Lab Exercise

Filters - Lab 3

Not all messages are of interest. That is, there is often a need to filter out some messages that enter a channel. Applications may only want to get messages formatted in a certain way (XML or JSON). Applications may only be interested in certain types of data (new sales messages but not messages about shipments). Other application may be interested in messages with certain message headers (for example when a message was created – only wanting to look at messages created after 5pm).

In this lab, you explore Spring Integration (SI) filters to examine messages in a channel and accept those of interest and discard the others.

Specifically, in this lab you will:

- Implement the SI MessageSelector interface and configure an SI filter.
- Explore and configure a built-in SI XPath filter to sort XML messages.
- Work with a built-in SI Validation filter to week out non-validating XML messages.

Lab solution folder: ExpressSpringIntegration\lab3\lab3-filters-solution & lab3-xml-filters-solution

Scenario – Filter File Messages

In this lab, you explore filters. You will work with two different Eclipse projects to explore filters. In the first project, you filter File messages (messages with File payload) looking for those File names that begin with a specified string. You provide the filter's logic by implementing the MessageSelector interface and configuring the filter in XML.

In the second project, you use two built-in SI filters to weed through XML messages that contain expected element data or don't validate (by XML Schema).

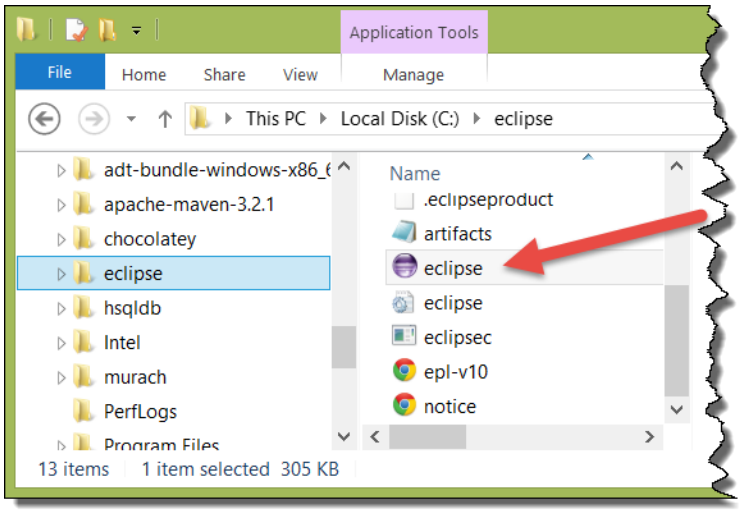
Step 1: Import the Maven Project

A Maven Project containing the base code for a File filtering application has already been created for you. You will use this project to begin your exploration of filters.

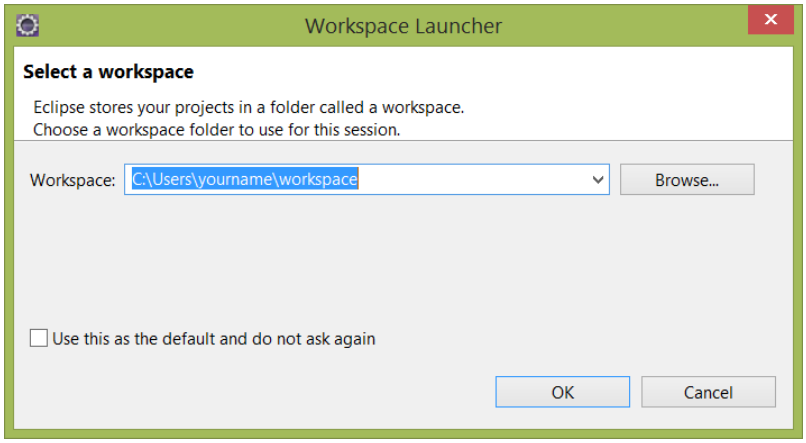
1.1 Start the Eclipse-based IDE. Locate and start Eclipse (or Eclipse-based) IDE.

1.1.1 Locate the Eclipse folder.

1.1.2 Start Eclipse by double clicking on the eclipse.exe icon (as highlighted in the image below).



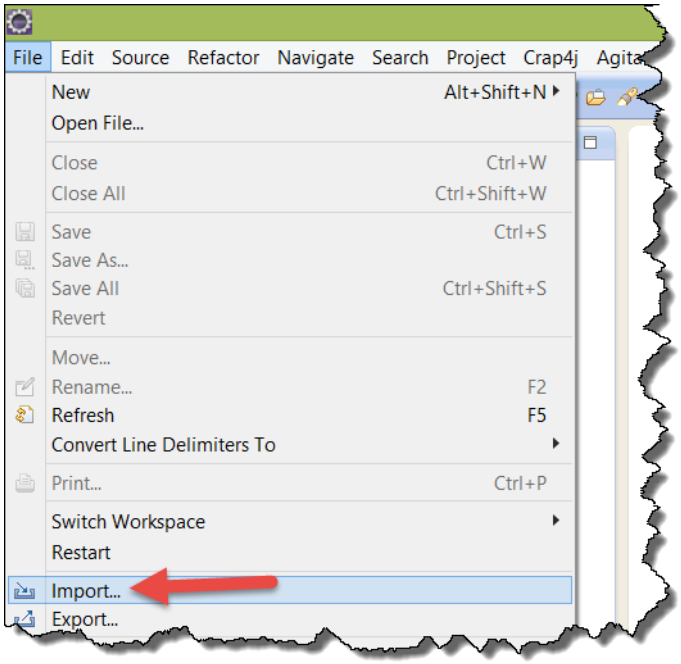
1.1.3 Open your workspace. Type in *C:\Users\<your username >\workspace* and click the *OK* button.



Note: As noted in the past labs, you may use an alternate location for your workspace, but the labs will always reference this location (c:\users\[your username]\workspace) as the default workspace location.

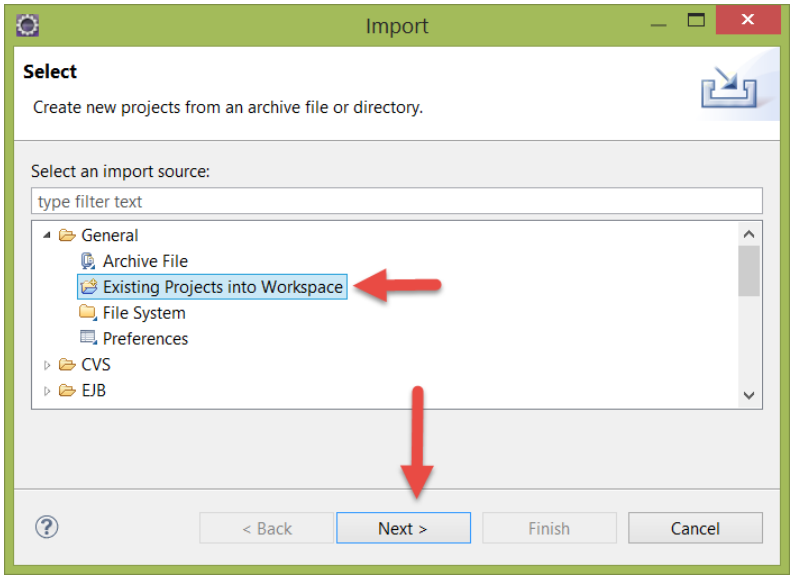
1.2 Import the new Maven Project.

1.2.1 Select File>Import... from the Eclipse menu bar.

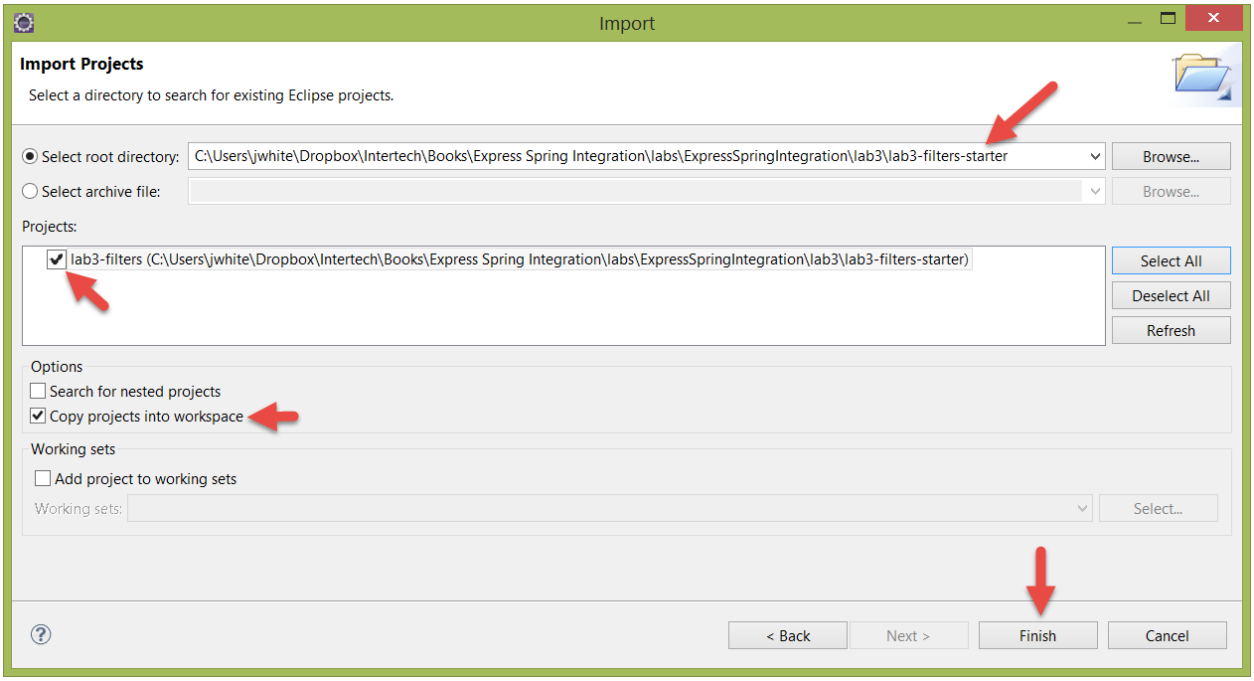


 Rev: 9

1.2.2 Locate and open the General folder in the Import window, and then select *Existing Projects into Workspace Project* from the options. Now push the *Next>* button.

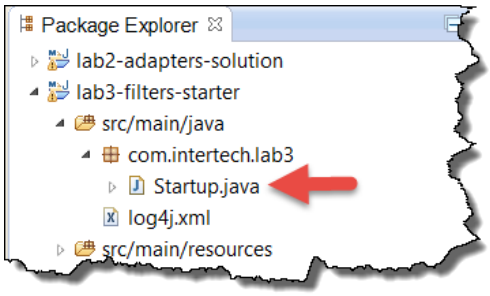


1.2.3 In the "Import" window, use the *Browse...* button to locate the lab3-filters-starter project folder located in ExpressSpringIntegration\lab3 (this folder is located in the lab downloads). Make sure the project is selected, and the *"Copy projects into workspace"* checkbox is also checked before you hit the *Finish* button (as shown below).

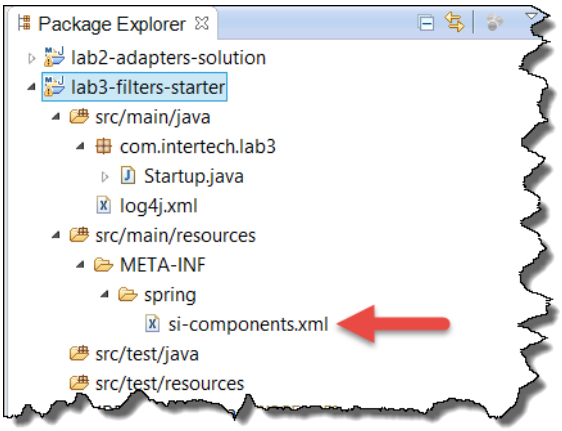


1.3 Explore the project. Examine the project for the Spring Integration components that are already present.

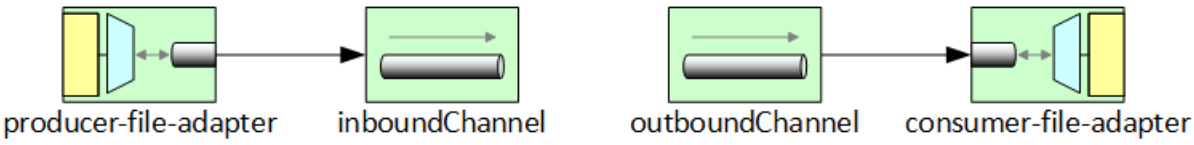
1.3.1 Examine the Startup.java. Expand the src/main/java folder in the Project Explorer view, and open the Startup.java file by double clicking on it. As in past labs, the Startup.java class is used to put the application in an infinite loop so that the SI components can do their work.



1.3.2 Examine the SI configuration. Expand the src/main/resources/META-INF/spring folder in the Project Explorer view, and open si-components.xml file by double clicking on it. The si-components.xml file contains the Spring configuration that includes the definitions for several SI components already.



1.3.3 In particular, note that a file inbound and outbound adapter are already defined in the configuration file. In addition, two channels are defined – as represented in the EIP diagram below.



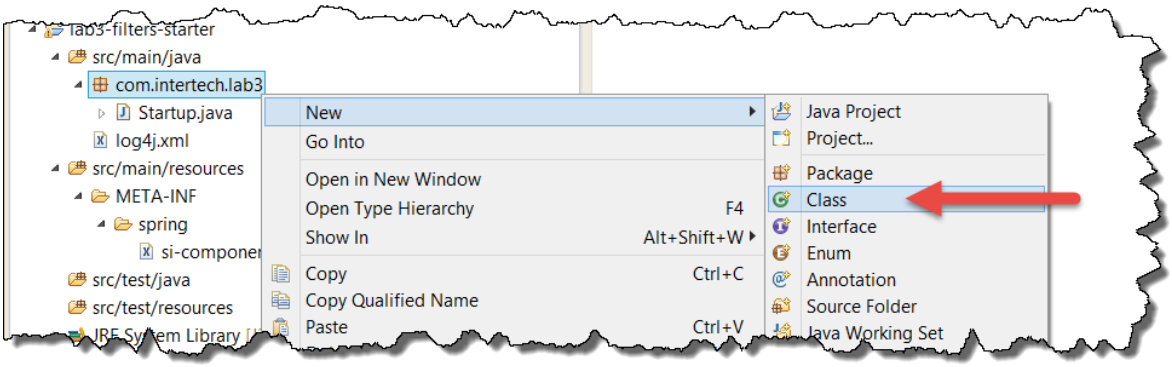
Note that no connection exists between the inbound and outbound channels at this time.

Step 2: Create the MessageSelector

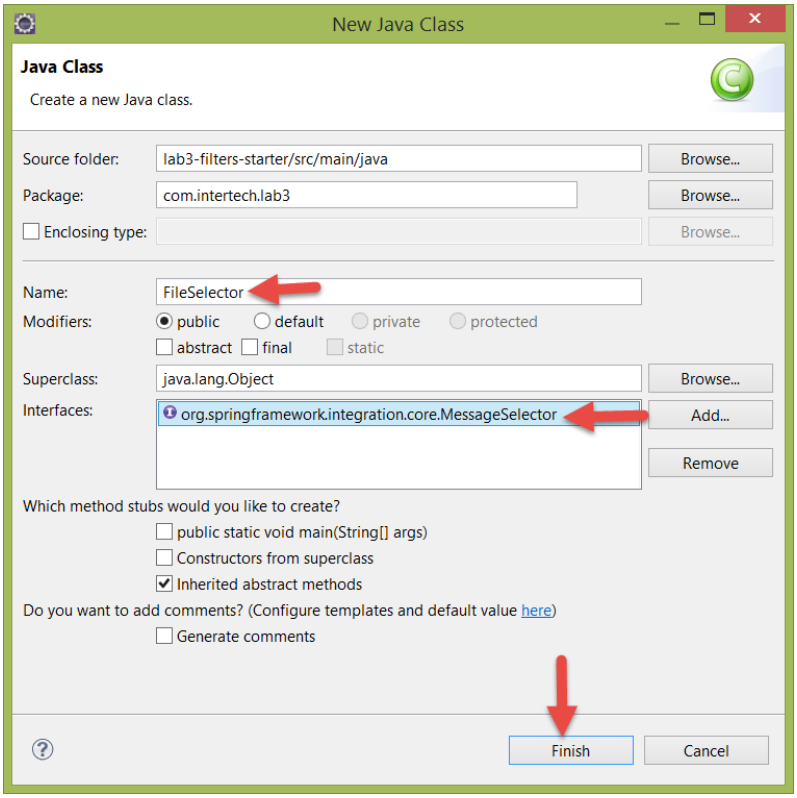
Filters use a MessageSelector under the covers to accept or reject messages from a message channel. In other words, the MessageSelector provides the business logic for determining which messages are filtered and those that are not. In this step, you create a MessageSelector – weeds out File messages where the File name does not start with a particular string.

2.1 Create the FileSelector.

2.1.1 Locate the com.intertech.lab3 package, right click on it and select New > Class from the resulting menu.



2.1.2 In the New Java Class window, enter FileSelector as the class name and add the org.springframework.integration.core.MessageSelector to the Interfaces list.



Notice that the wizard creates an accept(Message<?>) method that must be implemented for the interface.

2.2 Code the accept( ) method. Enter the following code for the method.

```
public boolean accept(Message<?> message) {
  if (message.getPayload() instanceof File
      && ((File) message.getPayload()).getName().startsWith("msg")) {
    return false;
  }
  return true;
}
```

Note that the method checks for the type of payload in the message. If it is a File payload and starts with the filename of "msg", the message is rejected.

Note: if you get stuck or feel like not typing in all the code yourself, you will find a working copy of the final FileSelector file at ExpressSpringIntegration\lab3\lab3-filters-solution.

2.2.1 Add the java.io.File import to the list of imports in the class.

```
import java.io.File;
import org.springframework.integration.core.MessageSelector;
import org.springframework.messaging.Message;
```

2.2.2 Save the class and make sure there are not compile errors.

Step 3: Add the Filter to the SI Components

Add a new filter component to the Spring Integration XML configuration. The filter will use the MessageSelector implementation you just created.

3.1 Add the FileSelector bean.

3.1.1 Locate the si-components.xml file in src/main/resource/META-INF/spring and open it by double clicking on the file.

3.1.2 Add a Spring bean of the FileSelector type, as shown below, into the configuration.

```
<bean id="selector" class="com.intertech.lab3.FileSelector" />
```

3.2 Add a SI filter component.

3.2.1 In the same si-components.xml file, add a SI filter that takes a message from the inbound channel, filters it and puts accepted messages in the outbound channel.

```
<int:filter input-channel="inboundChannel"
output-channel="outboundChannel" ref="selector" />
```

Note that the filter reference the selector bean.

Note: if you get stuck or feel like not typing in all the code yourself, you will find a working copy of the final si-component.xml file at ExpressSpringIntegration\lab3\lab3-filters-solution\si-component.xml.

3.2.2 Save the configuration file and make sure there are no errors in the file.

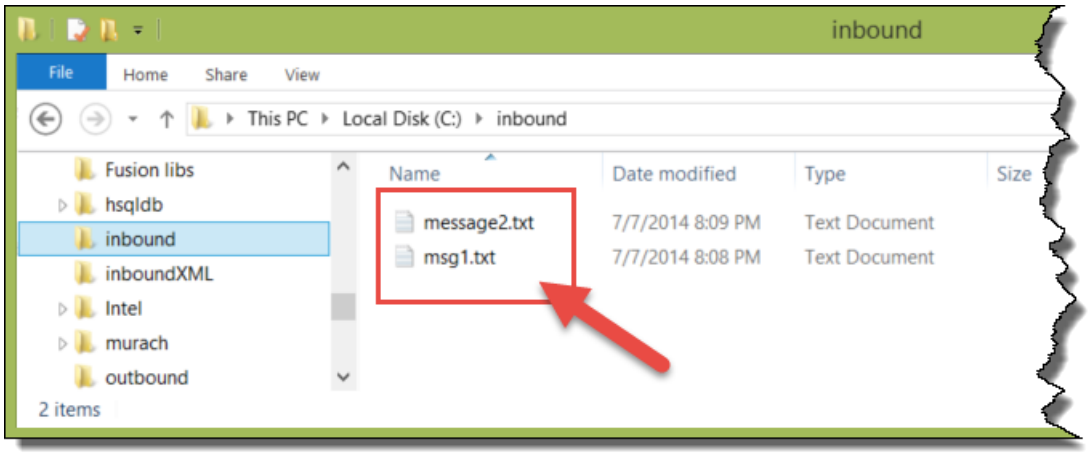
Step 4: Test the Filter

Add messages into the inbound file folder and then test the application to see the filter do its job.

4.1 Add the files to the inbound message folder.

4.1.1 In the si-components.xml, find the producer-file-adapter. Note the location of the directory. It is set, by default, to file:c://inbound. This is the location where messages will be taken into the application by the adapter. Create this message folder - changing the location to suit your needs and your file system (change the producer-file-adapter to reflect your location).

4.1.2 Add files into the inbound message folder. Make sure some (at least one) file has a name that begins with "msg". Make sure some (at least one) file has a name that does not begin with "msg".

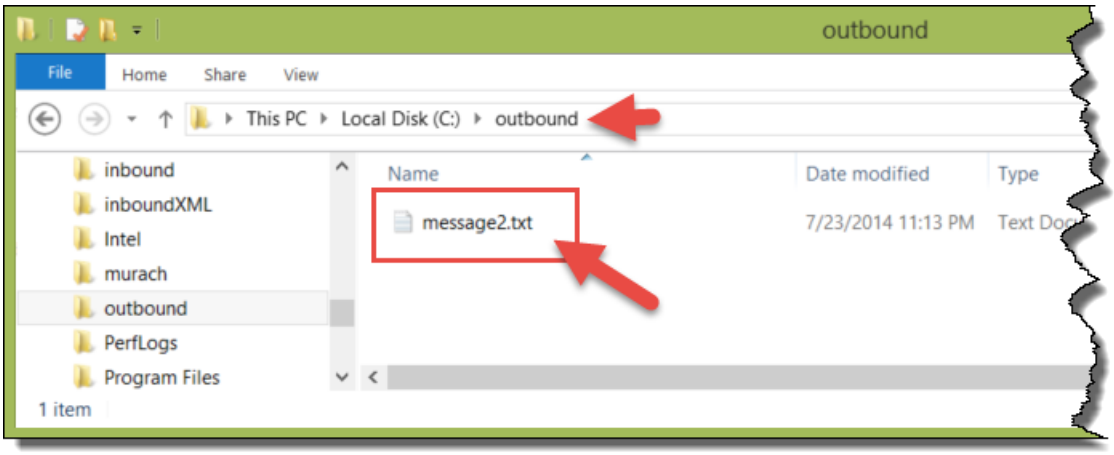


4.1.3 In the si-components.xml, also locate the outbound-channel-adapter. Note the location of the directory. It is set, by default, to file:c://outbound. This is the location where accepted messages will be deposited. Create the outbound folder in your file system. You can use a different folder name if you are also willing to change the adapter's directory name.

4.2 Test the application. Test the application to see the Files in the inbound file folder are filtered based on file name.

4.2.1 Locate the Startup.java file in the source folder. Right click on file and select Run As > Java Application from the resulting menu. Nothing should display in the Console view.

4.2.2 Using a Windows Explorer, open the folder specified as the File output adapter's directory (in step 4.1.3 above). See that the accepted messages are now in the folder. Rejected messages (files with names that begin with "msg") have been filtered and not in the folder.

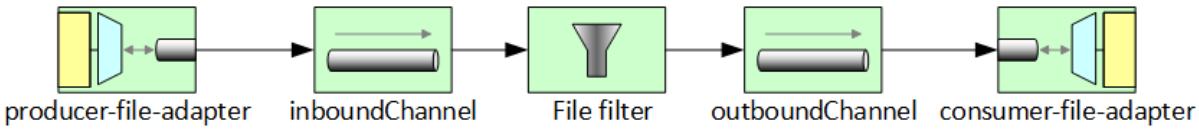


4.3 Stop the application. Recall the application is running in an infinite loop to allow for the constant publishing and consuming of messages. Stop the application now.

4.3.1 In the Console view, click on the red square to terminate the Startup application.

4.3.2 The Console view should now indicate that the application is "<terminated>". Clear the Console view by clicking on the Clear Console icon.

4.3.3 Below is the EIP model for your completed application.



Scenario – Filter XML Messages with XPath

XML is popular in many messaging systems to include Spring Integration. XML describes the data while also providing the data. When dealing with many messages, it can be inconvenient to convert the XML messages to objects or even strings and parse the data for relevant messages. Spring Integration already comes with a built-in XPath filter that allows you to define a filter with an XPath expression to select only messages of interest.

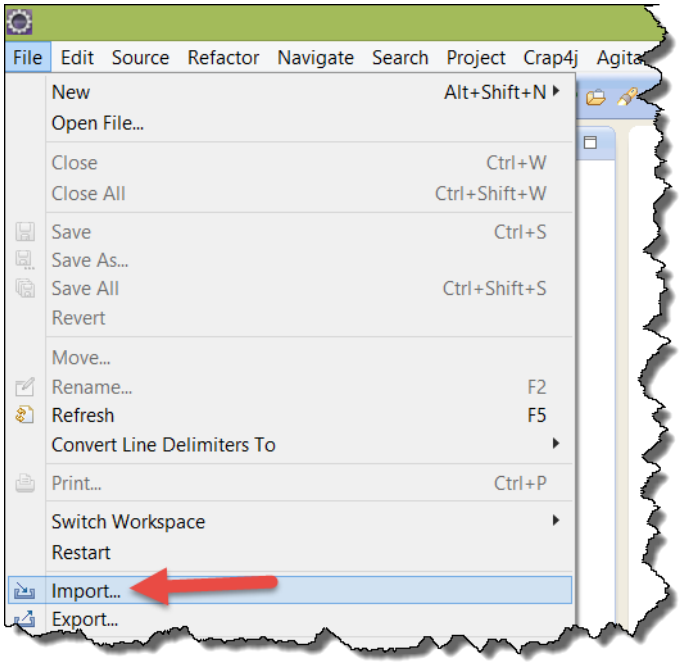
Step 5: Import the Maven Project

XML is popular in many messaging systems to include Spring Integration. XML describes the data while also providing the data. When dealing with many messages, it can be inconvenient to convert the XML messages to objects or even strings and parse the data for relevant messages. Spring Integration already comes with a built-in XPath filter that allows you to define a filter with an XPath expression to select only messages of interest.

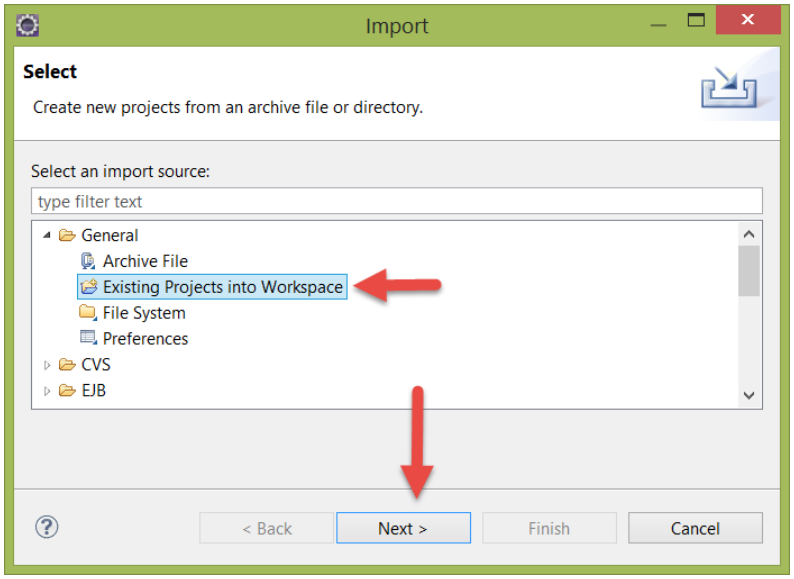
A Maven Project containing the base code for an XPath filtering application has already been created for you. You will use this project to begin your exploration of filters.

5.1 Import the new Maven Project.

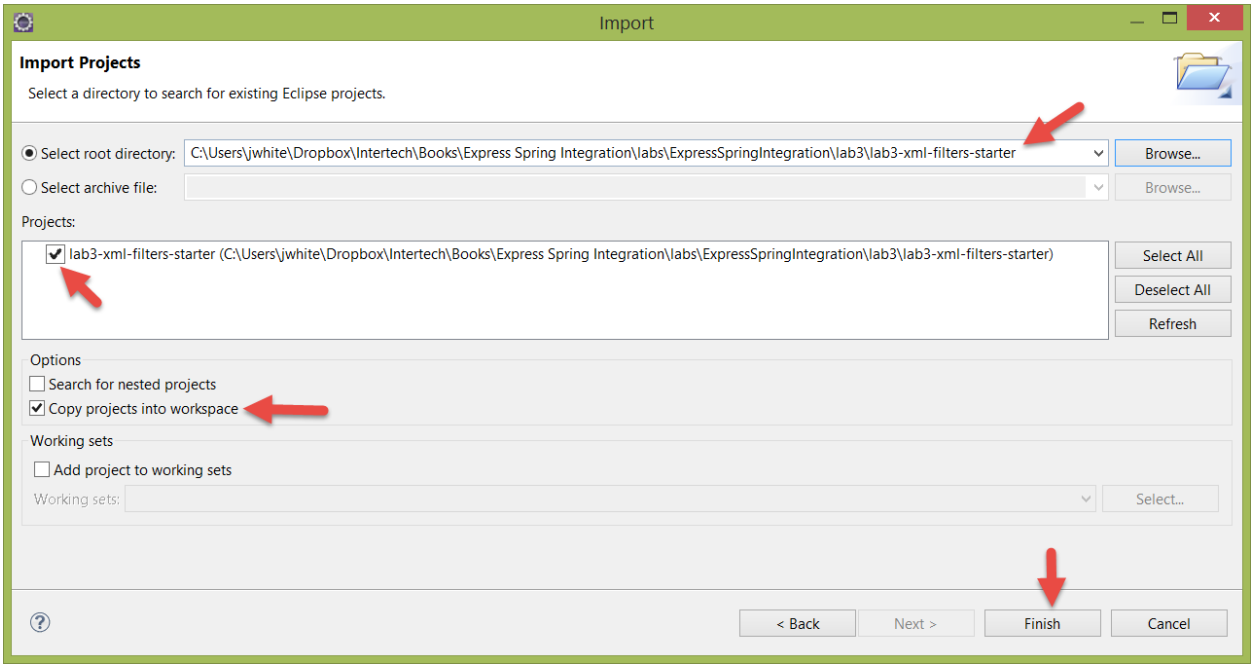
5.1.1 Select File>Import... from the Eclipse menu bar.



5.1.2 Locate and open the General folder in the Import window, and then select *Existing Projects into Workspace Project* from the options. Now push the *Next>* button.

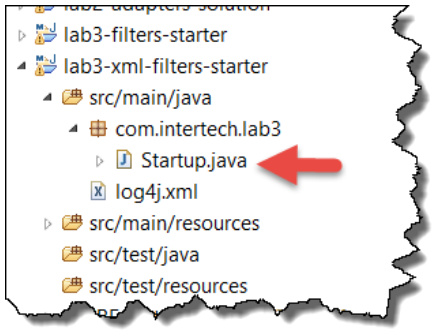


5.1.3 In the "Import" window, use the *Browse…* button to locate the lab3-xml-filters-starter project folder located in ExpressSpringIntegration\lab3 (this folder is located in the lab downloads). Make sure the project is selected, and the *"Copy projects into workspace"* checkbox is also checked before you hit the *Finish* button (as shown below).

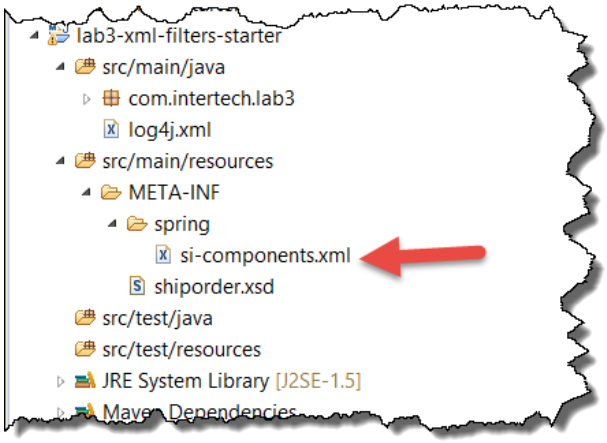


5.2 Explore the project. Examine the project for the Spring Integration components that are already present.

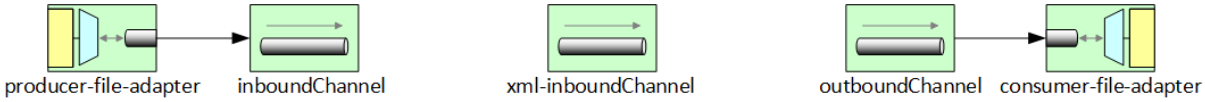
5.2.1 Examine the Startup.java. Expand the src/main/java folder in the Project Explorer view, and open the Startup.java file by double clicking on it. As in past labs, the Startup.java class is used to put the application in an infinite loop so that the SI components can do their work.



5.2.2 Examine the SI configuration. Expand the src/main/resources/META-INF/spring folder in the Project Explorer view, and open si-components.xml file by double clicking on it. The si-components.xml file contains the Spring configuration that includes the definitions for several SI components already.



5.2.3 In particular, note that again, a file inbound and outbound adapter are already defined in the configuration file. In addition, three channels are defined – as represented in the EIP diagram below.

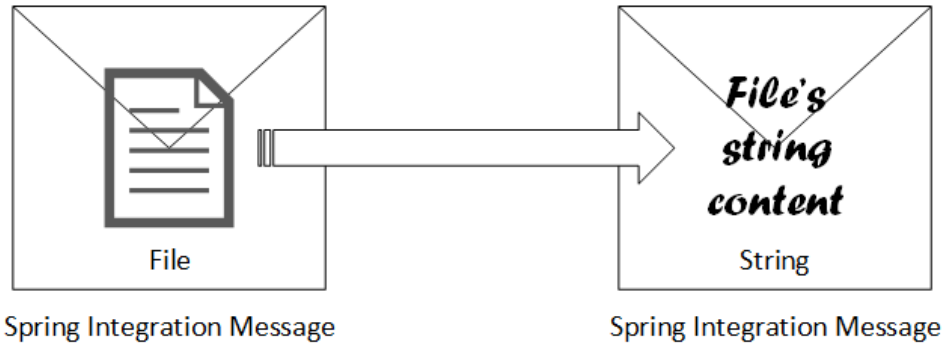


Note that no connection exists between the inbound and outbound channels exist at this time.

Step 6: Add an XPath filter

6.1 Add the file to string transformer.

You will learn more about transforms in an upcoming tutorial. There are many types of transformer message endpoints. Spring provides one called a file-to-string transformer. This transformer turns a file message (that is a SI message with a file as its payload) into a string message (a SI message with a string as its payload). The string that fills the resulting message is from the contents of the file.



Why is this important? In order for an XPath filter to accept or reject the XML in a message, it must first be able to access the XML contents rather than the file that holds the XML.

6.1.1 Locate the si-components.xml file in src/main/resource/META-INF/spring and open it by double clicking on the file.

6.1.2 Add the file to string transformer to the configuration. Attach the transformer to the inboundChannel message channel and the outbound xml-inboundChannel.

```
<int-file:file-to-string-transformer
  id="file-2-string-transformer" input-channel="inboundChannel"
  output-channel="xml-inboundChannel" charset="UTF-8" />
```

Note: if you get stuck or feel like not typing in all the code yourself, you will find a working copy of the final si-component.xml file at ExpressSpringIntegration\lab3\lab3-xml-filters-solution\si-component.xml.

6.2 Add a SI XPath filter component. While you could build a custom filter using a MessageSelector implementation (like you did in the first part of this lab), SI already comes with a built-in XPath filter. That way, you can filter XML messages without having to write code to parse the message and digest its contents with your own complex String manipulation code.

6.2.1 Note that the configuration file already contains the Spring Integration XML namespace at the top of the file.

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:int="http://www.springframework.org/schema/integration"
  xmlns:int-file=
    "http://www.springframework.org/schema/integration/file"
  xmlns:int-mail=
    "http://www.springframework.org/schema/integration/mail"
  xmlns:int-xml=
    "http://www.springframework.org/schema/integration/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:int-stream=
    "http://www.springframework.org/schema/integration/stream"
  xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/integration
http://www.springframework.org/schema/integration/spring-
integration.xsd
http://www.springframework.org/schema/integration/stream
http://www.springframework.org/schema/integration/stream/spring-
integration-stream.xsd
http://www.springframework.org/schema/integration/file
http://www.springframework.org/schema/integration/file/spring-
integration-file.xsd
http://www.springframework.org/schema/integration/xml
http://www.springframework.org/schema/integration/xml/spring-
integration-xml.xsd">
```

6.2.2 Add an SI XPath expression component that defines the content in the XML message you are looking for. This is not the filter itself, but is the expression component that defines the XPath expression that the filter will use to filter. In this case, the expression says you are looking for a <country> element in the message that contains the content of "USA".

```
<int-xml:xpath-expression id="filterXpathExp"
  expression="//country='USA'"></int-xml:xpath-expression>
```

Note: if you are looking for more help on XPath, see http://www.w3schools.com/XPath/.

6.3 Add an SI XPath filter component.

6.3.1 In the si-components.xml file, add a SI XPath filter that takes a message from the XML inbound channel, uses the XPath expression component above to filter the XML messages and puts accepted messages in the outbound channel.

```
<int-xml:xpath-filter id="xpathFilter"
  input-channel="xml-inboundChannel" match-type="exact"
  output-channel="outboundChannel"
  xpath-expression-ref="filterXpathExp">
</int-xml:xpath-filter>
```

Note that the filter references the XPath expression component.

6.3.2 Save the configuration file and make sure there are no errors in the file.

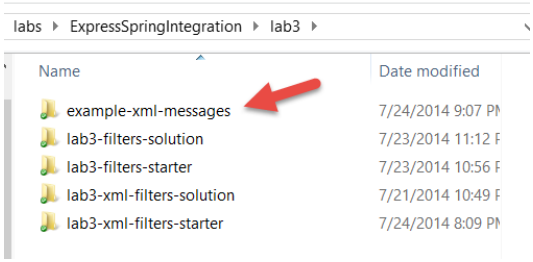
Step 7: Test the XPath Filter

Add XML messages into the inbound file folder and then test the application to see the filter do its job.

7.1 Add the files to the inbound message folder.

7.1.1 In the si-components.xml, find the producer-file-adapter. Note the location of the directory. It is set, by default, to file:c://inboundXML. This is the location where messages will be taken into the application by the adapter. Create this message folder - changing the location to suit your needs and your file system (change the producer-file-adapter to reflect your location).

7.1.2 Some sample XML messages have been provided to you. Find them in the ExpressSpringIntegration\lab3 folder.



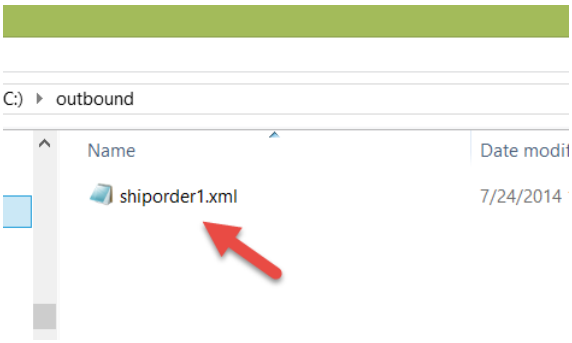
If you open the 3 messages, you will note that each are demo shipment orders. Each contain a <country> element. One of the messages has a country element that has USA as the country (satisfying the XPath expression). Copy the messages from the example-xml-messages to the inboundXML folder.

Note that the outbound location is the same as in the last part of this lab.

7.2 Test the application. Test the application to see the files in the inboundXML file folder are filtered based on <country> element content.

7.2.1 Locate the Startup.java file in the source folder. Right click on file and select Run As > Java Application from the resulting menu. Nothing should display in the Console view.

7.2.2 Using a Windows Explorer, open the folder specified as the File output adapter's directory. See that the accepted messages are now in the folder. Rejected messages (files with names that do not contain a <<country> with content of USA) have been filtered and not in the folder.

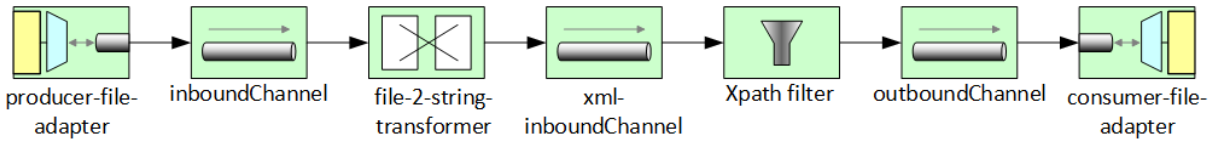


7.3 Stop the application. Recall the application is running in an infinite loop to allow for the constant publishing and consuming of messages. Stop the application now.

7.3.1 In the Console view, click on the red square to terminate the Startup application.

7.3.2 The Console view should now indicate that the application is "<terminated>". Clear the Console view by clicking on the Clear Console icon.

7.3.3 Below is the EIP diagram for your completed application.

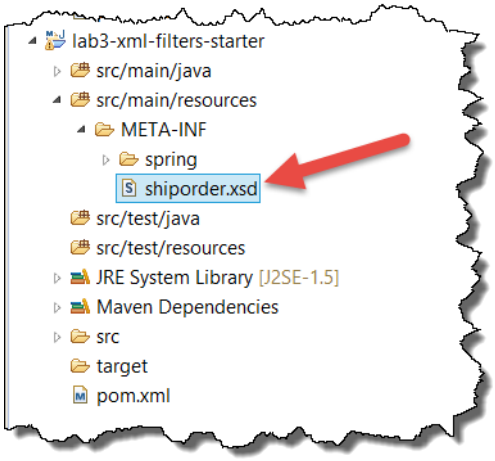


Step 8: Create a validation filter

SI also comes with a ready-made validating filter. When dealing with XML messages that are supposed to conform to an XML schema, it is often helpful to weed out XML messages that do not conform to the schema. Using SI's built-in validating filter, all you have to do is specify the location of the appropriate schema (.xsd) file and XML messages are filtered out of a channel when they do not align with the schema definition.

8.1 Locate the schema file in the project. A schema file for validating the shiporder XML messages has already been created for you and added to the project.

8.1.1 Locate the shiporder.xsd file in the src/main/resources/META-INF folder in the lab3 project.



8.1.2 Open the schema by double clicking on the file to explore its contents. This schema file defines the legal XML elements and attributes for a shiporder message.

Note: If you would like to learn more about XML schemas and their use, see http://www.w3schools.com/schema/default.asp.

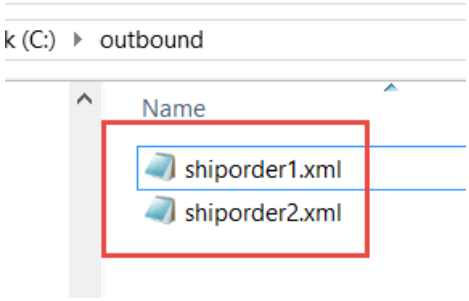
8.2 Add a validating filter. By this point in the tutorials, you have started to work enough with SI applications and SI components to start to get a feel for how they are configured and how the framework works. So it is time to put some of those skills to the test. Replace the XPath filter with a validating filter to accept only those XML shiporder messages that comply with the schema you saw in step 8.1. The basic template is shown below. You need to supply the in and out message channels along with the location of the schema file it is to use to do the filtering.

```
<int-xml:validating-filter id="validation-filter"
  input-channel=" " output-channel=" "
  schema-location=" ">
</int-xml:validating-filter>
```

Note: If you get stuck, consult the solutions for this lab.

8.3 Test the application. Make sure the outbound file folder is clear of messages before testing the application. Then start the application and insure that only the

valid messages are moved to the outbound folder. You should find two of the messages (out of the 3 original messages) are valid.



Filters are another type of Spring Integration message endpoint. They help to constrain the messages that enter other SI components or another applications. SI comes with many built-in filters, but you have also seen in this lab how to create your own custom filter with a MessageSelector implementation.

Lab Solution

si-components.xml - for the File filter lab

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:int="http://www.springframework.org/schema/integration"
  xmlns:int-
file="http://www.springframework.org/schema/integration/file"
  xmlns:int-
mail="http://www.springframework.org/schema/integration/mail"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:int-
stream="http://www.springframework.org/schema/integration/stream"
  xsi:schemaLocation="
  http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/integration
http://www.springframework.org/schema/integration/spring-
integration.xsd
  http://www.springframework.org/schema/integration/stream
http://www.springframework.org/schema/integration/stream/spring-
integration-stream.xsd
  http://www.springframework.org/schema/integration/file
http://www.springframework.org/schema/integration/file/spring-
integration-file.xsd">

  <!-- Adapter for reading files -->
  <int-file:inbound-channel-adapter id="producer-file-adapter"
    channel="inboundChannel" directory="file:c://inbound"
    prevent-duplicates="true">
    <int:poller fixed-rate="5000" />
  </int-file:inbound-channel-adapter>

  <int:channel id="inboundChannel" />

  <int:filter input-channel="inboundChannel" output-
channel="outboundChannel"
    ref="selector" />
  <bean id="selector" class="com.intertech.lab3.FileSelector" />

  <!-- a direct channel -->
  <int:channel id="outboundChannel" />

  <!-- Adapter for writing files -->
  <int-file:outbound-channel-adapter
    channel="outboundChannel" id="consumer-file-adapter"
directory="file:c://outbound" />

  <int:poller id="defaultPoller" default="true"
    max-messages-per-poll="5" fixed-rate="200" />

</beans>
```


FileSelector.java

```
package com.intertech.lab3;

import java.io.File;
import org.springframework.integration.core.MessageSelector;
import org.springframework.messaging.Message;

public class FileSelector implements MessageSelector {

  public boolean accept(Message<?> message) {
    if (message.getPayload() instanceof File
        && ((File) message.getPayload()).getName().startsWith("msg"))
{
      return false;
    }
    return true;
  }
}
```

si-components.xml – for the XML filters lab

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:int="http://www.springframework.org/schema/integration"
  xmlns:int-
file="http://www.springframework.org/schema/integration/file"
  xmlns:int-
mail="http://www.springframework.org/schema/integration/mail"
  xmlns:int-
xml="http://www.springframework.org/schema/integration/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:int-
stream="http://www.springframework.org/schema/integration/stream"
  xsi:schemaLocation="
  http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/integration
http://www.springframework.org/schema/integration/spring-
integration.xsd
  http://www.springframework.org/schema/integration/stream
http://www.springframework.org/schema/integration/stream/spring-
integration-stream.xsd
  http://www.springframework.org/schema/integration/file
http://www.springframework.org/schema/integration/file/spring-
integration-file.xsd
  http://www.springframework.org/schema/integration/xml
http://www.springframework.org/schema/integration/xml/spring-
integration-xml.xsd">

  <int-file:inbound-channel-adapter id="producer-file-adapter"
    channel="inboundChannel" directory="file:c://inboundXML"
    prevent-duplicates="true">
    <int:poller fixed-rate="5000" />
  </int-file:inbound-channel-adapter>
```



```
  <int:channel id="inboundChannel" />

  <int-file:file-to-string-transformer
    id="file-2-string-transformer" input-channel="inboundChannel"
    output-channel="xml-inboundChannel" charset="UTF-8" />

  <int:channel id="xml-inboundChannel" />

  <!-- <int-xml:xpath-filter id="xpathFilter" -->
  <!-- input-channel="xml-inboundChannel" match-type="exact" output-
channel="outboundChannel" -->
  <!-- xpath-expression-ref="filterXpathExp"> -->
  <!-- </int-xml:xpath-filter> -->
  <!-- <int-xml:xpath-expression id="filterXpathExp" -->
  <!-- expression="//country='USA'"></int-xml:xpath-expression> -->

  <int-xml:validating-filter id="validation-filter"
    input-channel="xml-inboundChannel" output-
channel="outboundChannel"
    schema-location="META-INF/shiporder.xsd">
  </int-xml:validating-filter>

  <int:channel id="outboundChannel" />

  <int-file:outbound-channel-adapter
    channel="outboundChannel" id="consumer-file-adapter"
directory="file:c://outbound" />

  <int:poller id="defaultPoller" default="true"
    max-messages-per-poll="5" fixed-rate="200" />

</beans>
```