

# *Creating Custom Interfaces to Stream Objects as JSON*

Open **demo-eureka-fastpass-console** application

## 1. Create an interface and add Message Channels

By default, the operation name is given to Channel name in RabbitMQ.

### **TollSource.java**

```
package eureka.demo;

import org.springframework.cloud.stream.annotation.Output;
import org.springframework.messaging.MessageChannel;

public interface TollSource {

    @Output("fastpassTollChannel")
    MessageChannel fastpassToll();

    @Output("standardTollChannel")
    MessageChannel standardToll();

}
```

## 2. Use the TollSource Interface to publish messages to appropriate operation channel

Open TollPublisher class and create a new method to write Java object in the stream as JSON data

***Comment out the method of earlier Use case***

## TollPublisher.java

```
package eureka.demo;

import
org.springframework.cloud.stream.annotation.EnableBinding;
import org.springframework.cloud.stream.messaging.Source;
import org.springframework.context.annotation.Bean;
import
org.springframework.integration.annotation.InboundChannelAdapter
;
import org.springframework.integration.annotation.Poller;
import org.springframework.integration.core.MessageSource;
import org.springframework.integration.support.MessageBuilder;

//@EnableBinding(Source.class)
@EnableBinding(TollSource.class)
public class TollPublisher {

    /*@InboundChannelAdapter(channel = Source.OUTPUT)
    public String sendTollCharge() {

        return "{station:\"20\", customerid:\"100\",
timestamp:\"2019-11-09-12T03:15:00\"}";
    }*/

    @Bean
    @InboundChannelAdapter(channel = "fastpassTollChannel",
poller=@Poller(fixedDelay = "2000"))
    public MessageSource<Toll> sendTollCharge() {

        return () -> MessageBuilder.withPayload(new Toll("20",
"100", "2019-11-09T12:04:00")).build();

    }

    class Toll {

        public String stationId;
        public String customerId;
```

```

        public String timestamp;

        public Toll(String stationId, String customerId,
String timestamp) {
            this.stationId = stationId;
            this.customerId = customerId;
            this.timestamp = timestamp;
        }
    }
}

```

3. The data being sent to the channel is JSON format. So, add these properties

*Comment out earlier entry of destination*

**application.properties**

```
#spring.cloud.stream.bindings.output.destination = fastpasstoll
```

```
spring.cloud.stream.bindings.fastpasstollchannel.destination =
fastpasstoll
```

```
spring.cloud.stream.default.contentType = application/json
```

START **Eureka Server**

4. START **demo-eureka-fastpass-console**

5. START **demo-eureka-toll-intake**


6. OUTPUT

```

40
41 }
42
demo-eureka-toll-intake - DemoEurekaTollIntakeApplication [Spring Boot App] /Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java
2019-11-10 17:08:19.172 INFO 1418 --- [main] o.s.i.monitor.Integ
2019-11-10 17:08:19.216 INFO 1418 --- [main] c.demo.DemoEurekaTc
{"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}
{"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}
{"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}
{"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}

```

7. There are 2 new channels based on TollSource operations

3.7.21Erlang 22.1.5

OverviewConnectionsChannelsExchangesQueuesAdmin

## Exchanges

▼ All exchanges (10)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D			
amq.direct	direct	D			
amq.fanout	fanout	D			
amq.headers	headers	D			
amq.match	headers	D			
amq.rabbitmq.trace	topic	D I			
amq.topic	topic	D			
fastpasstoll	topic	D	1.0/s	2.0/s	
output	topic	D	0.00/s		
standardTollChannel	topic	D			

8. Click on **Queue** Tab

9. Add a Queue to temporarily receive published messages as shown below

RabbitMQ 3.7.21 Erlang 22.1.5

Overview Connections Channels Exchanges **Queues** Admin

### Queues

▼ All queues (2)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Overview			Messages			Message rates			+/-
Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
fastpasstoll.anonymous.-TUGQc4VQ-S-_RygFPpS6g	AD Excl ML	running	0	0	0	1.0/s	1.0/s	1.0/s	
wiretap	D	running	3,222	0	3,222	1.0/s	0.00/s	0.00/s	

▼ Add a new queue

Name:  \*

Durability:

Auto delete:

Arguments:  =

Add Message TTL ? | Auto expire ? | Max length ? | Max length bytes ? | Overflow behaviour ?

Dead letter exchange ? | Dead letter routing key ? | Maximum priority ?

Lazy mode ? Master locator ?

9.1 Enter the Queue Name in **"Name:"** field

9.2 Click on **Add queue** button

9.3 Click on **wiretap** queue

Overview			Messages			Message		
Name	Features	State	Ready	Unacked	Total	incoming		
fastpasstoll.anonymous.-TUGQc4VQ-S-_RygFPpS6g	AD Excl ML	running	0	0	0	1.0/		
wiretap	D	running	3,261	0	3,261	1.0/		

9.4 Under **Bindings** section, add:  
the channel name in **From exchange** field  
“#” in **Routing key** field

to receive messages as shown below

**▼ Bindings**

From	Routing key	Arguments	
(Default exchange binding)			

⇓

This queue

---

Add binding to this queue

From exchange:

fastpasstoll

\*

Routing key:

#

Arguments:

=

String

⌵

Bind

9.5 In the same page, click on Get Message(s) button

**▼ Get messages**

Warning: getting messages from a queue is a destructive action. ?

Ack Mode:

Nack message requeue true

⌵

Encoding:

Auto string / base64

⌵

?

Messages:

1

Get Message(s)

10. You can observe the Payload here

Get Message(s)

Message 1

---

The server reported 3477 messages remaining.

Exchange	fastpasstoll
Routing Key	fastpasstoll
Redelivered	•
Properties	<div>priority: 0</div> <div>delivery_mode: 2</div> <div>headers: contentType: text/plain</div> <div>originalContentType: application/json; charset=UTF-8</div> <div>content_type: text/plain</div>
Payload	<div>71 bytes</div> <div>Encoding: string</div> <div>{"stationId": "20", "customerId": "100", "timestamp": "2019-11-09T12:04:00"}</div>

## *Handle Multiple Stream Listeners based on message headers*

Go to **demo-eureka-fastpass-console** app

11. Create a Random object

**TollPublisher.java**

Random **random** = **new** Random();

12. Change the return statement in the MessageSource method to set header

Rewrite the method as shown below

### **TollPublisher.java**

```
@Bean
@InboundChannelAdapter(channel = "fastpassTollChannel",
poller=@Poller(fixedDelay = "2000"))
public MessageSource<Toll> sendTollCharge() {

    return () -> MessageBuilder.withPayload(new Toll("20",
"100", "2019-11-09T12:04:00")).setHeader("speed",
random.nextInt(8) * 10).build();

}
```

13. Go to **demo-eureka-toll-intake** app

Add 2 stream listener methods to process based on header info

*Comment out existing stream listener method*

### **DemoEurekaTollIntakeApplication.java**

```
@StreamListener(target = Sink.INPUT, condition =
"headers['speed'] > 40")
public void logFast(String msg) {
    System.out.println("FAST - " + msg);
}

@StreamListener(target = Sink.INPUT, condition =
"headers['speed'] <= 40")
public void logSlow(String msg) {
    System.out.println("SLOW - " + msg);
}
```



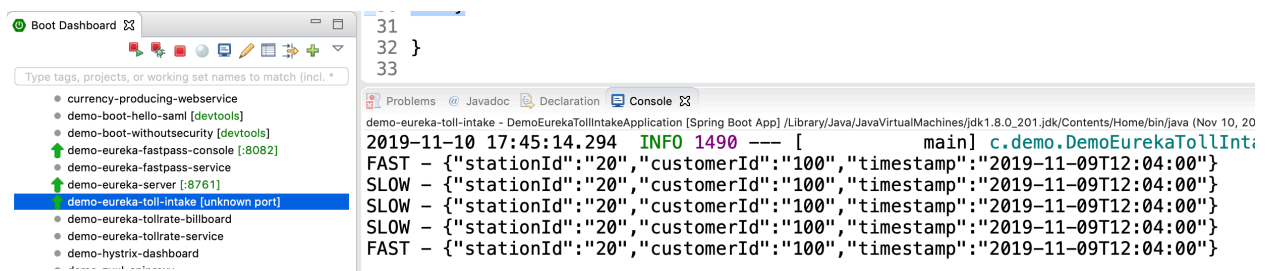
14.

START Eureka Server

START fastpass-console

START toll-intake

OUTPUT



The screenshot shows an IDE interface. On the left, a 'Boot Dashboard' window displays a list of projects. The project 'demo-eureka-toll-intake' is highlighted in blue, with its status shown as '[unknown port]'. Other projects listed include 'currency-producing-webservice', 'demo-boot-hello-saml', 'demo-boot-withoutsecurity', 'demo-eureka-fastpass-console', 'demo-eureka-fastpass-service', 'demo-eureka-server', 'demo-eureka-tollrate-billboard', 'demo-eureka-tollrate-service', and 'demo-hystrix-dashboard'. On the right, a 'Console' window shows the output of the 'demo-eureka-toll-intake' application. The output includes a timestamp '2019-11-10 17:45:14.294', an 'INFO' log level, and a message from 'c.demo.DemoEurekaTollInt:'. The message contains three JSON objects: a 'FAST' object, a 'SLOW' object, and another 'SLOW' object, all with 'stationId': '20', 'customerId': '100', and 'timestamp': '2019-11-09T12:04:00'.

```
31  
32 }  
33  
demo-eureka-toll-intake - DemoEurekaTollIntakeApplication [Spring Boot App] /Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java (Nov 10, 20  
2019-11-10 17:45:14.294 INFO 1490 --- [          main] c.demo.DemoEurekaTollInt:  
FAST - {"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}  
SLOW - {"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}  
SLOW - {"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}  
SLOW - {"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}  
FAST - {"stationId":"20","customerId":"100","timestamp":"2019-11-09T12:04:00"}
```