Freescale FRDM-K64F: Using MDK Middleware

MDK Version 5 Tutorial

AN271, March 2015, V 1.10

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_271.asp

Abstract

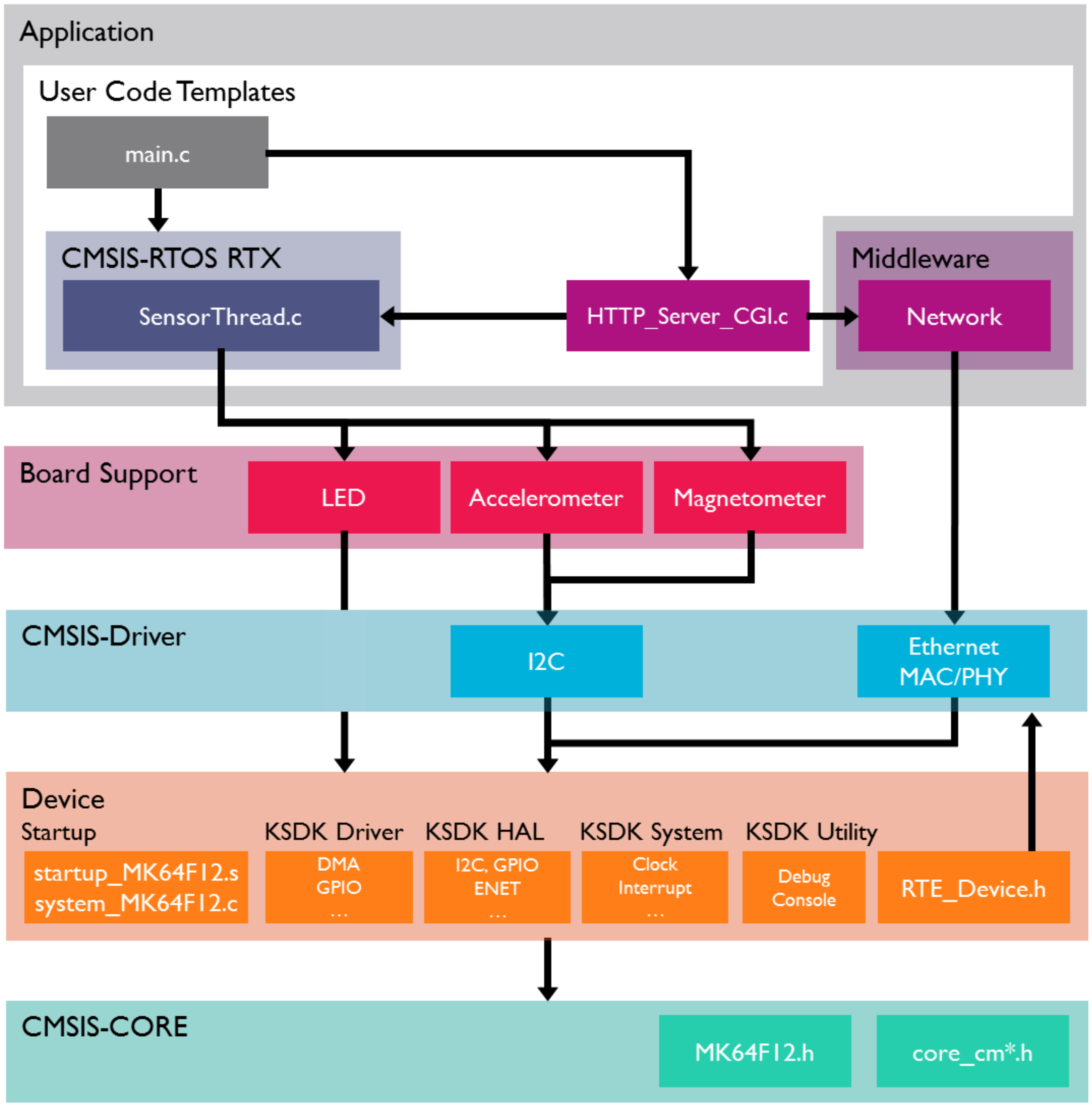
This tutorial runs you through the development of a middleware application on the FRDM-K64F development board. The application reads data from a MEMS sensor (accelerometer and magnetometer) and displays it on a website using CGI and JavaScript. The web server is a software component of the MDK-Professional middleware. The application uses I2C and Ethernet (MAC + PHY) CMSIS-Drivers and makes use of the underlying Freescale Kinetis SDK HAL.

Contents

Introduction

This workshop explains how to create a software framework for a sophisticated microcontroller application using CMSIS and Middleware components. During this workshop a demo application is created that implements the following functions:

- Read incoming data from an external MEMS sensor (accelerometer and magnetometer).
- Display this content using CGI and JavaScript on a webpage as text representation and with a graphical interface.



Software Stack

The application is created by using user code templates. These templates are part of software components such as the Middleware or CMSIS-RTOS.

CMSIS-RTOS RTX is a real-time operating system that is part of MDK and adheres to the CMSIS specification. It is used to control the application.

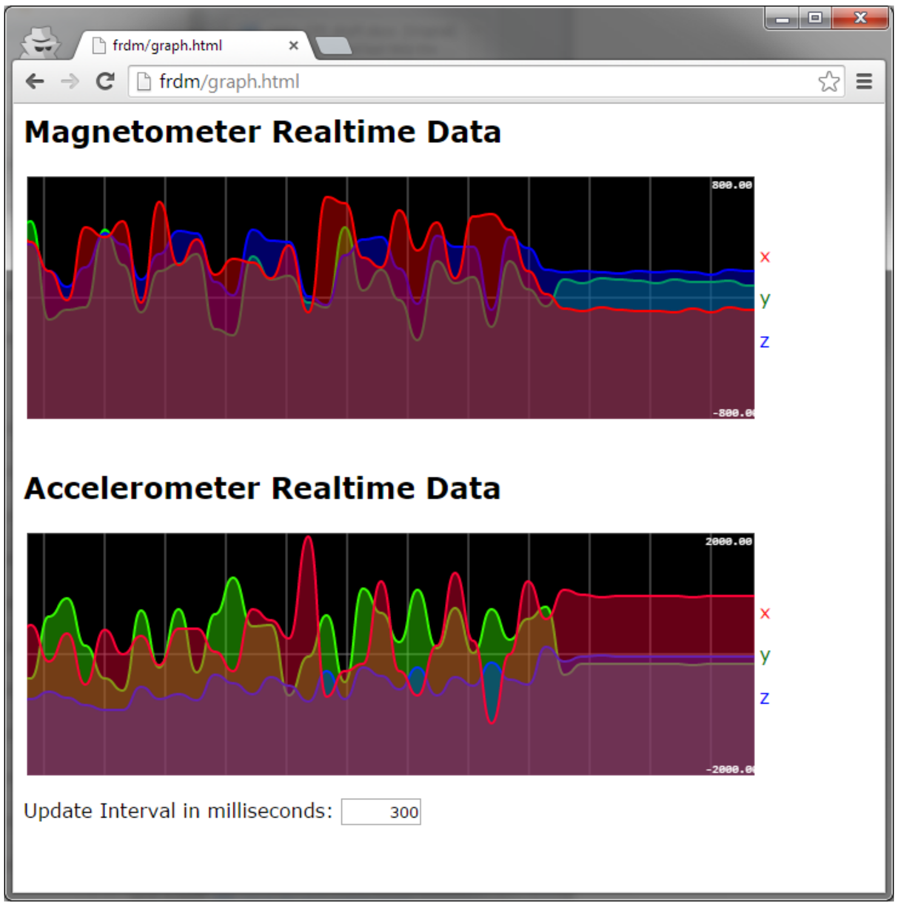
The **board support** files enable the user to quickly develop code for the hardware that is used here. It provides a simple API to control LEDs and get access to the external devices (accelerometer/magnetometer). Other components provide support for push buttons for example.

Middleware provides stacks for TCP/IP networking, USB communication, graphics, and file access. The Middleware used in this application is part of MDK-Professional and uses several CMSIS-Driver components.

CMSIS-Driver is an API that defines generic peripheral driver interfaces for middleware making it reusable across compliant devices. It connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. CMSIS-Driver are available for several microcontroller families and are part of the device family packs (DFPs).

The DFP contains the support for the **device** in terms of startup and system code, a configuration file for the CMSIS-Driver and a device family specific software framework with hardware abstraction layer (HAL).

The basis for the software framework is **CMSIS-Core** that implements the basic run-time system for a Cortex-M device and gives the user access to the processor core and the device peripherals. The device header files adhere to the CMSIS-Core standard and help the user to access the underlying hardware.


frdm/graph.html
frdm/graph.html
Magnetometer Realtime Data
800.00
-800.00
x
y
z
Accelerometer Realtime Data
2000.00
-2000.00
x
y
z
Update Interval in milliseconds:
300

Prerequisites

To run through the workshop you need to install the following software. Directions are given below:

- MDK-ARM Version 5.14 or later (https://www.keil.com/demo/eval/arm.htm).
- A valid MDK-Professional license.
- ARM::CMSIS 4.3 or higher
- Keil::MDK-Middleware 6.3 or higher
- Keil::Kinetis_SDK_DFP 2.0 or higher
- Freescale FRDM-K64F board
- Micro-USB cable
- Ethernet-cable to connect to Ethernet TCP/IP network

Install MDK-ARM and the listed software packs (http://www.keil.com/mdk5/install).

Setup the J-LINK on-board debugger

The Freescale OpenSDA platform provides an open on-board debugger platform that can be reprogrammed to behave like many different popular debuggers. SEGGER created a firmware which runs on the Freescale OpenSDA platform, making it J-Link compatible. This tutorial will make use of MDK-ARMs JLINK support. Enable the JLINK once as follows:

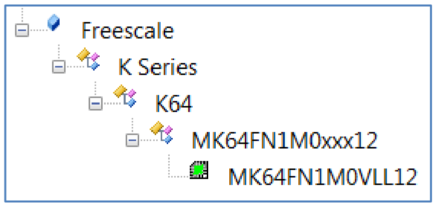
1. Acquire the latest firmware for OpenSDA V2 from https://segger.com/opensda.html Do ***not*** use V2.1
2. Attach the USB cable to your PC. Do not connect it to the K64 board yet.
3. Keep RESET pressed while you connect the USB cable to the SDAUSB port J26 on the FRDM-K64F.
4. Release RESET. A drive called BOOTLOADER will show up on your PC.
5. Copy the file JLink_OpenSDA_V2.bin onto the BOOTLOADER drive. After download and reprogramming is done the green LED next to the SDA connector starts to blink once a second.
6. If this is the first time you operate a J-Link on your PC: Run the InstDrivers.exe from *C:\Keil\ARM\Segger\USBDriver\* (Note: You need Administration Rights on your PC)
7. If this does not work, try running InstDriversCDC.exe from C:\Keil_v5\ARM\Segger\USBDriver\CDC\.
8. Reconnect the USB to start the new SDA firmware. The green LED will be on. If it blinks, the install failed.
9. The J-Link will be stored in Flash memory. It does not need to be reinstalled unless it somehow gets corrupted.

Install the MDK-Professional License

1. Obtain the PSN number that will be used to license your copy of MDK.
2. Start µVision in Administrator mode.
3. Select File/License Manager.
4. For more information and instructions visit: www.keil.com/download/license/

Create a Simple Project

1. Start µVision.
2. Create a new µVision Project: Select Project/New µVision Project…
3. Create a folder for your project and give it a name. Enter this folder.
4. Enter a project name in the Name: box. Click OK. The Select Device… window opens.
5. Select Freescale MK64FN1M0VLL12 as shown here:
6. Click on OK. The Manage Run-Time Environment window opens.

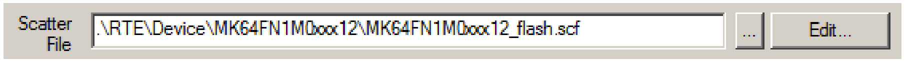


Configure the Run-Time Environment:

7. Make these selections:
Board Support::LED API:LED
CMSIS::RTOS(API)::Keil RTX
Device::Startup
8. Note there are some orange blocks. Click on Resolve and µVision will automatically select the required files.
9. All blocks will now be green. Select OK and they are added to your project and listed in the Project window.
10. In the Project window, expand the heading Target 1.
11. Right click on Source Group 1 and select Add New item to Group Source Group 1…
12. The Add New Item… window opens.
13. Select User Code Template and then expand the CMSIS heading.
14. Select Add "CMSIS-RTOS main function". Click on Add and it is added to your project under Source Group 1.

Configure Target Options:

15. Click on the Options for Target icon or ALT-F7
16. Click on the Target tab. Select Use MicroLib.
17. Click on the Output tab. Unselect Browse information.
18. Click on the C/C++ tab. In the Define box: add NDEBUG. Confirm Optimization to Level 0(-O0).
19. Select the Linker tab. Unselect Use Memory Layout from Target dialog.
20. In the Scatter file box, select .\RTE\Device\MK64FN1M0VLL12\MK64FN1M0xxx12_flash as shown below: Click


Scatter File .\RTE\Device\MK64FN1M0xxx12\MK64FN1M0xxx12_flash.scf

Open to add this scatter file. Note the file initially has no filename extension. This can change in a future release.
21. Connect the board USB SDAUSB connector to your PC. The usual USB dual-tone should be heard.
22. Select the Debug tab. Select J-Link / J-Trace Cortex in the USE: box on the right side of this window.
23. Select Settings and set the Port box to SW. In the SW-Device the ARM CoreSight SW-DP must be visible.
24. Click on OK twice to close the Options for Target window.

Add a Header File:

25. In main.c, right click on line 7 and select Insert "# include file".
26. Select #include 'Board_LED.h'. This line will be added to main.c.

Configure CMSIS-RTOS RTX:

27. In the Project window, expand the CMSIS heading and double-click on RTX_Conf_CM.c to open it.
28. Click on the Configuration Wizard tab at the bottom of this window.
29. Click on Expand All button and make the following changes:
30. Set RTOS Kernel Timer input Clock frequency to 120 000 000 Hz. (120 MHz)
31. Increase both the Default and Main Thread Stack sizes to 1024.
32. Click on File/Save All or .

Add files to main.c:

1. Near line 9 in main.c add this line: extern void hardware_init (void);
2. Add these lines to configure main.c like this:

```
        Int main (void)  {
              osKernelInitialize();         // initialize CMSIS-RTOS
              hardware_init();
              LED_Initialize();
              osKernelStart();              // start thread execution

              while(1) {
                     osDelay(500);
                     LED_On(1);     //red LED
                     osDelay(500);
                     LED_Off(1);
          }
        }
```

3. Click on File/Save All or .

Compile and Run Program:

4. Click on Build or F7 to compile the program. There will be no errors and no warnings.
5. Enter Debug mode by clicking on the Debug icon. The program will be programmed in the Flash memory.
6. Click on the RUN icon. The red LED will blink at a rate of 1 second. 1 ms * (2) 500 ticks = 1 sec.
7. If the LED does not blink, go back to find the mistake.
8. Stop the program with the STOP icon. **and** exit Debug mode .

Create a new RTX Thread

1. Right click on Source Group 1 in the Project window and select Add New item to Group Source Group 1…
2. The Add New Item… window opens.
3. Select User Code Template and then expand the CMSIS heading.
4. Highlight "CMSIS-RTOS Thread" and name it SensorThread.c. Select Add. This file is added to your project.
5. Normally you would create a unique name for the new thread rather than the provided for in the template (Thread). This is to prevent conflicts if you create other threads in a similar fashion. In this case, since we are adding only one thread, you do not have to make such changes.
6. In main.c near line 8, add: extern void Init_Thread(void);
7. Near line 18 in main.c, right before the osKernelStart(); call, add the line Init_Thread();.

Add blink LED to the thread SensorThread:

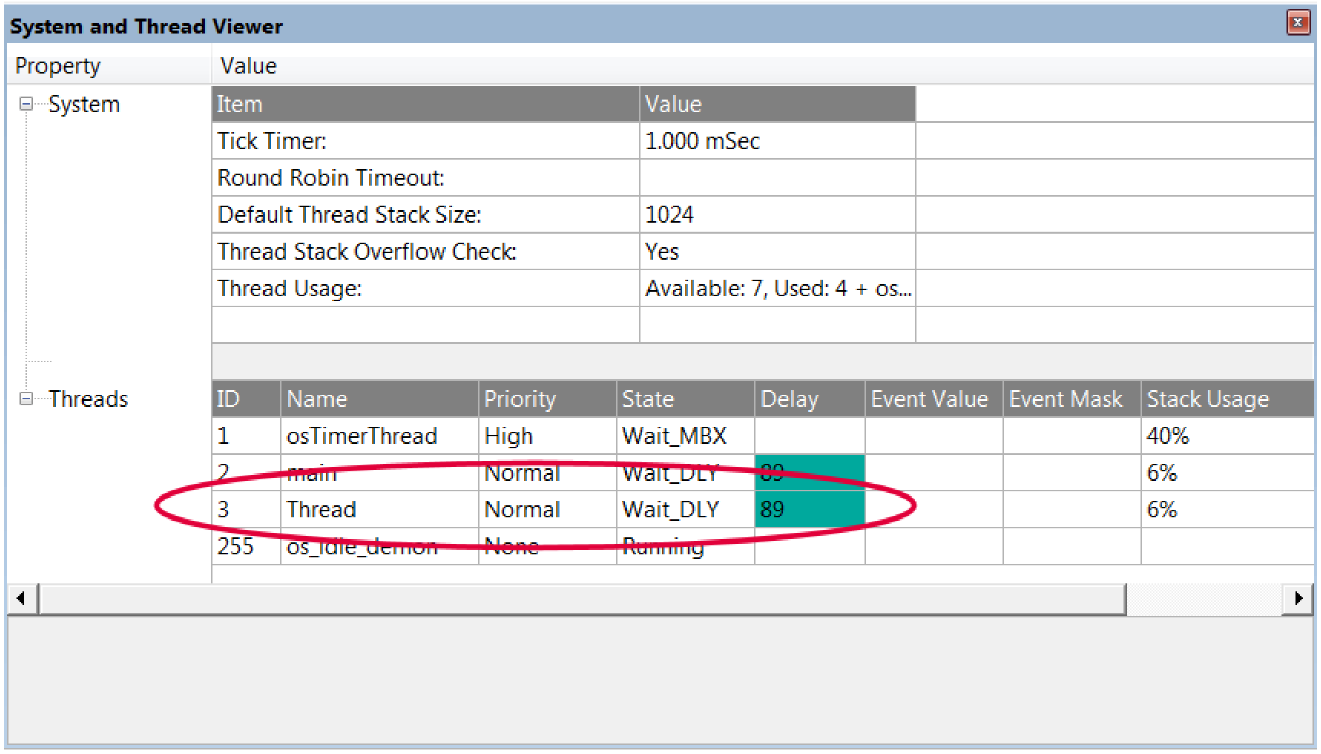
1. Right click in SensorThread.c near line and select Insert '#include file'. Select Board_LED.h to add it.
2. In beginning of the While(1) loop in SensorThread.c, add this code:

```
osDelay(500);
LED_Off(2);       //blue LED
osDelay(500);
LED_On(2);
```

1. Click on File/Save All or .
2. Click on Build or F7 to compile the program. Enter Debug mode by clicking on the Debug icon .
3. Click on the RUN icon. The red and blue LEDs will blink alternatively. If not, check your steps for errors.

4. Select Debug/OS Support and select System and Thread Viewer to verify that the Thread starts properly:

This window shows the Threads and their State. Note the program spends most of its time in the idle demon.



System and Thread Viewer

Property	Value
System	

Item	Value
Tick Timer:	1.000 mSec
Round Robin Timeout:	
Default Thread Stack Size:	1024
Thread Stack Overflow Check:	Yes
Thread Usage:	Available: 7, Used: 4 + os...

Threads

ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
1	osTimerThread	High	Wait_MBX				40%
2	main	Normal	Wait_DLY	89			6%
3	Thread	Normal	Wait_DLY	89			6%
255	os_idle_demon	None	Running				

5. Stop the program with the STOP icon and exit Debug mode .

What we have so far:

1. A program running RTX RTOS with two threads: main and Thread as shown in the screen above. In addition is the User Timer thread and os_idle_demon.
2. The thread main (main() is always a thread by default) blinks the red LED.
3. The thread Thread blinks the blue LED.

Interface the FOSX8500 MEMS Sensor

Add Accelerometer and Magnetometer support to SensorThread.c:

1. Open the Manage Run-Time Environment window:
2. Select these Components:
   Board Support::Accelerometer(API)::Accelerometer
   Board Support::Magnetometer(API)::Magnetometer
3. There will be some orange blocks. Click on the RESOLVE button to automatically select the required files.
4. Click on OK to close the Run-Time Environment window.
5. In SensorThread.c, right click near line 4 and select Insert "# include file".
6. Select Board_Accelerometer.h and this file will be inserted in SensorThread.c.
7. Similarly, select Board_Magnetometer.h near line 5.
8. In SensorThread.c, create two global variables of these structures to store the MEMS output values near line 7:

```
ACCELEROMETER_STATE g_accel;
MAGNETOMETER_STATE g_magneto;
```

9. At the beginning of the thread Thread and before the while(1) loop, add these lines near line 27:

```
Accelerometer_Initialize();
Magnetometer_Initialize();
```

10. In the while(1) loop in the thread Thread near line 30, add these lines:

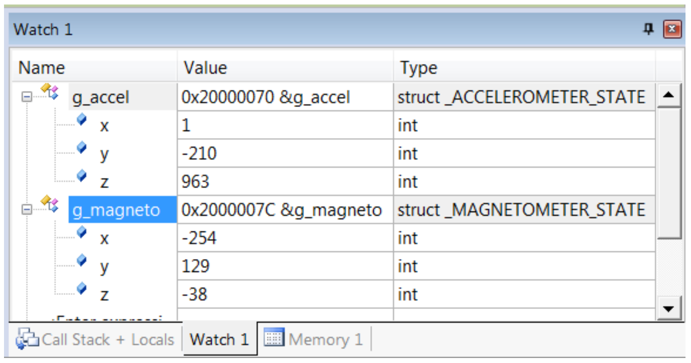
```
Accelerometer _GetState (&g_accel);
Magnetometer_GetState (&g_magneto);
```

11. In the Project window under the Device heading, double click on hardware_init.c to open it.
12. In hardware_init.c, at the end of the function hardware_init() in the file hardware_init.c near line 80, add this line: This must be the last source line in this function , just before the last "}".

```
configure_i2c_pins(0);
```

Modify I2C settings:

13. In the **Device** heading, open RTE_Device.h and select the Configuration Wizard tab
14. Enable **I2C0** and set **I2C0_SDA** to **PTE25** and **I2C0_SCL** to **PTE24**.
15. Click on File/Save All or .
16. Click on Build or F7 to compile the program. Enter Debug mode by clicking on the Debug icon .
17. Click on the RUN icon.
18. In SensorThread.c, Right click on the variable g_magneto and select Add 'g_magneto to… and select Watch 1
19. Repeat for g_accel. The Watch window will open and display these two structures.
20. Right click on g_accel and unselect Hexadecimal Display. Do the same for g_magneto.
21. Expand the structures as shown below:
22. Move the board and they will update in real-time as shown below:
23. This concludes the example of creating a small program that displays the MEMS data using RTX as the RTOS.
24. When you have finished, stop the program .
25. Exit Debug mode .



Name	Value	Type
g_accel	0x20000070 &g_accel	struct _ACCELEROMETER_STATE
x	1	int
y	-210	int
z	963	int
g_magneto	0x2000007C &g_magneto	struct _MAGNETOMETER_STATE
x	-254	int
y	129	int
z	-38	int

Integrate the HTTP Server

Add Network Middleware and CMSIS Driver support:

1. Open the Manage Run-Time Environment window:
2. Select these Components *in this order*:

   Network::Socket::UDP Network::Service::Web Server Compact
   Set **Network::Interface::ETH** to 1.
   CMSIS Driver::Ethernet MAC(API)::Ethernet MAC
   CMSIS Driver::Ethernet PHY(API)::KSZ8081RNA
3. There will be orange blocks. Click the Resolve button to assign the appropriate additional files.
4. Click on OK to close this window.
5. In main.c, right click near line 4, select insert '#include file' and then select rl_net.h.
6. In main.c, add this line right after Init_Thread(); net_initialize();
7. In the while(1) loop, add these two lines: net_main(); and osThreadYield();
8. Delete or comment out the 4 lines for the LED and the osDelay. Your main.c will look like this:

```
int main (void) {
  osKernelInitialize ();      // initialize CMSIS-RTOS
  hardware_init();
  LED_Initialize();
  Init_Thread();
  net_initialize();
  osKernelStart ();            // start thread execution
  while(1) {
  net_main();
  osThreadYield();
  }
}
```

Modify hardware_init.c:

9. In the file hardware_init.c, in the function hardware_init(),just after uint8_t I; near line 67, add this line:

```
MPU->CESR &= ~1;
```

10. In the file hardware_init.c, at the end of the function hardware_init(), near line 81 add this line:
    Do not include it in the for loop.

```
configure_enet_pins(0);
```

Modify Ethernet settings:

11. In the Project window in the Network heading, open Net_Config.c and select the Configuration Wizard tab.
12. Click Expand All and change the NET_HOST_NAME to "frdm"
13. In the Device heading, open RTE_Device.h and select the Configuration Wizard tab
14. Enable ENET and set the Mode to RMII and the Clock Source to OSCERCLK.
15. Open Net_Config_HTTP_Server.h and select the Configuration Wizard tab.
16. Unselect EnableUserAuthentication. This is how you require a password or not.
17. Click on File/Save All or .
18. On the next page we will test what you have so far to ensure everything is working correctly.

Testing your project:

5. In main.c near line 12, add this code: This will turn on the green LED when an IP address is granted by DHCP.

```
void dhcp_client_notify(uint32_t if_num, dhcpClientOption opt, const uint8_t *val, uint32_t len) {
  if (opt == dhcpClientIPaddress) {
              LED_Off(0);     //green LED
              LED_On(0);
  }
}
```

19. Connect the K64F board to a Ethernet network with a DHCP server. Almost any router will provide this.
20. Click on File/Save All or .
21. Click on Build or F7 to compile the program. There will be no errors or warnings.
22. Enter Debug mode by clicking on the Debug icon .
23. Click on the RUN icon.
24. In a few seconds the green LED will illuminate indicating it has acquired an IP address from the DHCP server. The function above `dhcp_client_notify` has successfully run and turned the green LED on.
25. If you do not see this indication, check back looking for mistakes. It is easy to miss a step. Check carefully.
26. You can ping the Freedom K64F with this command: ping frdm
27.

What we have so far:

1. We have configured the K64F to run an http server.
2. We have demonstrated it has acquired an IP address by turning on the green LED.

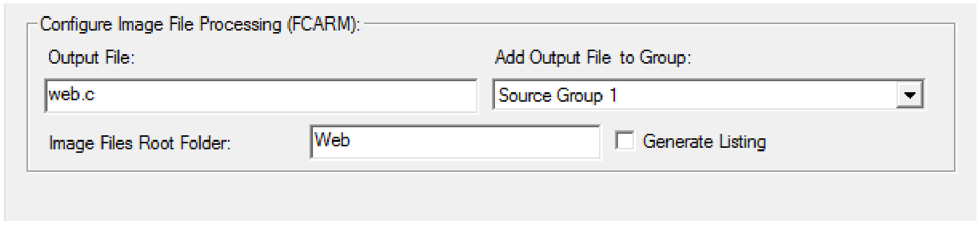
Interface to CGI

Add the Group Web and image file loc.cgi:

1. In the project window, right click on Target 1 and select Add Group…
2. A group called New Group is added. Change its name to "Web" (without the quotes).
3. Right click on Web and select Add New item to Group 'Web". Select Image File (.*) and name it loc.cgx.
4. In the Location box, add **\Web** to the default folder displayed. ***Important step!***
5. Click Add and loc.cgx will be added to the Group Web and also to the folder Web. ***Important step!***
6. Note a message appears that FCARM needs to be configured. Close this window. We will do this shortly.
7. Right click on loc.cgx and select Options for File 'loc.cgi'…
8. Unselect Image File Compression and click on OK.
9. **Note:** Turn off Image File Compression for all html and js type files. This can prevent issues with commented JavaScript sections.

Configure FCARM:

1. Using Microsoft Explorer, navigate to where your project is located. This is stated in the upper header of µVision.
2. Create a new folder called Web. This is where FCARM will create and store web.c.
3. Select Target Options or ALT-F7. Select the Utilities tab. Fill in the fields as shown here:
4. Click OK to return to µVision's main menu.


Configure Image File Processing (FCARM):
Output File: web.c
Add Output File to Group: Source Group 1
Image Files Root Folder: Web
☐ Generate Listing

Configure loc.cgi:

1. Double click on loc.cgi to open it. It will be blank. Add these two lines:

```
c L
.
```

The second line is a simple period. You can put a carriage return (Enter) at the end.

2. Click on File/Save All or .

Explanation of loc.cgi:

File	Details
loc.cgx	CGI file that is used as JSON data provider for index.html and graph.html to retrieve sensor data from the target on demand.

loc.cgx

The loc.cgx is a very simple file that should return a string to the calling client. The string contains the latest MEMS sensor data in a JSON representation. This string will look like this:

```
{"x":"630", "y":"-280", "z":"-134", "ax":"17", "ay":"1", "az":"977"}
```

Explanation of the contents of loc.cgx:

c indicates the *cgi_script(…)* callback should be called.

L will be passed as environment parameter to the callback.

. indicates the end of a cgi script to the line based interpreter.

HTTP_Server_CGI.c

1. Right click on Source Group 1 and select Add New Item to Group 'Source Group 1'…
2. Select User Code Template and under the Network header, select HTTP Server CGI. Click on Add.
3. HTTP_Server_CGI.c is added to your project and also opened for editing.
4. On line 13 just after the last #include, right click and select Insert '#include file'. Select "Board_LED.h"
5. Similarly insert #include "Board_Accelerometer.h" and #include "Board_Magnetometer.h"
6. Add two extern references as shown after the #include lines:
7. `extern MAGNETOMETER_STATE  g_magneto;` and `extern ACCELLERATOR_STATE g_accel;`

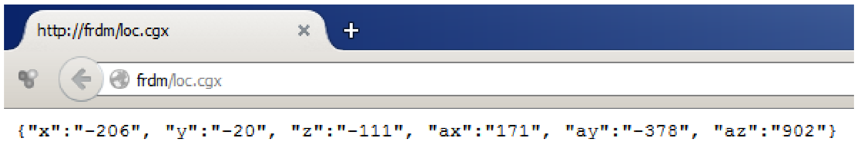
```
13. #include "Board_LED.h"
14. #include "Board_Magnetometer.h"         // ::Board Support:Magnetometer
15. #include "Board_Accelerometer.h"
16.
17. extern MAGNETOMETER_STATE  g_magneto;
18. extern ACCELEROMETER_STATE g_accel;
```

When L is passed from the script interpreter line prepare a JSON formatted string using sprint.

1. In HTTP_Server_CGI.c near line 121 is the function cgi_script as shown below:
2. Modify the template to match this one shown below:
3. Near 140 is the function cgx_content_type (void) as shown: (location depends on the source you added)
4. Modify the return (NULL) statement to match that shown below on line 151: return ("application/json");

```
121.// Generate dynamic web data from a script line.
122.uint32_t cgi_script (const char *env, char *buf, uint32_t buflen, uint32_t *pcgi) {
123.  uint32_t len = 0;
124.
125.    LED_On(2);                   //green LED
126.    switch (env[0]) {
127.        case 'L' :
128.            sprintf(buf, "{\"x\":\"%d\", \"y\":\"%d\",    \
129.                                           \"z\":\"%d\", \"ax\":\"%d\",   \
130.                                           \"ay\":\"%d\", \"az\":\"%d\"}" \
131.          ,g_magneto.x, g_magneto.y, g_magneto.z, g_accel.x, g_accel.y, g_accel.z);
132.
133.          len = strlen(buf);
134.      break;
135.    }
136.    LED_Off(2);
137.
138.  return (len);
139.}
140.


149.// Override default Content-Type for CGX script files.
150.const char *cgx_content_type (void) {
151.    return ("application/json");
152.}
```

1. Click on File/Save All or . Connect the FRDM K64F board to an Ethernet system with a DHCP capability.
2. Build Enter Debug mode: Click on the RUN icon. The green LED indicates a DHCP IP address.
3. You can access this script by pointing a web browser over the network to http://frdm/loc.cgx as shown below:
4. Press Refresh on your browser and the variables will update.
5. **Note:** some browsers might not be reliable when refreshing this.



```
{"x":"-206", "y":"-20", "z":"-111", "ax":"171", "ay":"-378", "az":"902"}
```

6. Stop and exit Debug mode .

Create the web application

File	Details
index.html	Page with embedded JavaScript that displays the raw values of the Sensor as text.
graph.html	Page with embedded JavaScript that graphically displays senor data on a real-time graph.
loc.cgx	CGI file that is used as JSON data provider for index.html and graph.html to retrieve sensor data from the target on demand.
jquery.min.js	JQuery is one of the most standard JavaScript libraries. Simplifies the handling of JSON and also HTML object manipulation is this tutorial.
smoothie.js	A JavaScript charting library for streamed data. Very compact and easy to use. Uses MIT license.

The following section requires basic knowledge of HTML and the JavaScript HTML DOM.

HTML Introduction: http://www.w3schools.com/htmL/html_intro.asp

HTML DOM: http://www.w3schools.com/js/js_htmldom.asp

Create index.html

1. In the Project window, right click on the Web heading. Select Add New Item to Group … and select Image File.
2. Name this file index.html and add the directory \Web to the default folder in the Location: box.
3. Click Add and index.html will be added to group Web and is also created in the .\Web folder.
4. Right click on index.html and select Options for Target. Unselect "Image File Compression".
5. Double click on index.html to open it. Enter the code as follows. You can also obtain a file to copy and paste.
6. In the head tag of the file include jquery.min.js. The load order of JavaScript files is important. Since our script block depends on jQuery it should be included first. Here are the first three lines in index.html:

```
1.  <html>
2.    <head>
3.      <script language="javascript" type="text/javascript" src="jquery.min.js.gz"></script>
```

Next open another <script> section. This will contain the JavaScript code for the dynamic update of sensor data.
If you are using jQuery the first time it is interesting to know that $ is a short form of the *jQuery()* identifier.

$(function() {…}*);* is the jQuery framework function that is called when the complete html document has been loaded and rendered in the browser.
function update() uses a timer to call itself every 300 ms. This creates a time structure for the getData() function.

*Function getData()*calls *$.getJSON,* which retrieves a JSON data string from loc.cgx and parses it into the variable *loc.*
Using the ID of the input boxes defined in our HTML structure, $('#ID').val() the lines 9 to 14 change the values according to the data retrieved from the getJSON call.
getJSON already made a type conversion and the variables are stored as integers. This would allow additional math easily if required.

```
4.      <script type="text/javascript">
5.        $(function() {
6.        var json_sample = [];
7.          function getData() {
8.            $.getJSON("loc.cgx", function(loc) { json_sample = loc; });
9.            $('#val_x').val(json_sample.x);
10.           $('#val_y').val(json_sample.y);
11.           $('#val_z').val(json_sample.z);
12.           $('#val_ax').val(json_sample.ax);
13.           $('#val_ay').val(json_sample.ay);
14.           $('#val_az').val(json_sample.az);
15.         }
16.         function update() {
17.           getData();
18.           setTimeout(update, 300);
19.         }
20.         update();
21.       });
22.     </script>
```

The following section contains the stylesheet for our document. It simply changes the font family used for the complete document. It can easily be extended with additional CSS rules. (http://www.w3.org/Style/CSS/)

```
    <style>
    html *
   {
      font-family: Verdana  !important;
   }
    </style>
  </head>
```

The body contains the actual rendered items. Notice the ids assigned to the <input…> boxes. Those are important as they are used to address the input box from within our JavaScript.

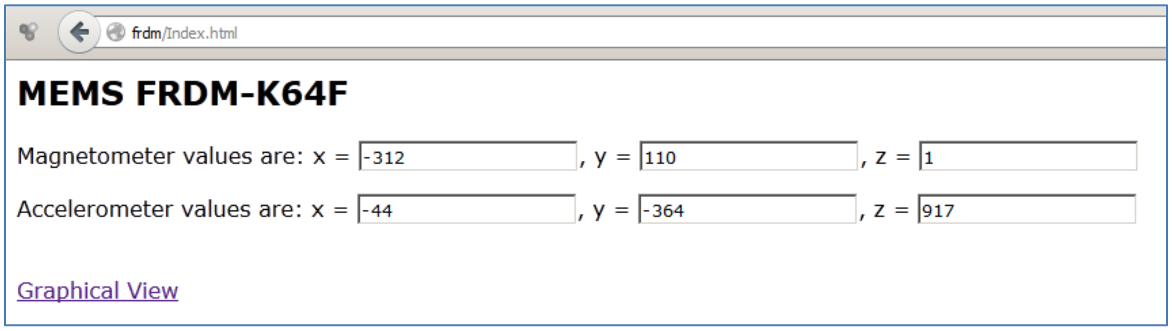
```
  <body>
    <h2>MEMS FRDM-K64F</h2>
    <p>Magnetometer values are:
    x = <input id="val_x" type="text" value="">,
    y = <input id="val_y" type="text" value="">,
    z = <input id="val_z" type="text" value=""></p>
    <p>Accelerometer values are:
    x = <input id="val_ax" type="text" value="">,
    y = <input id="val_ay" type="text" value="">,
    z = <input id="val_az" type="text" value=""></p>
    </br>
    <a href="graph.html" target=_blank>Graphical View</a>

  </body>
</html>
```

This completes the creation of index.html.

1. Click on File/Save All or .
2. Copy the file jquery.min.js.gz to your .\Web folder. This file will be supplied to you.
3. In the Project window, right click on Web and select Add Existing Files to Group 'Web'. Set Files of Type to *.*.
4. Navigate to your \Web folder and select the file jquery.min.js.gz. Click on Add and then Close.
5. Right click on index.html and select Options for File 'index.html'. Unselect Image File Compression. Click OK.
6. Repeat for jquery.min.js.gz.

7. Build Enter Debug mode: Click on the RUN icon. If asked to update web.c, click Yes.
8. The Green LED will light in a few second indicating the acquisition of an IP address assigned by DHCP.
9. Open a web browser on the network and enter this URL: http://frdm/index.html. Just frdm can work sometimes.
10. This window below opens and displays the MEMS variables. These are the same variables displayed in the Watch window. These values change as you move the KL64 board.

7. When you are finished, Stop and exit Debug mode .


frdm/Index.html

MEMS FRDM-K64F

Magnetometer values are: x = -312, y = 110, z = 1

Accelerometer values are: x = -44, y = -364, z = 917

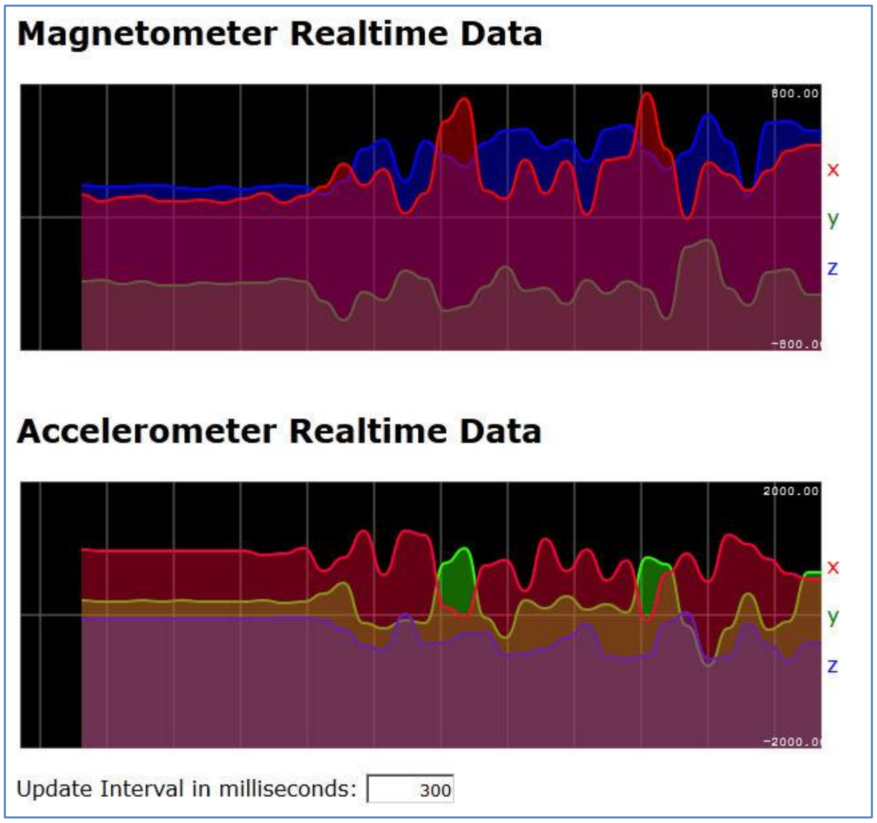
Graphical View

Create graph.html

While it is interesting to watch the values of the magnetometer and accelerometer change as you move the board, it is hard to tell from the display of numbers any trends or historical data. graph.html uses the JavaScript smoothie.js.gz and jquery.min.js.gz to display these variables in a graphical format. You can examine graph.html in µVision to see how it works.

1. Copy the supplied files graph.html and smoothie.js.gz to your \Web folder. These are provided to you.
2. Right click on graph.html and select Options for File graph.html'. Unselect Image File Compression. Click OK.
3. Repeat for smoothie.js.gz.
4. Click on File/Save All or .
5. Build Enter Debug mode: Click on the RUN icon. If asked to reload web.c, click Yes.
6. The green LED of the K64F board will illuminate indicating a DHCP IP address has been acquired.
7. In http://frdm/index.html, the numbers soon will start to change again.
8. Click on the button: Graphical View
9. This display will result: Moving the board will cause the data to change as shown here:

NOTE: If you use any Keil ULINK instead of the CMSIS-DAP you can use µVision's Logic Analyzer (LA) to display up to four variables. They will update in real-time. An Ethernet connection is not needed for the LA.


Magnetometer Realtime Data
800.00
-800.0
x
y
z
Accelerometer Realtime Data
2000.00
-2000.0
x
y
z
Update Interval in milliseconds:
300

In the JavaScript of both, index.htm and graph.html **getJSON()** is used to retrieve JSON data from the target system. getJSON is simple to use, but has some shortcomings. One of them is the absence of a timeout when the target system does not respond. For a better control over the JSON communication jQuery's AJAX API is advised (http://api.jquery.com/jQuery.ajax/).

Appendix A: Optimize the JavaScript ROM Usage

The JavaScript files have been [provided to you compressed using gzip. This is done to save program RO space. JavaScript files can be provided uncompressed so you can easily read them during development work and also compressed for final production.

All modern browsers support automatic decompression of zipped resources. Depending on the used libraries this can drastically reduce the required ROM space and loading time of pages. Using the project described in this application note, these are the size of the executable file: You can clearly see the RO-data in these cases is less than half of uncompressed.

Without gzip-compression:

Program Size: Code=43636 RO-data=**117624** RW-data=244 ZI-data=28684

With gzip-compression:

Program Size: Code=43636 RO-data=**45320** RW-data=244 ZI-data=28684

How to use gzip Compression in your project:

Here are the steps needed to compress your JavaScript files using gzip. You do not have to do this in the application note since we already did this for you. You can un-gzip them to look inside if you like.

1. Download gzip Binaries from http://gnuwin32.sourceforge.net/packages/gzip.htm
2. Unzip the \bin\ folder to your projects web folder.
3. Zip jquery and smoothie libraries with the commands:
4. .\bin\gzip.exe -c jquery.min.js > jquery.min.js.gz
5. .\bin\gzip.exe -c scmoothie.js > smoothie.js.gz
6. Patch the headers of html files that refer to the JavaScript libraries, like:

```
<html>
  <head>
    <script language="javascript" type="text/javascript" src="jquery.min.js.gz"></script>
```

Document Resources

Books:

- ***Free!*** **Getting Started with MDK 5:** www.keil.com/mdk5/.
- There is a good selection of books available on ARM processors. A good list of books on ARM processors is found at: www.arm.com/support/resources/arm-books/index.php
- µVision contains a window titled Books. Many documents including data sheets are located there.
- **A list of resources is located at:** www.arm.com/products/processors/cortex-m/index.php
  Click on the Resources tab. Or select the Cortex-M processor you want in the Processor panel on the left.
- Or search for the Cortex-M processor you want on www.arm.com.
- The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
- The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
- Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano.

Application Notes: **www.keil.com/appnotes**

- ***NEW!*** ARM Compiler Qualification Kit: www.keil.com/safety
- Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
- CAN Primer using NXP LPC1700: www.keil.com/appnotes/files/apnt_247.pdf
- CAN Primer using the STM32F Discovery Kit www.keil.com/appnotes/docs/apnt_236.asp
- Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
- Porting an mbed project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
- MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
- Using µVision with CodeSourcery GNU www.keil.com/appnotes/docs/apnt_199.asp
- RTX CMSIS-RTX Ports in MDK 5 Eval Version: C:\Keil_v5\ARM\Pack\ARM\CMSIS\
- Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
- Lazy Stacking on the Cortex-M4 www.arm.com and search for DAI0298A
- ***NEW!*** Cortex-M Processors for Beginners: http://community.arm.com/docs/DOC-8587
- Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
- Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp
- Migrating fom Cortex-M4 to Cortex-M4 Processors: www.keil.com/appnotes/docs/apnt_270.asp

Useful ARM Websites

- CMSIS Standards: www.arm.com/cmsis/ and www.keil.com/cmsis/
- ARM and Keil Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content
- ARM University Program**:** www.arm.com/university.
- ARM Accredited Engineer Program: www.arm.com/aae
- mbed™: http://mbed.org

Keil Direct Sales In USA: sales.us@keil.com or 800-348-8051. **Outside the US:** sales.intl@keil.com

Keil Distributors: See www.keil.com/distis/ **DS-5 Direct Sales Worldwide:** orders@arm.com

Keil Technical Support in USA: support.us@keil.com or 800-348-8051. Outside the US: support.intl@keil.com.