**Filters**

**Filters**

Filters provide the ability to add business logic to routes and to control the path a message takes through the processing logic. They are configured and stored as part of the route to provide a processing pipeline for the messages.

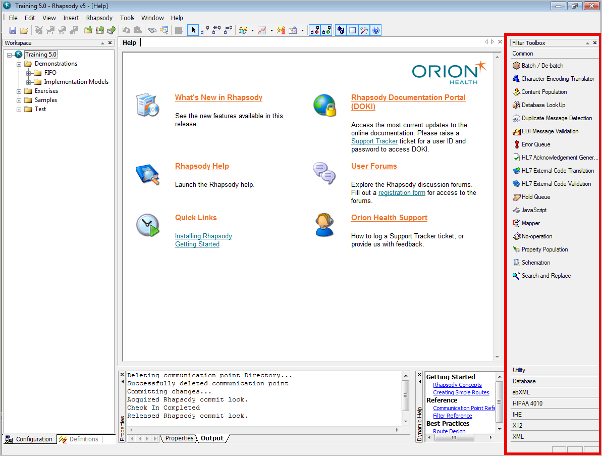
Filters range in complexity from single functions (for example, the **Search and Replace**filter) to complex message processors capable of substantially modifying a message or completely replacing the message content (for example, the **JavaScript** and **Mapper**filters).

The information you learn in Rhapsody 106 (this course) and Rhapsody 105 (Communication Points) will be brought together in Rhapsody 107, when we build and configure a more complex route than the examples seen so far.

### The Filter Toolbox

The filters are displayed in the IDE in the Filter Toolbox, displayed by default to the right of the **Route Canvas**. A filter may be added to a route by a drag and drop operation.

The **Filter Toolbox** may be removed from the display by clicking on the Hide button (**X** - Hide docked window) at the upper right of the toolbox, and re-instated by selecting **View/Toolbox** item from the top level menu (deselecting that item will also remove the toolbox from the interface).   
  
The **Filter Toolbox** is a dockable element and can be moved to another location or left free floating by clicking on the top bar of the Toolbox and dragging to another location.



### The Filter Toolbox Groupings

|  |  |
| --- | --- |
| The Toolbox contains in excess of 50 filters, arranged in an accordion style menu structure to simplify access. The groupings provided are:   * **Common:** the frequently used filters including No-operation, JavaScript and Database filters (amongst others). * **Utility:** predominantly encoding and archive filters. * **Database:** the database filters. * **ebXML:** support for ebXML messages. * **HIPAA 4010** * **X12:** HIPAA 5010 support. * **XML:** XML management tools including the ability to apply an XSLT stylesheet to a message.   Note that there are also a small number of deprecated filters which are maintained to provide backwards compatibility for configurations imported from older Rhapsody versions.   These may be accessed by selecting the backwards compatible filters option in the Rhapsody options dialog (**Tools/Rhapsody Options/Display** tab).  New filters may be added from time to time to augment the filter tool set. In addition, site specific filters may be developed with the Rhapsody Development Kit and included in the filter list. | filter_toolbox |

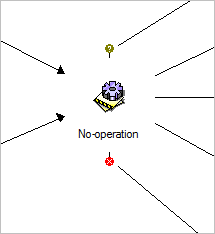
**Filters and Route Construction**

Each filter accesses two message objects;

* **Input message object:** a read only object received from the route.
* **Output message object:** the message object which the filter generates to output on its output connections.

**Filter Attachment Points**

Each filter supports four attachment points to allow connection to other components in the route;

* **Input**: by convention indicated by connectors at the left hand side of the filter, but denoted by an inbound arrow head; a filter may have many input connections.
* **Output**: by convention, indicated by connectors to the right of the filter; a filter may have many output connections, each of which causes a copy of the message to be forwarded to the next filter in the route, governed by any conditional rules applied on the connector.
* **No-match connector**: indicated by the green "lollipop" connection point at the top of the filter and used for re-direction of messages when conditions are defined for output and a message does not match any of the conditions. Messages will only be forwarded on this connection if they cannot be sent on any output connection.
* **Error connector**: indicated by the red "lollipop" connection point below the message; messages will be forwarded by this connection if any error is encountered during processing in the filter, if there is no path connected to the connection point, any message output by the error connector will be placed in the Error Queue.  
    
  

Each filter connection point permits multiple connections to be attached, allowing, for example, several paths to converge on a filter, or several paths to exit the filter. When multiple paths exit a filter, a copy of the message is sent along each path, and FIFO rules apply to all copies as a group.

The **No-Match** and **Error** connectors may be switched off by selecting the appropriate option in the **View** menu or the Rhapsody Toolbar.

Filters must have at least an input and an output connection to be utilized when processing a message.

Connections are created using the connector icon from the IDE toolbar.

When a component is selected, it is highlighted by a blue rectangle.

The direction of the connection (on dragging) will be indicated by an arrowhead on the connector.

|  |  |
| --- | --- |
| Filter connection selection | Filter connection split |

Filters may be dropped onto an existing connection. The connector will be highlighted (bold line) to indicate that the drop action is now permitted, and the filter will cause the connector to be split at that point.

Note that if the connector is not highlighted during the process, the filter will overlay the connector but not intersect it (it is not connected).

**The Filter Dialog**

As is the case for the Communication Points, filters use a common tabbed dialog with filter specific configuration detail being presented in the Configuration tab. The tabs provided are;

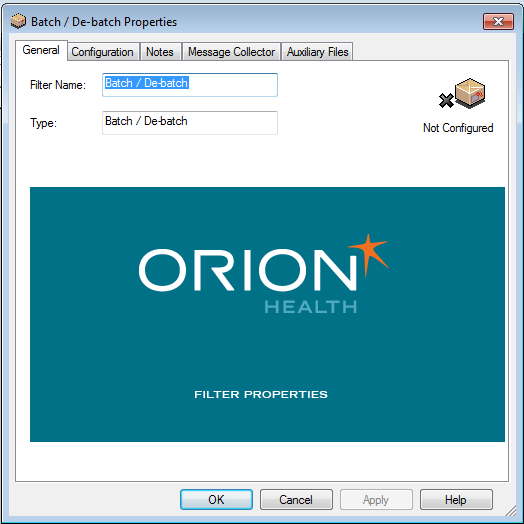
* General
* Configuration
* Notes
* Message Collector
* Auxiliary Files

Changing any parameter in any of the tabs will cause the underlying route to be checked out. The route must be checked in to ensure that the changes are posted to the engine and implemented during message processing.

### General Tab

The General tab permits the filter name to be changed and also describes the type of the filter.

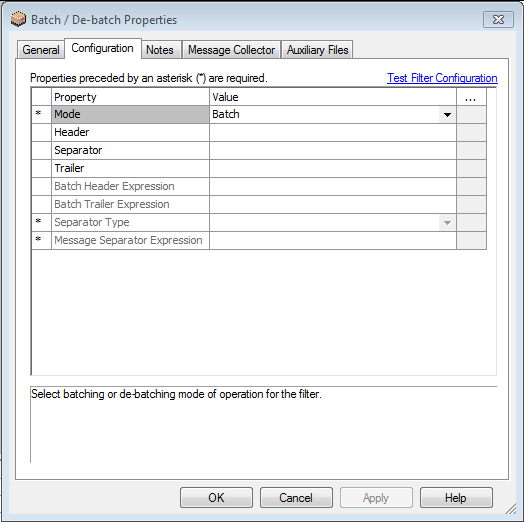
|  |  |
| --- | --- |
| Please Note | This field is read-only. |



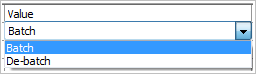
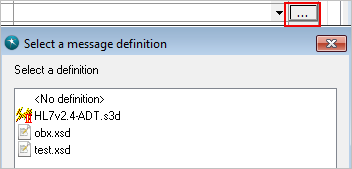
**Configuration Tab**

The Configuration tab presents the configurable parameters for the filter, with required parameters being highlighted with an asterisk (\*).

Contextual help is available for each row of the dialog and is displayed at the bottom of the dialog, and more extensive help is accessed by clicking the **Help** button.



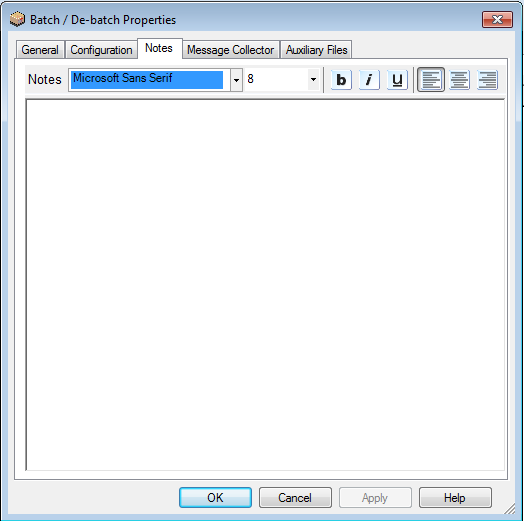
The Value fields may be one of the following;

* **Text entry:** enter free text, the name of a message property (note that you will need to know the property name to enter) or a Rhapsody Global Variable.  
    
  Filter config tab text
* **Drop-down menu:** select from the available options.  
    
  
* **Select from an external dialogue:** this option is indicated by the presence of an ellipsis in the cell adjacent to the field.  
    
  Clicking on the ellipsis will open the relevant dialogue.  
    
  

### Notes tab

The **Notes** tab allows the entry of free text content to describe the filter usage and implementation details.

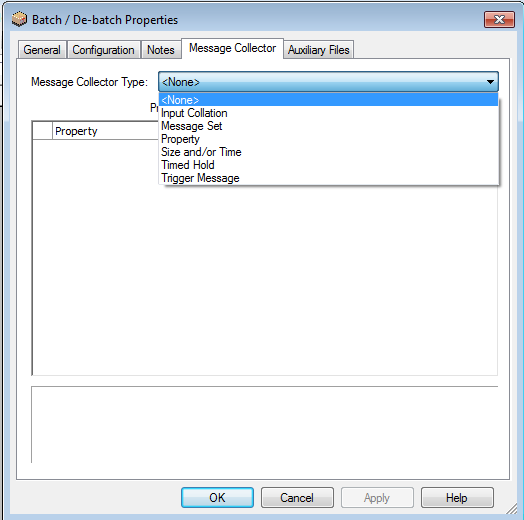
The content entered in this tab is stored as part of the route configuration and included when configuration documentation is generated in the IDE.



### Message Collector Tab

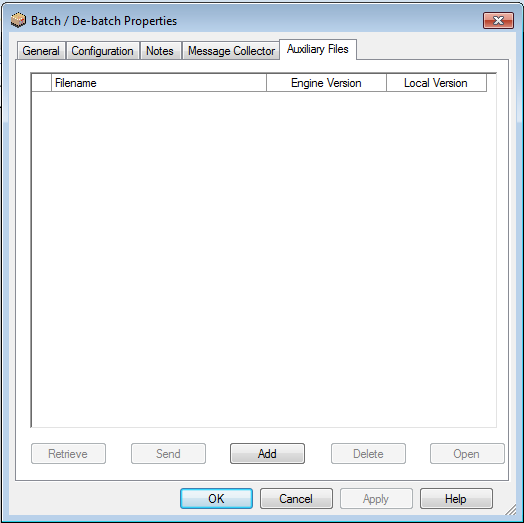
The **Message Collector** tab provides additional functionality to control the release of the message from the filter.   
  
Options include the ability to hold the message for a time or to hold the message until a set of messages are received by the filter.

Once the hold condition has been satisfied, the messages being held are released in the order defined by the collector. This may be in time order, or in a defined message sequence.



### Auxiliary Files Tab

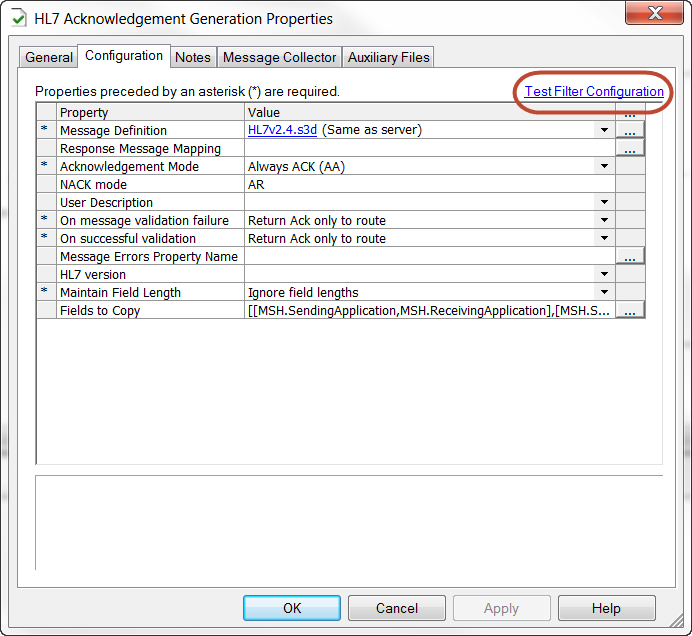
The **Auxiliary Files** tab provides the ability to add additional support files which may be required for the operation of this filter.   
  
The most frequent use is for the **Database** filters where the configuration details are automatically attached and additional driver files may be attached (particularly if the database is not one of the supported databases).



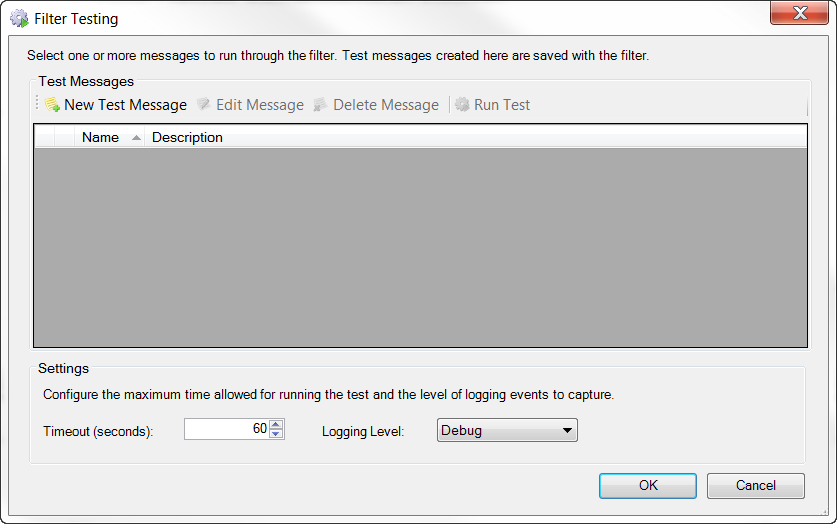
### Filter Testing

The **Filter Testing** functionality allows the user to validate the behavior of the filter prior to checking in, and speeds up the route development process.

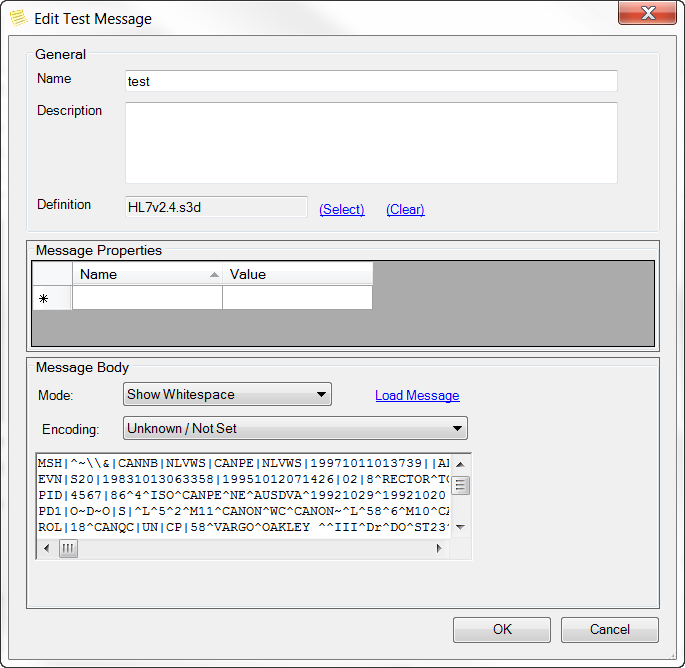
Once the filter configuration is complete, click on the **Test Filter Configuration link** on the **Configuration** tab of the dialog**.**

****

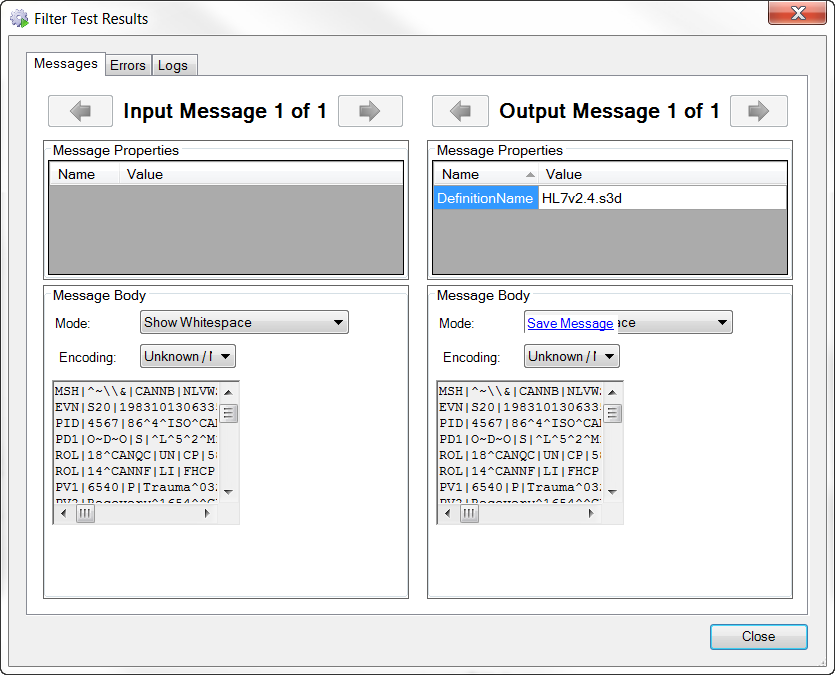
The **Filter Testing** dialog will open, permitting one or more test messages to be attached to the test. Once the messages have been added, the test may be run against all selected messages and the results will be shown on the output dialog, including error messages and the log for the test.



If message properties are required, they may be set up on a message by message basis during the Add and Edit test message processes.



Once the test process is complete, clicking on the **OK** button on the testing dialog will exit, saving the test messages as part of the filter configuration. These will be preserved in the exported configuration. If the messages are not required to be maintained in the configuration, using the **Cancel**button to exit will discard the test messages.



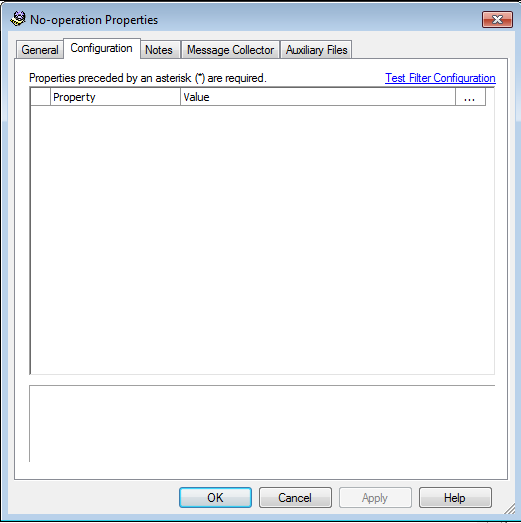
### No Operation Filter

### The No-operation Filter

The **No-operation** filter provides no processing functionality and is commonly used to control the pathway taken by the message through a route or to manage the visual layout of the route in the IDE for maintainability purposes. It has no configurable parameters.

Path management is controlled by the use of conditional connectors on the **Output** of the filter, allowing routing decisions to be made based on message content or properties. When a conditional connector is applied, only messages matching the conditions are able to traverse the connector.

If there no other Output connectors through which the message may traverse, the message will be sent to the **No-match** connection point. If there is no connector attached to the **No-match** connection point, the message will be sent to the **Error** connection point and will be placed into the **Error Queue** (if there is no specific error handling configured).



### Hold Queue

The **Hold Queue** filter provides the ability to send messages to the **Hold Queue** for review and further action.

The **Hold Queue** is a special output queue which is similar to sending the messages to a communication point. However, messages in the **Hold Queue** can only be accessed from the **Hold Queue** view in the Web Management Console.

In addition, messages on the **Hold Queue** will not be included in the results of a search including communication points in **Output** mode because they have not passed through an output communication point.

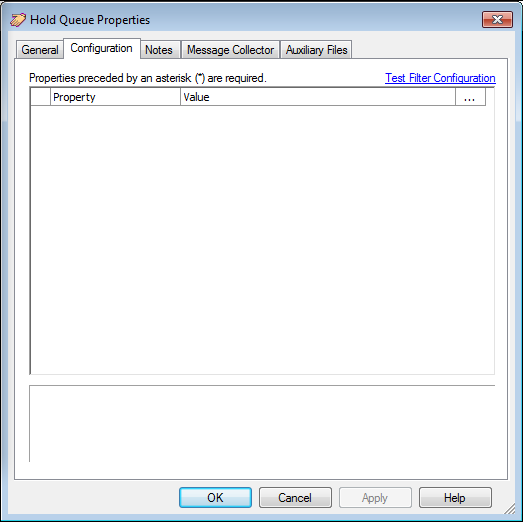
**Hold Queue** filters are commonly used in the development process to temporarily control the flow of messages along a route. The Web Management Console provides the ability to release messages singly or in bulk from the queue to better monitor their processing.

### Hold Queue Filter Important Note

While the Hold Queue filter can be used in extreme cases, this filter should **NEVER** be used as a permanent route feature in Production routes, because messages must be released from the **Hold Queue** by means of manual intervention.

Consequently, it is possible for messages to be overlooked and accumulate in the **Hold Queue**.

This can cause issues both in the application space and at Rhapsody restart as it will prolong the startup process for live queue validation.



**HL7 Acknowledgement Generator**

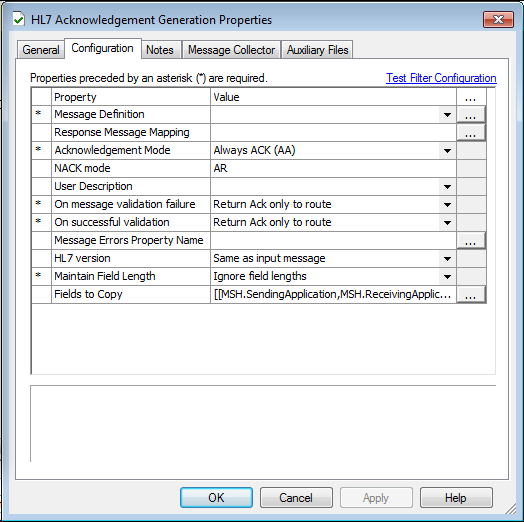
The **HL7 Acknowledgement Generation** filter provides the ability to check the incoming message against the definition and to generate an Acknowledgement message. Dependent on configuration, the filter replaces the message completely and outputs the acknowledgement to transmit back to the sender.

The filter is able to:

* Generate a positive acknowledgement for all input messages.
* Validate the input message against a definition and output the appropriate acknowledgement.
* Validate the message and also check message properties for errors before generating the response; this allows for field value checking prior to the HL7 Acknowledgement filter, for example.

Configuring the filter requires the following steps:

* Select a definition from the list of definitions known to the engine.   
  Note: if the required definition is not in the list, the filter editing must be paused temporarily and the appropriate definition loaded and checked in before resuming.
* Define the Acknowledgement mode.
* Define the behavior when an issue is detected. The message can be completely replaced or the message and the acknowledgement returned for further processing.  Properties are attached to the messages indicating the reason for the failure.
* Define the behavior required when a successful validation occurs. Again, new properties are defined and attached to the message to retain the validation state.



### Javascript Filter

The **JavaScript filter** provides extensive support for manipulating the message and its content, ranging from extracting values to updating content and completely replacing the message if required. The JavaScript implementation conforms to the ECMAScriptstandard.

The filter **Configuration** tab provides an editing panel into which the code can be added, and also provides boilerplate examples of common structures.  
  
In common with all filters, the **JavaScript** filter accepts a read-only message object as the input object and generates a writeable message object as the output object.

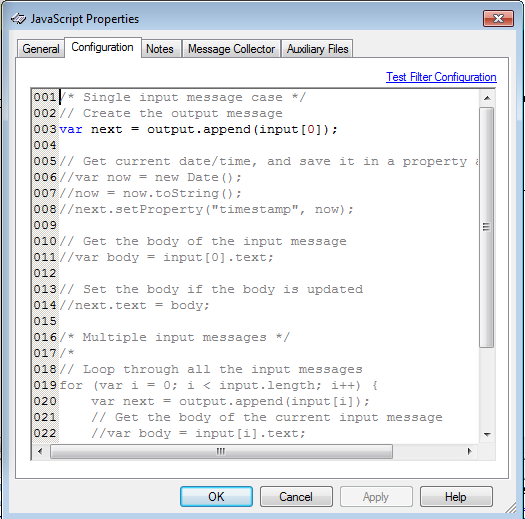
While the general case is that a single output message is generated, multiple output message objects may be generated (see the append function: **output.append()** creates a new object).

### Common Tasks of the Javascript Filter

Common tasks include:

|  |  |
| --- | --- |
| Extract a field value from a specified message path. | var fieldValue = input[0].getField(path\_to\_field) |
| Set a message property from the extracted value. | next.setProperty("New Property", fieldValue) |
| Update a field value. | next.setField(path\_to\_field, new\_field\_value) |
| Extract the complete message body. | var body = next.text ; |
| Completely replace the message body. | next.text = body ; |

When the message has a structure recognized by the engine, the path to a field may be specified in a XPath like notation using **/** or **.** for messages with an associated Symphonia definition (for example, **MSH/MessageType/MessageType**) , or XPath notation for XML messages.



**Exercise 106: Configuring a Filter**

**Filter and Route Processing**

Filters provide the ability to apply business logic to messages as they pass through the route pipeline. Each filter is optimised to provide a specific function to the pipeline and may be configured to provide branching as the message exits the filter.

Filters are managed and configured as integral components of the route functionality. Consequently, a change to any filter or connection in a route will cause the route to be checked out, and requires check in of the route once updates are complete.

When a message enters a route, it may be parsed according to a supplied message definition and message properties may be set, either by extraction of values from message fields or by setting explicit values.

**Filter Functionality**

The filter functionality extends the capabilities of the route by providing extensive processing functionality including (but not limited to):

* validation / acknowledgement of messages.
* extraction / setting of message properties.
* extraction / setting of field values within the message.
* re-parsing the message using a filter specific message definition.
* replacement of message content either by query of external sources or processing the existing message content.
* de-batching / batching message content.
* encryption / decryption of message content
* transformation / mapping message structure to a new structure.

Each filter also includes a test harness which will allow sample messages to be attached to the filter to simplify validation of the filter operation and re-testing in the event of downstream changes. The test harness details are maintained as part of the filter configuration and may be exported into a local copy (RLC) if desired.

Filter configuration is managed by a dialog similar to the communication point dialog containing:

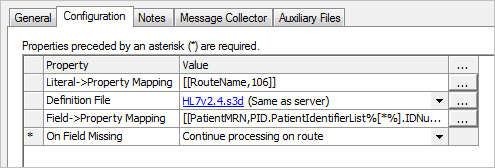
* **General configuration**: name of the filter.
* **Configuration**: configuration parameters specific to the functionality of the filter.
* **Notes**: repository for filter specific documentation.
* **Message Collector**: specification of rules for managing messages as sets.
* **Auxiliary files**: permits the attachment of additional resources to the filter; most commonly utilised for the database filters.

**Property Population Filter**

This exercise examines the extraction and management of properties at the route and filter level and will focus on the configuration requirements for the **Property Population** filter.

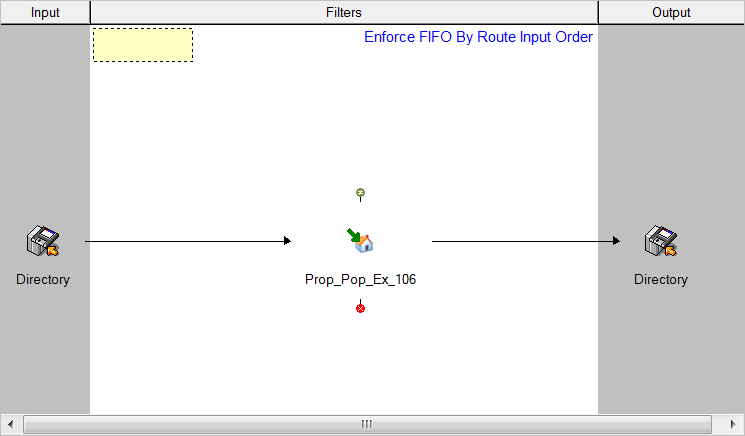
The **Property Population** filter allows **Message Properties** to be set either from a static string or by extracting a value from the message content. In the latter case, the filter requires specification of a message definition. Note that a number of other filters may also set **Message Properties** either implicitly or explicitly as part of their operation.

**Message Properties** are typically used to:

* control the pathway a message follows through the engine.
* control or augment the processing of the message by subsequent filters.
* provide information about the message for troubleshooting.  
    
  

**Exercise Tasks**

In this exercise, we will use the **Property Population** filter to set a **Message Properties** (metadata). These **Message Properties** can then be used after output from the filter, for example to control the message path using a conditional connector.

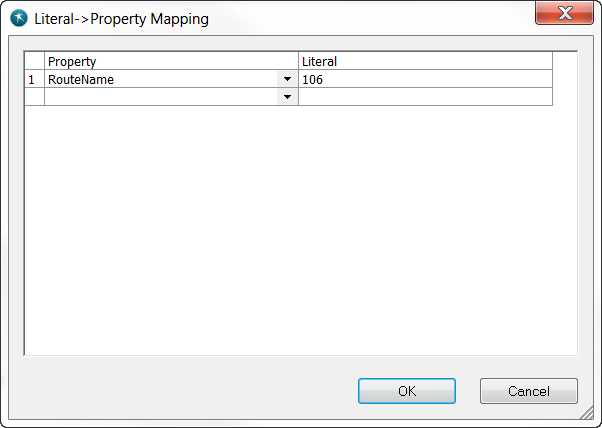


1. Create a route to use for this exercise and configure it with a bidirectional **Directory** communication point to provide input and output for the route, and connect them.
2. Add a **Property Population** filter to the route/connector.
3. Double click on the filter to open the properties dialog.
   * The filter properties dialog will open showing the **Configuration** tab.
4. **General** tab - assign a name to the filter.
5. Open the **Configuration** tab to view the property setting options. This tab permits assignment of properties by:
   * **Literal -> Property Mapping** - assigning a Literal value (that is, a character string) to a **Message Property**.
   * **Field -> Property Mapping** - specification of a message definition and the extraction of a field value from the message body and assigning it as a **Message Property**.

**1) Literal -> Property Mapping**

Exercise - How to create a **Message Property** named **RouteName** and assign the literal value of **106**:

1. Click on the **Assignment** button for the **Literal -> Property Mapping** (indicated by the ellipsis **[...]** in the row) to open the **Assignment** dialog.
2. Type the name for the Property in the **Property** cell (in this case **RouteName**).
3. Type the explicit value you wish to assign in the **Literal** cell (in this case **106**).
4. Select **OK** to exit the dialog when the assignments are complete.

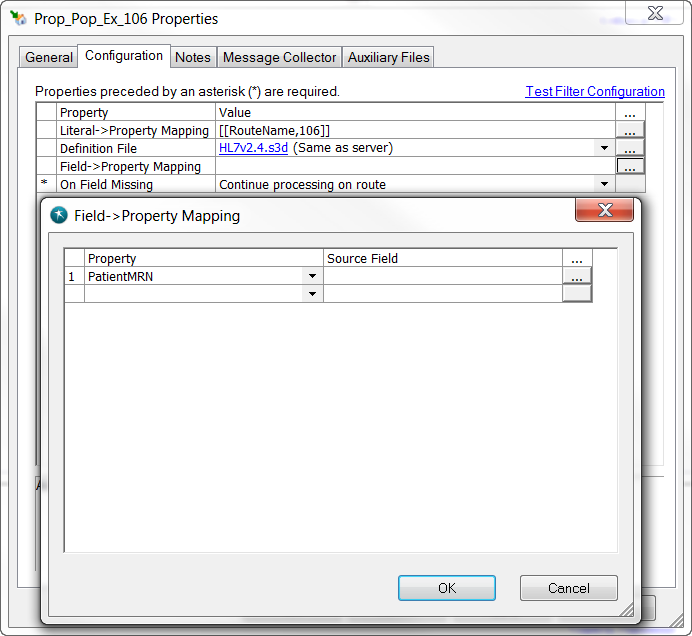


Additional Literal properties may be defined by typing a property name in an empty **Property** cell on the next row, and repeating the process.

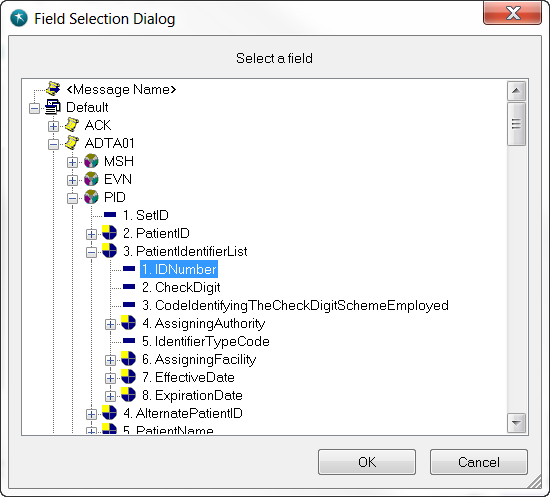
**2) Field -> Property Mapping**

A **Message Property** may be set by extracting a value from a message field by assigning a **Definition File** to the filter, and then specifying the field from where the value will be extracted.

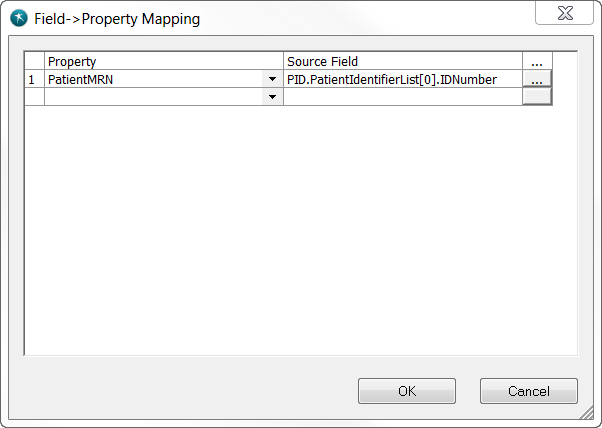
1. Assign the appropriate **Definition File** by selecting the **Assignment** button in the **Definition File** row. The definition selection dialog will open, displaying the list of definition files loaded into the engine.
2. Select the **HL7v2.4.s3d** definition and click **OK**.
   * The **Field -> PropertyMapping** row of the dialog will now be enabled.



1. Select the **Assignment** button in this row to open the **Property Assignment** dialog, shown above.
2. Create a property named **PatientMRN** by defining the **Property** name.
3. Select the field from which to extract the property (by selecting the **Assignment** button [....], which opens the **Field Selection Dialog)**.
   * The **Field Selection Dialog** provides a list of all messages available in the definition.



1. Select the **ADTA01** message message.
2. Access the segment and field hierarchy to select the field; **PID/PatientIdentifierList/IDNumber**
3. Click **OK** to return to the **Filter** dialog.  
     
   Note that:
   * The property will be extracted whenever the selected path is present in the message.
   * The field path utilises the period character (**.**) as the field separator; Rhapsody recognises both **/** and **.** as interchangeable field separators.
   * **PatientIdentifierList** is a repeating field; the field reference selected includes the wildcard [**\***] to extract all values of **IDNumber** present in the message. To extract only the first instance of **IDNumber**, replace the **\*** character with **0**, as shown below.



Additional properties may be defined in subsequent rows by repeating the process.

**Testing the route:**

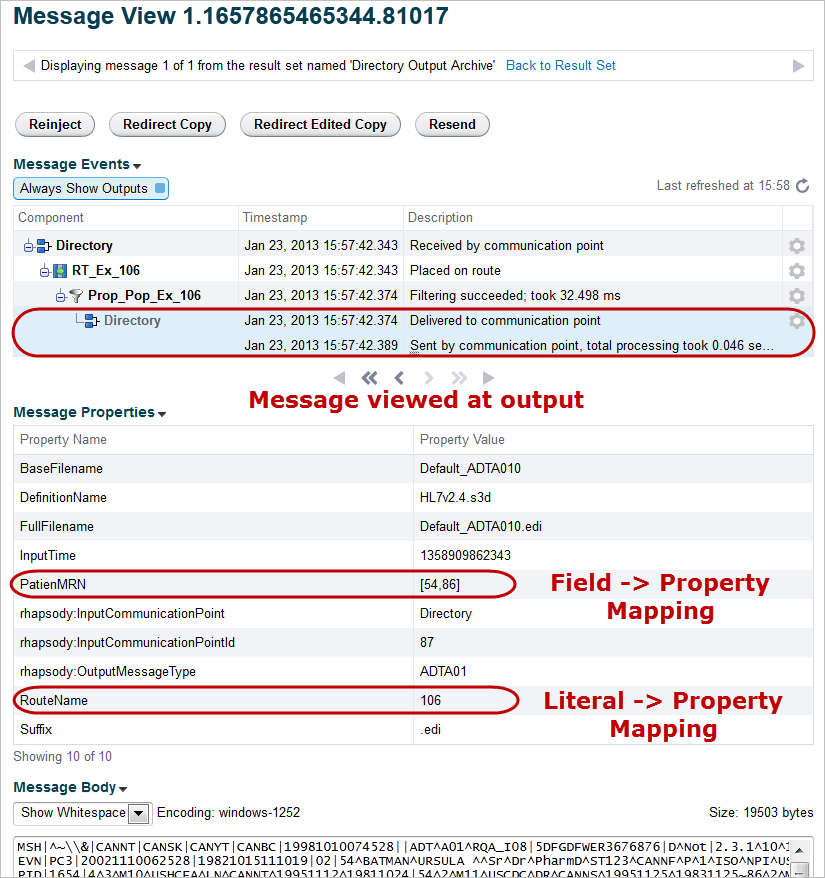
1. Ensure all components are checked in and running.
2. Copy a test message to the input folder.
3. Use the the Message View in the WMC to navigate through the Message Events tree.
4. Verify that the two Message Properties are added (after leaving the **Property Population** filter).

### 106 Model Answer (1)

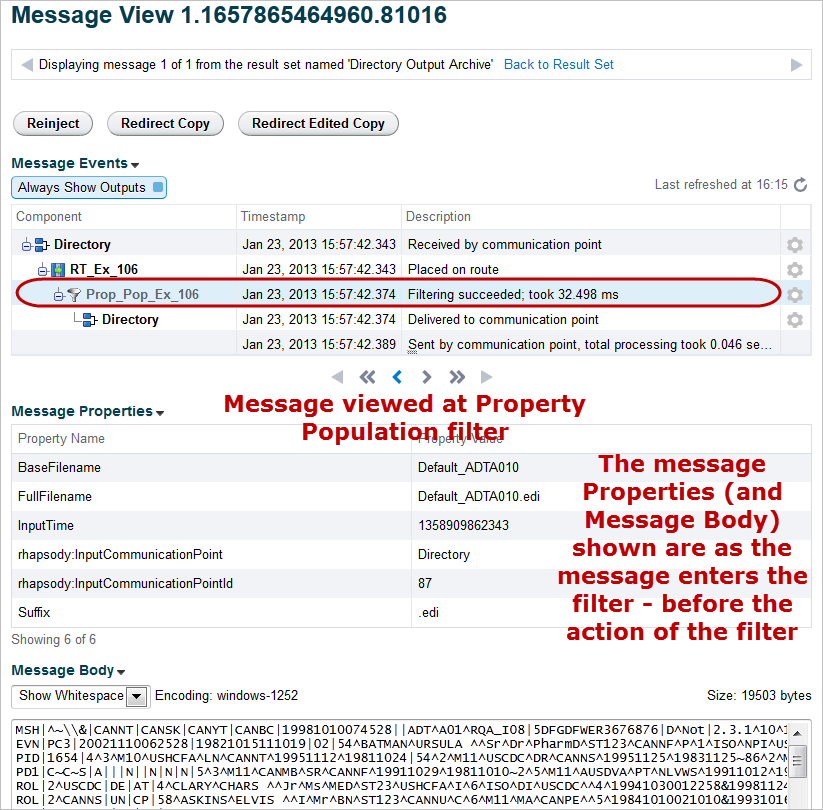
### What you expect to see:

**Message View at output**:

In order to view the results of a message after all changes have been made (to Message Properties or to the Message Body) it is important to view the message at the **output** of the **Events Tree.** In this example all repeats of the PatientMRN are shown in the Message properties (separated by a comma).



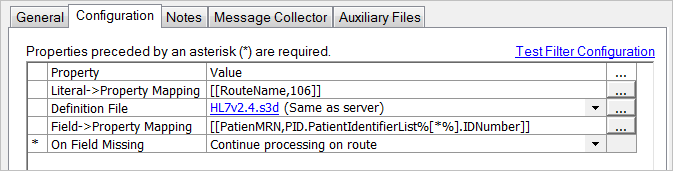
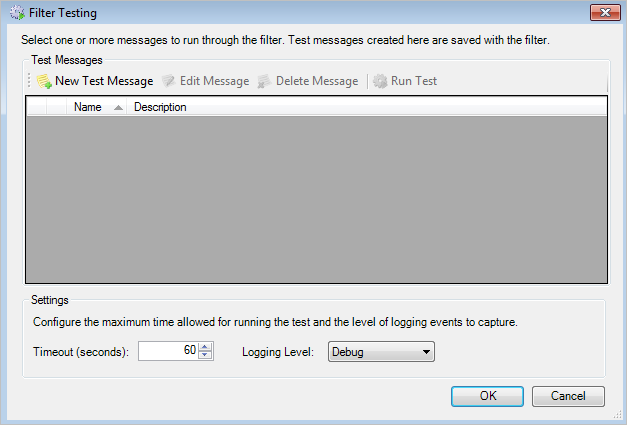
### What would you see if you look at the message at the Property Population filter in the Events Tree?



**Testing the Filter Configuration**

Filters should always be tested before they are used in a route. Sample test messages can be loaded and tested against a **Message Definition** to ensure that the filter is behaving as expected

**Exercise Tasks**

1. Open the **Property Population** filter Configuration dialog by double clicking on it.   
     
   
2. Select the **Test Filter Configuration** link which opens the **Filter Testing** dialog.  
     
   

The filter testing process requires:

* Assignment of one or more test messages (including properties likely to be set earlier on the route if they are required for testing).
* Run Test and examine ouput.
* Reconfigure filter if required.

**Test the Filter**

1. Add a test message by selecting the **New Test Message** link.
2. Define a name for the message (in the dialog which opens).
3. Add a message definition (if required).
4. Select the **Load Message** link.
5. Select a test message.
6. Add a **Description** to document the test.
7. Select **OK** to close the dialog and return to the **Filter Testing** dialog.
8. Ensure that a test messages is selected.
9. Select **Run Test** to execute the filter test.
10. Confirm that the **Message Properties** are correctly assigned and populated.
11. Make changes to the filter configuration and re-test if required.
12. Close the **Filter** dialog.

**Filter Test Results**

The **Filter Test Results** displays:

* **Messages**: displayed only if the test encounters no errors; displays **Message Properties** implicitly and explicitly assigned by the filter processing.
* **Errors**: if errors are encountered, detailed error messages are displayed.
* **Logs**: displays the log messages generated by the filter processing (normally blank).

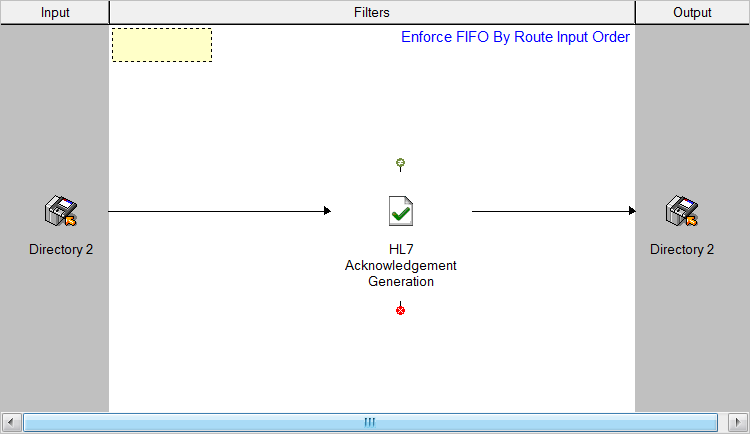
**Test the Route**

Send a test message through the route and examine the **Message Properties** in the **Web Management Console** (at the output of the **Events Tree** in the **Message View**).

**The HL7 Acknowledgement Generation Filter**

The HL7 standard requires that a system receiving an HL7 message should acknowledge receipt (back to sender) using an ACK message type. The **HL7 Acknowledgement Generation** filter provides a convenient method of generating the ACK message for return to the sending system.

In this exercise, we will use the **HL7 Acknowledgement Generation** filter to create an HL7 acknowledgement. The acknowledgement is returned to the sender (to the bi-directional Directory communication point).



The **HL7 Acknowledgement Generation** filter requires a **Message Definition** against which to generate the ACK.

**Configure the Filter**

1. Select HL7v2.4.s3d **Message Definition** from the list of definitions loaded into the engine. All other parameters are left as the default:
   * **Acknowledgement Mode**: Always ACK (AA) (the default).
   * **On message validation failure**: Return ACK only to the route.
   * **On successful validation**: Return ACK only to the route.
   * **Maintain field length**: Ignore field lengths.

No further configuration is required.

**Test the Filter**

1. Select the **Test Filter Configuration** link which opens the **Filter Testing** dialog.
2. Add a test message by selecting the **New Test Message** link.
3. Define a name for the message (in the dialog which opens).
4. Add a message definition.
5. Select the **Load Message** link.
6. Select a test message.
7. Add a **Description** to document the test.
8. Select **OK** to close the dialog and return to the **Filter Testing** dialog.
9. Ensure that a test messages is selected.
10. Select **Run Test** to execute the filter test.
11. Confirm that the ACK message is correctly generated (examine the **Message Properties** and the **Message Body**).
12. Make changes to the filter configuration and re-test if required.
13. Close the **Filter** dialog.

**Test the Route**

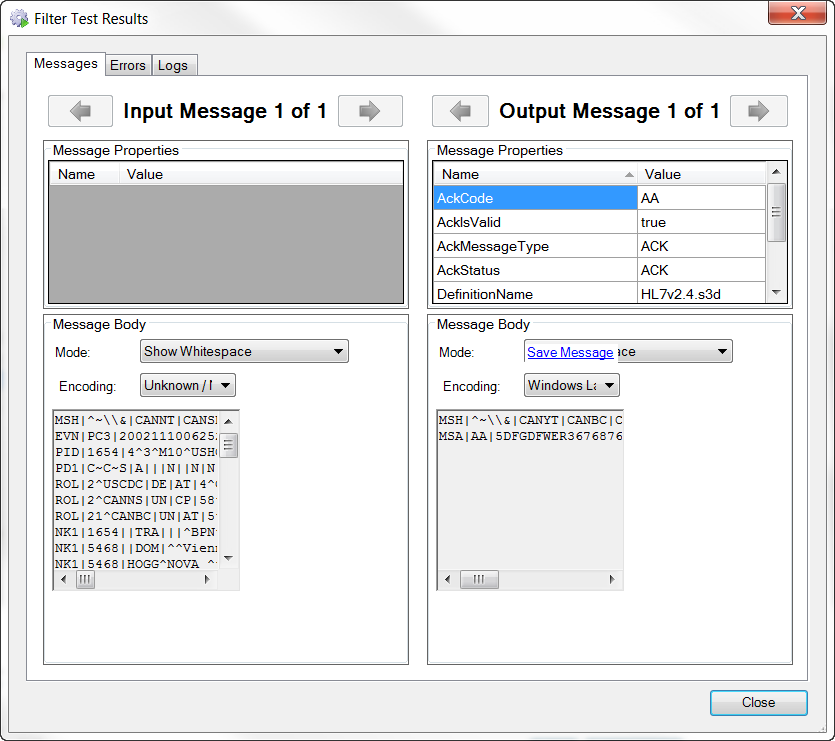
Send a test message through the route and examine the **Message Properties** and the **Message Body** in the **Web Management Console** (at the output of the **Events Tree** in the **Message View**).

### 106 Model Answer (3)

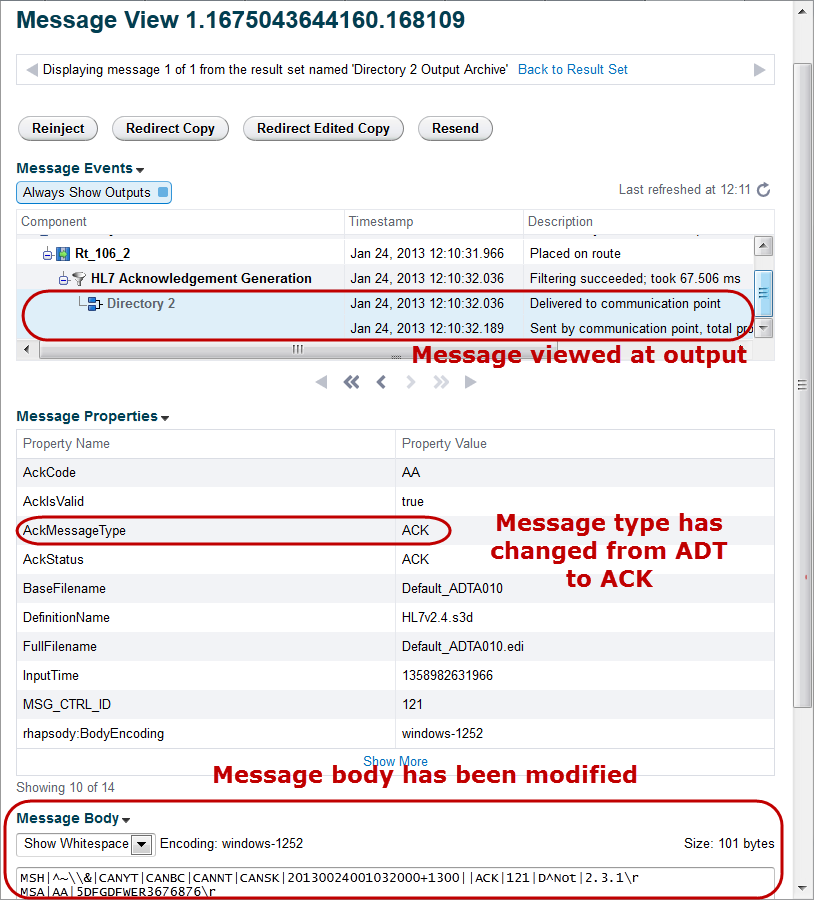
### What you expect to see:

**Filter Test:**

Notice that the message type is converted to ACK and that the message body is changed (stripped to minimal content).



**Route Test:**

****