Subscriber code example

The following is a simple example of a **Subscriber** handler for subscribing to (consuming) messages from the Integration Mesh.

```
import com.perspectium.api.Message;
import com.perspectium.logging.PerspectiumLogger;
import com.perspectium.replicator.ASubscriber;

/**
 * @author davidloo
 *
 * The EchoSubscriber demonstrates how to use the Subscriber interface to access
 * Perspectium's Cloud ESB for messages targeted for this agent.
 *
 * This subscriber is loaded into a Replicator Agent runtime by configuring it
 * in the agents config.xml
 *
 */
public class EchoSubscriber extends ASubscriber {
    final static PerspectiumLogger Log = PerspectiumLogger.create(EchoSubscriber.class);

    private String fMessageText;

    /*
     * This is the main call back API you will need to override in the subscriber
     * side of the agent.
     *
     * When the agent retrieves a message, it will decode, decrypt and marshall it
     * into a Message object before calling processMessage and passing it.
     *
     */
    @Override
    public boolean processMessage(Message message) {
        fMessageText = message.getValue();

        if (fMessageText != null)
            Log.info(String.format("%s - Received echoed message: %s", fTaskName, fMessageText));
        else
            Log.info("Message body was empty");

        return true;
    }

    /* (non-Javadoc)
     * @see com.perspectium.replicator.AgentTask#initialize(java.lang.String)
     */
    @Override
    public void initialize(String taskName) {
        super.initialize(taskName);
    }

    /* (non-Javadoc)
     * @see com.perspectium.replicator.ISubscribe#postProcessMessage()
     */
    public void postProcessMessage() {
        Log.info("in postProcessMessage of EchoSubscriber");
    }
}
```

intialize

First let's turn our attention to the initialize(String taskName) method. Your initialize method will be called with the taskName string set to the value of your <task_name> directive you defined within the conf/agent.xml configuration file. This is where you'll want to perform any configuration validation to ensure you have everything you need setup prior to your processMessage() method being called.

As you can see in the example, all we care about is making sure that the parent method gets called.

processMessage

The framework takes care of consuming the message from the Integration Mesh, performing the decryption and then calling your processMessage method with the Perspectium Messsage. As you can see, our processMessage method makes sure the message isn't empty and then logs the message text to the console. The processMessage method returns true in both cases implying that the message was processed without an error. When the method returns true the message is further processed for reporting. If the method returns false then this step is skipped.

postProcessMessage

This method is called in order for your handler to perform any final processing. For example, if you had been buffering messages in order to perform an optimized write to the local file system then that worked would be finished within this method.

You've successfully created a Subscriber handler which will pull messages from your Integration Mesh queue and provide them to your handler via the processMessage method.

For **Sharer** handler code example, see Sharer code example.