Writing Agent SDK handlers

When you leverage the Agent SDK to consume or produce messages, you're writing a Java-based handler that is plugged into the Perspectium Framework for performing various data/service integration tasks.

The various handlers leverage the Perspectium Message class to represent the message or data that's being shared via the Perspectium Integration Mesh.

Perspectium Message

The Message class represents a **Perspectium Message** that is being produced or consumed through the Integration Mesh. The class has the following properties:

1. Topic
2. Type
3. Key
4. Name
5. Value

The meaning of these properties and their values are context dependent and typically coordinated between a producer and consumer. The producer will drive the value and meaning of each of these properties. In most cases, topic, type, key and name are mandatory. The value property represents the message's payload.

Name Overloading

The Message class provides the getName() method which is used to retrieve the Name value. In some cases, the handler leverages this property to mean more than one thing. For example, when Ssharing a table from a ServiceNow instance, the value is structured such that it contains the name of the table followed by a '.' period separator and then the action associated with the table row contained within the Value keys value. For example getName() could return a String such as **incident.insert**. The row contained within the value key's value is a record from the **incident** table and the action is to **insert** the record.

Writing Handlers

A handler is the actual Java class that's defined or plugged into the framework by using the <handler> directive to perform the replication.

In the Perspectium Framework, a **Sharer** will produce messages to be shared to the Integration Mesh for consumption by another application. A **Subscriber** will consume messages from the Integration Mesh. So a **Sharer** will share messages to the Integration Mesh that a **Subscriber** will subscribe to and do action on (such as save the records into a database or another application).

Each handler will have access to its <task> section of the overall configuration. Two types of handlers that can be configured within the <agent> tag, the **Subscriber** <subscribe> and the **Sharer** <share>. Each of these directives will have zero or more nested <task> elements. Each <task> element will likely have unique directives that are required so the handler can perform its work.

Writing a Sharer

Each **Sharer** handler you write must extend the abstract class **ASharer** and implement the method **public void processMessages()**.

As a Sharer-type handler you are sharing or producing data that is being replicated. The data that you are replicating is placed into a Message and then shared to the Perspectium Framework using the publish(<Message>) method available as part of the SDK. The framework handles encryption of the message prior to being placed into the Message Bus with TripleDES.

See SDK Sharer example here.

Writing a Subscriber

Each **Subscriber** handler you write must extend the abstract class **ASubscriber** and implement the method **public boolean processMessage (Message message)** throws SubscribeException.

As a Subscriber-type handler you're subscribing to or consuming data from the Integration Mesh. The Perspectium Framework schedules your handler to run based on the type of scheduling that gets configured at the <task> level. The framework calls your processMessage(Message message) method with an available message from the Message Bus. The Message you receive is unencrypted for you by the framework.

See SDK Subscriber example here.

Optionally, the agent supports reporting errors and acknowledgements back to the sender. This feature includes an extended SDK API for customizing reporting. For more information about the reporting feature and to view a sample Subscriber, see Reporting with the Agent SDK.

Quick Start

To embed the SDK agent in your own Java code, make sure you have the correct Agent Configuration and then follow these steps:

1

Create a folder named **conf** in the base directory of your source path, and create the following **agent.xml** in that folder:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<config>
        <agent>
                <share>
                        <task>
                                <message_connection user="sdk_user" password="sdk_user"
                                        queue="psp.out.replicator.sdk_user">amqp://sdktest-
cluster-1-amqp.perspectium.net
                                </message_connection>
                                <task_name>sdk_echo_sharer</task_name>
                                <handler>EchoSharer</handler>
                                <polling_interval>30</polling_interval>
                                <encryption_key>The cow jumped over the moon</encryption_key>
                                <echo_message>This is a simple message that is being shared
or echoed.</echo_message>
                                <topic>replicator</topic>
                                <type>echo</type>
                                <key>echo</key>
                                <name>example_echo_sharer.insert</name>
                        </task>
                </share>
                <subscribe>
                        <task>
                                <message_connection user="sdk_user" password="sdk_user"
                                        queue="psp.out.replicator.sdk_user">amqp://sdktest-
cluster-1-amqp.perspectium.net
                                </message_connection>
                                <task_name>sdk_echo_subscriber</task_name>
                                <handler>EchoSubscriber</handler>
                                <max_reads_per_connect>10</max_reads_per_connect>
                                <polling_interval>20</polling_interval>
                                <decryption_key>The cow jumped over the moon</decryption_key>
                        </task>
                </subscribe>
        </agent>
</config>
```

2

Next, create an instance of the ReplicatorAgent class and call the start() method:

```
ReplicatorAgent replicatorAgent = new ReplicatorAgent();
replicatorAgent.start();
```

Here is an example of a main java class:

```
package com.perspectium;

import org.apache.commons.configuration.ConfigurationException;
import com.perspectium.replicator.ReplicatorAgent;
import com.perspectium.logging.PerspectiumLogger;

public class Main {
    public static void main(String[] args) {
        PerspectiumLogger Log = PerspectiumLogger.create(Main.class);
        ReplicatorAgent replicatorAgent = new ReplicatorAgent();
        try {
                replicatorAgent.start();
        } catch (ConfigurationException e) {
                Log.error(e.toString());
        }
    }
}
```

To stop the the SDK agent call the stop method:

```
replicatorAgent.stop();
```