PlugIn Manual

Contents

[1. Installation and deployment of PlugIn development environment](#h.6m57nfaq6tx9)

[2. Introduction of PlugIn](#h.g3tpiyg95rqj)

[2.1 Introduction of PlugIn configuration file](#h.y4dvbptgg1iq)

[3. PlugIn Component Development](#h.ikb9mkglkszf)

[3.1 Illustration for accessible AUTOSAR SWC Ports](#h.f15vt5ris29f)

[3.2 Introduce PlugIn template](#h.2o1ggmd78csw)

[3.3 Develop a new PlugIn](#h.iphnypp9pw6w)

[3.4 Introduce PlugIn configuration template](#h.tsby8melepy6)

[4. Upload PlugIn to the trusted server](#h.yxgs529jpvru)

[4.1 Preparation before upload PlugIn](#h.w14d02hymixg)

[4.2 Upload PlugIn .zip files and .xml file](#h.4syfphd19lv)

[5. PlugIn deployment on the Simulator](#h.x7t2xrp5oag3)

[6. PlugIn deployment on the MOPED](#h.5mkl3ove4t5z)

[Appendix](#h.vnkm10iaqpud)

# 1. Installation and deployment of PlugIn development environment

Assuming that Eclipse IDE are already installed properly, it starts with a new Java project and then imports cldc-1.1-jar5.0.jar ([CLDC lib](https://moped.sics.se/wordpress/wp-content/uploads/2014/12/cldc-1.1-java5.0.jar)) and api-0.0.1-SNAPSHOT.jar ([PlugIn Development lib](https://moped.sics.se/wordpress/wp-content/uploads/2014/12/api-0.0.1-SNAPSHOT.jar)) to the project (right-click the project, then select "Properties", then "Java Build Path", then "Add external JARs"). Finally, remove the default JDK from the project (right-click the project, then select “Properties”, then “Java Build Path”, then “Remove” JRE System Library).

Note that due to Squawk being constrained to its compiler CLDC by default, it is only imported CLDC 1.1 library as execution environment when develop new PlugIns.

# 2. Introduction of PlugIn

Currently, PlugIn can be performed either in MOPED platform or in Simulator environment and therefore, it has two corresponding entry methods - public static void main(String[] arg) method for MOPED platform, public void run() for Simulator. The core functions of the PlugIn follow up placing in one of entry methods. In run-time context, PlugIn is carried out as Isolate in the former environment and as a thread in the latter environment.

Instantiation of PlugIn is subjected to restriction on mandatory adding one parameter

namely String[] arg, which is used for assigning PlugIn port initialization.

PlugIn can be assembled with many PlugIn ports defined as attributes in Java code. Two types of PlugIn port can be instanced: PluginPPort (read data from the port), PluginRPort (write data to the port). Note that these PlugIn ports always need to be initialized before invoking read or write methods, and initialization codes must be placed in the public void init() method.

## 2.1 Introduction of PlugIn configuration file

In general, one application is consisted of a few PlugIns and one PlugIn configuration file (file postfix as .xml). The PlugIn configuration file is written in XML format and is mainly used for defining PlugIn name, location specifying the target ECU, description about PlugIn port with name, and links about how to wire among ports. For instance, PluginPPort, PluginRPort, VirtualPPort, VirtualRPort. More specification, PluginPPort and PluginRPort are assigned to name of PlugIn Port before-mentioned in the description of PlugIn port, while VirtualPPort and VIrtualRPort are assigned to ID of port. ID of virtual ports are referred in terms of system configuration file. Within the “link” tag, “from” tag represents PluginPPort or VirtualRPort and “to” tag represents PluginRPort or VirtualPPort.

# 3. PlugIn Component Development

## 3.1 Illustration for accessible AUTOSAR SWC Ports

Those ports can be used to read data from sensors or write data to actuators or even communication on between ECUs.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Port ID | Port Name | ECU | Type | Unit | Range | Simulator compatible |
| 0 | write data to external MQTT server | 2 | String | - | “%key%|%value%” | X |
| 1 | read value from ECU 3 | 2 | any | - | - |  |
| 2 | write value to ECU 3 | 2 | any | - | - |  |
| 3 | write Speed to Drive motor | 2 | Integer | - | {-100, 100} | X |
| 4 | write Steer to Steering servo | 2 | Integer | - | {-100, 100} | X |
| 5 | read front wheel speed from front wheel sensor (absolute value) | 2 | Integer | cm/s | >= 0 | X |
| 6 | read rear wheel speed from rear wheel sensor (absolute value) | 2 | Integer | cm/s | >= 0 | X |
| 7 | read voltage from ADC sensor | 2 | String | Volt | >= 0 | (X, only fake value) |
| 8 | read Position from position sensor | 2 | Long |  |  |  |
| 9 | write data to LED | 2 | String | - | PIN|VAL  {1|2|3}|{0|1}  PIN 1: red led  PIN 2: yel 1  PIN 3: yel 2  VAL 0: LED on  VAL 1: LED off | X |
| 10 | write data to external MQTT Server | 3 | String | - | “%key%|%value%” | X |
| 11 | write value to ECU 2 | 3 | any | - | - |  |
| 12 | read value from ECU 2 | 3 | any | - | - |  |
| 13 | read distance from forward-looking sensors | 3 | Integer | cm | >= 0 |  |
| 14 | read IMU from IMU sensor | 3 | String |  |  |  |

## 3.2 Introduce PlugIn template

In order to ease the PlugIn development, it is prepared the PlugIn template as follows:

package tests;

import sics.plugin.PlugInComponent;

import sics.port.PluginPPort;

import sics.port.PluginRPort;

public class NewClass extends PlugInComponent {

public PluginPPort pport;

public PluginRPort rport;

public NewClass() {}

public NewClass(String[] args) {

super(args);

}

public static void main(String[] args) {

NewClass instance = new NewClass(args);

instance.run();

}

@Override

public void init() {

// Initiate PluginPPort

pport = new PluginPPort(this, "pport");

rport = new PluginRPort(this, "rport", 0);

}

public void run() {

init();

// do functions, for example, read front wheel speed value from sensor and then publish through MQTT

while(true) {

// read front wheel speed value

Integer frontWheelData = (Integer)rport.read();

// Prepare published data, which is packaged in the format “key|value”

String pubData = “fs|” + String.valueof(frontWheelData);

// Publish data

pport.write(pubData);

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

;

}

}

}

}

## 3.3 Develop a new PlugIn

Step 1: Referring to the above PlugIn template, create a normal Java Class.

Step 2: Copy the template code in the new class and then replace “tests” in the first line with your package name, “NewClass” with your class name. The template codes include one PluginPPort and one PluginRPort attribute. Depending on your requirement, your new PlugIn could have more attributes of PlugIn port for reading or writing port. Correspondingly, these plugin ports must be initialized in the init method. The way for initializing PlugIn port can be referred to [PlugIn API documents](https://moped.sics.se/wordpress/wp-content/uploads/2014/12/apidocs.zip).

Step 3: Fill in the core function in the run method.

## 3.4 Introduce PlugIn configuration template

Likewise the PlugIn template, the PlugIn configuration template is also prepared, shown as followed:

<?xml version="1.0" encoding="UTF-8"?>

<app>

<vehicleName>MOPED</vehicleName>

<brand>SICS</brand>

<plugins>

<plugin>

<name>NewClass.suite</name>

<ecu>2</ecu>

<ports>

<port>

<name>pport</name>

</port>

<port>

<name>rport</name>

</port>

</ports>

</plugin>

</plugins>

<links>

<link>

<from>pport</from>

<to>0</to>

</link>

<link>

<from>5</from>

<to>rport</to>

</link>

</links>

</app>

In the XML template, the combination of ”vehicleName” tag and “brand” tag represents the vehicle type the application will be running on. Within “plugins” tag, it includes the definitions of PlugIn with name, target ECU, and ports. In the range of “links“ tag, it is defined about how to wire among ports.

NOTE that within “from” or “to” tag, the numbers should be referred to System.xml configuration (referring to Appendix), which is written down when the type of vehicle is built.

TIPS: the PlugIn configuration file is either written in Eclipse (right-click project, then select “NEW”, then select “Other”, then select “XML”, then select “XML File”) or any plain text editor.

# 4. Upload PlugIn to the trusted server

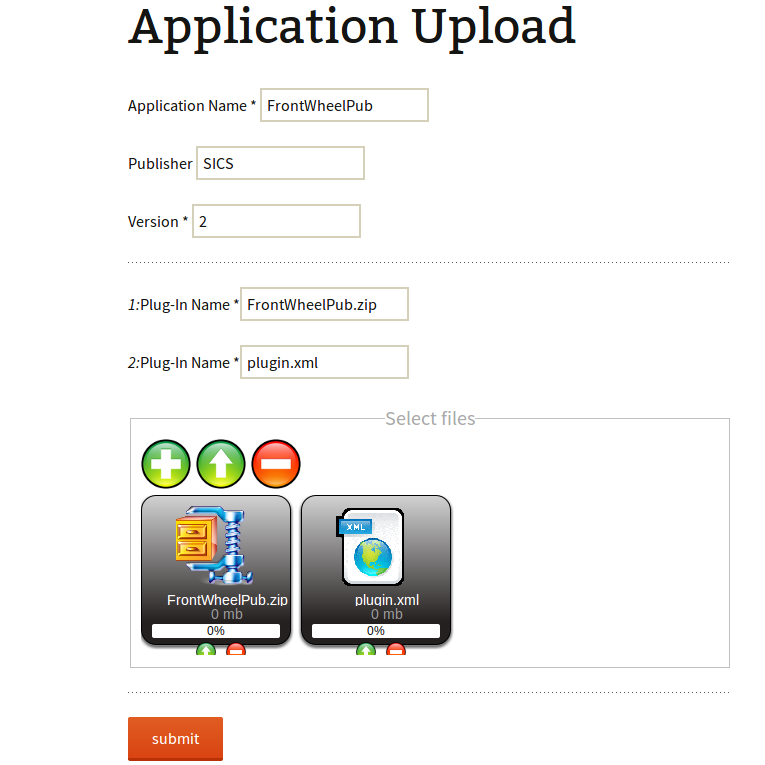
## 4.1 Preparation before upload PlugIn

First of all, register a new account by clicking “register” button in the right corner from [MOPED web portal](https://moped.sics.se/). Then, a new email would be received and meanwhile password needs be setted. Applying for permission of developer, the register user also sends an email to the [administrator](mailto:jax@sics.se) for granting developer permission.

## 4.2 Upload PlugIn .zip files and .xml file

Log in MOPED web portal with permission of developer and otherwise the application upload page cannot be accessible. Register users have no developer permission to upload PlugIns and have to apply for the developer permission from the administrator.

After login with permission of developer, click “The open innovation portal”, then click “Application Upload”.



Fill in the upload form with Application name, Publisher, Version (not constraint to Integer, could be string type too) and attach .zip files and .xml file by clicking PLUS button. If uploaded successfully, similar output will be popped up in the following:



# 5. PlugIn deployment on the Simulator

Step 1: download Simulator

The Simulator package includes simulator-0.0.1-SNAPSHOT.jar, apps, configs, db, and ecus. The simulator-0.0.1-SNAPSHOT.jar is normal Java jar executable file. The “apps” folder is the place where downloaded applications from server are stored. The “configs” folder is the place where a vehicle configuration file is store, which would be also scanned by the Simulator when it initiates in order to dynamically generate instances of ECU and a variety type of ports. The “db” folder is used as local database for saving PlugIn contexts. The “ecus” folder is a simulated file system for each ECU.

Step 2: drop a vehicle configuration file to the “configs” folder, if you want to change vehicle configuration (in the Simulator.zip, an example configuration is already included).

Step 3: run simulator: “java -jar simulator-0.0.1-SNAPSHOT.jar” from terminal.

Step 4: browse the “App Gallery” page in MOPED web portal and select one of applications by clicking green downloaded button in JDK column.

When the Simulator startups, using WirelessIno application on Android cell phone can control the car in the Simulator by following to the next few steps:

1. In the terminal, enter “ipconfig” on Windows or “ifconfig” on Linux, and capture IP address on current computer.
2. Startup WirelessIno on phone, then click settings, input IP address obtained from above and 9000 for port.

# 6. PlugIn deployment on the MOPED

browse the “App Gallery” page in MOPED web portal and select one of applications by clicking green downloaded button in Squawk column.

# Appendix

The example of system configuration description

<?xml version="1.0" encoding="UTF-8"?>

<vehicle>

<name>MOPED</name>

<brand>SICS</brand>

<vin>20UYA31581L000000</vin>

<ecm>

<server>

<!-- <ip>localhost</ip> -->

<!-- <ip>none</ip> -->

<ip>appz-ext.sics.se</ip>

<socketport>9999</socketport>

</server>

</ecm>

<ecus>

<ecu>

<id>2</id>

<swcs>

<swc>

<hasPirte>true</hasPirte>

<ports>

<!-- read data from SCU -->

<port>

<id>1</id>

</port>

<!-- write data to SCU -->

<port>

<id>2</id>

</port>

</ports>

</swc>

</swcs>

<sensors>

<!-- read speed data from Speed Sensor -->

<sensor>

<name>readSpeed</name>

<ports>

<port>

<id>5</id>

</port>

</ports>

</sensor>

<!-- read steer data from Steer Sensor -->

<sensor>

<name>readSteer</name>

<ports>

<port>

<id>6</id>

</port>

</ports>

</sensor>

<!-- read ADC data from ADC Sensor -->

<sensor>

<name>readAdc</name>

<ports>

<port>

<id>7</id>

</port>

</ports>

</sensor>

<!-- read Position data from Position Sensor -->

<sensor>

<name>readPosition</name>

<ports>

<port>

<id>8</id>

</port>

</ports>

</sensor>

<!-- write data to LED -->

<sensor>

<name>writeLED</name>

<ports>

<port>

<id>9</id>

</port>

</ports>

</sensor>

</sensors>

<actuators>

<!-- PUB port -->

<actuator>

<name>pubVCUData</name>

<ports>

<port>

<id>0</id>

</port>

</ports>

</actuator>

<!-- write data to Speed Actuator -->

<actuator>

<name>speedActuator</name>

<ports>

<port>

<id>3</id>

</port>

</ports>

</actuator>

<!-- write data to Steer Actuator -->

<actuator>

<name>steerActuator</name>

<ports>

<port>

<id>4</id>

</port>

</ports>

</actuator>

</actuators>

</ecu>

<ecu>

<id>3</id>

<swcs>

<swc>

<hasPirte>true</hasPirte>

<ports>

<!-- write data to VCU -->

<port>

<id>11</id>

</port>

<!-- read data from VCU -->

<port>

<id>12</id>

</port>

</ports>

</swc>

</swcs>

<sensors>

<!-- read distance data from forward-looking Sensor -->

<sensor>

<name>readDistance</name>

<ports>

<port>

<id>13</id>

</port>

</ports>

</sensor>

<!-- read IMU data from IMU Sensor -->

<sensor>

<name>readIMU</name>

<ports>

<port>

<id>14</id>

</port>

</ports>

</sensor>

</sensors>

<actuators>

<!-- PUB port -->

<actuator>

<name>pubSCUData</name>

<ports>

<port>

<id>10</id>

</port>

</ports>

</actuator>

</actuators>

</ecu>

</ecus>

<links>

<!-- Between VCU and SCU -->

<link>

<type>2</type>

<from>11</from>

<to>1</to>

</link>

<link>

<type>2</type>

<from>2</from>

<to>12</to>

</link>

</links>

</vehicle>

TIPS: In System.xml file, “sensors” and “actuators” tags are only used for Simulator to dynamically generate “sensor” and “actuator” instances. In other cases, they can be ignored or even removed from file.