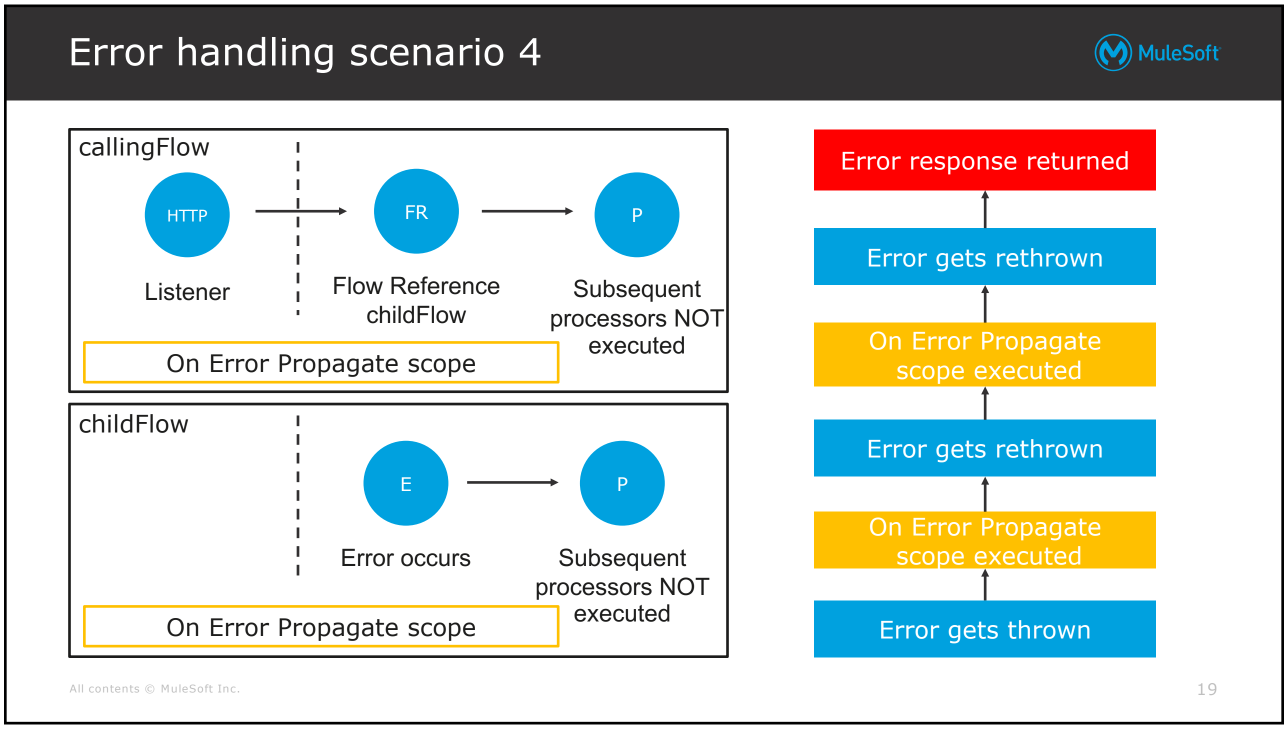


Error handling scenario 3

callingFlow
HTTP
Listener
E
Error occurs
P
Subsequent processors NOT executed
On Error Continue scope
Success response returned
On Error Continue scope executed
Error gets thrown

18

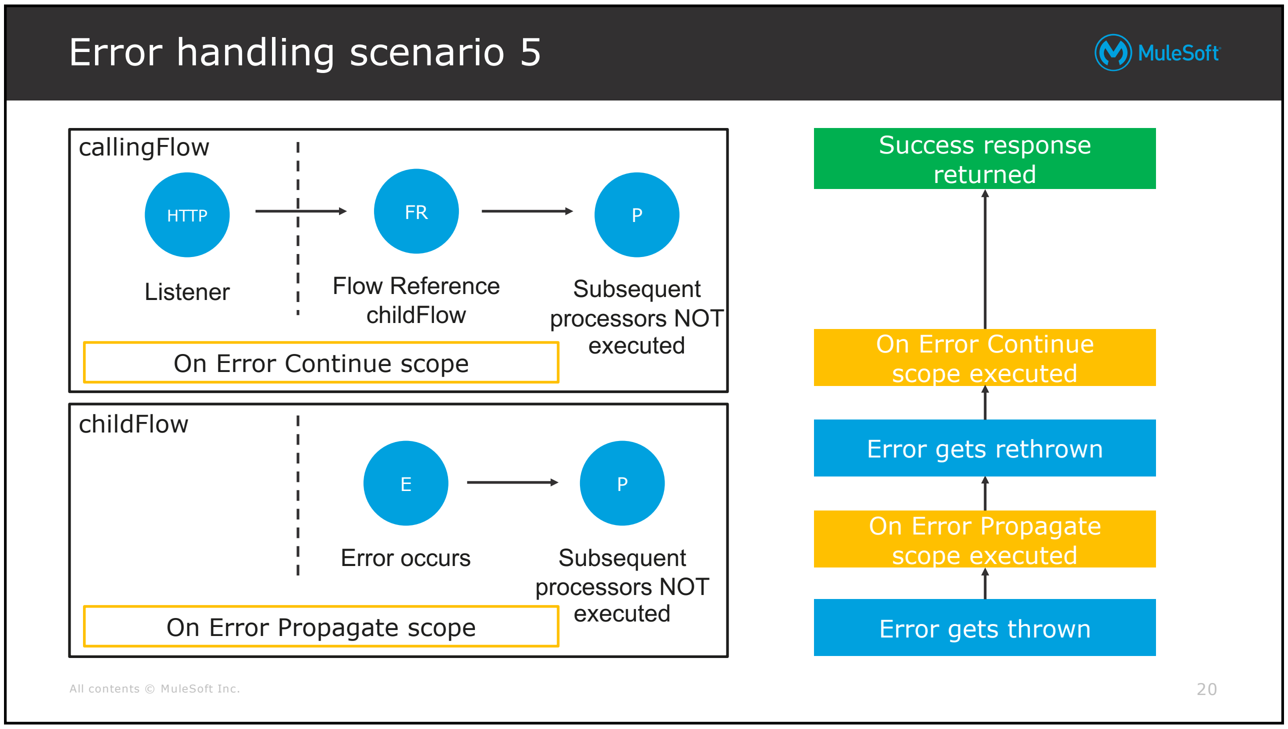
18



Error handling scenario 4

19

19

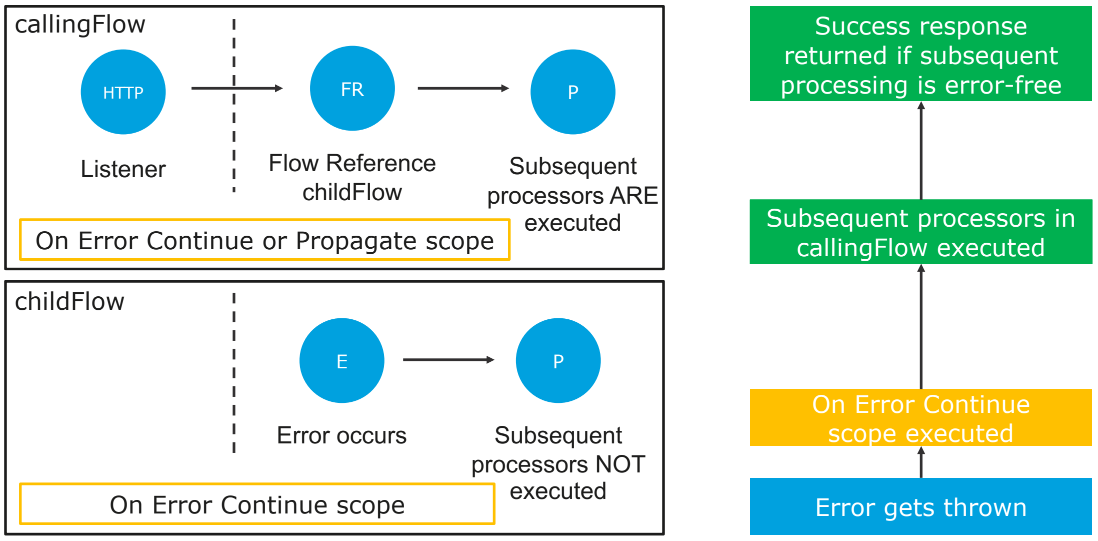


Error handling scenario 5

20

20

Error handling scenario 6


callingFlow
HTTP
Listener
FR
Flow Reference childFlow
P
Subsequent processors ARE executed
On Error Continue or Propagate scope
childFlow
E
Error occurs
P
Subsequent processors NOT executed
On Error Continue scope
Success response returned if subsequent processing is error-free
Subsequent processors in callingFlow executed
On Error Continue scope executed
Error gets thrown

21

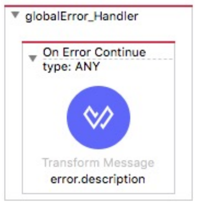
21

Handling errors at the application level

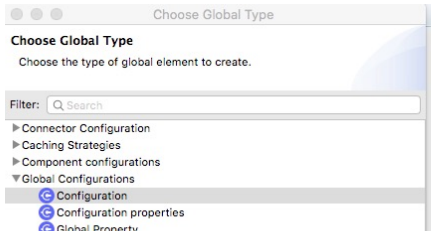
22

Defining a default error handler for an application

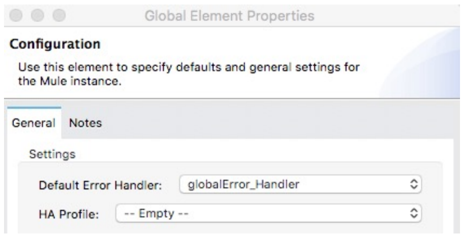
- Add an error handler outside a flow
  - Typically, put it in the global configuration file



- Specify this handler to be the application's default error handler


Choose Global Type

Choose the type of global element to create.


Global Element Properties

Configuration

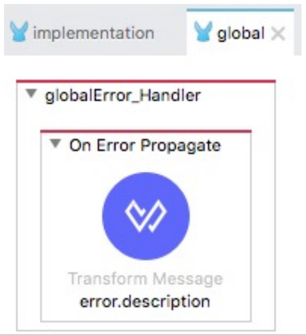
Use this element to specify defaults and general settings for the Mule instance.

Default Error Handler: globalError_Handler

HA Profile: -- Empty --

Walkthrough 10-2: Handle errors at the application level

- Create a global error handler in an application
- Configure an application to use a global default error handler
- Explore the differences between the On Error Continue and On Error Propagate scopes
- Modify the default error response settings for an HTTP Listener

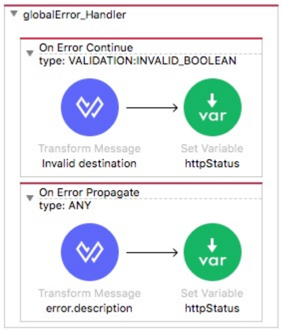


Handling specific types of errors

25

Adding multiple error handler scopes

- Each error handler can contain one or more error handler scopes
  - Any number of On Error Continue and/or On Error Propagate
- Each error handler scope specifies when it should be executed
  - The error is handled by the *first* error scope whose condition evaluates to true

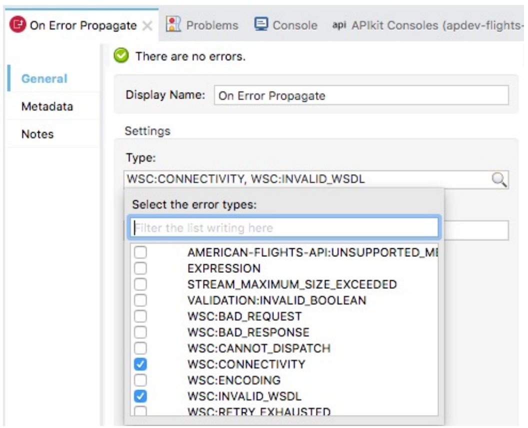


26

Specifying scope execution for specific error types

MuleSoft

- Set the **type** to ANY (the default) or one or more types or errors



On Error Propagate | Problems | Console | APIkit Consoles (apdev-flights-

General
Metadata
Notes

There are no errors.

Display Name: On Error Propagate

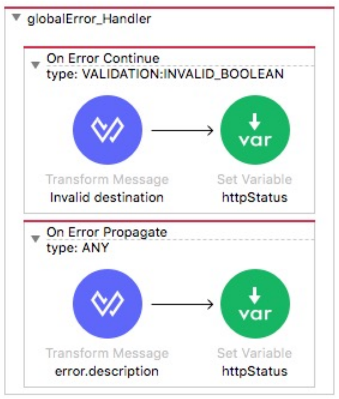
Settings

Type: WSC:CONNECTIVITY, WSC:INVALID_WSDL

Select the error types:

Filter the list writing here

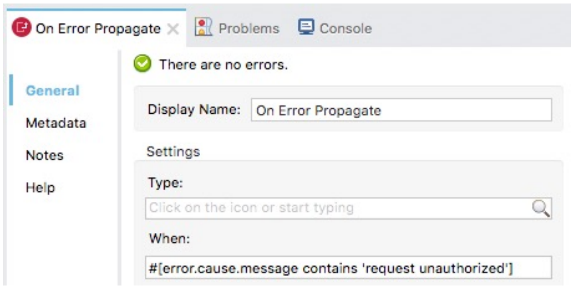
- [ ] AMERICAN-FLIGHTS-API:UNSUPPORTED_ME
- [ ] EXPRESSION
- [ ] STREAM_MAXIMUM_SIZE_EXCEEDED
- [ ] VALIDATION:INVALID_BOOLEAN
- [ ] WSC:BAD_REQUEST
- [ ] WSC:BAD_RESPONSE
- [ ] WSC:CANNOT_DISPATCH
- [✓] WSC:CONNECTIVITY
- [ ] WSC:ENCODING
- [✓] WSC:INVALID_WSDL
- [ ] WSC:RETRY_EXHAUSTED



Specifying scope execution upon a specific condition

MuleSoft

- Set the **when** condition to a Boolean DataWeave expression
  - HTTP:UNAUTHORIZED
  - error.errorType.namespace == 'HTTP'
  - error.errorType.identifier == 'UNAUTHORIZED'
  - error.cause.message contains 'request unauthorized'
  - error.cause.class contains 'http'



On Error Propagate | Problems | Console

General
Metadata
Notes
Help

There are no errors.

Display Name: On Error Propagate

Settings

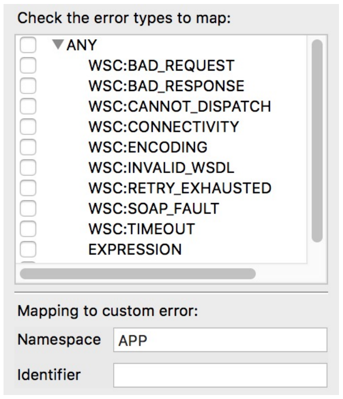
Type: Click on the icon or start typing

When: #[error.cause.message contains 'request unauthorized']

Walkthrough 10-3: Handle specific types of errors

MuleSoft

- Review the possible types of errors thrown by different processors
- Create error handler scopes to handle different error types

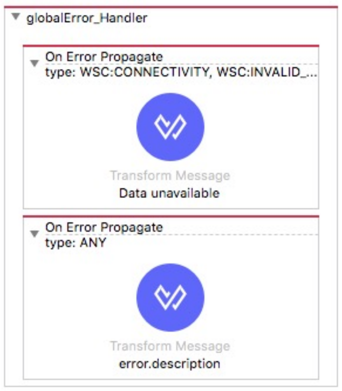


Check the error types to map:

- ANY
  - WSC:BAD_REQUEST
  - WSC:BAD_RESPONSE
  - WSC:CANNOT_DISPATCH
  - WSC:CONNECTIVITY
  - WSC:ENCODING
  - WSC:INVALID_WSDL
  - WSC:RETRY_EXHAUSTED
  - WSC:SOAP_FAULT
  - WSC:TIMEOUT
  - EXPRESSION

Mapping to custom error:

Namespace	APP
Identifier	



globalError_Handler

On Error Propagate
type: WSC:CONNECTIVITY, WSC:INVALID_...

Transform Message
Data unavailable

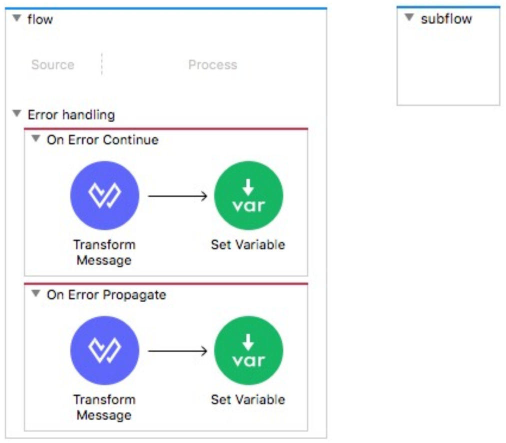
On Error Propagate
type: ANY

Transform Message
error.description

Handling errors at the flow level

Defining error handlers in flows

- All flows (except subflows) can have their own error handlers
- Any number of error scopes can be added to a flow's error handler



31

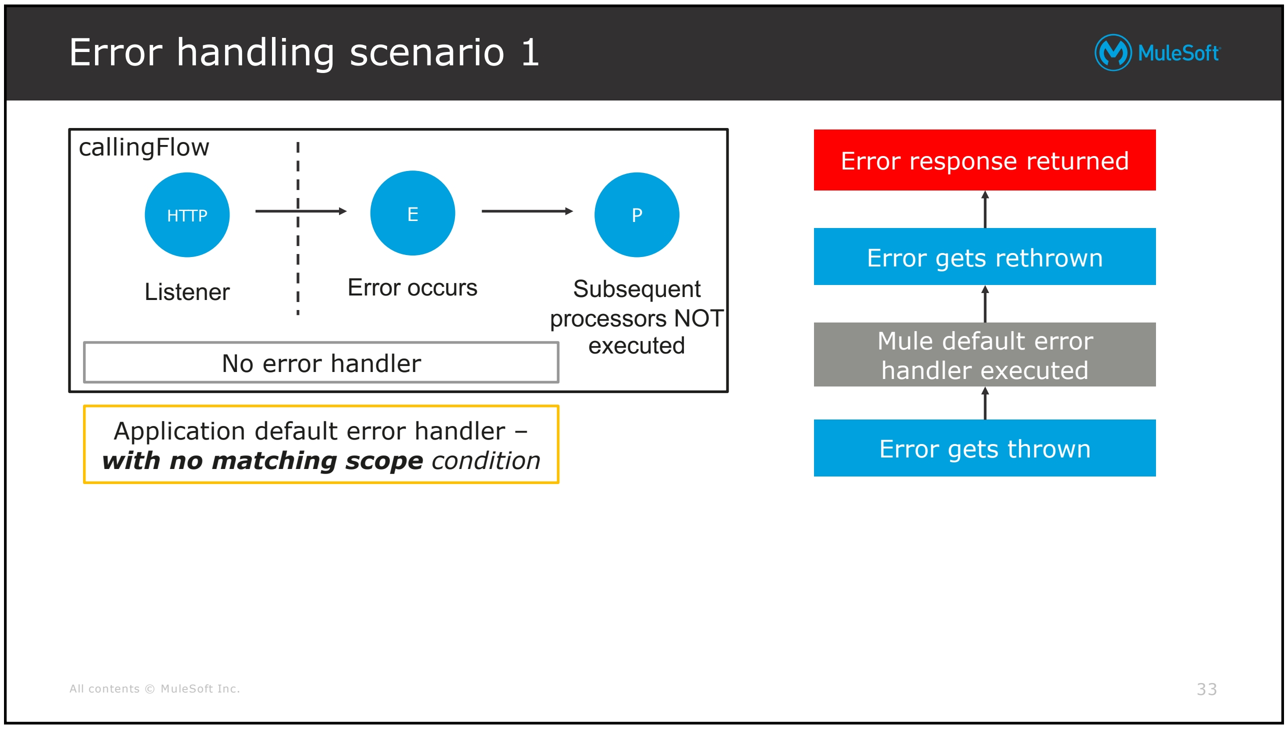
31

Which error scope handles an error?

- If a flow *has* an error handler
  - The error is handled by the *first* error scope whose condition evaluates to true
  - **If *no* scope conditions are true,** the error is handled by the **Mule default error handler** NOT any scope in an application's default error handler
    - The Mule default error handler propagates the error up the execution chain where there may or may not be handlers
- If a flow *does not have* an error handler
  - The error is handled by a scope in an **application's default error handler** (the first whose scope condition is true, which may propagate or continue) otherwise it is handled by the Mule default error handler

32

32

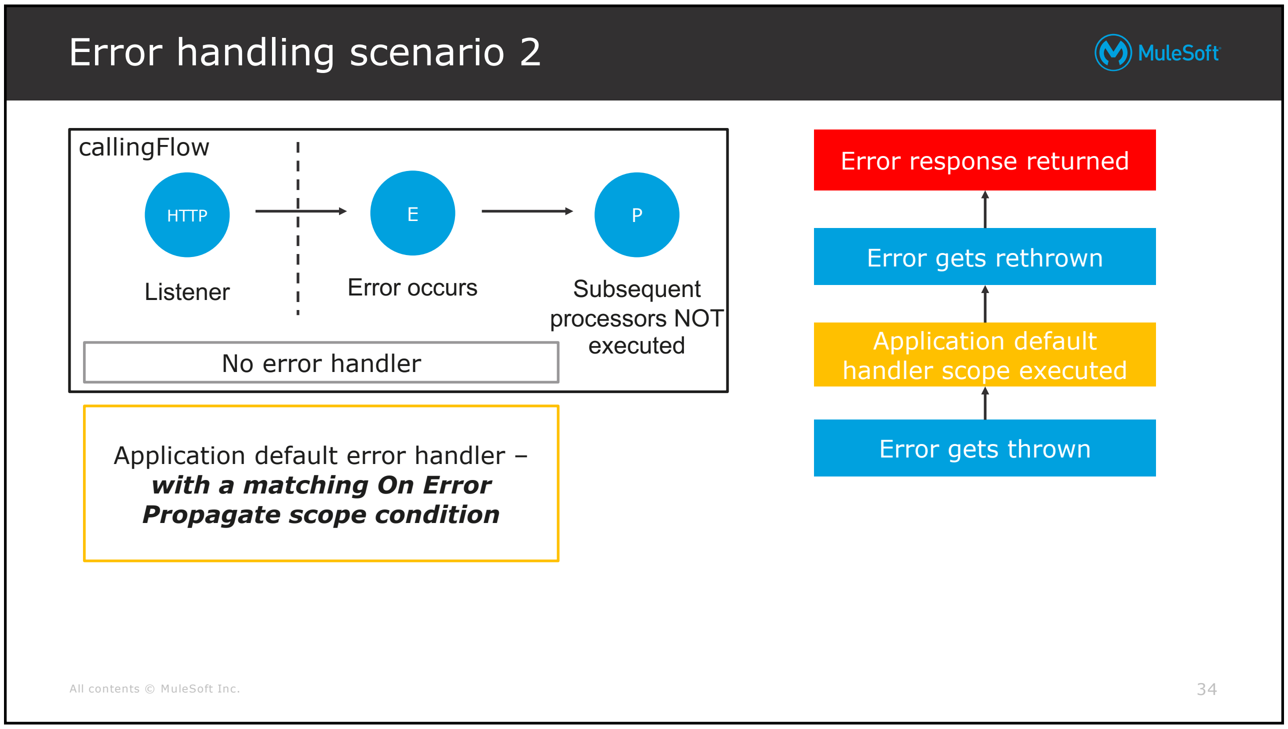


Error handling scenario 1

MuleSoft
callingFlow
HTTP
Listener
E
Error occurs
P
Subsequent processors NOT executed
No error handler
Application default error handler – ***with no matching scope*** *condition*
Error response returned
Error gets rethrown
Mule default error handler executed
Error gets thrown

33

33

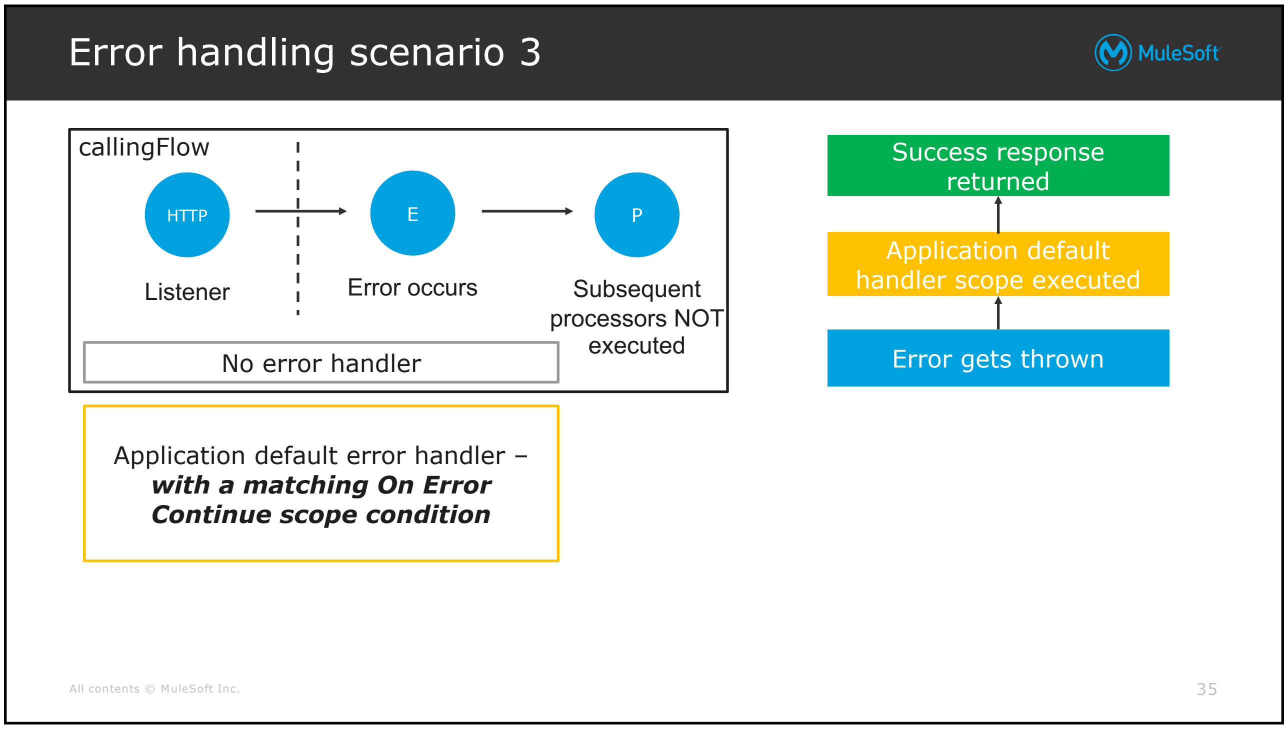


Error handling scenario 2

MuleSoft
callingFlow
HTTP
Listener
E
Error occurs
P
Subsequent processors NOT executed
No error handler
Application default error handler – ***with a matching On Error Propagate scope condition***
Error response returned
Error gets rethrown
Application default handler scope executed
Error gets thrown

34

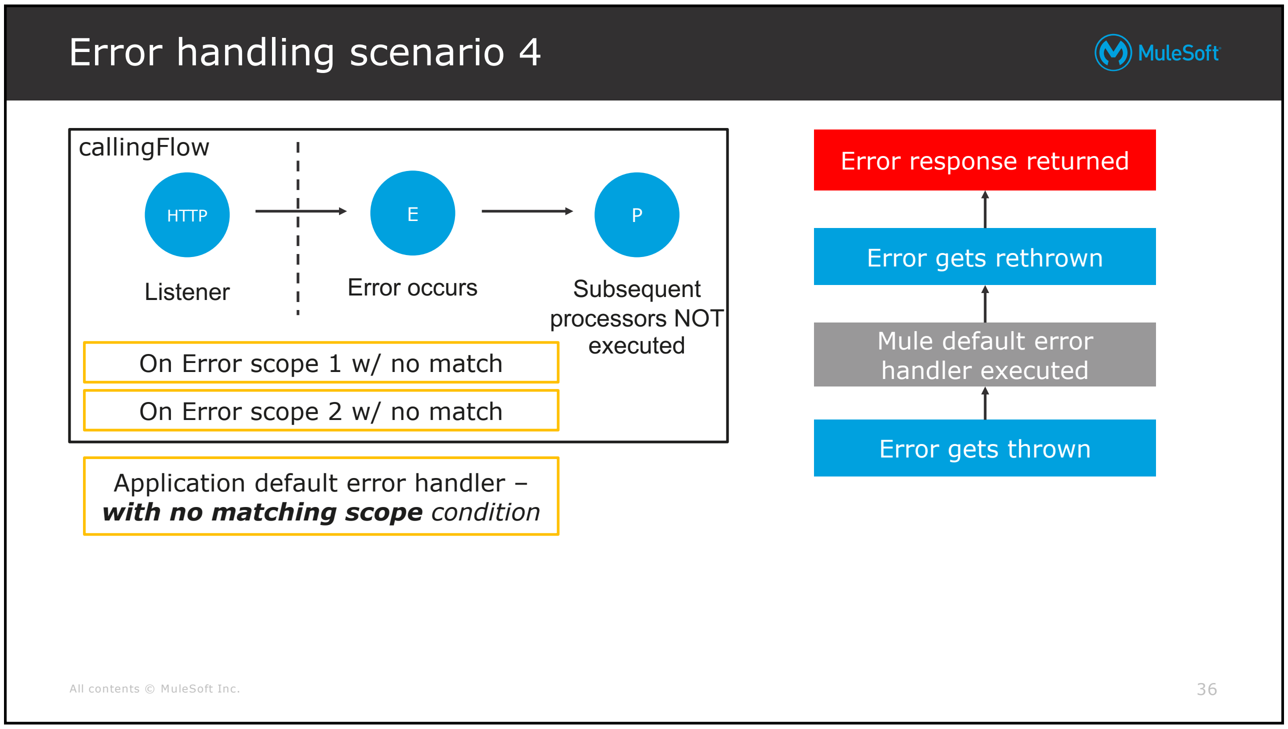
34



Error handling scenario 3

MuleSoft
callingFlow
HTTP
Listener
E
Error occurs
P
Subsequent processors NOT executed
No error handler
Success response returned
Application default handler scope executed
Error gets thrown

Application default error handler – ***with a matching On Error Continue scope condition***

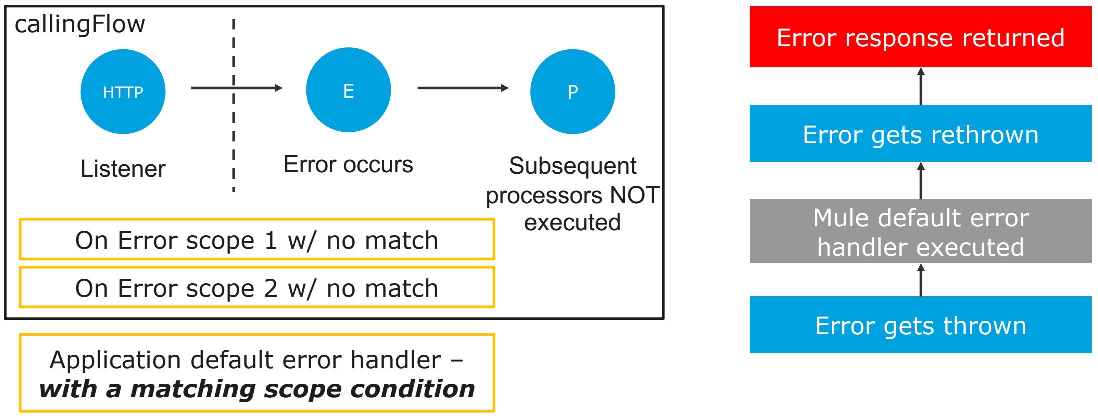


Error handling scenario 4

MuleSoft
callingFlow
HTTP
Listener
E
Error occurs
P
Subsequent processors NOT executed
On Error scope 1 w/ no match
On Error scope 2 w/ no match
Error response returned
Error gets rethrown
Mule default error handler executed
Error gets thrown

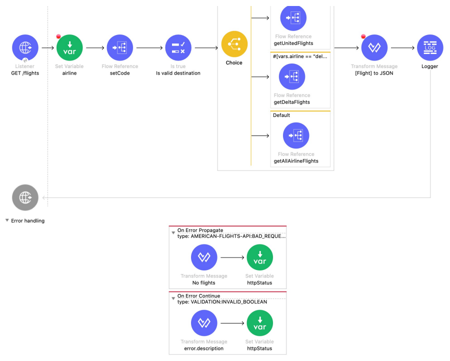
Application default error handler – ***with no matching scope** condition*

Error handling scenario 5


callingFlow
HTTP
Listener
E
Error occurs
P
Subsequent processors NOT executed
On Error scope 1 w/ no match
On Error scope 2 w/ no match
Application default error handler – *with a matching scope condition*
Error response returned
Error gets rethrown
Mule default error handler executed
Error gets thrown

Walkthrough 10-4: Handle errors at the flow level

- Add error handlers to a flow
- Test the behavior of errors thrown in a flow and by a child flow
- Compare On Error Propagate and On Error Continue scopes in a flow
- Set an HTTP status code in an error handler and modify an HTTP Listener to return it



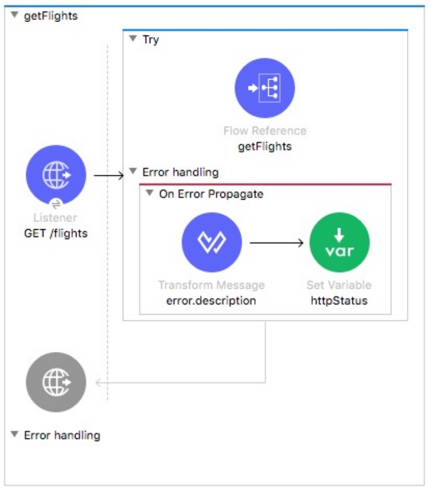
Handling errors at the processor level

39

Handling errors at the processor level

MuleSoft

- For more fine grain error handling of elements within a flow, use the Try scope
- Any number of processors can be added to a Try scope
- The Try scope has its own error handling section to which one or or more error scopes can be added



40

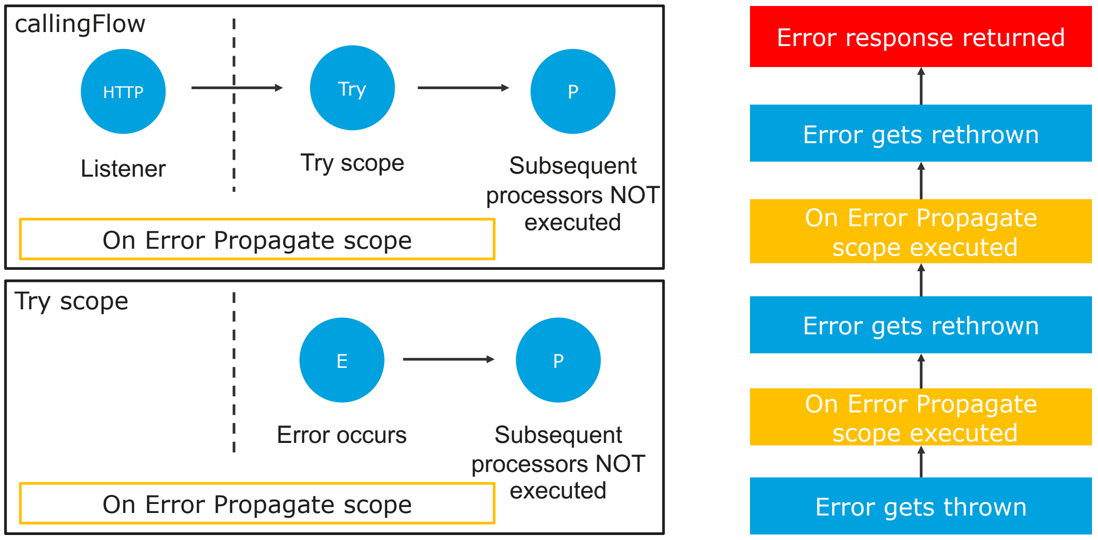
Error handling behavior in the Try scope

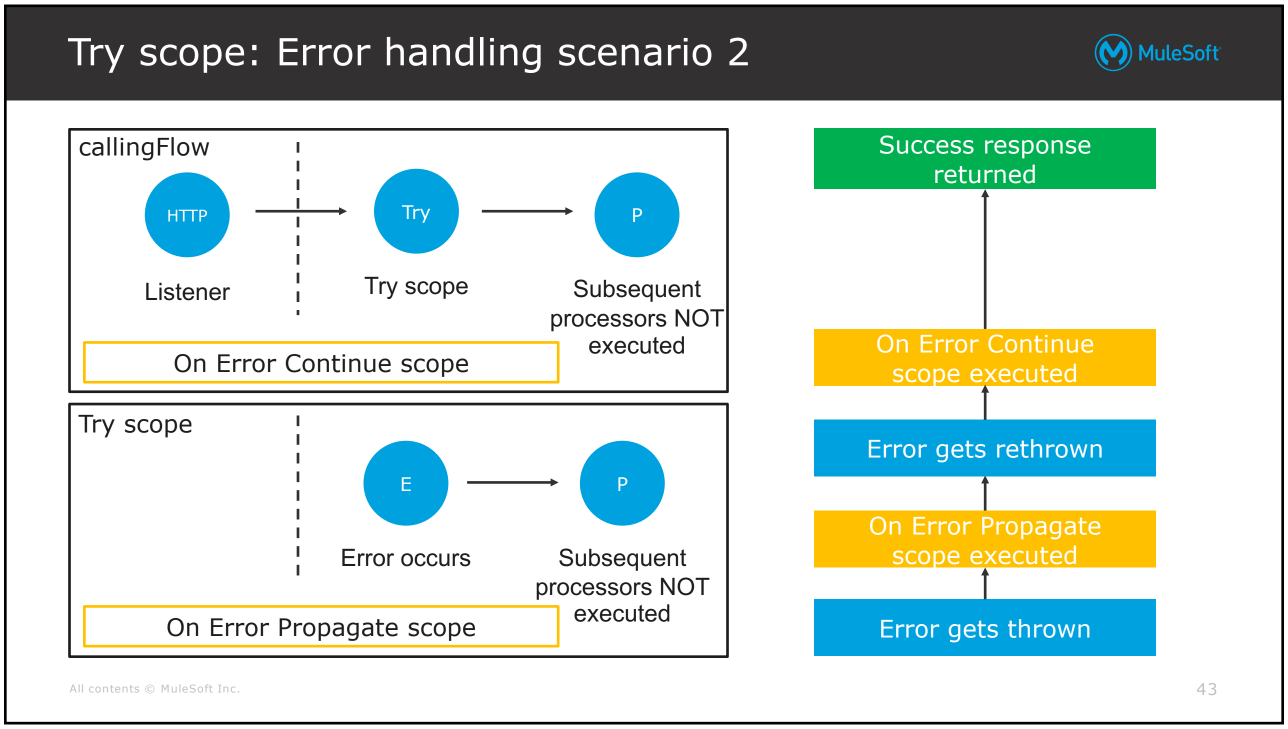
MuleSoft

- **On Error Propagate**
  - All processors in the error handling scope are executed
  - At the end of the scope
    - The rest of the *Try scope* is not executed
    - If a transaction is being handled, it is rolled back
    - The error is rethrown up the execution chain to the parent flow, which handles the error
  - An HTTP Listener returns an *error* response
- **On Error Continue**
  - All processors in the error handling scope are executed
  - At the end of the scope
    - The rest of the *Try scope* is not executed
    - If a transaction is being handled, it is committed
    - The event is passed up to the parent flow, which continues execution
  - An HTTP Listener returns a *successful* response

Try scope: Error handling scenario 1

MuleSoft

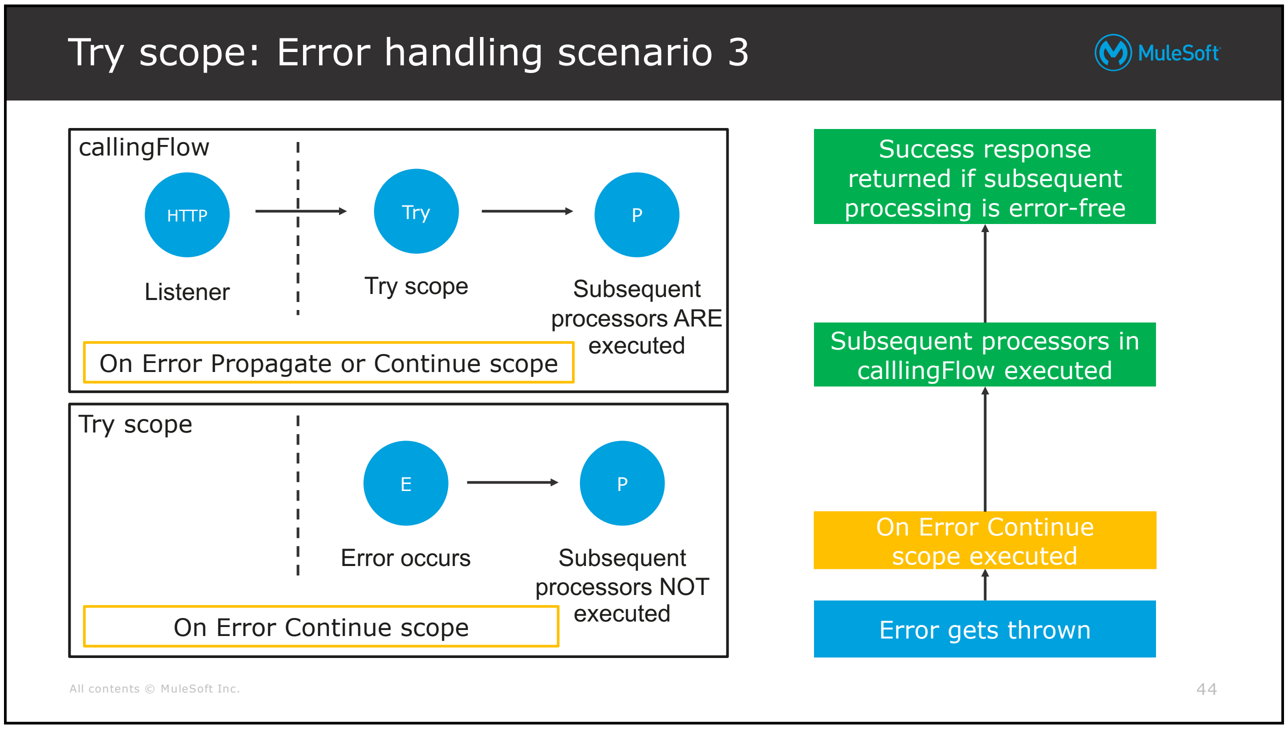




Try scope: Error handling scenario 2

MuleSoft
callingFlow
HTTP
Listener
Try
Try scope
P
Subsequent processors NOT executed
On Error Continue scope
Try scope
E
Error occurs
P
Subsequent processors NOT executed
On Error Propagate scope
Success response returned
On Error Continue scope executed
Error gets rethrown
On Error Propagate scope executed
Error gets thrown

43



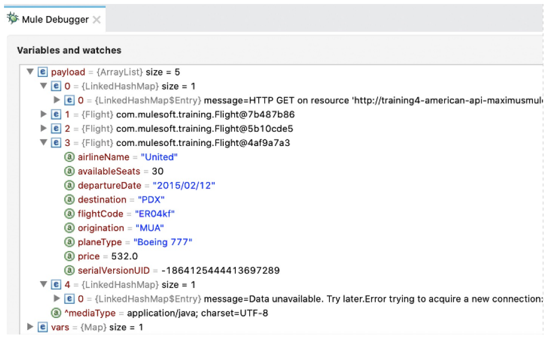
Try scope: Error handling scenario 3

MuleSoft
callingFlow
HTTP
Listener
Try
Try scope
P
Subsequent processors ARE executed
On Error Propagate or Continue scope
Try scope
E
Error occurs
P
Subsequent processors NOT executed
On Error Continue scope
Success response returned if subsequent processing is error-free
Subsequent processors in calllingFlow executed
On Error Continue scope executed
Error gets thrown

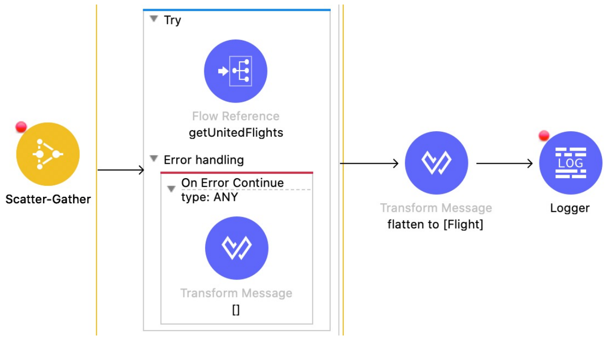
44

Walkthrough 10-5: Handle errors at the processor level

- Wrap the Flow Reference in each branch of a Scatter-Gather in a Try scope
- Use an On Error Continue scope in each Try scope error handler to provide a valid response so flow execution can continue



```
▼ payload = {ArrayList} size = 5
  ▼ 0 = {LinkedHashMap} size = 1
    ▶ 0 = {LinkedHashMap$Entry} message=HTTP GET on resource 'http://training4-american-api-maximusmul
  ▶ 1 = {Flight} com.mulesoft.training.Flight@7b487b86
  ▶ 2 = {Flight} com.mulesoft.training.Flight@5b10cde5
  ▼ 3 = {Flight} com.mulesoft.training.Flight@4af9a7a3
      airlineName = "United"
      availableSeats = 30
      departureDate = "2015/02/12"
      destination = "PDX"
      flightCode = "ER04kf"
      origination = "MUA"
      planeType = "Boeing 777"
      price = 532.0
      serialVersionUID = -1864125444413697289
  ▼ 4 = {LinkedHashMap} size = 1
    ▶ 0 = {LinkedHashMap$Entry} message=Data unavailable. Try later.Error trying to acquire a new connection:
    ^mediaType = application/java; charset=UTF-8
▶ vars = {Map} size = 1
```



All contents © MuleSoft Inc.

Mapping errors to custom error types

Mapping errors for more granular error handling

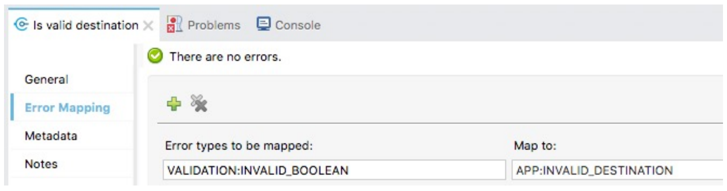
- If an app has two HTTP Request operations that call different REST services, a connectivity failure on either produces the same error
  - Makes it difficult to identify the source of the error in the Mule application logs
- To differentiate between these two errors, you can map each connectivity error to different custom error types
- These custom error types enable you to differentiate exactly where an error occurred

Mapping errors to custom error types

- For each module operation in a flow, each possible error type can be mapped to a custom error type
- You assign a custom namespace and identifier to distinguish them from other existing types within an application
  - Define namespaces related to the particular Mule application name or context
    - CUSTOMER namespace for errors with a customer aggregation API
    - ORDER namespace for errors with an order processing API
  - Do not use existing module namespaces

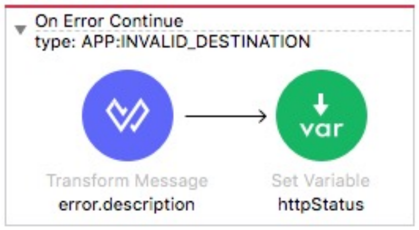
Walkthrough 10-6: Map an error to a custom error type

- Map a module error to a custom error type for an application
- Create an event handler for the custom error type



There are no errors.

Error types to be mapped:	Map to:
VALIDATION:INVALID_BOOLEAN	APP:INVALID_DESTINATION



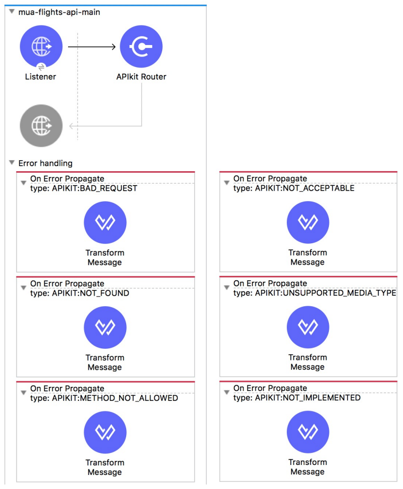
49

Reviewing and integrating with APIkit error handling

50

Error handling generated by APIkit

- By default, interfaces created with APIkit have error handlers with multiple On Error Propagate scopes that handle APIkit errors
  - The error scopes set HTTP status codes and response messages
- The main routing flow has six error scopes
  - APIKIT:BAD_REQUEST > 400
  - APIKIT:NOT_FOUND > 404
  - APIKIT:METHOD_NOT_ALLOWED > 405
  - APIKIT:NOT_ACCEPTABLE > 406
  - APIKIT:UNSUPPORTED_MEDIA_TPYE > 415
  - APIKIT:NOT_IMPLEMENTED > 501


mua-flights-api-main
Listener
APIkit Router
Error handling
On Error Propagate type: APIKIT:BAD_REQUEST
Transform Message
On Error Propagate type: APIKIT:NOT_ACCEPTABLE
Transform Message
On Error Propagate type: APIKIT:NOT_FOUND
Transform Message
On Error Propagate type: APIKIT:UNSUPPORTED_MEDIA_TYPE
Transform Message
On Error Propagate type: APIKIT:METHOD_NOT_ALLOWED
Transform Message
On Error Propagate type: APIKIT:NOT_IMPLEMENTED
Transform Message

51

Integrating with APIkit error handling

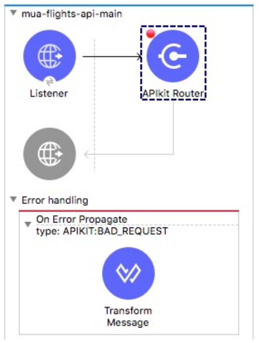
- You can modify the APIkit error scopes and add additional scopes
- You also need to make sure the error handling in the application works as expected with the new interface router
  - **On Error Continue**
    - Event in implementation is not passed back to main router flow
  - **On Error Propagate**
    - Error in implementation is propagated to main router flow
      - Lose payload and variables

52

52

Walkthrough 10-7: Review and integrate with APIkit error handlers

- Review the error handlers generated by APIkit
- Review settings for the APIkit Router and HTTP Listener in the APIkit generated interface
- Connect the implementation to the interface and test the error handling behavior
- Modify implementation error scopes so they work with the APIkit generated interface



58

53

Handling system errors

54

Applications can have two types of errors

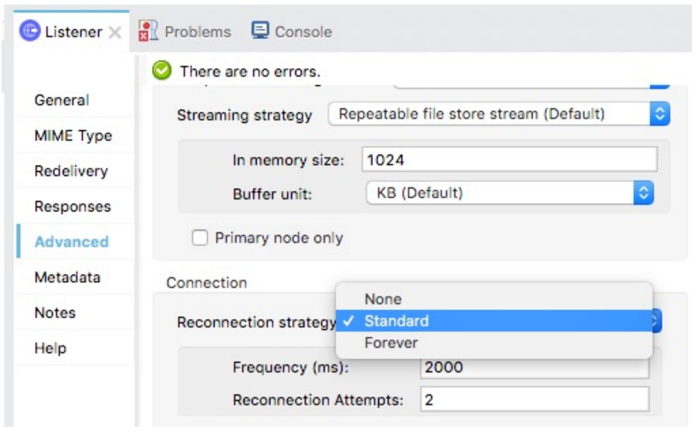
- Messaging errors
  - Thrown within a flow whenever a Mule event is involved
- System errors
  - Thrown at the system-level when *no* Mule event is involved
  - Errors that occur
    - During application start-up
    - When a connection to an external system fails
  - Handled by a system error handling strategy
    - Non configurable
    - Logs the error and for connection failures, executes the reconnection strategy

55

55

Reconnection strategies

- Set for a connector (in Global Elements Properties) or for a specific connector operation (in Properties view)



There are no errors.

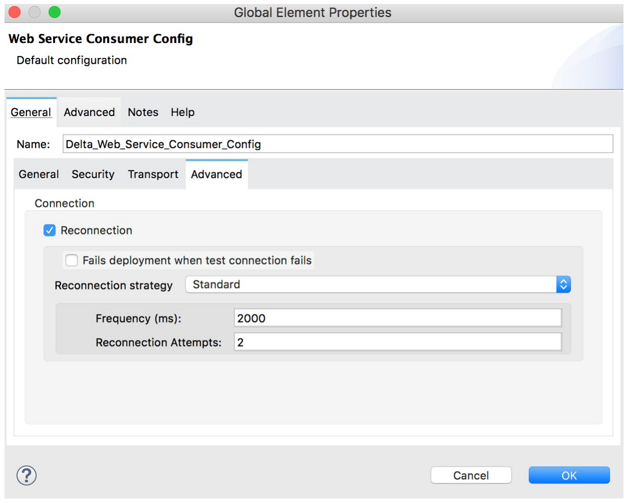
Streaming strategy: Repeatable file store stream (Default)
In memory size: 1024
Buffer unit: KB (Default)
Primary node only
Connection
Reconnection strategy: Standard
Frequency (ms): 2000
Reconnection Attempts: 2

56

56

Walkthrough 10-8: Set a reconnection strategy for a connector

- Set a reconnection strategy for the Web Service Consumer connector



Name: Delta_Web_Service_Consumer_Config

Connection

☑ Reconnection

☐ Fails deployment when test connection fails

Reconnection strategy: Standard

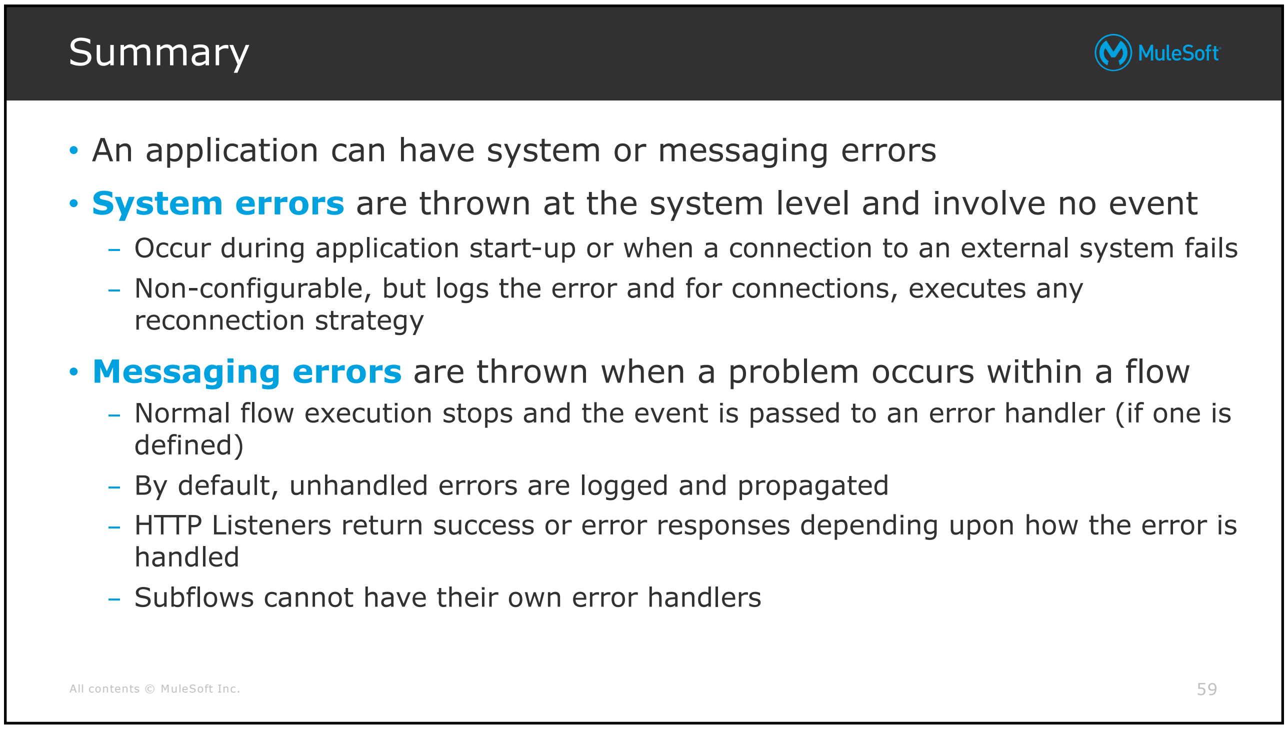
Frequency (ms): 2000

Reconnection Attempts: 2

57

Summary

58

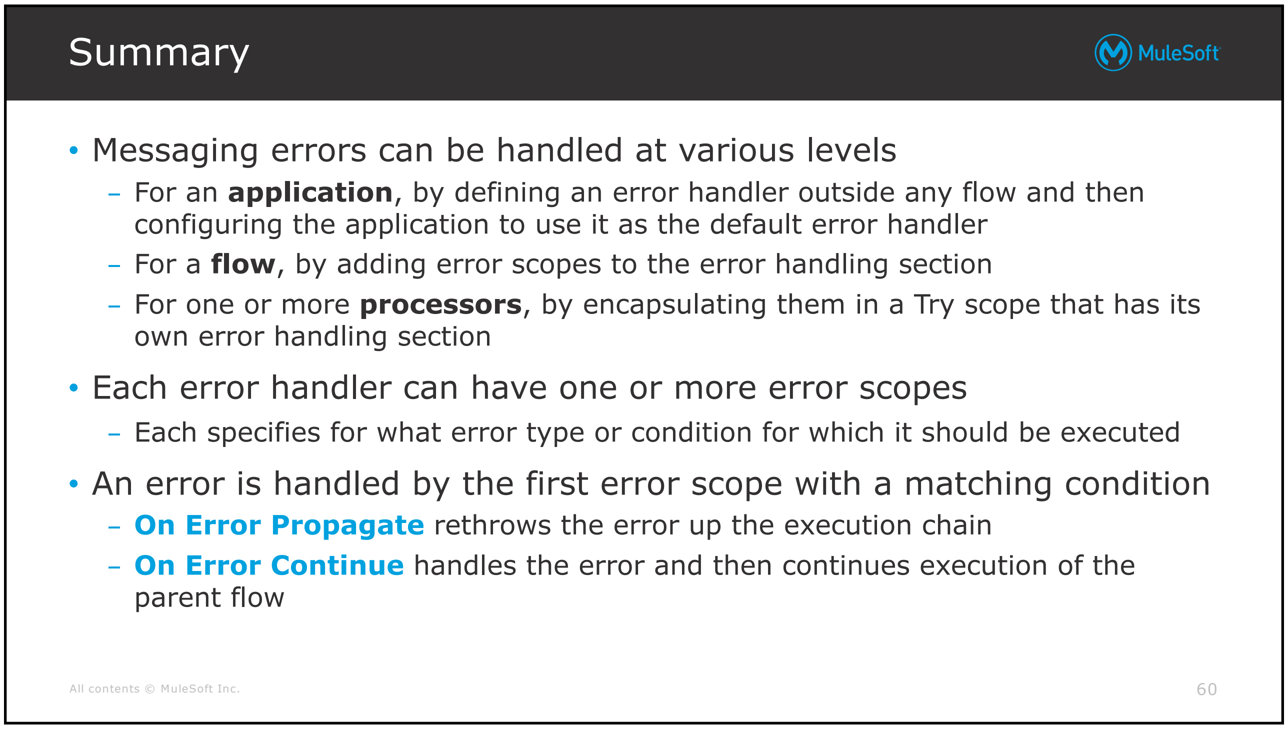


Summary

- An application can have system or messaging errors
- **System errors** are thrown at the system level and involve no event
  - Occur during application start-up or when a connection to an external system fails
  - Non-configurable, but logs the error and for connections, executes any reconnection strategy
- **Messaging errors** are thrown when a problem occurs within a flow
  - Normal flow execution stops and the event is passed to an error handler (if one is defined)
  - By default, unhandled errors are logged and propagated
  - HTTP Listeners return success or error responses depending upon how the error is handled
  - Subflows cannot have their own error handlers

59

59

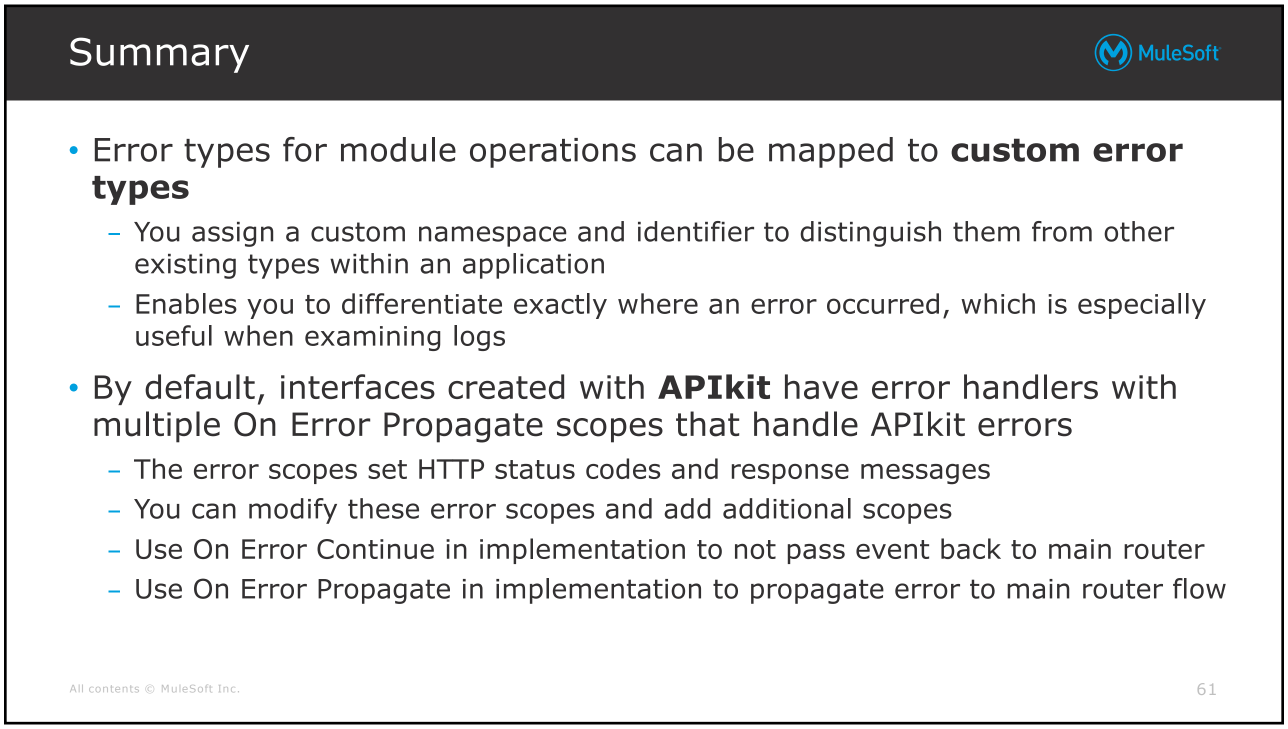


Summary

- Messaging errors can be handled at various levels
  - For an **application**, by defining an error handler outside any flow and then configuring the application to use it as the default error handler
  - For a **flow**, by adding error scopes to the error handling section
  - For one or more **processors**, by encapsulating them in a Try scope that has its own error handling section
- Each error handler can have one or more error scopes
  - Each specifies for what error type or condition for which it should be executed
- An error is handled by the first error scope with a matching condition
  - **On Error Propagate** rethrows the error up the execution chain
  - **On Error Continue** handles the error and then continues execution of the parent flow

60

60



Summary

- Error types for module operations can be mapped to **custom error types**
  - You assign a custom namespace and identifier to distinguish them from other existing types within an application
  - Enables you to differentiate exactly where an error occurred, which is especially useful when examining logs
- By default, interfaces created with **APIkit** have error handlers with multiple On Error Propagate scopes that handle APIkit errors
  - The error scopes set HTTP status codes and response messages
  - You can modify these error scopes and add additional scopes
  - Use On Error Continue in implementation to not pass event back to main router
  - Use On Error Propagate in implementation to propagate error to main router flow