Application Note AN-121

Setting up the Ethernet I/O Agent
Reach Technology, Inc.
01/28/2016

Reach Technology Inc.

www.reachtech.com

Sales: 510-770-1417 x112
sales@reachtech.com

Technical Support: 503-675-6464
techsupport@reachtech.com

1 Overview

This application note describes how to configure the Ethernet I/O agent (eio-agent) on a G2 Display Module.

The basic function of the application is to provide level indicators for water, fuel and temperature, an OIL LED, a speedometer, a Start button and a Stop button. When either button is pressed or released, messages are sent to the host controller via the I/O agent. When host controller messages are received for any of the indicators, the specified value is shown on that indicator.

2 QML Application Screen

The QML GUI application (AgentSampleProject) contains VerticalLevelIndicator elements for Water level, Fuel level, and temperature. There are two ImageButton elements for Start and Stop buttons, an LEDLight element for an OIL LED, and a Speedometer element.

The screen shown is from a 7” display module. The 4.3” display is similar.

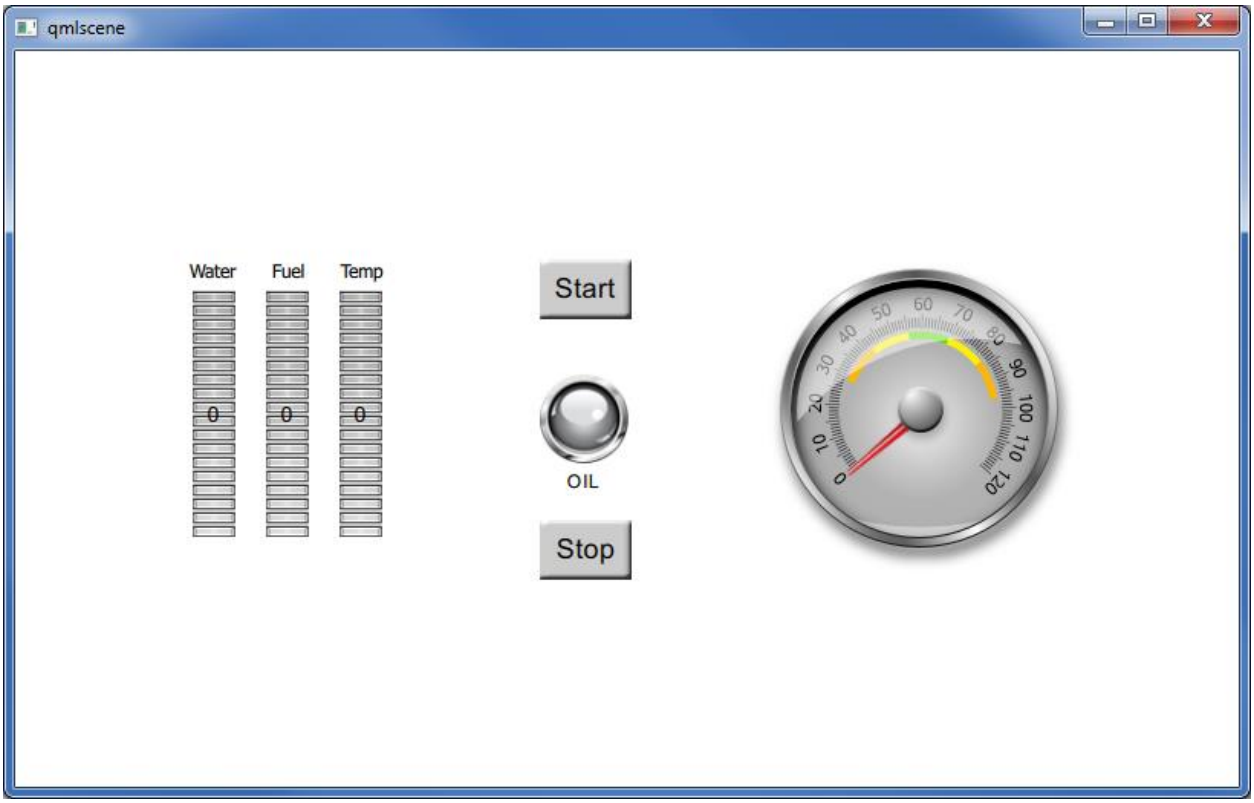

Figure 1 – QML Application Screen

3 Application Messages

The table below lists the messages used in this application.

Table 1 – Application Messages

Description	Direction	TIO Message	QML Message
Water Level	Incoming	“w=<value>”	“water.value=<value>”
Fuel Level	Incoming	“f=<value>”	“fuel.value=<value>”
Temp Level	Incoming	“t=<value>”	“temp.value=<value>”
OIL LED	Incoming	“o=<value>”	“oil.on=<value>”
Speedometer	Incoming	“s=<value>”	“speed.value=<value>”
Start Button	Outgoing	“st=<value>”	“Start.state=<value>”
Stop Button	Outgoing	“sp=<value>”	“Stop.state=<value>”

3.1 I/O Agent Messages

The messages received by the I/O agent can be application-specific, and the I/O agent may translate them from one form to another. For this application note, it is assumed that the I/O agent passes messages it receives straight through without translation.

Note that all messages from the host controller must be terminated with a new-line (ASCII LF, 0x0A).

3.2 TIO Messages

The “Incoming” TIO messages shown in Table 1 are as received from the I/O agent.

The translation between TIO messages and QML messages is done as specified in the project’s “translate.txt” file. Note that the I/O agent could easily just translate the messages it receives from the host controller directly into the QML form, and then the translations defined in translate.txt would not be needed (although the I/O agent would still send the messages to the tio-agent).

3.3 QML Messages

Each QML message is in the “object.property=value” format that the QML viewer uses. The “object” comes from the “objectName” property assigned to the QML element, and the “property” is the settable property as defined by the element itself.

4 Example Message Flows

4.1 Host Controller to Display Module

An example message to set the Water level indicator to a value of 7 would look like:

w=7

The I/O agent sends this to the tio-agent. The tio-agent receives this message and uses rules in translate.txt to translate it into the QML message:

water.value=7

4.2 Display Module to Host Controller

The user presses the Start button, and the QML Viewer sends this message (per the GUI code):

start.value=1

The tio-agent uses rules in translate.txt to translate this message into:

st=1

and sends this message to the I/O agent, which then sends it to the host controller.

5 Setup

This application note assumes that you have:

- The G2 Dev kit installed and running.
- G2Link installed on your Windows system.
- The Reach Linux VM installed and running.

6 Initial Testing

The QML application can be tested with the standard SIO agent running. You would just send any of the TIO Messages from Table 1 with a terminal emulator connected to the RS232 port on the Dev Kit. If you press one of the buttons, you will see the listed start or stop message. This allows you to verify that the QML application has been downloaded and is running correctly.

To download the QML application, start G2Link, click on `Select Serial Port` and select the COM port connected to the Debug port on the G2 Dev Kit, and click on `Apply and Close`. You should see the remaining buttons get enabled and the IP Address filled in with the G2 module's setting.

Click on `Publish and Run`, click on `Browse`, navigate to the `AgentSampleProject` folder for your display module, click `OK`, and then click `Go` to download the application to the G2 module. Once the application is downloaded, the QML viewer will restart and you should see the application screen on the display module.

Connect a terminal emulator to UART1/RS232 on the G2 Dev Kit (115200 baud, N81). Now you can type in any of the TIO Messages. For example, entering `w=5` followed by the `<enter>` key will set the Water level indicator to 5. Press the `Start` button and you will see the message `st=1`, and when you release the button you will see the message `st=0`.

7 Building and Installing reach-eio-agent

7.1 Building

1. Copy the `reach-eio-agent` directory to your Linux VM.
2. Open the project file (`reach-eio-agent.pro`) with `Qt Creator`.
3. If prompted, Configure Project for `G2H1`.
4. Click on `Build->Build All`.
5. The executable will be in your Projects directory, in in the directory `build-reach-eio-agent-G2H1-Debug`, and is named `eio-agent`.

7.2 Installing

1. On the VM desktop, click on the `Connect to Server` icon.
2. In the `Server` field, enter the IP address of your G2 Dev Kit (as shown in G2Link).
3. In the `Type` pull-down, select `Windows share`.
4. Leave the other fields blank and click `Connect`.
5. In the browser window that opens, double-click on `app`, then `bin`.
6. Delete any existing file named `eio-agent`.
7. Copy the new file `eio-agent` into the `bin` directory.

8. On the PC desktop, using G2Link, click on `View Files in Explorer`.
9. In the browser window that opens, double-click on `config`, then on `init.d`.
10. Copy the file `Scripts/eio-agent` to `init.d`.
11. Close the browser window.

7.3 Startup Scripts

1. In G2Link, click on `View->Advanced View`.
2. In the bottom text window, use the command "`mv /etc/rcS.d/S99sio-agent .`" to disable the sio-agent. (Note that the script may reside in `/etc/rc5.d` on some releases. If so, simple change the references below accordingly.)
3. Make a softlink in `/etc/rcS.d` to the `eio-agent` script with the commands (in the bottom text window):
   ```
   cd /etc/rcS.d
   ln –s ../init.d/eio-agent S99eio-agent
   ```
4. Use the command "`chmod 755 /application/bin/eio-agent`" to set the file permissions on the eio-agent.
5. Use the command `reboot` to restart the module with the eio-agent running.

8 Final Testing

Use telnet to send and receive the same messages as above. The IP address of the display module is shown in G2Link. The default port is 7880 (as defined in the eio-agent script). Be sure to configure your telnet agent to add LF characters to the end of the lines.

9 Source Code

9.1 QML Application

The QML Application code is the contents of the file mainview.qml in the QML project folder. Points of interest are the “objectName” fields in all elements that receive value settings, and the use of “connection.sendMessage()” in the buttons to send messages to the host controller.

The code shown below is from a 7” display module. The 4.3” display module code is essentially the same, but with a different QtQuick version, and different X/Y locations for the GUI elements.

```
import QtQuick 2.0
import "components"

Rectangle {
    id: page
    width: 800
    height: 480

    ImageButton {
        id: start_button
        x: 343
        y: 135
        width: 60
        height: 40
        text: "Start"
        imageUp: "images/internal_button_up.bmp"
        font.pixelSize: 18
        textColor: "#000000"
        imageDown: "images/internal_button_dn.bmp"
        font.bold: false
        font.family: "Arial"

        onButtonPress: {
//            oil.on=true
            connection.sendMessage("Start.state=1")
        }
        onButtonRelease: {
            connection.sendMessage("Start.state=0")
        }
    }

    ImageButton {
        id: stop_button
        x: 343
        y: 305
        width: 60
        height: 40
        text: "Stop"
        imageUp: "images/internal_button_up.bmp"
        font.pixelSize: 18
        textColor: "#000000"
        imageDown: "images/internal_button_dn.bmp"
        font.bold: false
```

```
            font.family: "Arial"

            onButtonPress: {
//                oil.on=false
                connection.sendMessage("Stop.state=1")
            }
            onButtonRelease: {
                connection.sendMessage("Stop.state=0")
            }
        }

        LEDLight {
            id: oil_led
            objectName: "oil"
            x: 272
            y: 191
            width: 202
            height: 99
            on: false
            font.pixelSize: 12
            textColor: "#000000"
            textPosition: "bottom"
            label: "OIL"
            fieldSpacing: 4
            font.bold: false
            font.family: "Arial"
            imageOff: "images/ledoff.png"
            imageOn: "images/ledon.png"
        }

        VerticalLevelIndicator {
            id: water_level
            objectName: "water"
            x: 116
            y: 156
            width: 28
            height: 161
            minValue: 0
            hintFontPixelSize: 14
            imageBase: "images/level.png"
            hintFontColor: "#000000"
            value: 0
            showHint: true
            imageOverlay: "images/level_overlay.png"
            maxValue: 18
            increment: 9
            startPosition: "bottom"
            hintFontFamily: "Arial"
        }

        VerticalLevelIndicator {
            id: fuel_level
            objectName: "fuel"
            x: 164
            y: 156
            width: 28
            height: 161
```

```
        minValue: 0
        hintFontPixelSize: 14
        imageBase: "images/level.png"
        hintFontColor: "#000000"
        value: 0
        showHint: true
        imageOverlay: "images/level_overlay.png"
        maxValue: 18
        increment: 9
        startPosition: "bottom"
        hintFontFamily: "Arial"
    }

    VerticalLevelIndicator {
        id: temp_level
        objectName: "temp"
        x: 212
        y: 156
        width: 28
        height: 161
        minValue: 0
        hintFontPixelSize: 14
        imageBase: "images/level.png"
        hintFontColor: "#000000"
        value: 0
        showHint: true
        imageOverlay: "images/level_overlay.png"
        maxValue: 18
        increment: 9
        startPosition: "bottom"
        hintFontFamily: "Arial"
    }

    Text {
        id: text1
        x: 114
        y: 136
        text: qsTr("Water")
        horizontalAlignment: Text.AlignHCenter
        font.pixelSize: 12
    }

    Text {
        id: text2
        x: 162
        y: 136
        width: 33
        text: qsTr("Fuel")
        horizontalAlignment: Text.AlignHCenter
        font.pixelSize: 12
    }

    Text {
        id: text3
        x: 210
        y: 136
        width: 33
```

```
            text: qsTr("Temp")
            font.pixelSize: 12
            horizontalAlignment: Text.AlignHCenter
        }

        Speedometer {
            id: speedometer
            objectName: "speed"
            x: 490
            y: 136
            width: 210
            height: 210
            needleImage: "images/needle.png"
            overlayImageHeight: 105
            min: 0
            needleImageHeight: 63
            overlayImage: "images/overlay.png"
            value: 0
            needleImageWidth: 8
            max: 120
            needleRotationY: 65
            needleRotationX: 5
            maxAngle: 133
            overlayY: 18
            overlayImageWidth: 148
            overlayX: 21
            needleY: 33
            needleX: 98
            backgroundImage: "images/meterbackground.png"
        }

    }
```