Tutorial: Howto call a ROS Service in the Happy Artist RMDMIA:

In this tutorial we teach Java programmers how to call a non-persistent **ROS** Service (A single request/response).

System Requirements:

- ROS
- ROS Turtlesim (Configuration instructions)
- Java 1.6 or greater
- RMDMIA vpr1_v7 slipstream or greater

Prerequisites:

- RMDMIA Client Configuration for ROS Turtlesim Tutorial

Overview:

This tutorial explains how to call a single request/response to a ROS *Service with the Happy Artist RMDMIA. The* ROS *Turtlesim is used in this example to demonstrate a service call to the turtlesim "/clear" service. A call to the "/clear" service clears the painted turtle trail.*

The following code demonstrates how to perform a non-persistent request/response service call:

REQUEST:

<code>

```
// Define the clear message. Clear is an empty message, therefore, this is a 4 byte length of 0 byte[].
final static byte[] CLEAR = {(byte)(0x00),(byte)(0x00),(byte)(0x00),(byte)(0x00)};
// Get the ROSNode from RCSMManager.
ROSNode rosNode = (ROSNode)getController().getRCSM().getProviderByName("ros");
// Call the service (wrapped in a try catch statement for an RCSMException),
sendServiceMessage is synchronized.
try
{
        rosNode.sendServiceMessage(CLEAR, rosNode.getService("/clear"));
  }
  catch (RCSMException ex){e.printStackTrace();}
```

</code>

RESPONSE:

The service response for a non-persistent service call is handled differently than a persistent service call response. A persistent service call is handled identically to a subscriber using the MessageHandlerInterface. Due to the need to disconnect a non-persistent service call after the response is received the MessageHandlerInterface process methods are used to perform this logic. Therefore, the registerDataHandler(DataHandlerInterface dataHandler) method was introduced for the purpose of handling response data. The reason registerDataHandler is not used instead of process in the MessageHandlerInterface for a single API method to handle messages is registerDataHandler requires more system resources, and introduces latency is response processing. Response latency is not a good trait for a robotic control system. The faster the system responds the better the robot performs. The sendServiceMessage is synchronized on the ROS thread, because most non-persistent service calls, are system calls like “/shutdown”, or “/clear” in turtlesim.

<code>

```
package my.example;
import
org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.AbstractServiceMessageHandler;

public ConsoleWriterServiceMessageHandler extends AbstractServiceMessageHandler
{
    public ConsoleWriterServiceMessageHandler()
    {
        // Always declare super() in the constructor.
        super();
        //  register data handler
        registerDataHandler(new ConsoleWriterDataHandler());
    }
    // In this example we will define an inner class for clarity of the DataHandlerInterface
    class ConsoleWriterDataHandler implements DataHandlerInterface
    {
        // Define the ROS Message Decoder to read byte[] to String
        ROSMessageDecoder decode = new ROSMessageDecoder();

        // Implement the process methods for incoming data
        public void process(byte[] message, int dataLength)
         {
            System.out.println("Hex Message (single-block): " +
org.happy.artist.rmdmia.utilities.BytesToHex.bytesToHex(message));
            System.out.println("Hex to Text Message (single-block): "  +
decode.convertHexToString(org.happy.artist.rmdmia.utilities.BytesToHex.bytesToHex(messag
e).toCharArray()));
        }

        // Implement the process methods for incoming data
        public void process(byte[] message)
        {
            System.out.println("Hex Message (multi-block): " +
org.happy.artist.rmdmia.utilities.BytesToHex.bytesToHex(message));
```

```
                        System.out.println("Hex to Text Message (multi-block): "  +
decode.convertHexToString(org.happy.artist.rmdmia.utilities.BytesToHex.bytesToHex(messag
e).toCharArray()));
                }
        }
}
```

</code>

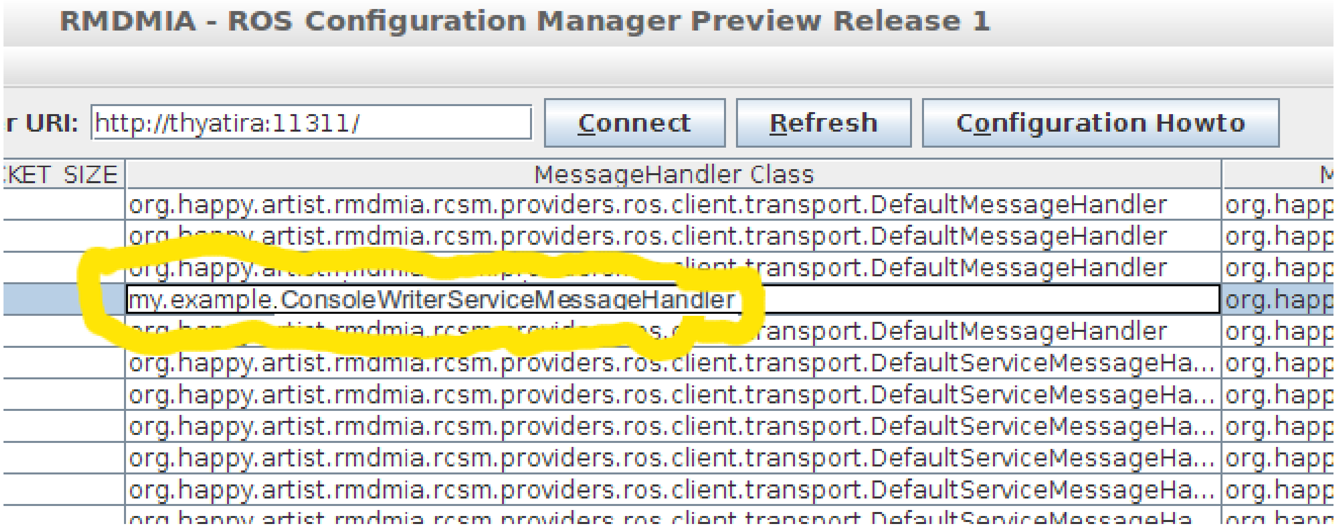
ConsoleWriterServiceMessageHandler

To test the above class:

1. compile, and add the generated ConsoleWriterServiceMessageHandler package structure with a class file to a jar file.

2. Copy the Jar to the RMDMIA plugins/rcsm directory.

3. Update the MessageHandler class for the associated topic in the ROS Configuration Manager, and save/exit.



RMDMIA - ROS Configuration Manager Preview Release 1

r URI: http://thyatira:11311/

KET SIZE	MessageHandler Class	M
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultMessageHandler	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultMessageHandler	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultMessageHandler	org.happ
	my.example.ConsoleWriterServiceMessageHandler	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultMessageHandler	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultServiceMessageHa...	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultServiceMessageHa...	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultServiceMessageHa...	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultServiceMessageHa...	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultServiceMessageHa...	org.happ
	org.happy.artist.rmdmia.rcsm.providers.ros.client.transport.DefaultServiceMessageHa	org.happ

4. Launch the RMDMIA, and the new message handler will load.