

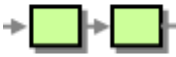
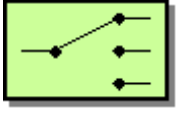
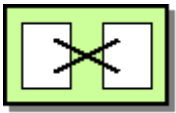
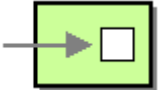


## Enterprise Integration Patterns with WSO2 Enterprise Integrator



Enterprise Application Integration (EAI) is key to connecting business applications with heterogeneous systems. Over the years, architects of integration solutions have invented their own blend of patterns in a variety of ways. However, most of these architectures have similarities, initiating a set of widely-accepted standards in architecting integration patterns. Most of these standards are described in the [Enterprise Integration Patterns Catalog](#).





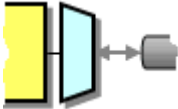


In this guide, we have shown how each pattern in the Patterns Catalog can be simulated using various constructs in **the ESB profile of WSO2 Enterprise Integrator (EI)**. Click on a topic in the list below for details.

### Messaging Systems


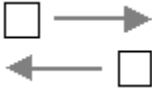
	<a href="#">Message Channels</a>	How one application communicates with another using messaging.
	<a href="#">Message</a>	How two applications connected by a message channel exchange a piece of information.
	<a href="#">Pipes and Filters</a>	How to perform complex processing on a message while maintaining independence and flexibility.
	<a href="#">Message Router</a>	How to decouple individual processing steps so that messages can be passed to different filters depending on conditions.
	<a href="#">Message Translator</a>	How systems using different data formats communicate with each other using messaging.
	<a href="#">Message Endpoint</a>	How an application connects to a messaging channel to send and receive messages.

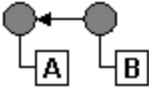
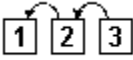

### Messaging Channels

	<a href="#">Point-to-Point Channel</a>	How the caller can be sure that exactly one receiver will receive the document or perform the call.
	<a href="#">Publish-Subscribe Channel</a>	How the sender broadcasts an event to all interested receivers.

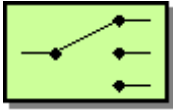

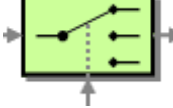



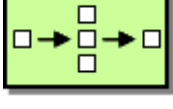
	<a href="#"><u>Datatype Channel</u></a>	How the application sends a data item such that the receiver will know how to process it.
	<a href="#"><u>Invalid Message Channel</u></a>	How a messaging receiver gracefully handles a message that makes no sense.
	<a href="#"><u>Dead Letter Channel</u></a>	What the messaging system does with a message it cannot deliver.
	<a href="#"><u>Guaranteed Delivery</u></a>	How the sender ensures delivery of a message, even if the messaging system fails.
	<a href="#"><u>Channel Adapter</u></a>	How to connect an application to the messaging system to send/receive messages.
	<a href="#"><u>Messaging Bridge</u></a>	How multiple messaging systems can be connected so that messages available on one are also available on the others.
	<a href="#"><u>Message Bus</u></a>	An architecture enabling separate applications to work together in a decoupled fashion such that applications can be easily added or removed without affecting the others.

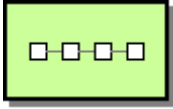
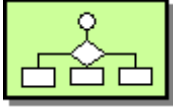

## **Message Construction**

	<a href="#"><u>Command Message</u></a>	How messaging can be used to invoke a procedure in another application.
	<a href="#"><u>Document Message</u></a>	How messaging can be used to transfer data between applications.
	<a href="#"><u>Event Message</u></a>	How messaging can be used to transmit events from one application to another.
	<a href="#"><u>Request-Reply</u></a>	How an application that sends a message gets a response from the receiver.
	<a href="#"><u>Return Address</u></a>	How a replier knows where to send the reply.


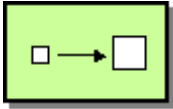
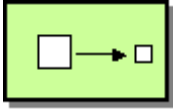
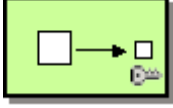

	<a href="#"><u>Correlation Identifier</u></a>	How a requester that has received a reply knows which request the reply is for.
	<a href="#"><u>Message Sequence</u></a>	How messaging can transmit an arbitrarily large amount of data.
	<a href="#"><u>Message Expiration</u></a>	How a sender indicates when a message should be considered stale and therefore should not be processed.
	<a href="#"><u>Format Indicator</u></a>	How a message's data format can be designed to allow for possible future changes.

## **Message Routing**

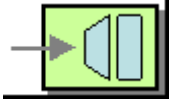
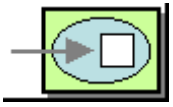
	<a href="#"><u>Content-Based Router</u></a>	How to handle a situation when the implementation of a single logical function (such as an inventory check) is spread across multiple physical systems.
	<a href="#"><u>Message Filter</u></a>	How a component avoids receiving uninteresting messages.
	<a href="#"><u>Dynamic Router</u></a>	How to avoid the dependency of a router in all possible destinations, while maintaining its efficiency.
	<a href="#"><u>Recipient List</u></a>	How to route a message to a list of dynamically specified recipients.
	<a href="#"><u>Splitter</u></a>	How to process a message if it contains multiple elements, each of which may have to be processed in a different way.
	<a href="#"><u>Aggregator</u></a>	How to combine the results of individual but related messages so that they can be processed as a whole.
	<a href="#"><u>Composed Msg. Processor</u></a>	How to maintain the overall flow when processing a message consisting of multiple elements, each of which may require different processing.
	<a href="#"><u>Scatter-Gather</u></a>	How to maintain the overall flow when a message needs to be sent to multiple recipients, each of which may send a reply.

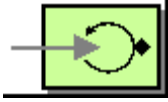
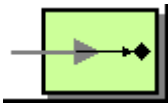
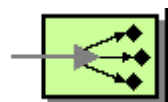
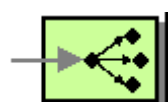
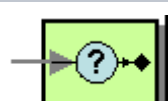

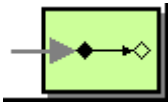
	<a href="#"><u>Routing Slip</u></a>	How to route a message consecutively through a series of steps when the sequence of the steps is not known at design time and may vary for each message.
	<a href="#"><u>Process Manager</u></a>	How to route a message through multiple processing steps, when the required steps may not be known at design time and may not be sequential.
	<a href="#"><u>Message Broker</u></a>	How to decouple the destination of a message from the sender and maintain central control over the flow of messages.

## Message Transformation



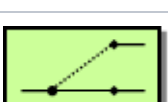
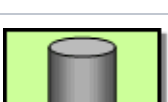
	<a href="#"><u>Envelope Wrapper</u></a>	How existing systems participate in a messaging exchange, which places specific requirements in the message format, such as message header fields or encryption.
	<a href="#"><u>Content Enricher</u></a>	How to communicate with another system if the message originator does not have all the required data items available.
	<a href="#"><u>Content Filter</u></a>	How to simplify dealing with a large message when you are interested only in a few data items.
	<a href="#"><u>Claim Check</u></a>	How to reduce the data volume of a message sent across the system without sacrificing information content.
	<a href="#"><u>Normalizer</u></a>	How to process messages that are semantically equivalent but arrive in a different format.
	<a href="#"><u>Canonical Data Model</u></a>	How to minimize dependencies when integrating applications that use different data formats.

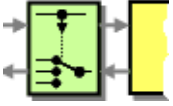
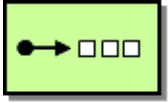

## Messaging Endpoints

	<a href="#"><u>Messaging Gateway</u></a>	How to encapsulate access to the messaging system from the rest of the application.
	<a href="#"><u>Messaging Mapper</u></a>	How to move data between domain objects and the messaging infrastructure, while keeping the two independent of each other.
	<a href="#"><u>Transactional Client</u></a>	How a client controls its transactions with the messaging system.

	<a href="#"><u>Polling Consumer</u></a>	How an application consumes a message when the application is ready.
	<a href="#"><u>Event-Driven Consumer</u></a>	How an application automatically consumes messages as they become available.
	<a href="#"><u>Competing Consumers</u></a>	How a messaging client processes multiple messages concurrently.
	<a href="#"><u>Message Dispatcher</u></a>	How multiple consumers on a single channel coordinate their message processing.
	<a href="#"><u>Selective Consumer</u></a>	How a message consumer selects which messages to receive.
	<a href="#"><u>Durable Subscriber</u></a>	How a subscriber avoids missing messages while it is not listening for them.
	Idempotent Receiver	How a message receiver deals with duplicate messages.
	<a href="#"><u>Service Activator</u></a>	How an application designs a service to be invoked via both messaging and non-messaging techniques.

## **System Management**

	<a href="#"><u>Channel Purger</u></a>	Removes unwanted messages, which can disturb tests or running systems, from a channel.
	<a href="#"><u>Control Bus</u></a>	Administers a messaging system that is distributed across multiple platforms and a wide geographic area.
	<a href="#"><u>Detour</u></a>	Routes a message through intermediate steps to perform validation, testing or debugging functions.
	<a href="#"><u>Message History</u></a>	Lists all applications that the message passed through since its origination.
	<a href="#"><u>Message Store</u></a>	Reports against message information without disturbing the loosely coupled and transient nature of a messaging system.

	<a href="#"><u>Smart Proxy</u></a>	Tracks messages on a service that publishes reply messages to the Return Address specified by the requester.
	<a href="#"><u>Test Message</u></a>	Ensures the health of message processing components by preventing situations such as garbling outgoing messages due to an internal fault.
	<a href="#"><u>Wire Tap</u></a>	Inspects messages that travel on a Point-to-Point Channel.