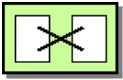


Enterprise Integration Patterns Tutorial Reference Chart



Message Translator

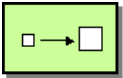
A Message Translator translates one application's message format into another.

Command Line:

```
call Translator <inputChannel> <outputChannel> <Transformation XSL File>
```

Example:

```
call Translator orderChannel orderEnrichedChannel Order2OrderItem.xsl
```



Content Enricher

A Content Enricher, a specialized transformer, augments a message with missing information either by accessing an external data source or by performing a computation on the message data. This kit comes with a pre-built Enricher that takes an Order Message and copies the Order ID and the number of items into each <Item> element:

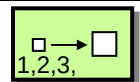
<pre><Order OrderID="2"> <Item>COFFEE</Item> <Item>LATTE</Item> </Order></pre>		<pre><Order> <Item OrderID="2" TotalItems="2">COFFEE</Item> <Item OrderID="2" TotalItems="2">LATTE</Item> </Order></pre>
--	--	--

Command Line:

```
call Enricher <inputChannel> <outputChannel>
```

Example:

```
call Enricher orderChannel orderEnrichedChannel
```



Sequencer Tagger

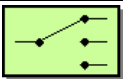
A Sequence Tagger is a special Enricher that adds consecutive numbers to messages. The XPath argument specifies where to insert the number into the XML document.

Command Line:

```
call SequenceTagger <inputChannel> <outputChannel> <xPath>
```

Example:

```
call SequenceTagger orderChannel orderTaggedChannel "/Order/@OrderID"
```



Message Router

A Message Router consumes one Message from a Message Channel and republishes it to one of multiple potential Channels, depending on a set of conditions.

Command Line:

```
call Router <inputChannel> <outputCh1> <condition1> <outputCh2>
```

Example:

```
call Router itemChannel coldBevChannel "Item = 'FRAPPUCINO'" hotBevChannel
```

Enterprise Integration Patterns Tutorial Reference Chart



Delay

A Delay consumes a message and republishes it after waiting for a specified interval.

Command Line:

```
call Delay <inputChannel> <outputChannel> <delayInMilleseconds>
```

Example:

```
call Delay invoiceChannel completedOrderChannel 1000
```



Barista

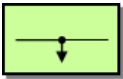
The Barista processes messages from an input channel. When the Barista is done processing one message it forwards it to the output channel. Essentially, a Barista is a Delay component.

Command Line:

```
call Barista <inputChannel> <outputChannel>
```

Example:

```
call Barista orderChannel orderCompletedChannel
```



Tee (Wire Tap)

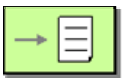
A Wire Tap publishes each incoming message to both the main channel and a secondary channel.

Command Line:

```
call Tee <inputChannel> <outputChannel> <secondaryOutputChannel>
```

Example:

```
call Tee orderChannel orderChannel2 logChannel
```



Logger

A Logger writes incoming messages to a log window. You can use it to manually inspect messages.

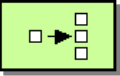
Command Line:

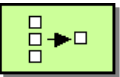
```
call Logger <inputChannel>
```

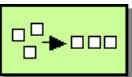
Example:

```
call Logger completedOrderChannel
```

Enterprise Integration Patterns Tutorial Reference Chart

	Splitter
A Splitter breaks out a composite message into a series of individual messages, each containing data related to one item:	
<pre><Order OrderID="2"> <Item>COFFEE</Item> <Item>LATTE</Item> </Order></pre>	 <pre><Item>COFFEE</Item> <Item>LATTE</Item></pre>
Command Line: call Splitter <inputChannel> <outputChannel> <XPath>	
Example: call Splitter orderChannel orderItemChannel "/Order/Item"	

	Aggregator
An Aggregator, a stateful filter, collects and store individual messages until a complete set of related messages has been received. Then, the Aggregator publishes a single message distilled from the individual messages. An Aggregator requires each message belonging to the same group to contain a correlation identifier with the same value. The correlationXPath specifies the location of this identifier within the message.	
Command Line: call Aggregator <inputChannel> <outputChannel> <correlationXPath>	
Example: call Aggregator completedItemChannel completedOrderChannel "/Item/@OrderID"	

	Resequencer
A Resequencer, a stateful filter, collects and re-orders messages so that they can be published to the output channel in a specified order. A Resequencer requires each message to be tagged with a consecutive message number. The XPath Expression has to point to the element that numbers each message.	
Command Line: call Resequencer <inputChannel> <outputChannel> <XPathExpression>	
Example: call Resequencer completedItemChannel completedOrderChannel "/Order/@OrderID"	

Enterprise Integration Patterns Tutorial Reference Chart

	Customer
The Customer component places orders in XML format on a channel. The format of an order is as follows: <pre><Order> <Item>COFFEE</Item> <Item>LATTE</Item> <Item>FRAPPUCINO</Item> </Order></pre>	
Command Line: call Customer <channel>	
Example: call Customer orderChannel	

	Manual Step
The Manual Step consumes messages and displays them in a list. The user can then inspect each message being held and release them in any arbitrary order. The Manual Step is a great tool to simulate out of order scenarios or to send a batch of messages at once. The message label argument is an optional XPath expression that determines what to display in the list for each message.	
Command Line: call ManualStep <inputChannel> <outputChannel> <MessageLabel>"	
Example: call ManualStep itemChannel completedChannel "concat('Order ', /Item/@OrderID)"	

Prerequisites

- ☐ Windows 2000, Windows XP or Windows Server 2003
- ☐ Microsoft .NET Framework 1.1
- ☐ The Message Queuing component of the Windows operating system. This component is part of the operating system, but is not installed by default. To install this component, enter the Control Panel, select Add or Remove Programs. On the icon bar on the left select Add/Remove Windows Components. Select the checkbox next to Message Queuing. Since this toolkit uses only local queues, you do not have to install the Active Directory Integration subcomponent.

Setup Instructions

1. Copy the ZIP file MessagingKitInstall.zip to a folder of your choice.
2. Extract the ZIP file (make sure to preserve the path name of the files)
3. Start the Message Queuing service:
 - a. Open a command prompt
 - b. Type " net start "Message Queuing"
 - c. You should see "The Message Queuing service was started successfully." (or you might see a message indicating that the service was already running).
4. Your working folder will be the folder "Exercises" that was created inside your folder.

Useful Tips

- If you create a batch script, remember use the keyword "call" before each component. This is required to allow the batch script to start a series of individual components.
- Beware of typos. If you mistype a channel name, messages will go into the ether.
- Remember to close all components after you complete an exercise. Lingering components might consume messages without your knowing.
- Use Tees and Loggers to see what is going on in your system. Remember to use a Tee so you can both log a message and pass it to a subsequent component.
- To see all message channels defined on your computer and how many messages they hold, do the following:
 1. Right-click on "My Computer" and chose Manage
 2. Expand the node "Services and Applications"
 3. Expand the node "Message Queuing"
 4. Expand the node "Private Queues"