Direct Request-Reply

In this tutorial, we will discuss another common Message Exchange Pattern (MEP) called request-reply. This is where an application (requestor) is expecting a response from a backend service (replier); those familiar with RESTful APIs should be familiar with this pattern, as opposed to the unidirectional pattern of publish/subscribe. This pattern is often implemented as a blocking API call, which simplifies application code from having to correlate asynchronous replies with outstanding requests.

Notice that “under the covers” a request-reply message interaction is just a bi-directional publish/subscribe interaction: the Requestor (sender) publishes a message which the Replier (receiver) receives, and then responds by publishing a response message back to the Requestor application.

## Requestor Helper Object

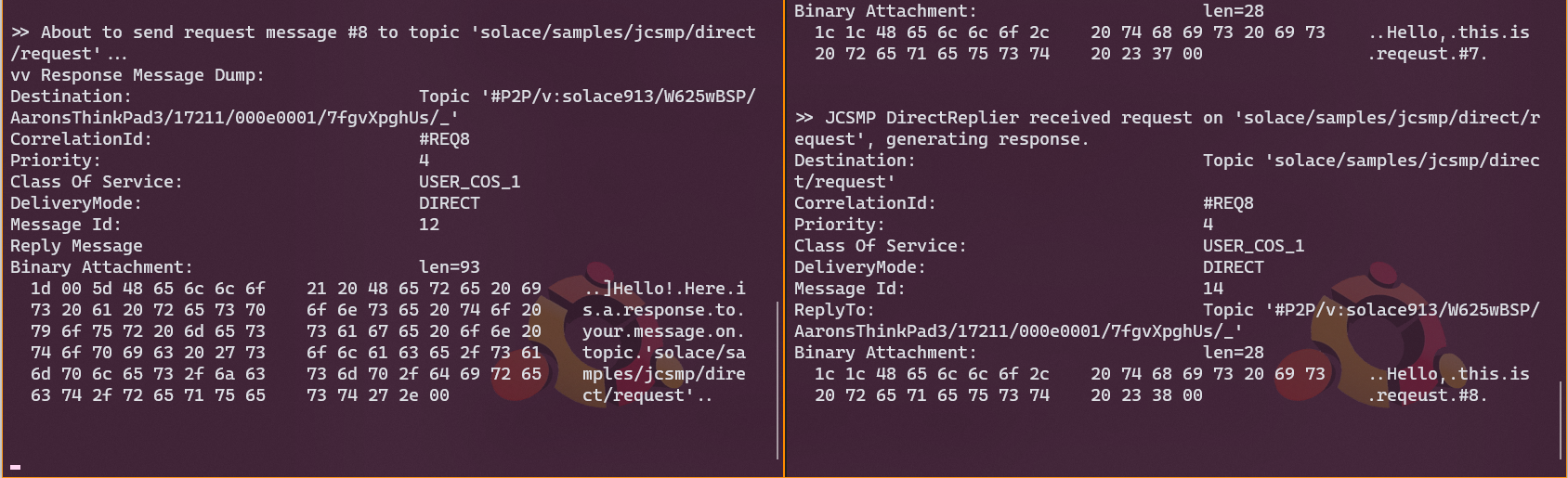
In both classic and NextGen Solace APIs, there are helper objects and methods designed to aid the application developer making request-reply calls. This allows a requestor application to make a single API request() call that blocks, and returns a message with the response when it is received. Similarly, a Replier application has helper methods to automatically respond to a request message that is received.

### Additional Implementation Details

Under-the-covers, this is implemented by automatically populating a message with a number of message properties:

* **Reply-To Topic**: when the Requestor publishes the request message, this field is populated so that the Replier application knows where to send the response. By default, the Requestor’s “inbox topic” is used, which is a topic similar to #P2P/abc123
* **Correlation ID**: this is a per-request identifier that is send in the request message that the Replier application will pass back via the response message. It allows the Requestor’s API to know which Request (if there are multiple outstanding from multiple threads) object/thread to unblock and deliver the response message to.
* **Reply Message Flag**: this is an optional message flag set by the Replier application when responding to a request message to indicate to the Requestor API that this incoming message is in response to a request.

If you are dump()ing the message payloads to the console on both Requestor and Replier applications, you should see these fields and values being set.



Requestor on left, Replier on right

## Running the Replier App

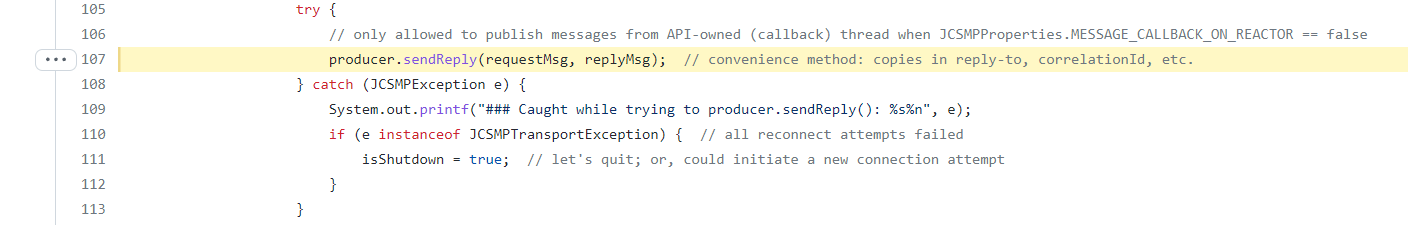
Before a Requestor can make a request, the backend Replier application must be running. This is especially true since these samples use Direct messaging, which will only pass messages from sender to receiver if both applications are connected at the same time.

The topic that the Replier subscribes to must be known in advance by the Requestor (or at least discoverable somehow). This concept is called a “*well-known topic*” or WKT. It is similar to both actors in a standard RESTful API knowing what the correct URL is for a given service call. However one big difference with Solace is if both Requestor and Replier are connected to different brokers in a distributed Event Mesh, the topics and subscriptions will get routed across the mesh and the applications can still communicate as if they were connected to the same broker.

Note that majority of the application initialization code is very familiar from other samples. The largest difference is inside the message receive function (either onReceive() for async, or after receive() for sync), where it checks to ensure the message is request message (by the presence of a “reply-to” topic metadata), and it constructs a response message based on the request.

[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/master/src/main/java/com/solace/samples/jcsmp/patterns/DirectReplier.java#L93-L104)

One the response message has been constructed, it uses an API helper function sendReply() – instead of just plain send() – which tells the API to copy the reply-to topic and correlation ID out of the request message to use in the response message.

[](https://github.com/SolaceSamples/solace-samples-java-jcsmp/blob/master/src/main/java/com/solace/samples/jcsmp/patterns/DirectReplier.java#L107)

<https://docs.solace.com/Solace-PubSub-Messaging-APIs/API-Developer-Guide/Receiving-Direct-Message.htm#reply>

## Requestor Application

Once the backend Replier application is running, the Requestor application can start up and start to submit requests. Since this is a Direct messaging application, it can use the API convenience Requestor object, which sends a request message and blocks the thread waiting for the response. As with the Replier application, this application must be able to both send and receive messages, so it must configure both a Producer and Consumer object.

Once connected, it can send a request message using the sendReply() method. If a timeout value of 0 is passed in, the call becomes non-blocking and returns immediately.

<https://docs.solace.com/Solace-PubSub-Messaging-APIs/API-Developer-Guide/Publishing-Messages-that.htm>

# Building the Samples

Examples for each?