Tutorial: Howto implement a ROCI (Robot Operator Control Interface) Provider in the Happy Artist RMDMIA:

In this tutorial we teach Java programmers how to implement a ROCI (Robot Operator Control Interface) Provider.

System Requirements:

- *Java 1.6 or greater*
- RMDMIA vpr1_v7 slipstream or greater

Tutorial Files:

- ros_control_sample_v1.0-src.zip
- ros_control_sample_v1.0-bin.zip

Prerequisites:

- Compilation of ROSTurtlesimController.java requires the following Jar files in the classpath from RMDMIA vpr1_v7 slipstream or greater,
    - plugins/rcsm/ros_v1.0.jar
    - plugins/rcsm/lib/RMDMIA_Controller_Manager.jar

Overview:

Effective non-expert supervised autonomy requires a messaging bridge between the Robot and the non-expert supervised user. A connection to the robot must be supported via an external robot control device (examples - phone, computer, human interaction). The Robot Controller device interfaces directly with the Robot Operator Control Interface Manager (ROCI Manager).

Robot Operator Control Interface Manager (ROCI Manager):

- *Supports unspecified number of concurrently running ROCI Provider implementations.*
- *(Robot/Operator) Robot Intelligent Agent can send messages to the Operator Supervisor via a ROCI Interface provider implementation.*

- *(Robot/Operator) Operator Supervisor can send messages to the RMDMIA via a ROCI Interface provider implementation.*

Requirements:

- Implement abstract class org.happy.artist.rmdmia.roci.provider.ROCIProvider class.
- Package the ROCIProvider implementation into a JAR file with the ROCI Provider Plugin Jar structure:
    - Required – Minimum empty text file. Provider plugin properties file
    - /conf/roci/<provider_name>/<provider_name>_v<version_double>.properties
    - Optional (strongly advised) - /conf/roci/plugin.properties – contains default plugin properties configuration for roci.properties
    - Required – Update /conf/roci.properties with the new plugin by numerically iterating the next number for the plugin id with the following 4 properties (where 1 is an example number if only 0. exists). Only set enabled to true if you want the plugin to load, otherwise specify false.:
        - 1.enabled=true
        - 1.version=1.0
        - 1.class=org.happy.artist.rmdmia.roci.plugins.ROSTurtlesimController
        - 1.name=ros_control_sample

The following is the API documentation for ROCIProvider:

---

org.happy.artist.rmdmia.roci

Class ROCIProvider

```
java.lang.Object
  └org.happy.artist.rmdmia.roci.ROCIProvider
```

All Implemented Interfaces:
ROCInterface

```
public abstract class ROCIProviderextends java.lang.Objectimplements ROCInterface
```

Abstract class for ROCI plugin implementors. The ROCIProvider class initializes the ROCI Provider Plugin into the RMDMIA system via an automatic plugin identifier, and registration of Plugin Properties into the file system. A ROCI Provider jar can provide a default Properties file into the RMDMIA ROCI Provider Jar distribution, that will automatically copy and load the Properties file to

the RMDMIA implementation file system for non-default properties modification. RMDMIA systems that do not provide access to the file system will use the default ROCI Provider Properties as defined in the ROCI JAR distribution.

Field Summary

static int	BYTES_DATA_TRANSFER_METHOD
static int	NO_TRANSFER_METHOD
static int	STREAMING_DATA_TRANSFER_METHOD

Constructor Summary

ROCIProvider()	

Method Summary

Controller	getController() Return the Controller.
int	getDataTransferMethod() Return the data transfer method the ROCIProvider implements.
int	getID() Return the ROCI ID.
java.util.Properties	getProperties() Return the ROCI Provider Properties Object.
java.lang.String	getPropertiesFilePath() Return the ROCI Plugin's associated Properties File Path.
void	onInitialized() onInitialized is called by ROCIManager following all Provider initialization.
void	setController(Controller controller) The Controller ID is set automatically by the ControllerManager.
void	setDataTransferMethod(int type) Set the data transfer method the ROCIProvider implements.
void	setID(int roci_id) set the ROCI ID.
void	setProperties(java.util.Properties properties) The setProperties class is used to load the default Properties at plugin install time.

void	**setPropertiesFilePath**(java.lang.String filePath) Set the ROCI Provider's Properties File Path.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface org.happy.artist.rmdmia.roci.ROCInterface
getInputStream, getName, getOutputStream, getVersion, initialize, isInitialized, processIncoming, processOutgoing, recycle, setInputStream, setOutputStream, shutdown

Field Detail

STREAMING_DATA_TRANSFER_METHOD

public static final int **STREAMING_DATA_TRANSFER_METHOD**

See Also:
Constant Field Values

BYTES_DATA_TRANSFER_METHOD

public static final int **BYTES_DATA_TRANSFER_METHOD**

See Also:
Constant Field Values

NO_TRANSFER_METHOD

public static final int **NO_TRANSFER_METHOD**

See Also:
Constant Field Values

Constructor Detail

ROCIProvider

public **ROCIProvider**()

Method Detail

getID

```
public int getID()
```

Return the ROCI ID.

Specified by:
getID in interface ROCInterface
Returns:
int ROCI ID.

---

setID

```
public void setID(int roci_id)
```

set the ROCI ID.

Specified by:
setID in interface ROCInterface
Parameters:
roci_id - int

---

setController

```
public void setController(Controller controller)
```

The Controller ID is set automatically by the ControllerManager. Controller implementers of the ROCIProvider do not need to know anything about this method. Automatically set by the ROCIManager.

Specified by:
setController in interface ROCInterface
Parameters:
controller - the Controller of this instance of this RCSM.

---

getController

```
public Controller getController()
```

Return the Controller. Used by the Provider to obtain a reference to the Controller.

Specified by:
`getController` in interface `ROCInterface`

Returns:
Controller reference.

---

setProperties

```
public void setProperties(java.util.Properties properties)
```

The setProperties class is used to load the default Properties at plugin install time. At load time the Properties will be transferred to either a properties file in the configuration directory, and/or appended to a global properties document with the ROCI_ID appended to the front of each associated property key with a "." separator in the name to identify all properties associated with that ROCI plugin. ROCI Plugin constructors are empty, and therefore the properties is an option to add Properties at startup.

Specified by:
`setProperties` in interface `ROCInterface`

Parameters:
`properties` - java.util.Properties

---

getProperties

```
public java.util.Properties getProperties()
```

Return the ROCI Provider Properties Object. Managed by the ROCIManager.

Specified by:
`getProperties` in interface `ROCInterface`

Returns:
java.util.Properties

---

setPropertiesFilePath

```
public void setPropertiesFilePath(java.lang.String filePath)
```

Set the ROCI Provider's Properties File Path.

Specified by:
`setPropertiesFilePath` in interface `ROCInterface`

Parameters:
`filePath` - String The Properties File Path.

---

getPropertiesFilePath

```
public java.lang.String getPropertiesFilePath()
```

Return the ROCI Plugin's associated Properties File Path.

Specified by:
getPropertiesFilePath in interface ROCInterface
Returns:
String The Properties File Path.

---

getDataTransferMethod

```
public int getDataTransferMethod()
```

Return the data transfer method the ROCIProvider implements. 0=streaming (setInputStream/setOutputStream methods), 1=bytes (processIncoming/processOutgoing methods). Default data transfer method is streaming.

Specified by:
getDataTransferMethod in interface ROCInterface
Returns:
int 0-streaming, 1=bytes

---

setDataTransferMethod

```
public void setDataTransferMethod(int type)
```

Set the data transfer method the ROCIProvider implements. 0=streaming (setInputStream/setOutputStream methods), 1=bytes (processIncoming/processOutgoing methods). Default data transfer method is streaming.

Specified by:
setDataTransferMethod in interface ROCInterface
Parameters:
`type` - int 0=streaming, 1=bytes

---

onInitialized

```
public void onInitialized()
```

onInitialized is called by ROCIManager following all Provider initialization. Implementing as a do nothing method that must be overridden to use. .

Specified by:
`onInitialized` in interface `ROCInterface`

Submit a bug or feature
For further API reference and developer documentation, see RMDMIA Documentation. That documentation contains tutorials, developer focused descriptions, concept details, definitions, acronyms, workarounds, and code examples.

The following code demonstrates a ROCI Provider implementation. This code implements teleoperation of the turtesim via a PS/3 Joystick operating through JInput:

<code>

```
package org.happy.artist.rmdmia.roci.plugins;


import java.io.*;
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;
import net.java.games.input.Component;
import net.java.games.input.Controller;
import net.java.games.input.ControllerEnvironment;
import net.java.games.input.Event;
import net.java.games.input.EventQueue;
import org.happy.artist.rmdmia.rcsm.RCSMException;
import org.happy.artist.rmdmia.rcsm.provider.CommunicationSenderInterface;
import org.happy.artist.rmdmia.rcsm.providers.ros.ROSNode;
import org.happy.artist.rmdmia.roci.ROCIProvider;
```

```
/**
 * The ROS Turtlesim Joystick Sample program implemented
 * as a ROCI Provider Plugin.
 *
 * @author Happy Artist
 *
 * @copyright Copyright ©2014 Happy Artist. All rights reserved.
 *
 */
public class ROSTurtlesimController extends ROCIProvider
{
    // ROCI Provider variables
    private final static String name="ros_control_sample";
    private final static double version = 1.0;
    private Properties properties;
    // Reference to rosNode
    private ROSNode rosNode;
    // initialize method variables
    private boolean isInitialized=false;
    // message= 4 bytes message_length + 20 bytes 5 float array.
    private byte[] message = new byte[52];
    // Thread to run the controller loop.
    private Thread thread;
    private CommunicationSenderInterface sender;
    // JInput variables
    private Component component;
    private EventQueue queue;
```

```
private Event event;
private Controller[] controllers;
private Component[] components;
private String cname;
// turtlesim movements.
private static byte[] GAMEPAD_UP=Movements.getUpMovement();
private static byte[] GAMEPAD_DOWN=Movements.getDownMovement();
private static byte[] GAMEPAD_LEFT=Movements.getLeftMovement();
private static byte[] GAMEPAD_RIGHT=Movements.getRightMovement();

// Define and instantiate Logger
private Logger logger = Logger.getLogger(ROSTurtlesimController.class.getName());
// set instance in constructor for message callback to reference.
private static ROSTurtlesimController instance=null;
public ROSTurtlesimController()
{
   this.instance=this;
}

/** return reference to constructed ROSTurtlesimController. */
public static ROSTurtlesimController getInstance()
{
   return instance;
}

/** Initialize the ROCI TelnetServer Plugin. */
public void initialize()
{
```

```
        logger.log(Level.INFO, "Initializing ROS Turtlesim Controller plugin.");
        // Set the ROCI Provider Data Transfer method to byte data transfer
        // method, even though no actual transfer method is required due to
        // obtaining a direct reference to ROSNode.
        setDataTransferMethod(this.NO_TRANSFER_METHOD);
        // Setup the ROCI Plugin properties.
        this.properties=getProperties();

        // Get the ROSNode from RCSMManager.
        this.rosNode = (ROSNode)getController().getRCSM().getProviderByName("ros");

        if(rosNode!=null&&rosNode.isInitialized())
        {
            logger.log(Level.INFO, "Testing ROS TurtleSim..");
            try
            {
                // get the topic/service lookup Map
                //this.topicServiceLookupMap = rosNode.getSenderLookupMap();
                this.sender=rosNode.getPublisherSenders()
[rosNode.getTopicIndex("/turtle1/cmd_vel")];
//                System.out.println("Turtle Sender Topic threadName: " +
((TCPROSPublisherCommunicator)sender).threadName);
                // hook up the PS/3 controller.
                this.thread = new Thread(new Runnable() {
                public void run()
                {
                    try
                    {
```

```
                if(ROSTurtlesimController.this.sender!=null)
                {
    //                System.out.println("calling startController on /turtle1/cmd_vel");
                   startController(ROSTurtlesimController.this.sender);
                }
                else
                {
                Logger.getLogger(ROSTurtlesimController.class.getName()).log(Level.INFO,
"Turtlesim Controller could not send message due to the Publisher not being initialized.");

                }
              } catch (IOException ex)
              {

Logger.getLogger(ROSTurtlesimController.class.getName()).log(Level.SEVERE, null, ex);
              }
            }
      });
            thread.start();

        }
        catch (Exception ex)
        {
            Logger.getLogger(ROSTurtlesimController.class.getName()).log(Level.SEVERE,
null, ex);
        }
```

```
            // Set isInitialized to false.
            this.isInitialized=true;
 //           this.monitor = new ConnectionMonitor(tableData, tableColumnNames,
rosNode.getCallerID(), rosNode.getSubscriberMessageManager(), rosNode,
rosNode.getProperties(), rosNode.getPropertiesFilePath());
        }
        else if(!rosNode.isInitialized())
        {
            // ROS Plugin disabled
             logger.log(Level.WARNING, "The \"ros\" RCSM Provider is not enabled. To start the
ROS TurtleSim controller, the \"ros\" RCSM Provider must be enabled.");
            // Set isInitialized to false.
            this.isInitialized=false;
        }
        else
        {
            // Could not find ROS Log this.
            logger.log(Level.WARNING, "A \"ros\" RCSM Provider was not found. TurtleSim
controller not started.");
            // Set isInitialized to false.
            this.isInitialized=false;
        }

        logger.log(Level.INFO, "ROCI ROS TurtleSim controller initialized");
    }

    /** Return the plugin version number. A version number of 0 is default of no version
number. */
```

```
public double getVersion()
{
    return version;
}

/** Return the ROCI Plugin name. */
public String getName()
{
    return name;
}

@Override
public void setOutputStream(OutputStream os) {
    throw new UnsupportedOperationException("Not supported yet.");
    }

@Override
public void setInputStream(InputStream is) {
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public InputStream getInputStream() {
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public OutputStream getOutputStream()
```

```
{
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public void processIncoming(byte[] data)
{
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public void processOutgoing(byte[] data)
{
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public boolean recycle()
{
    throw new UnsupportedOperationException("Not supported yet.");
}

@Override
public void shutdown()
{
    this.thread=null;
    // Set isInitialized to false.
    this.isInitialized=false;
```

```
    throw new UnsupportedOperationException("Not supported yet.");
}

/** Return boolean is initialized on ROCI Provider.
 * @return boolean returns false if ROCI Provider is not initialized.
 */
public boolean isInitialized()
{
    return isInitialized;
}

// Clear is an empty message, therefore, this is a 4 byte length of 0 byte[].
private final static byte[] CLEAR = {(byte)(0x00),(byte)(0x00),(byte)(0x00),(byte)(0x00)};
public void startController(CommunicationSenderInterface sender) throws IOException
{
    while(true)
    {
        this.controllers = ControllerEnvironment.getDefaultEnvironment().getControllers();
        if(controllers.length==0)
        {
            System.out.println("Found no controllers.");
            return;
        }

        for(int i=0;i<controllers.length;i++)
        {
            controllers[i].poll();
            this.queue = controllers[i].getEventQueue();
```

```
this.event = new Event();
while(queue.getNextEvent(event))
{
    this.component=event.getComponent();
    if((this.cname = component.getName()).equals("Base"))
    {
        sender.send(GAMEPAD_DOWN);
    }
    else if(cname.equals("Base 2"))
    {
        sender.send(GAMEPAD_LEFT);
    }
    else if(cname.equals("Top 2"))
    {
         sender.send(GAMEPAD_UP);                                                    }
    else if(cname.equals("Pinkie"))
    {
         sender.send(GAMEPAD_RIGHT);
    } // PS/3 button on down value is 1.0F, on release value is 0.0. Two events are
triggered on PS3 button press.
    else if(cname.equals("Extra 17")&&event.getValue()==1.0f)
    {
      try
      {
          rosNode.sendServiceMessage(CLEAR, rosNode.getService("/clear"));
      }
      catch (RCSMException ex)
      {
```

```
Logger.getLogger(ROSTurtlesimController.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            }
        }

        try
        {
            // Pause 20ms before polling the Joystick for new events.
            Thread.sleep(20);
        }
        catch (InterruptedException e)
        {
           e.printStackTrace();
        }
      }

   }
}

   class Movements
   {
      public static byte[] getUpMovement()
      {
         byte[] gamepad_up;
         gamepad_up = new byte[52];
         gamepad_up[0] = (byte)(0x30);
```

```
gamepad_up[1] = (byte)(0x00);
gamepad_up[2] = (byte)(0x00);
gamepad_up[3] = (byte)(0x00);
// Append linear x movement event to byte[].
gamepad_up[4] = (byte)(0x00);
gamepad_up[5] = (byte)(0x00);
gamepad_up[6] = (byte)(0x00);
gamepad_up[7] = (byte)(0x00);
gamepad_up[8] = (byte)(0x00);
gamepad_up[9] = (byte)(0x00);
gamepad_up[10] = (byte)(0x00);
gamepad_up[11] = (byte)(0x40);
// Append linear y movement event to byte[].
gamepad_up[12] = (byte)(0x00);
gamepad_up[13] = (byte)(0x00);
gamepad_up[14] = (byte)(0x00);
gamepad_up[15] = (byte)(0x00);
gamepad_up[16] = (byte)(0x00);
gamepad_up[17] = (byte)(0x00);
gamepad_up[18] = (byte)(0x00);
gamepad_up[19] = (byte)(0x00);
// Append linear theta z to byte[]
gamepad_up[20] = (byte)(0x00);
gamepad_up[21] = (byte)(0x00);
gamepad_up[22] = (byte)(0x00);
gamepad_up[23] = (byte)(0x00);
gamepad_up[24] = (byte)(0x00);
gamepad_up[25] = (byte)(0x00);
```

```
gamepad_up[26] = (byte)(0x00);
gamepad_up[27] = (byte)(0x00);

// Append angular x movement event to byte[].
gamepad_up[28] = (byte)(0x00);
gamepad_up[29] = (byte)(0x00);
gamepad_up[30] = (byte)(0x00);
gamepad_up[31] = (byte)(0x00);
gamepad_up[32] = (byte)(0x00);
gamepad_up[33] = (byte)(0x00);
gamepad_up[34] = (byte)(0x00);
gamepad_up[35] = (byte)(0x00);
// Append angular y movement event to byte[].
gamepad_up[36] = (byte)(0x00);
gamepad_up[37] = (byte)(0x00);
gamepad_up[38] = (byte)(0x00);
gamepad_up[39] = (byte)(0x00);
gamepad_up[40] = (byte)(0x00);
gamepad_up[41] = (byte)(0x00);
gamepad_up[42] = (byte)(0x00);
gamepad_up[43] = (byte)(0x00);
// Append angular theta z to byte[]
gamepad_up[44] = (byte)(0x00);
gamepad_up[45] = (byte)(0x00);
gamepad_up[46] = (byte)(0x00);
gamepad_up[47] = (byte)(0x00);
gamepad_up[48] = (byte)(0x00);
gamepad_up[49] = (byte)(0x00);
```

```
    gamepad_up[50] = (byte)(0x00);
    gamepad_up[51] = (byte)(0x00);
    return gamepad_up;
}

public static byte[] getDownMovement()
{
    byte[] gamepad_down=new byte[52];
    gamepad_down[0] = (byte)(0x30);
    gamepad_down[1] = (byte)(0x00);
    gamepad_down[2] = (byte)(0x00);
    gamepad_down[3] = (byte)(0x00);
    // Append linear x movement event to byte[].
    gamepad_down[4] = (byte)(0x00);
    gamepad_down[5] = (byte)(0x00);
    gamepad_down[6] = (byte)(0x00);
    gamepad_down[7] = (byte)(0x00);
    gamepad_down[8] = (byte)(0x00);
    gamepad_down[9] = (byte)(0x00);
    gamepad_down[10] = (byte)(0x00);
    gamepad_down[11] = (byte)(0xc0);
    // Append linear y movement event to byte[].
    gamepad_down[12] = (byte)(0x00);
    gamepad_down[13] = (byte)(0x00);
    gamepad_down[14] = (byte)(0x00);
    gamepad_down[15] = (byte)(0x00);
    gamepad_down[16] = (byte)(0x00);
    gamepad_down[17] = (byte)(0x00);
```

```
gamepad_down[18] = (byte)(0x00);
gamepad_down[19] = (byte)(0x00);
// Append linear theta z to byte[]
gamepad_down[20] = (byte)(0x00);
gamepad_down[21] = (byte)(0x00);
gamepad_down[22] = (byte)(0x00);
gamepad_down[23] = (byte)(0x00);
gamepad_down[24] = (byte)(0x00);
gamepad_down[25] = (byte)(0x00);
gamepad_down[26] = (byte)(0x00);
gamepad_down[27] = (byte)(0x00);

// Append angular x movement event to byte[].
gamepad_down[28] = (byte)(0x00);
gamepad_down[29] = (byte)(0x00);
gamepad_down[30] = (byte)(0x00);
gamepad_down[31] = (byte)(0x00);
gamepad_down[32] = (byte)(0x00);
gamepad_down[33] = (byte)(0x00);
gamepad_down[34] = (byte)(0x00);
gamepad_down[35] = (byte)(0x00);
// Append angular y movement event to byte[].
gamepad_down[36] = (byte)(0x00);
gamepad_down[37] = (byte)(0x00);
gamepad_down[38] = (byte)(0x00);
gamepad_down[39] = (byte)(0x00);
gamepad_down[40] = (byte)(0x00);
gamepad_down[41] = (byte)(0x00);
```

```
    gamepad_down[42] = (byte)(0x00);
    gamepad_down[43] = (byte)(0x00);
    // Append angular theta z to byte[]
    gamepad_down[44] = (byte)(0x00);
    gamepad_down[45] = (byte)(0x00);
    gamepad_down[46] = (byte)(0x00);
    gamepad_down[47] = (byte)(0x00);
    gamepad_down[48] = (byte)(0x00);
    gamepad_down[49] = (byte)(0x00);
    gamepad_down[50] = (byte)(0x00);
    gamepad_down[51] = (byte)(0x00);
    return gamepad_down;
}

public static byte[] getLeftMovement()
{
    byte[] gamepad_left=new byte[52];
    gamepad_left[0] = (byte)(0x30);
    gamepad_left[1] = (byte)(0x00);
    gamepad_left[2] = (byte)(0x00);
    gamepad_left[3] = (byte)(0x00);
    // Append linear x movement event to byte[].
    gamepad_left[4] = (byte)(0x00);
    gamepad_left[5] = (byte)(0x00);
    gamepad_left[6] = (byte)(0x00);
    gamepad_left[7] = (byte)(0x00);
    gamepad_left[8] = (byte)(0x00);
    gamepad_left[9] = (byte)(0x00);
```

```
gamepad_left[10] = (byte)(0x00);
gamepad_left[11] = (byte)(0x00);
// Append linear y movement event to byte[].
gamepad_left[12] = (byte)(0x00);
gamepad_left[13] = (byte)(0x00);
gamepad_left[14] = (byte)(0x00);
gamepad_left[15] = (byte)(0x00);
gamepad_left[16] = (byte)(0x00);
gamepad_left[17] = (byte)(0x00);
gamepad_left[18] = (byte)(0x00);
gamepad_left[19] = (byte)(0x00);
// Append linear theta z to byte[]
gamepad_left[20] = (byte)(0x00);
gamepad_left[21] = (byte)(0x00);
gamepad_left[22] = (byte)(0x00);
gamepad_left[23] = (byte)(0x00);
gamepad_left[24] = (byte)(0x00);
gamepad_left[25] = (byte)(0x00);
gamepad_left[26] = (byte)(0x00);
gamepad_left[27] = (byte)(0x00);

// Append angular x movement event to byte[].
gamepad_left[28] = (byte)(0x00);
gamepad_left[29] = (byte)(0x00);
gamepad_left[30] = (byte)(0x00);
gamepad_left[31] = (byte)(0x00);
gamepad_left[32] = (byte)(0x00);
gamepad_left[33] = (byte)(0x00);
```

```
    gamepad_left[34] = (byte)(0x00);
    gamepad_left[35] = (byte)(0x00);
    // Append angular y movement event to byte[].
    gamepad_left[36] = (byte)(0x00);
    gamepad_left[37] = (byte)(0x00);
    gamepad_left[38] = (byte)(0x00);
    gamepad_left[39] = (byte)(0x00);
    gamepad_left[40] = (byte)(0x00);
    gamepad_left[41] = (byte)(0x00);
    gamepad_left[42] = (byte)(0x00);
    gamepad_left[43] = (byte)(0x00);
    // Append angular theta z to byte[]
    gamepad_left[44] = (byte)(0x00);
    gamepad_left[45] = (byte)(0x00);
    gamepad_left[46] = (byte)(0x00);
    gamepad_left[47] = (byte)(0x00);
    gamepad_left[48] = (byte)(0x00);
    gamepad_left[49] = (byte)(0x00);
    gamepad_left[50] = (byte)(0x00);
    gamepad_left[51] = (byte)(0x40);
    return gamepad_left;
}
public static byte[] getRightMovement()
{
    byte[] gamepad_right=new byte[52];
    gamepad_right[0] = (byte)(0x30);
    gamepad_right[1] = (byte)(0x00);
    gamepad_right[2] = (byte)(0x00);
```

```
gamepad_right[3] = (byte)(0x00);
// Append linear x movement event to byte[].
gamepad_right[4] = (byte)(0x00);
gamepad_right[5] = (byte)(0x00);
gamepad_right[6] = (byte)(0x00);
gamepad_right[7] = (byte)(0x00);
gamepad_right[8] = (byte)(0x00);
gamepad_right[9] = (byte)(0x00);
gamepad_right[10] = (byte)(0x00);
gamepad_right[11] = (byte)(0x00);
// Append linear y movement event to byte[].
gamepad_right[12] = (byte)(0x00);
gamepad_right[13] = (byte)(0x00);
gamepad_right[14] = (byte)(0x00);
gamepad_right[15] = (byte)(0x00);
gamepad_right[16] = (byte)(0x00);
gamepad_right[17] = (byte)(0x00);
gamepad_right[18] = (byte)(0x00);
gamepad_right[19] = (byte)(0x00);
// Append linear theta z to byte[]
gamepad_right[20] = (byte)(0x00);
gamepad_right[21] = (byte)(0x00);
gamepad_right[22] = (byte)(0x00);
gamepad_right[23] = (byte)(0x00);
gamepad_right[24] = (byte)(0x00);
gamepad_right[25] = (byte)(0x00);
gamepad_right[26] = (byte)(0x00);
gamepad_right[27] = (byte)(0x00);
```

```
// Append angular x movement event to byte[].
 gamepad_right[28] = (byte)(0x00);
 gamepad_right[29] = (byte)(0x00);
 gamepad_right[30] = (byte)(0x00);
 gamepad_right[31] = (byte)(0x00);
 gamepad_right[32] = (byte)(0x00);
 gamepad_right[33] = (byte)(0x00);
 gamepad_right[34] = (byte)(0x00);
 gamepad_right[35] = (byte)(0x00);
 // Append angular y movement event to byte[].
 gamepad_right[36] = (byte)(0x00);
 gamepad_right[37] = (byte)(0x00);
 gamepad_right[38] = (byte)(0x00);
 gamepad_right[39] = (byte)(0x00);
 gamepad_right[40] = (byte)(0x00);
 gamepad_right[41] = (byte)(0x00);
 gamepad_right[42] = (byte)(0x00);
 gamepad_right[43] = (byte)(0x00);
 // Append angular theta z to byte[]
 gamepad_right[44] = (byte)(0x00);
 gamepad_right[45] = (byte)(0x00);
 gamepad_right[46] = (byte)(0x00);
 gamepad_right[47] = (byte)(0x00);
 gamepad_right[48] = (byte)(0x00);
 gamepad_right[49] = (byte)(0x00);
 gamepad_right[50] = (byte)(0x00);
 gamepad_right[51] = (byte)(0xc0);
```

return gamepad_right;

}

}

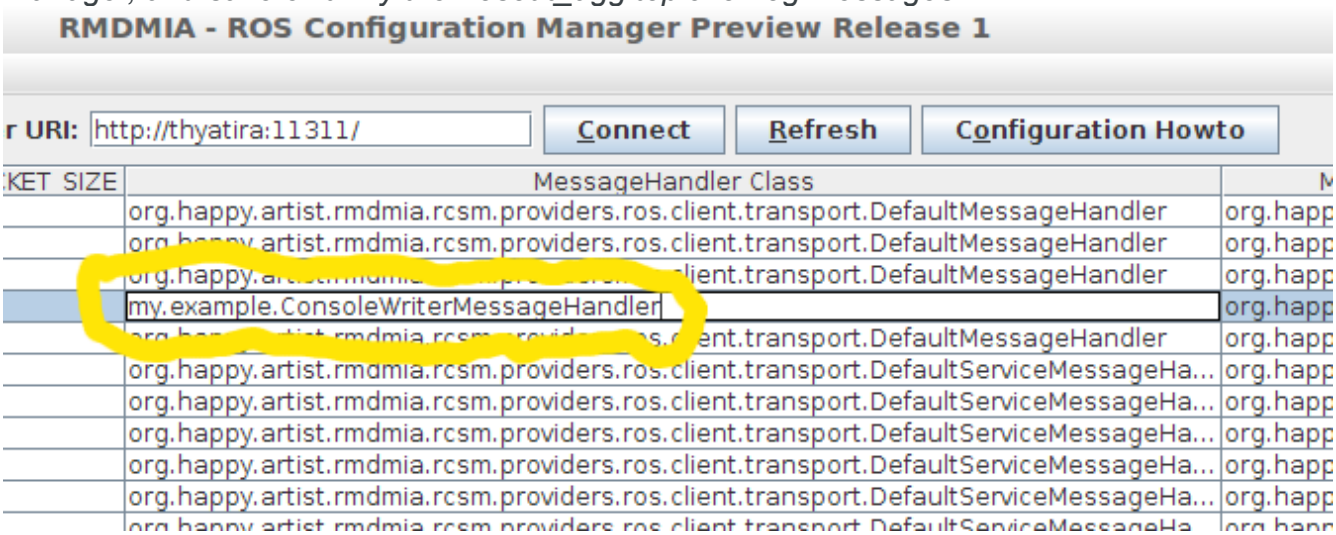
</code>

To test the above class:

1. *compile, and add the generated ConsoleWriterMessageHandler 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 service in the ROS Configuration Manager, and save/exit. Try the /rosout_agg topic for log messages.*



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