Spring Integration Support

In Spring-based applications, Spring Integration enables lightweight messaging and supports integration with external systems via declarative adapters.

The Java Agent automatically discovers entry points based on the MessageHandler interface. The Java Agent automatically discovers exit points for all Spring Integration Release 2.2.0 channels except DirectChannel.

The Java Agent supports tracking application flow through Spring Integration messaging channels. For pollable channels:

- A continuing transaction is tracked if the Spring Integration framework is polling for messages.
- If the application code polls for messages in a loop, the span of each loop iteration is tracked as a transaction. Tracking begins when the loop begins and end it when the iteration ends.

Entry points

Originating transactions begin with MessageHandler.handleMessage() implementations. If the incoming message is already recognized by the Java Agent then a continuing transaction is started.

Exit points

Exit points are based on MessageChannel.send().

Most message channels exist inside the JVM. To represent this arrangement in a flow map, the Controller UI shows a link from the tier to the message channel component name (bean name) and then back to the tier.

Spring Integration Support Customizations

The following sections describe typical scenarios for tuning the default Spring Integration monitoring.

Track Application Flow Before Message Handler is Executed

In cases where a lot of application flow happens before the first MessageHandler gets executed, you should enable tracking the application flow as follows:

- Find a suitable POJO entry point and configure it.
- Set the enable-spring-integration-entry-points node property to false. This property is set to true by default.
- Restart the application server
Limit Tracking of Looping Pollable Channels

To safeguard against cases where `pollableChannel.receive()` is not called inside a loop, you can ensure that the Java Agent tracks a pollable channel loop only if it happens inside a class/method combination similar to that defined in the following example.

You can configure the `spring-integration-receive-marker-classes` node property for each class/method combination that polls messages in a loop, in which case only those class/methods identified in this node property are tracked.

```java
class MessageProcessor {
    void process() {
        while(true) {
            Message message = pollableChannel.receive()
        }
    }
}
```

For example, for the loop above, set the `spring-integration-receive-marker-classes` node property as follows and restart the application server:

```
spring-integration-receive-marker-classes=MessageProcessor/process
```

*Note:* The `spring-integration-receive-marker-classes` node property must be configured before the method `process()` gets executed for any changes to take effect. Restart the application server after setting this property.