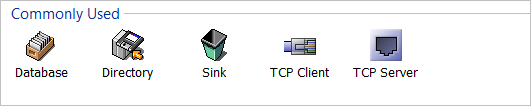
**Configuring Communication Points**

**Configuring Commonly Used Communication Points**

Analysis of representative client configurations indicates that the most commonly used communication points are Database, Directory, Sink, TCP Client and TCP Server.



Each of these share the **Common** tabs in the property dialog and have protocol specific parameters configurable through the **Configuration** tab.   
  
This section discusses the configuration process for each of these communication points.   
  
The discussion will cover the most frequently used parameters.

### Directory

The Directory communication point is optimised to work with files in the file system of the host server or folders shared by a remote system. It supports all modes of connection (that is: **Input**, **Output**, **Bi-directional, In->Out and Out->In**).   
  
In overview, its role is to input files from a defined location or write messages to files in a defined location.   
  
The Directory communication point provides extensive control for managing its operation and also supports extracting messages from a file (**de-batching**) or writing messages into an uncompressed or compressed archive file (**batching**).

### Directory - Input Mode

### Input mode

In **Input** mode, the communication point polls a defined directory at regular time intervals (5 seconds by default).   
  
When a new file is detected, it is read in and placed into the queue, and then the file is deleted to ensure it is not accessed again. If desired, the file may be moved to an archive location.

A mask may be applied to include or exclude filename patterns from the input stream, and the order of input can also be defined by **name**, **file access time** or **custom**.   
  
For example, to read in only files of type .txt, specify the pattern as \*.txt.

The pattern should provide a pattern which will match the complete file name. Note that \* is the only wildard recognised in the pattern.

### Debatch Operation

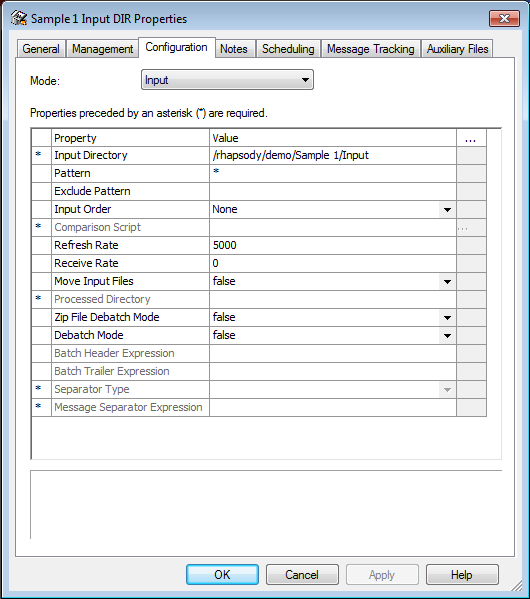
The **debatch** operation is the most efficient method for debatching files containing multiple messages because the process acts on the stream of data searching for the separator pattern defined.

This operation is also supported by a de-batching filter which is more memory hungry.

The path for the input directory should be the absolute path to the directory on the server hosting the Rhapsody engine.

While the configuation dialog supports the use of forward slash ( **/** ) as well as back-slash ( **\** ) in the path, the path is interpreted by the server operating system.  The relevant separator should be used (forward slash - **/** - for Unix and Linux hosts, backslash -**\\** - for Windows hosts).   
  
UNC (Universal Naming Convention) paths are supported for accessing directories on remote servers provided that the Rhapsody server is a Windows host and the Rhapsody engine runs as a Windows domain user.   
  
The path specification for a UNC path is as normally specified for the share:

**\\\\Host\\share\\path\_to\_file**



### Directory - Output Mode

### Output Mode

In **Output** mode, the Directory communication point writes the messages out into a file in the defined Output directory. By default, each message is written to an individual file unless the Batch Mode is set to **true**.

The filename may be defined using the **Base Filename** parameter. If this is not defined, the system default is used (date and time including milliseconds).   
  
No file type is appended by default.

The source filename and type may be retained by using the **BaseFilename** and **Suffix** properties extracted and attached to the message by the engine.   
  
This can be achieved by setting the following:

**BaseFilename** parameter to the property **$BaseFilename**

**Suffix** parameter to **$Suffix**

### Concatenation Not Supported

The configuration fields do not support concatenation of property values and strings. If a complex string is required, this should be created and stored as a message property prior to the Output communication point, and the configuration parameter should reference this property.   
  
For example, if the date is appended to the **BaseFilename**, it is useful to insert a hyphen ( - ) prior to the date. This would require creating a property on the route which concatenates the **BaseFilename** property and the separator ( **-** ).

### Conflicting Filenames

Conflicting file names are handled by appending a version number to the filename.

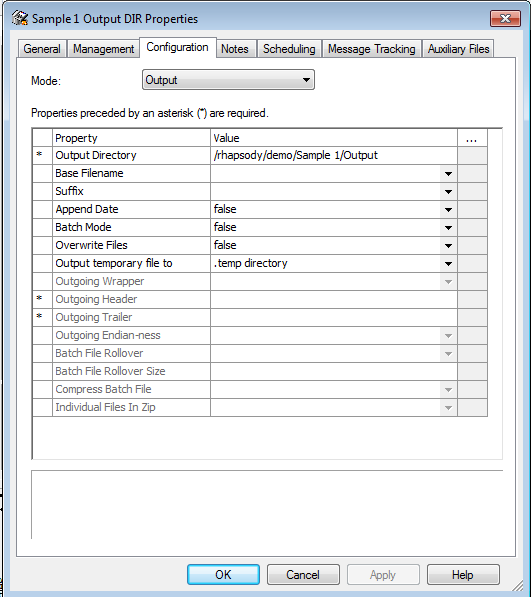
For example, if **test.csv** exists in the Output directory, the new file will be **test(1).csv**

Alternatively, the communication point may be configured to append the current date and time to the filename.

The normal mode of operation for the Directory communication point in Output mode is to store each message in a separate file.

### Batched Messages

If desired, the messages can be batched (that is, a number of messages stored in a single file) with controls available to start a new file based on either time or size.   
  
The batch output file can also be compressed using the **zip** protocol if desired, with the messages appended to a single file which is then compressed, or with each message stored in a separate file in a compressed archive.   
  
Note that if a compressed batch output is configured, the **Suffix** field should be set to **.zip** to ensure the file is handled correctly by the Operating System.



### Directory - Bidirectional, In->Out and Out->In

### Bidirectional, In->Out and Out->In modes

When the communication point is configured for **Bi-directional**, **In->Out** or **Out->In** modes, the parameters for both input and output traffic are displayed for configuration and the communication point **must** be located at both the input and output of the route.

### TCP Server

The **TCP Server** communication point opens a nominated TCP port and listens for connections from remote servers.   
  
Although configured by default to handle a connection from only a single server, it can be configured to handle multiple connections and to use connection tracking to ensure a response goes back to the originating server.

All communication modes are supported by the TCP Server. It is characterized by the need for the remote host to establish a TCP (that is, network layer) connection to the communication point before message transfer is able to occur.

### TCP Server - Input Mode

In **Input** mode, the **TCP Server** communication point opens the port defined by the **Local Port** parameter. The value of the port number may be selected arbitrarily in the range 1025 - 65335 (port numbers below 1025 are reserved for system use). In general it is wise to avoid ports associated with well known protocols.

Once the port is open, the **TCP Server** waits for a remote host to initiate a TCP connection with it on the nominated port. Once the connection has been made, the TCP Server waits for messages to be transmitted by the remote host (that is, it remains in a passive, listener state).

In general, the port is made available on all network interfaces configured on the server, but it is possible to lock the port to a specific network interface on the host by defining the **Local Address** of the interface to use (defined by IP address).

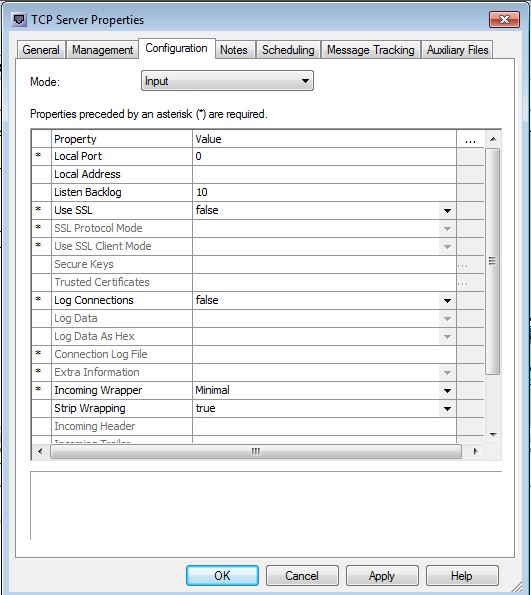
Messages are typically transmitted using codes to wrap the message which indicate where the start and end of the messages is, otherwise the data stream is unintelligible. Most commonly the HL7 standard wrappers are defined using the **Minimal** setting for the Incoming Wrapper. This requires that the message is preceded by the character "**\x0B**" (that is, the character represented by the hexadecimal code 0B), and has the characters "**\x1C\x0D**" appended.

If desired, the **Wrapper** codes may be set to:

* user defined values
* a hybrid model
* none (this requires that the data stream be managed such that the message can be correctly detected).

In the normal case, the **Wrapper** characters delimit the message rather than forming part of the message body. The option is therefore available to strip the **Wrapper** characters from the message before placing it into the message queue.

|  |  |
| --- | --- |
|  | Support is available for connection debugging using the **Log Connections** option and its associated parameters. |



### TCP Server - Server Output

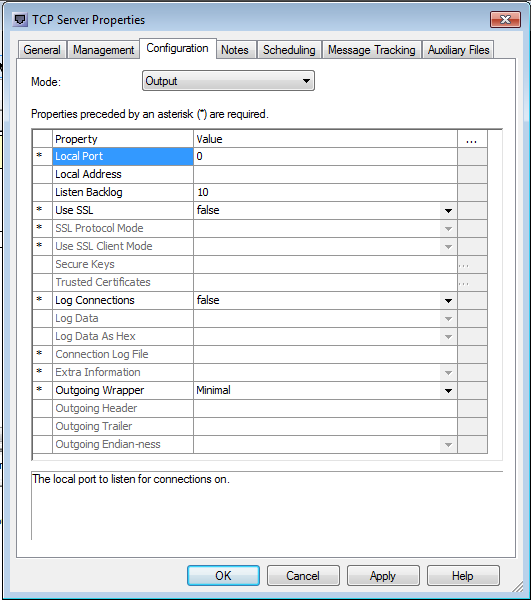
The behavior of the **TCP Server** in **Output** mode is similar to that of Input mode except the communication point transmits messages to the remote host.

Operation requires that the remote host initiates the TCP connection with the **TCP Server** communication point. Once the connection is in place, the communication point will transmit messages as they are received from the route.

This mode of operation is less frequently used but provides support for the situation where the remote host is not visible to the Rhapsody server.

For example, if the remote host is behind a firewall or a connection utilizing NAT (Network Address Translation) to manage IP addresses more effectively, the TCP connection must be initiated by the remote host.

The **TCP Server** communication point also requires specification of the message wrapper options and provides a range of logging options.



### TCP Server - Bidirectional, In->Out and Out->In

When the communication point is configured for Bi-directional, In->Out or Out->In modes, the parameters for both input and output traffic are displayed for configuration.

### TCP Client

The **TCP Client** communication point is designed to connect to a remote system and then to send or receive messages.

### TCP Client - Input Mode

In **Input** mode, the **TCP Client** communication point establishes a TCP connection with the remote server, and then waits to receive messages over the connection. This requires specification of the connection details for the remote host. The required details are its IP address or Host name and the TCP port to connect to on the remote host.

Normally, Rhapsody selects the appropriate network interface to use for the connection, but in a system with multiple interfaces, the specific interface to use may be specified with the **Local Address** parameter. Similarly, the **TCP port** used to originate the connection is usually randomly assigned in the range 1025 to 65535, but this may be locked to a specific port using the **Local Port** parameter.

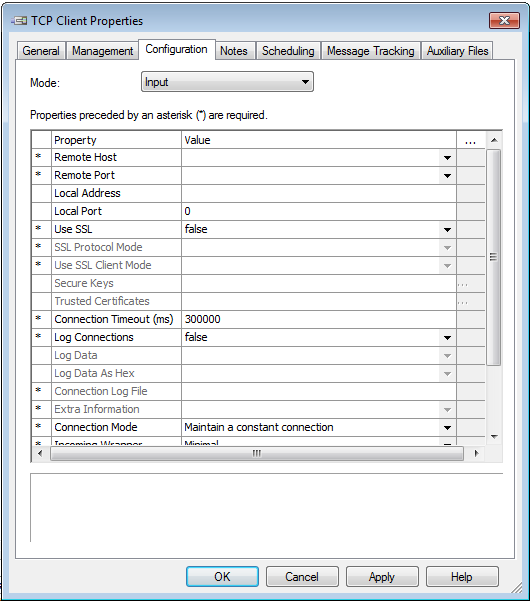
The Message Wrapper codes define the start and end of the message and are set by default to the Minimal HL7 wrapper set (message is preceded by "**\x0B**" and terminated by "**\x1C\x0D**" characters). Other wrapper code models may be selected from User defined or Hybrid models, or no wrapper codes defined.

The **Strip Wrapping** parameter enables the wrapper codes to be removed from the message where they do not form part of the body.

The **Connection Mode** parameter allows the connection to be permanently maintained in the case where messages are transmitted frequently, which saves the overhead of creating and tearing down the TCP connection for each message.

When the message traffic is infrequent, the **Connection Mode** can be set to create and tear down the connection for each message.

The TCP Client communication point provides support for de-bugging connection issues by means of the **Connection Log**. If **Log Connections** is enabled, each connection attempt will be noted. Additionally, the data can be logged either as text or as hexadecimal, the latter being useful for investigating issues with the message wrappers.



**TCP Client - Output**

When configured in **Output** mode, the **TCP Client** communication point establishes a connection to a remote host and then sends messages as they are made available from the route.

The connection details for the remote host are specified using the **Remote Host** and **Remote Port**parameters. The Remote Host is specified as either the IP address or the host name of the remote server.

Normally, Rhapsody selects the appropriate network interface to use for the connection, but in a system with multiple interfaces, the specific interface to use may be specified with the **Local Address**parameter. Similarly, the TCP port used to originate the connection is normally randomly assigned in the range 1025 to 65535, but this may be locked to a specific port using the Local Port parameter.

The Message Wrapper codes define the start and end of the message and are set by default to the Minimal HL7 wrapper set (message is preceded by "**\x0B**" and terminated by "**\x1C\x0D**" characters). Other wrapper code models may be selected from;

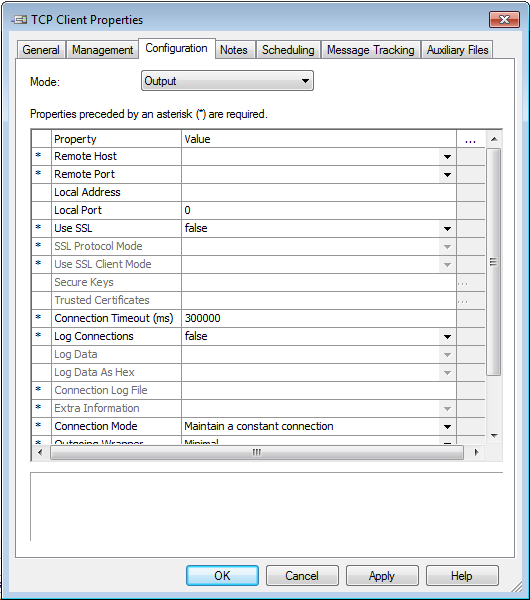
* User defined
* Hybrid models
* No wrapper codes defined

**Strip Wrapping Parameter**

The **Strip Wrapping** parameter enables the wrapper codes to be removed from the message where they do not form part of the body.

The **Connection Mode** parameter allows the connection to be permanently maintained in the case where messages are transmitted frequently which saves the overhead of creating and tearing down the TCP connection for each message. When the message traffic is infrequent, the Connection Mode can be set to create and tear down the connection for each message.

The **TCP Client** communication point provides support for de-bugging connection issues by means of the **Connection Log**. If **Log Connections** is enabled, each connection attempt will be noted. Additionally, the data can be logged either as text or as hexadecimal, the latter being useful for investigating issues with the message wrappers.



### TCP Client Bidirectional, In->Out and Out->In

The **TCP Client** communication point supports the **Bidrectional**, **In->Out** and **Out->In** connection modes.

The **Out->In** mode is frequently used when transmitting HL7 messages to support the requirement that a message be acknowledged before the next message can be transmitted.

### Sink

The **Sink** communication point provides a means to cleanly discard messages. It has no configurable parameters presented on the **Configuration** tab.

**Database**

The **Database** communication point provides support for reading from and writing to database tables. Native support is provided for:

* Microsoft SQL Server
* Oracle
* mySQL
* postgreSQL

Support is also available for third party databases for which jdbc drivers are available.

The Database communication point supports **Input**, **Output**, **Out->In** and **In->Out** connection modes.

Configuration of a **Database** Communication Point requires defining the connection details for the database (including driver, connection and authentication details) and defining the SQL statement(s) required to execute the task.

Extensive support for defining the SQL statements is provided via the **Database Configuration Editor** which provides support for multiple statements and includes the ability to selectively execute statements.

On input, the communication point queries a database and the record(s) are returned as an XML message.

For output, the message received from the route may be of any structure provided the fields can be referenced in the SQL statements, and fields from the message are used to update or add records to the database.

Note that definition of the SQL statements will be dealt with as a separate topic.

**Database - Input Mode**

In **Input** mode, the **Database** communication point executes SQL statements to query a database and return results. These results are returned in an XML format, and the communication point may be constrained to return a multi-row result as a single message or as multiple messages, each containing a single row from the query.

If required, the **Database** communication point will retain a pointer to assist in retrieving new records (that is, records which have been added to the tables since the last query).

The **Database** communication point is designed to repetitively query a database if a query returns records to assist in clearing data from the tables. The operation can be simply described as follows:

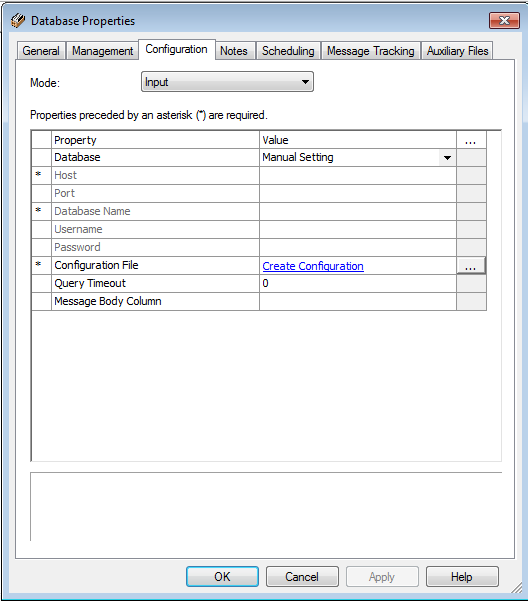
* Query database.
* Records returned as a message.
* Immediately query the database again.
* Records returned as a message.
* Immediately query the database again.
* No records returned.
* Sleep for the defined polling period.
* Query the database.

Consequently, if the input statements are not correctly defined, the Database communication point will repetitively query the database and return a very large number of messages quite rapidly.

**Configuration Requirements**

Configuration requires:

* Definition of the database connection details (note that if configured to **Manual Setting**, the details must be configured in the SQL Configuration file)
* Definition of the SQL statements in the Configuration file created by the **Database Configuration Editor**.



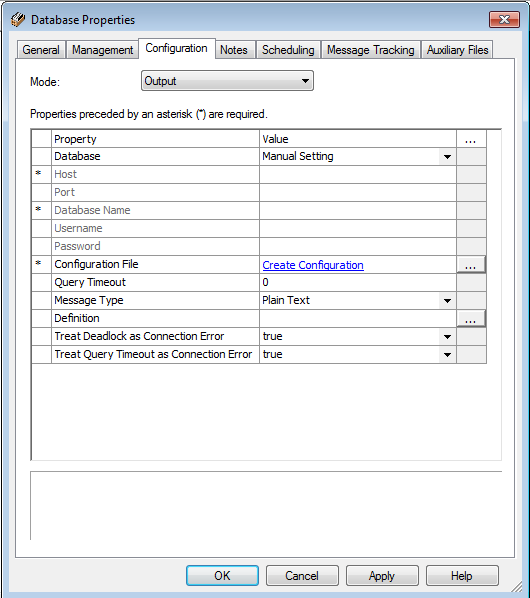
**Database - Output Mode**

In **Output** mode, the **Database** communication point will execute queries against a database as defined by the Configuration file. In the general case, the SQL statements in the configuration will extract fields from the message and write those to the database.

**Configuration Requirements**

Configuration requires:

* Definition of the database connection details.  
  Note that if configured to **Manual Setting**, the details must be configured in the SQL Configuration file.
* Definition of the SQL statements in the Configuration file created by the **Database Configuration Editor.**
* Definition of the message type as Plain text or XML.  
  If XML, the communication will automatically parse the message to make the fields available  
  If Plain Text, a definition should be supplied to allow parsing of the fields.



**Database Out->In**

In **Out->In** mode the configuration options are presented as for Output mode.

The Configuration file must be defined to support the selected mode.

* **Out->In** mode: The Database communication point acts in the same manner as the Database Lookup filter.  
    
  The configuration file is used to define Message Property creation and field substitution from input statements in the configuration file. If there are no input statements in the configuration, the message is passed through to the communication point input unchanged.

**Exercise 105B - Communication Points**

**Overview**

The preceding exercises have made use of the Directory, Sink and TCP Client communication points to illustrate the process of building and configuring routes and to illustrate the behaviour of some key configuration parameters.

We will now take a deeper look at the Directory and TCP communication points to examine the protocol specific details for these components.

In these exercises, we will undertake the following:

* **Directory communication point**
  + Input, Output and Bi-directional connection modes.
  + De-batching messages on Input and batching on Output.
* **TCP Client communication point**
  + Connecting to a remote client.
  + Using Rhapsody Variables.
  + Out->In connection mode.

**Connection modes**

Rhapsody provides the ability for communication points to operate in an asynchronous mode or a synchronous mode. The specific modes available to a communication point vary by type and purpose (for example, the Sink communication point only operates in an asynchronous mode - Output).

**Asynchronous Mode**

The asynchronous modes support the following modes;

* **Input**: the communication point is able to receive messages.
* **Output**: the communication point is able to send messages.
* **Bi-directional**: the communication point is both able and configured to send and receive messages independently of each other.  
    
  Note that if a communication point is in bi-directional mode, it must be present on the **Input** side of a route and on the **Output** side of a route - not necessarily the same route.

**Synchronous Mode**

The synchronous modes support the following modes:

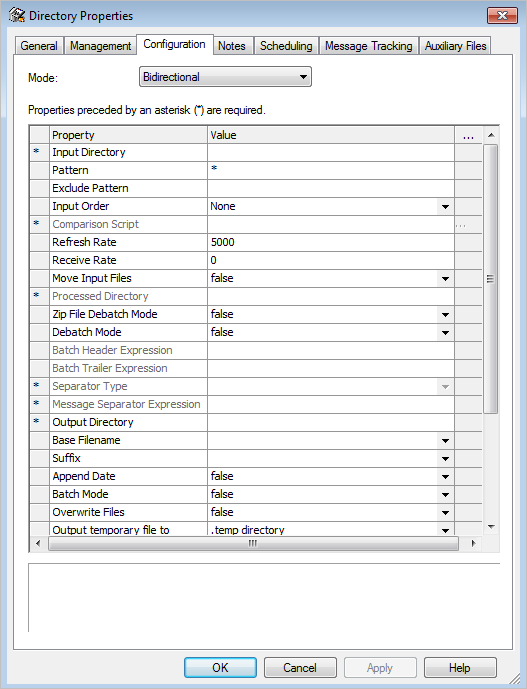
* **In->Out**: the communication point receives a message, then blocks input until it has sent the message (or its derivative after processing); then it is able to receive again.
* **Out->In**: the communication point sends a message, then blocks output until it receives a response.

Both of these synchronous modes require that the communication point be configured on the **Input** and the **Output** side of a route (note, not necessarily the same route).

### 1) The Directory Communication Point

The **Directory** communication point provides file oriented support for messaging, allowing messages to be read from files in a directory or written to files.

### Input Mode

In **Input** mode, the **Directory** communication point polls the nominated directory at regular intervals specified by a configuration parameter. If one or more files are present in the directory which have filenames matching the specified pattern, the contents of each file will be read into separate messages. The default configuration matches all files.

Once the file content has been read into the message, the file is deleted (or moved to a nominated processed files directory if desired). The Rhapsody engine must therefore have both read and write access to the input directory and the files to be able to input the files. Files are read only from the nominated directory; sub-folders and files in any sub-folders are ignored.

Support is also available to de-batch multiple messages held in a file based on a nominated separator (for example, an end of line marker), or to uncompress messages held in a zip archive. In both cases, each message extracted becomes an independent message in Rhapsody.

### Output Mode

In **Output** mode, the **Directory** communication point writes each message to a separate file in the nominated output directory. A pattern may be specified for constructing file names, and the messages may be batched into a file (multiple messages per file) or placed in a compressed archive file.

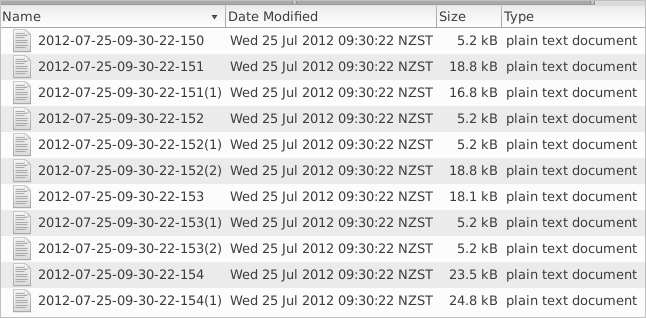
**Locations**

The **Directory** communication point requires that the directory locations be specified (as consistent with the **Connection Mode**).

**Exercise Tasks**

This exercise works with a Directory communication point in **Bidirectional** mode (which means that it both reads messages from files and writes output messages to files).

1. Create a working folder for this exercise.
2. Create a new route.
3. Create a Directory communication point and configure as follows:
   * Connection Mode: **Bidirectional**
   * **Input Directory**:   
     For example:  
     **/rhapsody/Users/your\_user\_name/105B/Input**
   * **Output Directory**:   
       
     For example:  
     **/rhapsody/Users/your\_user\_name/105B/Output**
4. Add the **Directory** communication point to both the **Input** and **Output** sections of the route (because the mode is **Bidirectional this is required** ).
5. Connect the input and output communication points.
6. Check in the route and start all the components.
7. Copy the test messages from exercise 105A into the **Input** folder. They will be detected by the communication point within 5 seconds (the default polling period) and input to the route, then output to the **Output** folder.
8. Verify that the files are in the **Output** folder.  
     
   Note that as we have not defined a filename structure for **Output**, the default naming convention will be used (output date and time to the millisecond).



It is common for multiple messages to be output within the same instant. In these cases, a copy number is added in parentheses.

**Locations continued**

The **Directory** locations in the previous task were specified using the full path to the locations.

Rhapsody supports the use of global values (Rhapsody Variables) to simplify repetetive use of values and also migration of values between server environments. The **Variable Manager** is accessed from the **View**, **Rhapsody Variables Manager**.

Rhapsody Variables may be used in the **Configuration** dialog wherever a value can be typed in.  This is done by:

* EITHER typing in the reference to the variable using the syntax **$(variable\_name).**
* OR by selecting the name from a popup list.

The default configuration for this course defines the variable **Users** to provide a path to the **/rhapsody/Users** directory. This variable can be combined with a sub-directory specification to simplify defining the path to the **Input** and **Output** directories. This can reduce the risk of typographical error in defining the path.

**Exercise Tasks**

1. Modify the configuration of the **Directory** communication point to specify:
   * Input Directory: **$(Users)/your\_user\_name/path\_to\_input\_folder**
   * Output Directory: **$(Users)/your\_user\_name/path\_to\_output\_folder**
2. Check in the changes.
3. Confirm that the operation of the communication point is unchanged (as in the previous exercise).

Notice that the Output Directory section of the dialog provides both a path component (**Output Directory**) and a filename component (**Base Filename**).

The **Output Directory** parameter is static and only evaluated when the component starts, but the **Base Filename** parameter is dynamic (unless batching is specified). Consequently, a message property may be constructed as the message traverses the route which includes both a sub-directory path as well as the file name.

The **Output** configuration also includes the ability to define the suffix for output messages. This is appended to the constructed filename.

**Masks**

File selection may be further refined by means of pattern masks which define the set of file names to be included or excluded by the communication point. The **Pattern** and **Exclude Pattern** of the **Configuration** dialog provide the ability to refine the selection of files from the **Input** directory.

**Pattern Specification**

The **Pattern** specification requires a complete mask to match the filename. The default uses the wildcard **\*** to provide a complete match.  The pattern may be built using a combination of text and  **\*** to define the portions of the file names that are not important.

For example:

* **\*** - defines a complete match
* **\*.txt** - defines all filenames ending in **.txt**
* **Input\*** - defines all filenames beginning with Input

Note that **\*** is the only wildard recognized in the pattern.

**Exclude Pattern Specification**

The **Exclude Pattern** also requires a complete mask specification and will not read files with names matching the exclude mask. For example;

* **\*.txt** will not input files ending with the term **.txt**  
  Note that the pattern matches to the filename string, so the period (**.**) is part of the matched string.

**Exercise Tasks**

* 1. Modify the **Exclude pattern** of the Directory communication point to exclude all test filenames that include the string **".t"** in their name. Hint - you will need to use wildcards.
  2. Ensure that the route and components are checked in and running.
  3. Copy the test messages to the **Input** directory.

Note that after a few seconds, all files should have been removed from the **Input** directory (and passed on to the route) except for those with the exclude pattern in their names (**ADTA01.tmp** and **ADTA01.txt**).

**Working with Batches**

**De-batch mode**

It is common for file based messaging to include a batch of messages in a single file. The **Directory** communication point supports extracting the individual messages from the file using:

* EITHER a marker for the start of each message.
* OR a separator at the end of each messages.

Using the **Directory** communication point to de-batch the messages requires:

* Analysis of the file to identify the type and specification for the separator.
* Configuration of the Directory communication point:
  + **De-batch Mode**: **true**
  + **Separator Type**: as required
  + **Separator Expression**: as derived from the analysis.

The **Addresses.csv** test file is a list of addresses in CSV format. This file was used in earlier exercises and it was noted that the end of each line is marked by the Linefeed character (represented as **\n**).

**Exercise Tasks (de-batch)**

1. Reconfigure the **Directory** communication point:
   * **De-batch Mode:** true
   * **Separator Type:** Separates messages
   * **Message Separator Expression:\n**
2. Check in all components and ensure that they are running.
3. Copy the **Addresses.csv** file to the input folder.
4. Verify the results in WMC and in the output folder.

**Batch mode**

In **output** mode, the **Directory** communication point may be configured to stream all ouput messages into a batched or archive file. As it is possible for files to grow in size rapidly in this mode, options are available to roll the file over (that is, close the file and open a new output file) based on size or time constraints.

**Exercise Tasks (batch)**

1. Modify the route to contain **two** Directory comunication points (1 input, 1 output).
2. Reconfigure the **output Directory** communication point:
   * **Batch Mode:** true (other settings left as default).
3. Check in all components and ensure that they are running.
4. Copy the set of test messages to the input folder.
5. Verify the results in WMC and in the output folder.

### 2) The TCP Client Communication Point - Output

Rhapsody supports message transmission over a network via TCP communications. The **TCP Client Communication Point** manages communication when it is the responsibility of the Rhapsody engine to initiate communications with remote servers, while the **TCP server** supports opening a network port for a remote host to initiate a connection.

This exercise examines the key configuration parameters for the TCP Client communication point;

* Specification of **Remote Host** and **Port**.
* Purpose and specification of the **Wrapper.**
* The **Out->In Connection Mode** (synchronous behavior)**.**

### Remote Host and Port

The **Remote Host** field in the **TCP Client** configuration identifies the host  to which a connection should be made. The **hostname** may be a valid hostname or an IP address (network address) or a Rhapsody Variable which contains the hostname details.

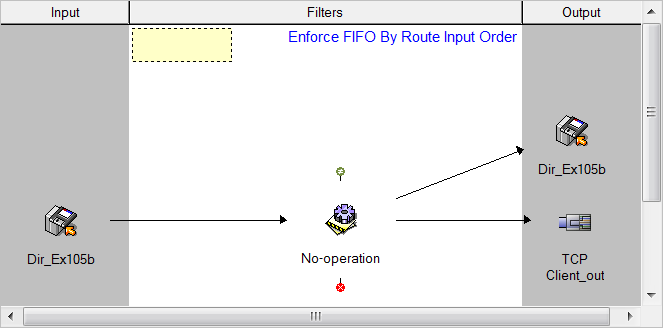
The **Remote Port** identifies the network port to which the connection will be made, and may be specified either as the port number (typically in the range 1025 - 65536) or a Rhapsody Variable which contains the value.

The Rhapsody Variables **TCP\_Remote\_Host** and **TCP\_Remote\_Port** may have been pre-defined. You can define or change Rhapsody Variables in the Rhapsody Variables Manager (from the IDE toolbar dropdown menu View, Rhapsody Variables Manager).

#### Rhapsody Variable Assignments

|  |  |
| --- | --- |
| **Variable Name** | **Value** |
| TCP\_Remote\_Host | localhost |
| TCP\_Remote\_Port | 10003 |

### Exercise Tasks

1. Create a **TCP Client** Communication Point configured as follows:
   * **Connection Mode**: **Output**
   * **Remote Host**: **$(TCP\_Remote\_Host)**
   * **Remote Port**: **$(TCP\_Remote\_Port)**
   * All other required parameters as default values.
2. Add a **No-operation** filter to the route and connect it to the **TCP Client** communication point as shown below.  
     
   
3. Check in all of the modified components.
4. Start all components.
5. Ensure the **TCP Client** Communication Point starts without error.
   * (hint - check the communication point details page in the WMC).
6. Copy a test message to the input folder.
7. Verify that the test message is sent by the **TCP Client Communication Point.**
   * Hint - use the WMC to check the message at the **TCP Client Communication Point** output of the Message Events tree.

### Wrapper

Sending and receiving messages over a TCP connection requires that the communication point is able to identify the start and the finish of the message (otherwise it may effectively have a continuous stream of data).

The **Wrapper** configuration parameter allows definition of characters which will be in the data stream to identify the start and end of a message. Any other characters which are received by the communication point which are not within a message will be silently discarded.

The default **Wrapper** characters are defined as the **HL7 MLLP** characters (Minimal Lower Layer Protocol). The communication point also supports:

* **Hybrid** (MLLP plus a checksum)
* User defined
* No wrapper

### TCP Client - Out->In Mode I

The **synchronous** modes of operation (**In->Out, Out->In**) permit the communication point to manage the messaging flow, if required (**application level**).

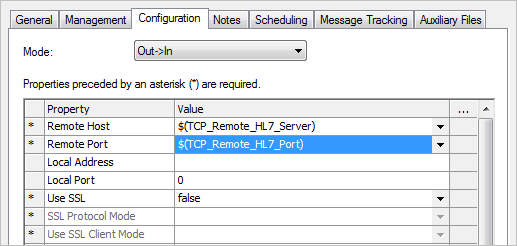
The **Out->In** mode is used in HL7 messaging to ensure that the remote host acknowledges the receipt of the message before the next message is transmitted to it.

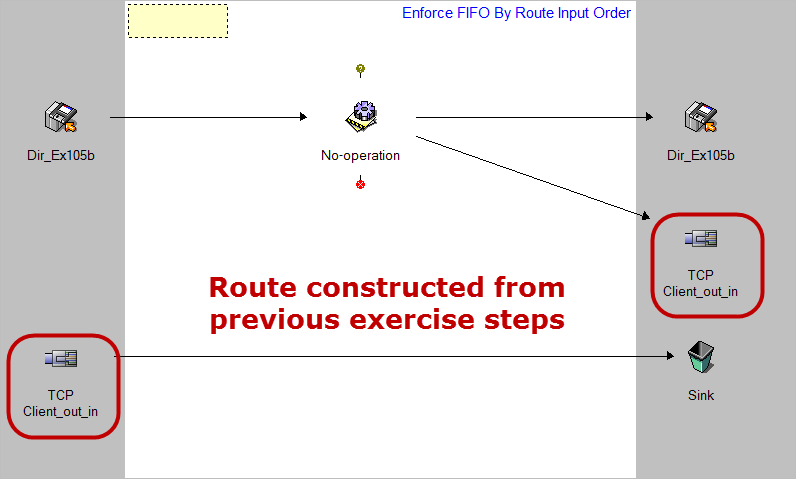
### Exercise Tasks (response received)

This exercise establishes a working connection to a remote server and receives a response from the remote server.

#### Rhapsody Variable Assignments

|  |  |
| --- | --- |
| **Variable Name** | **Value** |
| TCP\_Remote\_HL7\_Server | localhost |
| TCP\_Remote\_HL7\_Port | 10004 |
| TCP\_Blocking\_Port | 10002 |

1. Create/modify the **TCP Client Communication Point** to be configured as follows:
   * **Connection Mode**: **Out->In**
   * **Remote Host**: **$(TCP\_Remote\_HL7\_Server)**
   * **Remote Port**: **$(TCP\_Remote\_HL7\_Port)**  
       
     
2. Add this communication point to the route constructed in the last exercise (it needs to be in the output and input zones).



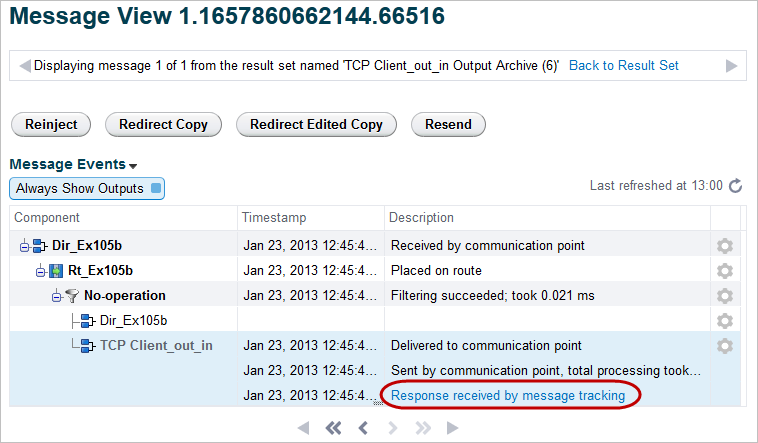
1. Create a Sink communication point and add it to the route.
2. Connect the route and ensure all components are checked in and running.
3. Copy one of the HL7 test messages from Exercise 105A to the input folder.
4. Use the WMC to examine the message at the**TCP Client\_out\_in** communication point.
5. Using the Message View, confirm that a response was received from the server.
6. Compare the message at the output (**View Output Archive** link) and at the input (**View Input Archive** link) for this communication point.

Note that once the response is received, the message events indicate **Response received by message tracking**. In synchronous mode, this indicates that the communication point has received the response to the message which has been output. The next message can then be sent.

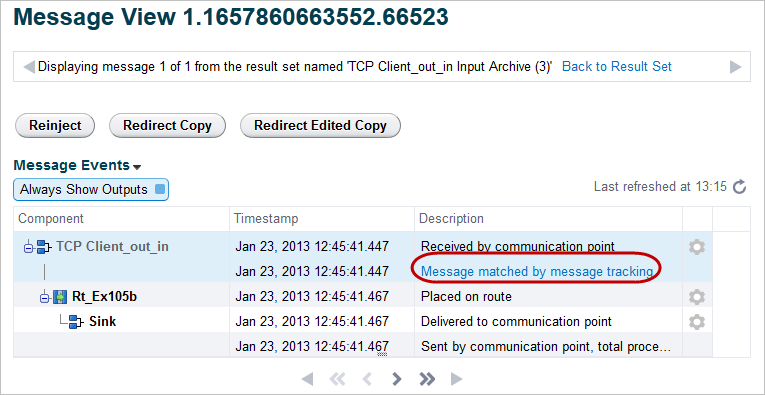
### 105B Model Answer (1)

### What you expect to see:

**Message View at output**:



**Message View at input**:



### TCP Client - Out->In Mode II

### Exercise 2 (no response received)

The **TCP Client** communication point provides support to manage the behavior when a **network** connection to the remote host fails. The retry behavior is configured on the **General** tab (**Connection Retries** panel) of its configuration dialog (see Exercise 105A).

This exercise illustrates the sequence of events if the remote server does not respond to a message in a timely manner.

The **Out->In Properties** dialog provides support for the situation where a response is not received from the server. This permits recovery when the message is sent but no response is received within the period defined by the **Response Timeout** and ensures that the communication point does not wait for the response indefinitely and block output of later messages.

#### Rhapsody Variable Assignment

|  |  |
| --- | --- |
| **Variable Name** | **Value** |
| TCP\_Remote\_HL7\_Server | localhost |
| TCP\_Remote\_HL7\_Port | 10004 |
| TCP\_Blocking\_Port | 10002 |

### Exercise Tasks (no response received)

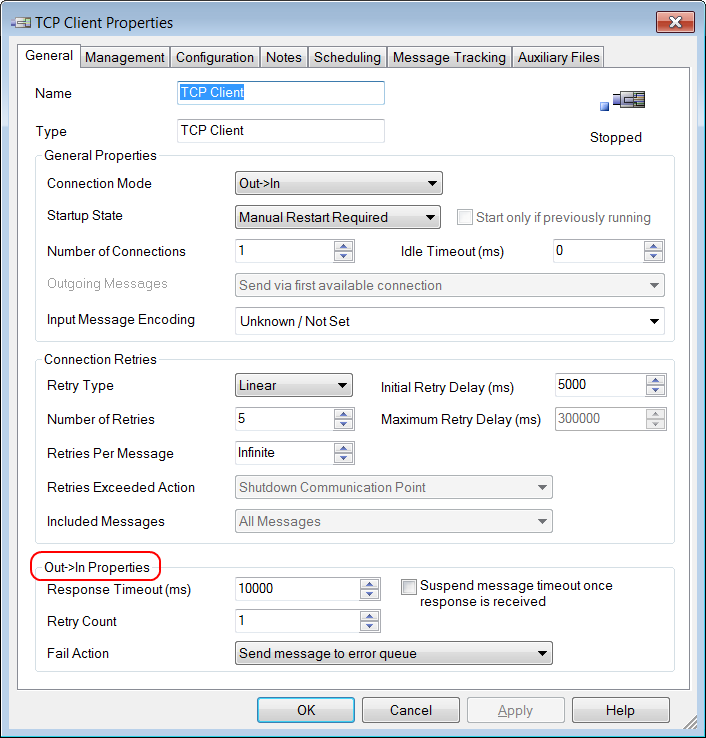
Reconfigure the **TCP Client** communication point as follows:

**Configuration** tab

* Remote Port: $(TCP\_Blocking\_Port)

**General** tab (Out -> In Properties))

* Response Timeout: 10000
* Retry Count: 1
* Fail Action: Send message to error queue



This configuration attempts to send the message to the server. The communication point will wait for the response for 10 seconds. If no response is received from the server, the message is re-sent and Rhapsody will wait for a further 10 seconds and then invoke the **Fail Action** if no response is received. This is the expected behavior for this exercise.

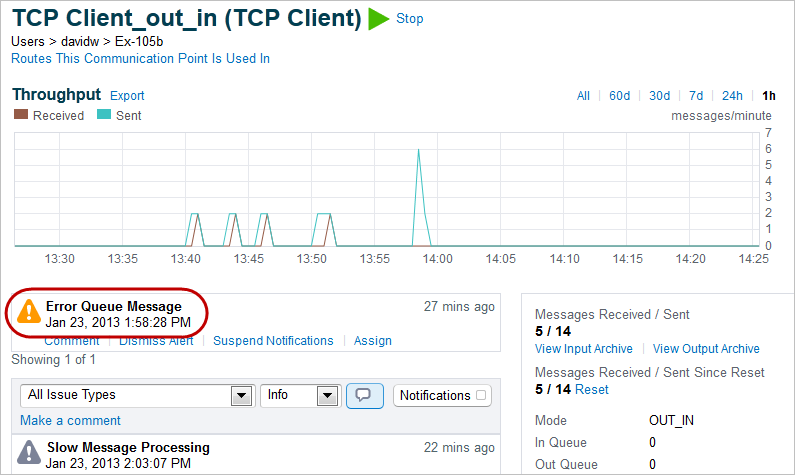
1. Ensure all components are checked in and running.
2. Copy one of the HL7 test messages from Exercise 105A to the input folder.
3. Use the WMC to examine the message at the **TCP Client\_out\_in** communication point.
4. Use the **Message View** to confirm that a response was not received from the server and that the message was sent to the **Error Queue.**

### 105B Model Answer (2)

### What you expect to see:

**TCP Client communication Point details:**

Here you will see an alert that a message has been delivered to the **Error Queue:**

****

**Message View at output**:

At the output **Message View** the **Message Events** will list the initial sending of the message and the subsequent resend 10 seconds later (because the **Response Timeout** requires that the response be received within 10 seconds, otherwise the message is re-sent).

