Application Note AN-112

G2C1 CAN Sample Program
Reach Technology, Inc.
12/17/2015

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 a QML application used to demonstrate the CAN IO agent (reach-can-agent) in the G2C1 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 CAN bus. When CAN messages are received for any of the indicators, the specified value is shown on that indicator.

2 QML Application Screen

The QML GUI application (CANSampleProject) 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.

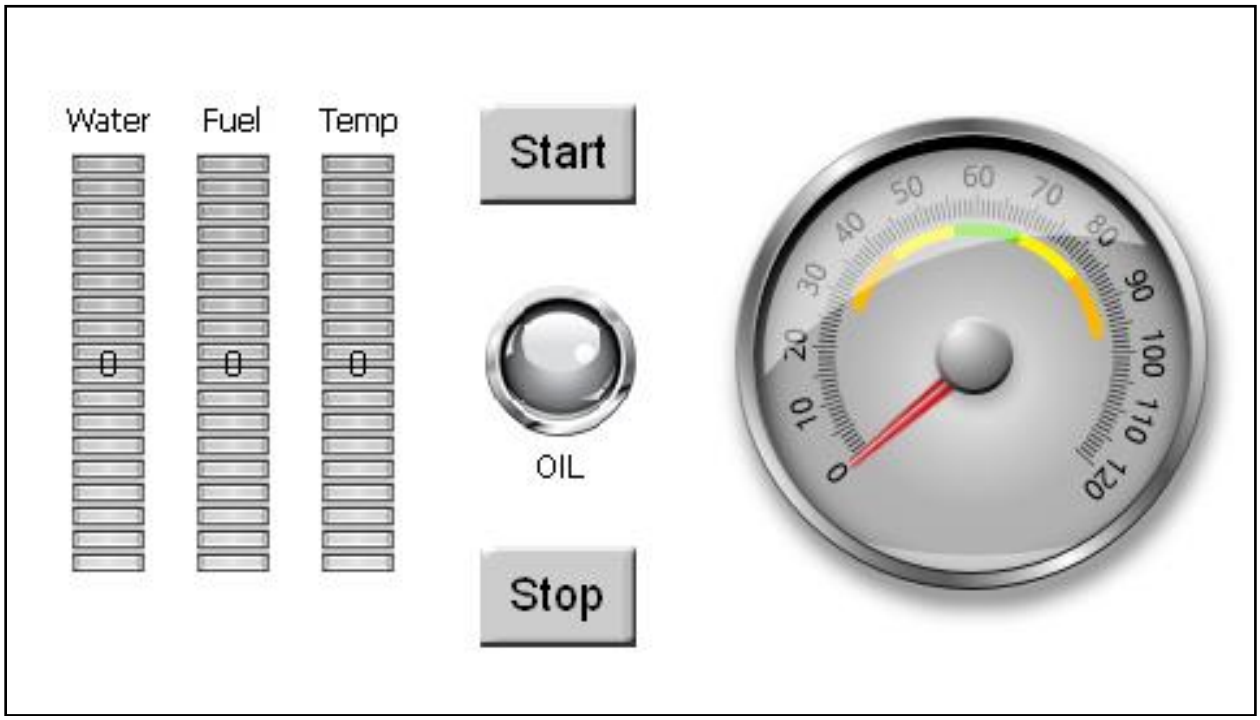

Figure 1 – QML Application Screen

3 Application Messages

The table below lists the messages used in this application.

Table 1 – Application Messages

Description	CAN ID	CAN DLC	CAN Data	TIO Message	QML Message
Water Level	0x01	1	<value>, 0,0,0,0,0,0,0	“w=<value>”	“water.value=<value>”
Fuel Level	0x02	1	<value>, 0,0,0,0,0,0,0	“f=<value>”	“fuel.value=<value>”
Temp Level	0x03	1	<value>, 0,0,0,0,0,0,0	“t=<value>”	“temp.value=<value>”
OIL LED	0x10	1	<value>, 0,0,0,0,0,0,0	“o=<value>”	“oil.on=<value>”
Speedometer	0x20	1	<value>, 0,0,0,0,0,0,0	“s=<value>”	“speed.value=<value>”
Start Button	0x80	1	<value>, 0,0,0,0,0,0,0	“st=<value>”	“Start.state=<value>”
Stop Button	0x81	1	<value>, 0,0,0,0,0,0,0	“sp=<value>”	“Stop.state=<value>”

3.1 CAN Messages

The CAN ID, DLC and Data fields define the CAN messages. All values are binary, ranging from 0 to 255 (as appropriate to the object referenced). For the Level objects and the Speedometer, the <value> field in the Data is the specified level or speed to indicate. For the Oil LED, <value> is the on/off specification, with 1 for ‘on’ and 0 for ‘off’. For the Start and Stop buttons, <value> is 1 for ‘pressed’ and 0 for ‘not pressed’. The translation between CAN messages and TIO messages is hard-coded within the can-agent, making the can-agent application specific.

3.2 TIO Messages

Each TIO message is in a short-hand style, roughly translating the CAN messages to/from ASCII. The <value> fields have the same meaning as in the CAN messages, but the fields are all in ASCII. Messages sent to the tio-agent must be terminated with the ASCII LF character (0x0A).

The translation between TIO messages and QML messages is done as specified in the project’s “translate.txt” file. Note that the can-agent could easily just translate the messages into the QML form, and then the translations in translate.txt would not be needed.

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

ID 0x01
DLC 1
Data 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00

The can-agent translates this into a TIO message that looks like:

w=7

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

The can-agent receives this message and translates it into the CAN message:

ID 0x80
DLC 1
Data 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00

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 `CANSampleProject` folder, 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-can-agent

7.1 Building

1. Copy the `reach-can-agent` directory to your Linux VM.
2. Open the project file (`reach-can-agent.pro`) with `Qt Creator for Reach Display Module Applications`.
3. Click on `Build->Build All`.
4. The executable will be in your Projects directory, in in the directory `build-reach-can-agent-G2C2-Debug`, and is named `reach-can-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 the existing file named `can-agent`.
7. Copy the new file `reach-can-agent` into the `bin` directory, right click on `reach-can-agent` and rename it to `can-agent`.

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/can-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/rc5.d/S99sio-agent .`" to disable the sio-agent.
3. Make a softlink in `/etc/rc5.d` to the `can-agent` script with the commands (in the bottom text window):
   ```
   cd /etc/rc5.d
   ln –s ../init.d/can-agent S99can-agent
   ```
4. Use the command "`chmod 755 /application/bin/can-agent`" to set the file permissions on the can-agent.
5. Use the command `reboot` to restart the module with the can-agent running.

8 Unresolved Issues

1. The TIO agent currently only looks for “/tmp/sioSocket” when making a server socket, so the can-agent must use this same name. Perhaps the agents should all use a more generic name, but this requires a change to all existing agents.

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.

```
import QtQuick 1.1
import "components"

Rectangle {
    id: page
    width: 480
    height: 272

    ImageButton {
        id: start_button
        x: 182
        y: 36
        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: 182
        y: 206
        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: 184
        y: 107
        width: 58
        height: 58
        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: 25
        y: 56
        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: 73
        y: 56
        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: 121
    y: 56
    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: 23
    y: 36
    text: qsTr("Water")
    horizontalAlignment: Text.AlignHCenter
    font.pixelSize: 12
}

Text {
    id: text2
    x: 71
    y: 36
    width: 33
    text: qsTr("Fuel")
    horizontalAlignment: Text.AlignHCenter
    font.pixelSize: 12
}

Text {
    id: text3
    x: 119
    y: 36
    width: 33
    text: qsTr("Temp")
    font.pixelSize: 12
```

```
        horizontalAlignment: Text.AlignHCenter
    }

    Speedometer {
        id: speedometer
        objectName: "speed"
        x: 270
        y: 36
        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"
    }

}
```

9.2 CAN Agent Message Translation

These two functions are from the file can_server_socket.c in the reach-can-agent project. The application-specific message translation occurs in the code blocks bracketed with `#ifdef CUSTOMER_XLATE`. The translations shown below implement the message translations as described in the section Application Messages.

```
/**
 * Reads a single message from the socket connected to the
 * tcp/ip server port. If no message is ready to be received, the call
 * will block until one is available.
 *
 * @param socketFd the file descriptor of for the already open
 *                 socket connecting to the tio-agent
 * @param msgBuff address of a contiguous array into which the
 *                message will be written upon receipt from the
 *                tio-agent
 * @param bufferSize the number of bytes in msgBuff
 *
 * @return int 0 if no message to return (handled here), -1 if
 *         recv() returned an error code (close connection) or
 *         >0 to indicate msgBuff has that many characters
 *         filled in
```

```
 */
int canServerSocketRead(int socketFd, char *msgBuff)
{
    int cnt;
    int i;
    struct can_frame frame;
    memset(&frame, 0, sizeof(frame));
    cnt = read(socketFd, &frame, sizeof(frame));

    if (cnt < 0)
    {
        LogMsg(LOG_INFO, "%s(): recv() failed, client closed\n",
               __FUNCTION__);
        close(socketFd);
        return -1;
    }
    else
    {
        LogMsg(LOG_INFO, "CAN message - ID: 0x%lx (%ld)  dlc: 0x%02X
data: %02X %02X %02X %02X %02X %02X %02X %02X\n",
               frame.can_id, frame.can_id,
               frame.can_dlc,
               frame.data[0], frame.data[1], frame.data[2],
               frame.data[3], frame.data[4], frame.data[5],
               frame.data[6], frame.data[7]);

#define CUSTOMER_XLATE
#ifdef  CUSTOMER_XLATE

        // YOUR CAN-to-TIO TRANSLATE CODE GOES HERE

        // example translate - see CAN_messages.docx for message details
        switch (frame.can_id)
        {
            case 0x01:          // Water Level
                sprintf(msgBuff, "w=%d\n", (unsigned char)frame.data[0]);
                break;
            case 0x02:          // Fuel Level
                msgBuff[0] = 'f';
                msgBuff[1] = '=';
                sprintf(&msgBuff[2], "%d\n", frame.data[0]);
                break;
            case 0x03:          // Temp Level
                msgBuff[0] = 't';
                msgBuff[1] = '=';
                sprintf(&msgBuff[2], "%d\n", frame.data[0]);
                break;
            case 0x10:          // OIL LED
                msgBuff[0] = 'o';
                msgBuff[1] = '=';
                sprintf(&msgBuff[2], "%d\n", frame.data[0]);
                break;
            case 0x20:          // Speedometer
                msgBuff[0] = 's';
                msgBuff[1] = '=';
                sprintf(&msgBuff[2], "%d\n", frame.data[0]);
                break;
```

```
            // button messages not received by display module
            case 0x80:          // Start Button
            case 0x81:          // Stop Button
            default:
                // could issue an error here...
                LogMsg(LOG_ERR, "Unrecognized CAN message\n");
                break;
        }
        cnt = strlen(msgBuff);
#else
        // pass received messages (ASCII strings) to the TIO agent.
        // note that the strings should end with a new-line char (0x0A).
        strncpy(msgBuff, (char *)frame.data, 8);
        cnt = frame.can_dlc;
        msgBuff[cnt] = '\0';
#endif
        LogMsg(LOG_INFO, "%s: cnt: %d, msgBuff: %s\n", __FUNCTION__,
                    cnt, msgBuff);
        return cnt;
    }

}


void canServerSocketWrite(int socketFd, const char *buff)
{
    int cnt = strlen(buff);
    if (cnt > 8)
        cnt = 8;
    int i = 0;
    struct can_frame frame;

    memset(&frame, 0, sizeof(frame));

#ifdef  CUSTOMER_XLATE

        // YOUR TIO-to-CAN TRANSLATE CODE GOES HERE

        // example translate - see CAN_messages.docx for message details
        if (strncmp(buff, "st=", 3) == 0)       // Start Button
        {
            frame.can_id = 0x80;
            frame.can_dlc = 1;
            frame.data[0] = (buff[3] == '0' ? 0 : 1);
        }
        else if (strncmp(buff, "sp=", 3) == 0)  // Stop button
        {
            frame.can_id = 0x81;
            frame.can_dlc = 1;
            frame.data[0] = (buff[3] == '0' ? 0 : 1);
        }
        else
        {
            // could issue an error here...
            LogMsg(LOG_ERR, "Unrecognized TIO message: %s\n", buff);
        }
#else
```

```
    frame.can_id = 0;
    strncpy((char *)frame.data, buff, cnt);
    frame.can_dlc = cnt;
#endif

    if (write(socketFd, &frame, sizeof(frame)) < 0)
    {
        LogMsg(LOG_ERR, "CAN BUS: write() failed, %d\n",
            socketFd);
        perror("what's messed up?");
    }
    else
    {
#ifdef  CUSTOMER_XLATE
        LogMsg(LOG_INFO, "%s: sent CAN message - ID: 0x%lx (%ld)  dlc:
               0x%02X  data: %02X %02X %02X %02X %02X %02X %02X %02X\n",
               __FUNCTION__,
               frame.can_id, frame.can_id,
               frame.can_dlc,
               frame.data[0], frame.data[1], frame.data[2],
               frame.data[3], frame.data[4], frame.data[5],
               frame.data[6], frame.data[7]);
#else
        LogMsg(LOG_INFO, "%s: sent = %s", __FUNCTION__, frame.data);
#endif
    }
}
```