Hermes™ IoT Development Platform Developer's Guide

1 Introduction

The Hermes™ IoT Development Platform allows you to develop your own applications using RFMicron's passive wireless sensors based on the Magnus®-S chip. Hermes allows you to control the AMS Radon Reader to read sensor tags and display the results to the user on a display or send the data to a remote user through either a UART, TCP, CAN, I2C or SPI interface.

2 The Developer's Tool

Everything you need to develop a custom application for Hermes is included in the software image pre-installed in the kit and also available for download from www.github.com/RFMicron/Hermes. The image includes:

- The Debian version of the Linux Operating System which includes the Linux Socket Interface
- The Qt Toolkit
- The QWT library
- The AMS Radon Reader Application Programming Interface (API)
- The GNU Compiler Collection (GCC)
- The GDB debugger

You can use the Integrated Development Environment or the programming editor of your choice available for Linux.

To begin development, you can learn about the Hermes Demo application which is described in this document.

3 Hermes Demo Software Architecture

The Hermes Demo application uses the Model-View-Controller (MVC) design pattern. There are two views, the Graphical User Interface (GUI) view and the Remote User Interface (RUI) view. These two views translate user requests for setting the AMS Radon Reader settings and controlling its operations. In response to screen selections or commands received through one of the remote interfaces (i.e. UART, TCP, CAN, I2C, or SPI), the Controller carries out those requests by utilizing the operations available from the Model. The Model in turn maintains the state of the Reader which is its settings and results

of reads and controls the Reader. The Model makes use of the Reader API, which makes available a set of commands to set the Reader's settings and read tags among other operations. The Model signals the GUI when the settings of the Reader have been changed or when there are results of reads available. The GUI subsequently requests that data from the Model to display it to the user. The RUI signals the GUI when it wants to display the commands it has received through one of the remote interfaces and the responses it has sent back. The RUI gets commands to search for tags, measure tags, send back Reader settings, and set Reader settings. Please see Figure 1 for a depiction of the modules.

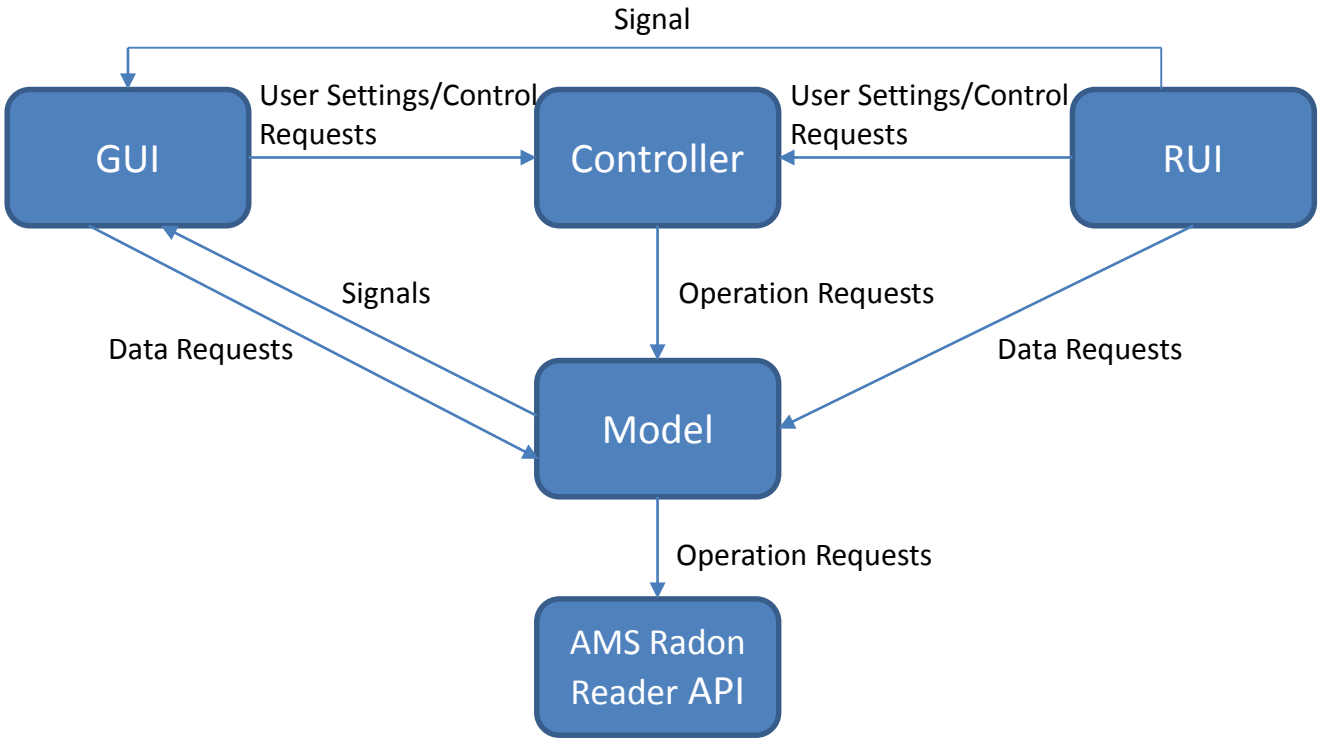

Figure 1. Demo Application Software Architecture

When the user requests to start the Temperature or Moisture demonstrations available on the application, which use the Reader to perform continuous reads and is an intensive operation, the application spawns a Chart thread to offload that operation from the GUI thread. This allows the GUI thread to be available to handle user selections on the screen so that the screen does not become unresponsive. When one of the remote interfaces is started on the application, the interface selected is continuously monitored for commands and once a command is received, that command is processed and a response generated and sent back. This is an intensive operation as well. Therefore, the application spawns a

RUI thread to offload this operation from the GUI thread to again prevent the screen from becoming unresponsive. Please see Figure 2 for a depiction of these threads.

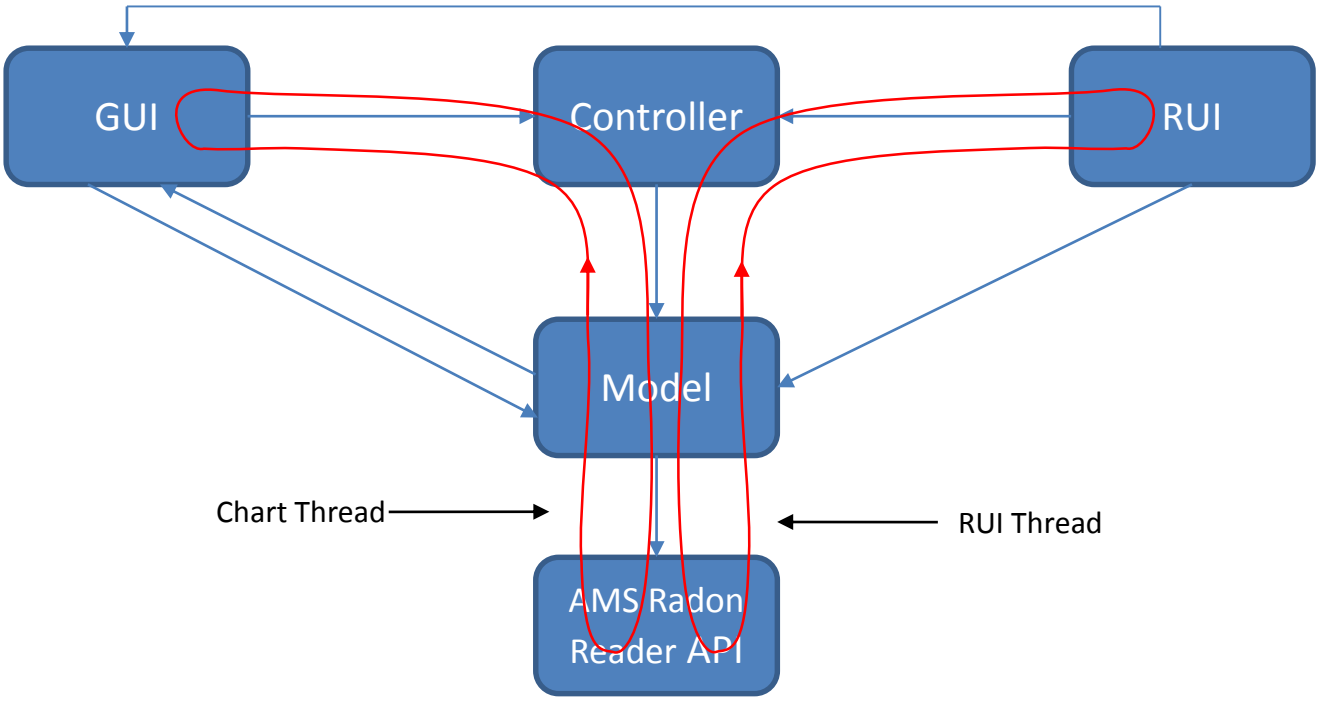

Figure 2. Demo Application Worker Threads

4 Hermes Application Programming Interface

The Hermes Demo application was developed in C++/C and utilizes the Qt Toolkit along with the QWT graphical library for the Temperature and Moisture demonstrations. The application makes use of the Linux Socket interface to communicate with the Reader and to communicate with external devices through one of the remote interfaces available. We have developed our own API for the AMS Radon Reader, which makes available a set of commands to perform the operations that demonstrate the capabilities of RFMicron's passive sensors and more. The API makes use of the commands made available by the AMS Radon Reader firmware, which we also modified for the API. Please see Figure 3 below for a depiction of the APIs available with the Hermes platform.

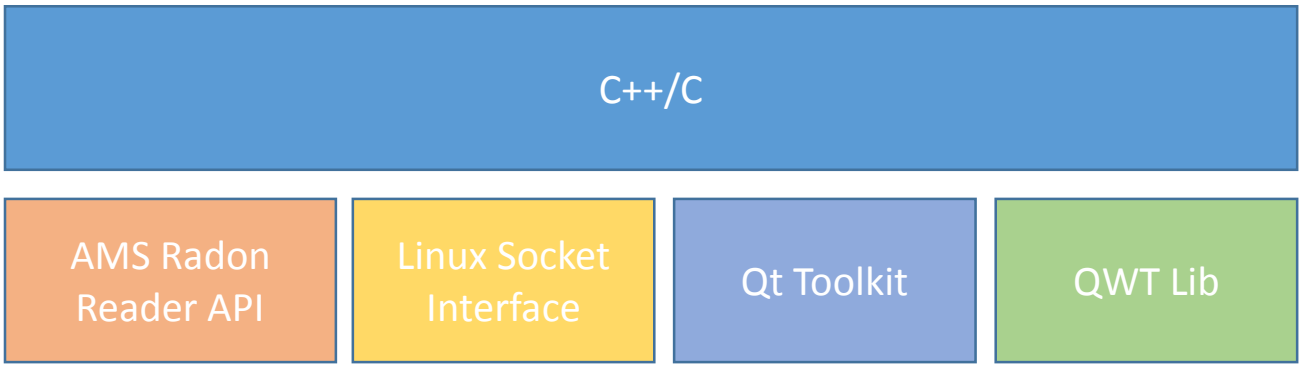

Figure 3. Hermes Platform APIs

5 The Modules of the Demo Application

Now that we have described the Software Architecture of the Demo application and the APIs it uses, we can move to describing its modules in more detail. We will focus on the most important operations performed by those modules.

5.1 GUI

The GUI module is depicted in the figure below. The module is defined in the *gui_view* files and it utilizes the *hermes*, *chart*, *chart_thread* and *configdialog* files to carry out its operations. The *configdialog* files in turn utilize the *pages* files to carry out its operations. The most important operations these files contain are as follows:

- **gui_view:** Defines the resources needed for the GUI, most importantly the hermes wizard.
- **hermes:** Defines the Main, Temperature, and Moisture windows (or pages) of the GUI, which includes all the controls and indicators for each window.
- **chart:** Defines both the Temperature and the Moisture charts.
- **chart_thread:** Defines the Chart thread that is spawned to offload the intense process of collecting temperature and sensor codes for the Temperature and Moisture charts from the GUI thread.
- **configdialog:** Defines the Details, Settings, Calibration and Help dialogs found in the Temperature and Moisture Demo windows.
- **pages:** Defines the pages of the dialogs mentioned above, which includes all the controls and indicators for each dialog.

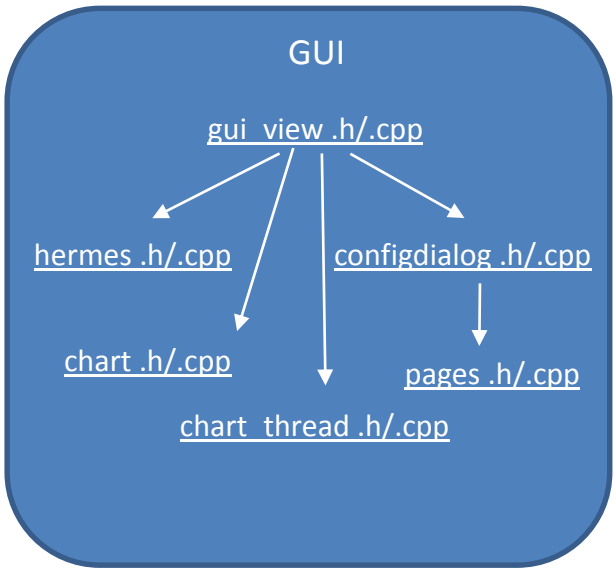

Figure 4. Demo Applications GUI Module

5.2 Controller

The Controller module is depicted in the figure below. The module is defined in the *kit_controller* files and no other file fall within this module. The module does perform some of the most important operations of the system which are as follows:

- **short launchRUI(QString interfaceType):**
  - Starts the RUI listening on interfaceType, which can be UART, TCP, CAN, I2C or SPI.
  - Returns 0 if successful.
- **void stopRUI():**
  - Stops the RUI. Hermes stops listening for commands.
- **RUIThread* getRUIThreadPointer():**
  - Returns the pointer to the RUI Thread. The pointer is used to connect the thread to the console on the GUI so that it can display status messages.
- **void turnReaderOn():**
  - Applies power to the AMS Radon Reader, turning it on.
- **void turnReaderOff():**
  - Removes power from the AMS Radon Reader, turning it off.
- **double measureTempCodeForCalibration():**
  - Returns the current Temp Code for the tag currently selected. Used to calibrate tag.
- **void searchForTempTags():**
  - Performs a search for Temp tags in the field using the Reader. Results are put in the Model's Temp Tags list.

- **void searchForMoistureTags():**
  - Performs a search for Moisture tags in the field using the Reader. Results are put in the Model's Moisture Tags list.
- **void measureTempTags():**
  - Performs a read of the tags's On-Chip RSSI codes and Temp codes. Results are put in the Model's Temp Tag list.
- **void measureMoistureTags():**
  - Performs a read of the tags's On-Chip RSSI codes and Sensor codes. Results are put in the Model's Moisture Tag list.
- **int clearTempTags():**
  - Clears the Temp Tags list and measurement time history along with the Temperature Demo chart. Returns 0 always.
- **int clearMoistTags():**
  - Clears the Moisture Tags list and measurement time history along with the Moisture Demo chart. Returns 0 always.
- **int setMoistLinearFit(bool setting):**
  - Enables the use of the Linear Fit algorithm to produce measurements from raw Sensor, On-Chip RSSI, and Temp code reads. Returns 0 always.
- **int setTempAutoPower(bool setting):**
  - Sets Temp Auto Power on or off. Returns 0 always.
- **int setMoistAutoPower(bool setting):**
  - Sets Moisture Auto Power on or off. Returns 0 always.
- **int setTempMaxPowerLevel(int dBm):**
  - Sets Temp Max Power level. Returns 0 always. dBm can be 18 – 30.
- **int setMoistMaxPowerLevel(int dBm):**
  - Sets Moisture Max Power level. Returns 0 always. dBm can be 18 – 30.
- **int setTempSamplesPerMeasurement(QString s):**
  - Sets the minimum number of samples used to produce a measurement for On-Chip RSSI and Temp code. Returns 0 always. s can be 2, 5, 10, 20.
- **int setMoistSamplesPerMeasurement(QString s):**
  - Sets the minimum number of samples used to produce a measurement for On-Chip RSSI and Moisture code. Returns 0 always. s can be 2, 5, 10, 20.
- **int setTempOnChipRssiTargetMin(int t):**
  - Sets the minimum value for the On-Chip RSSI code for a Temp code read to be valid. Returns 0 always. t can be between 0 and 31. Min < Max.
- **int setTempOnChipRssiTargetMax(int t):**
  - Sets the maximum value for the On-Chip RSSI code for a Temp code read to be valid. Returns 0 always. t can be between 0 and 31. Max > Min.
- **int setMoistOnChipRssiTargetMin(int t):**
  - Sets the minimum value for the On-Chip RSSI code for a Sensor code read to be valid. Returns 0 always. t can be between 0 and 31. Min < Max.
- **int setMoistOnChipRssiTargetMax(int t):**

  - Sets the maximum value for the On-Chip RSSI code for a Sensor code read to be valid. Returns 0 always. t can be between 0 and 31. Max > Min.
- **int setMoistThreshold(int threshold):**
  - Sets the threshold value for the Sensor code read to indicate that the tag is detecting moisture. Returns 0 always.
- **int setMoistAboveThreshold(bool above):**
  - Sets if the Sensor code read has to go above or below the Moisture Threshold to indicate that the tag has detected moisture. Returns 0 always.
- **int setBandRegion(FreqBandEnum band):**
  - Sets the frequency band and tunes the Reader for that band. Band can be FCC, ETSI, PRC, JAPAN, FCC_center, and ETSI_center. Returns 0 if successful.

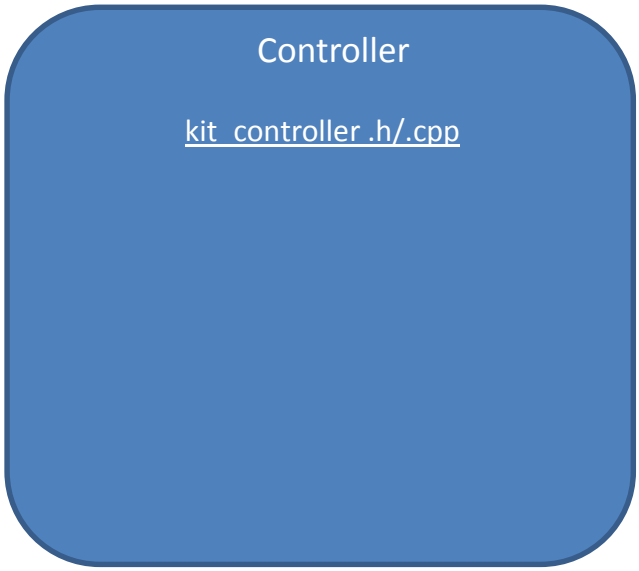

Figure 5. Demo Applications Controller Module

5.3 RUI

The RUI module is depicted in the figure below. The module is defined in the *rui_view* files and it utilizes the *rui_thread* and *interfaces* files to carry out its operations. The *interfaces* files in turn utilize the *can*, *i2c*, *spi*, *tcp_server*, and *uart* files. The most important operations contained in these files are as follows:

- **rui_view:** Defines the resources needed for the RUI, most importantly the interfaces object.

- **rui_thread:** Defines the RUI thread that is spawned to offload the intense process of listening for commands, processing them, and sending back responses from the GUI thread.
- **interfaces:** Defines the generic interface that is used to connect Hermes to an external device either through the UART, TCP, CAN, SPI or I2C interfaces.
- **can:** Defines the CAN Linux socket
- **i2c:** Defines the I2C Linux socket
- **spi:** Defines the SPI Linux socket
- **tcp_server:** Defines the TCP Server Linux socket
- **uart:** Defines the UART Linux socket

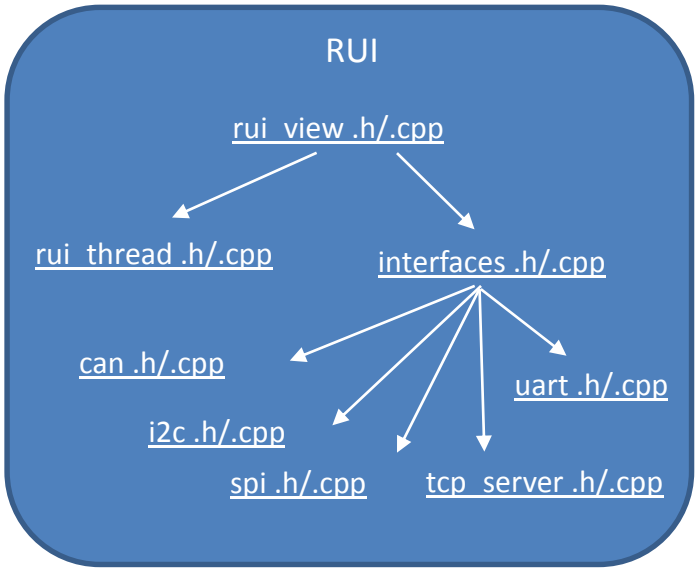

Figure 6. Demo Applications RUI Module

5.4 Model

The Model module is depicted in the figure below. The module is defined in the *kit_model* files and it utilizes the *sensorTag*, *GPIO*, *util*, and *utility* files to carry out its operations. Some of the most important system operations are contained in these files and are as follows:

- **kit_model:** Defines the resources needed for the Model, most importantly the reader object, the sensorTag lists, and the functions available to set the Reader settings and perform reads. The most important operations available in the Model will be described below.

- **sensorTag:** Defines the data structures used to store sensor reads, sensor measurements, and sensor data.
- **GPIO:** Defines the object through which the BeagleBone's GPIOs can be accessed, which is used to turn the Reader on/off and to connect the UART, I2C or SPI ports on the board to the BeagleBone when one of those interfaces is selected.
- **util:** Defines helper functions for GPIO.
- **utility:** Defines helper functions for performing data conversions.

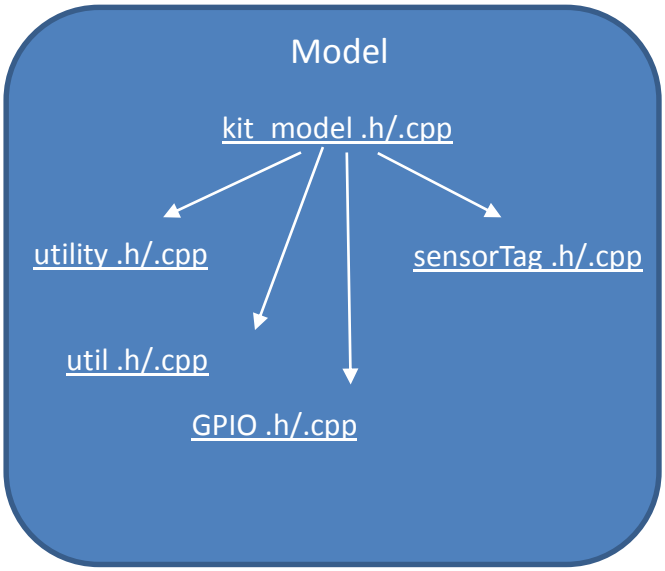

Figure 7. Demo Applications Model Module

Some of the most important operations performed by Hermes are defined in the *kit_model* files and are described below:

- **void turnReaderOn():**
  - Applies power to the AMS Radon Reader, turning it on.
- **void turnReaderOff():**
  - Removes power from the AMS Radon Reader, turning it off.
- **void selectForMeasurement(QString measurementType, QString tagLabel, bool select):**
  - Marks the tag with tagLabel in the measurementType (Temperature or Moisture Demo) in the (Temperature or Moisture) Tag list.
- **int setPower(char value):**
  - Sets the attenuation factor for the AMS Reader chip (AS3993). Value can be between 6 and 19. Returns 0 if successful.

- **int setFrequencyBand(FreqBandEnum band):**
  - Sets the frequency band and tunes the Reader for that band. Band can be FCC, ETSI, PRC, JAPAN, FCC_center, and ETSI_center. Returns 0 if successful.
- **void addTagToList(TagData tag, QString measurementType):**
  - Adds the tag returned by the Reader to the measurementType (Temperature or Moisture) Tag list.
- **float calculateTemperature(QString tagEpc, int minOnChipRssi, int maxOnChipRssi, int minNumberSuccessfulReads):**
  - Returns the temperature in degrees C from the temp code average for a tag.
- **void addSensorReading(TagData tag, QString measurementType):**
  - Adds a tag's raw Sensor, Temp, and On-Chip RSSI code to the measurementType (Temperature or Moisture) Tag list.
- **int findTags(int numInventories, QString measurementType):**
  - Search for tags in the area performing a number of inventories (numInventories) of the type measurementType (Temperature or Moisture). Returns 0 if successful.
- **int searchForTempTags():**
  - Perform a search for Temp tags using the findTags function adjusting the power until the user clicks the Stop button on the Temperature screen. Returns 0 always.
- **int searchForMoistTags():**
  - Perform a search for Moisture tags using the findTags function adjusting the power until the user clicks the Stop button on the Moisture screen. Returns 0 always.
- **int setSelectsForReading():**
  - Sets up the Select commands in the Reader in preparation for reading Temp or Moisture tags. It does not perform the Select commands. The Select commands are performed when the Reader is instructed to perform an inventory. Returns 0 if successful.
- **int writeDataToTag(QString epc, char bankCode, int address, QString dataHexString):**
  - Writes data to the bankCode (0=Reserved, 1=EPC, 2=TID, 3=User) at the address (in Hex), the dataHexString for the tag with epc. Returns 0 if successful.
- **int readTags(int numInventories, QString measurementType):**
  - Perform a number of inventories (numInventories) of the measurementType (Temperature or Moisture) type to read the tags Sensor, Temp and On-Chip RSSI codes. Returns 0 if successful.
- **bool findOptimumPower(int tagIndex, QString measurementType):**
  - Adjust the power level for the tag with tagIndex of the measurementType (Temperature or Moisture) to obtain a valid read. This is executed only if

Auto Power is set to true in the Temperature or Moisture Demo settings. Returns true if the optimum power was found.

- **int autotune(int c):**
  - Instructs the Reader to perform an auto tuning for the frequency c and place the resulting antenna tuning parameters in the tuning table for that frequency for future use during inventories. Returns 0 if successful.
- **int measureTempTags():**
  - Reads Temp tags in the area to obtain their Temp and On-Chip RSSI codes, using auto power if set. Returns 0 if successful.
- **int measureMoistTags():**
  - Reads Moisture tags in the area to obtain their Sensor and On-Chip RSSI codes, using auto power if set. Returns 0 if successful.
- **double measureTempCodeForCalibration():**
  - Returns the Temp code for a tag currently selected to use for calibrating the tag. Returns -1000 if it fails.
- **void updateTempTagsSignal(QList<SensorTag>):**
  - Signals the GUI thread to update the Temperature chart with the list of sensor tags passed.
- **void updateTempTagSelectionsSignal():**
  - Signals the GUI thread to check the Temp Tag list and update the Temperature Demo chart because a tag has been selected or unselected for measurement.
- **void updateMoistTagsSignal(QList<SensorTag>):**
  - Signals the GUI thread to update the Moisture chart with the list of sensor tags passed.
- **void updateMoistTagSelectionsSignal():**
  - Signals the GUI thread to check the Moisture Tag list and update the Moisture Demo chart because a tag has been selected or unselected for measurement.
- **void antennaTuningSignal(int, int):**
  - Signals the GUI thread to update the splash screen or the tuning progress bar of the Temperature or Moisture Other Settings tab with the latest percentage completion of the Set Frequency Band operation.
- **void bandChangedSignal(FreqBandEnum):**
  - Signals the GUI thread to update the Band Region Combo box of the Temperature or Moisture Demo Other Settings tab with the new band selected.

5.5 AMS Radon Reader API

The AMS Radon Reader API module is depicted in the figure below. The module is defined in the *ams_radon_reader* files. The file contains the API for the Reader and the data

structure in which raw tag information returned from the Reader is stored. The API was developed to mirror as much as possible the commands that the AMS Radon Reader Firmware makes available. For some commands encompassing multiple sub-commands, we decided to create separate, individual commands for those sub-commands in our API. We customized the inventoryGen2() firmware command by creating two inventory types to meet our performance and data needs. One inventory type (tidAndFast = 0x03) is used for obtaining Temperature Calibration data and the TID and another (tidAndFast = 0x05) for obtaining Sensor, Temp, and On-Chip RSSI code from our tags. In the API we created our own command (getTagData()) to obtain inventory results (or tag data) that doesn't have a counterpart in the firmware. This commands is described below.

- **short getTagData(vector<TagData> &tags, char &inventoryType, char &inventoryResult, char &numberOfTagsFound):**
  - This command fills a vector of tag data structures with the tag data returned by the Reader firmware. It also returns the inventory type that generated the data, the inventory results (whether the data is valid or not), and the number of tags found. The return value of the function indicates whether the command was executed successfully or not.
  - The TagData data structure or class is in the *ams_radon_reader.h* file and it's simple and self-explanatory and hence, we will not discuss it here.

Other than the changes described above, the Reader API follows the firmware command set closely and we will refer the reader to the AMS Radon Kit's Firmware documentation for a description of those commands.

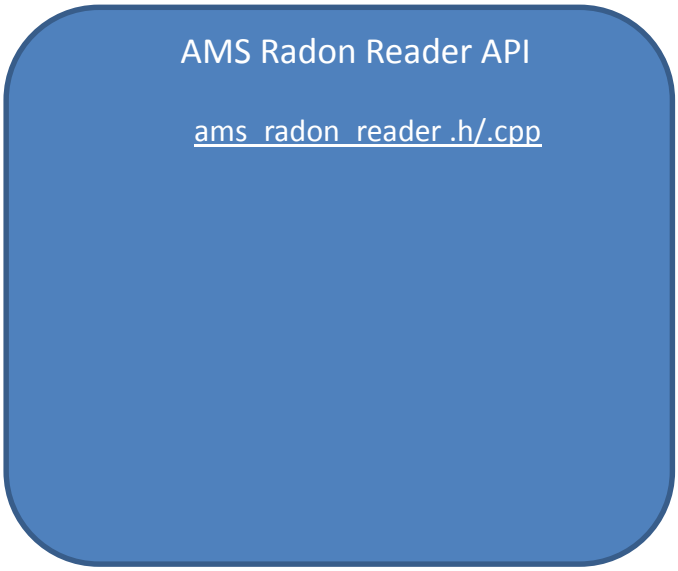

Figure 8. Demo Applications AMS Radon Reader API Module

6 Related Resources

We have discussed mainly the software architecture of the Demo application and the AMS Radon Reader API. For a discussion of the other tools available to the developer, please refer to the following resources:

- **C++/C:** http://www.cplusplus.com/doc/tutorial/
- **Linux Socket Interface:** http://www.linuxhowtos.org/C_C++/socket.htm
- **Qt Toolkit:** http://doc.qt.io/
- **QWT Library:** http://qwt.sourceforge.net/
- **GCC:** https://gcc.gnu.org/
- **GDB:** https://www.gnu.org/software/gdb/

7 Notices

RFMicron, Inc., ("RFMicron") conditionally delivers this document to a single, authorized customer ("Customer"). Neither receipt nor possession hereof confers or transfers any rights in, or grants any license to, the subject matter of any drawings, designs, or technical information contained herein, nor any right to reproduce or disclose any part of the contents hereof, without the prior written consent of RFMicron.

RFMicron reserves the right to make changes, at any time and without notice, to information published in this document, including, without limitation, specifications and product descriptions. This document supersedes and replaces all information delivered prior to the publication hereof. RFMicron makes no representation or warranty, and assumes no liability, with respect to accuracy or use of such information.

Customer is solely responsible for the design and operation of its applications and products using RFMicron products. It is the Customer's sole responsibility to determine whether the RFMicron product is suitable and fit for Customer's applications and products planned, as well as for the planned application and use of Customer's end user(s). RFMicron accepts no liability whatsoever for any assistance provided to Customer at Customer's request with respect to Customer's applications or product designs. Customer is advised to provide appropriate design and operating safeguards to minimize the risks associated with its applications and products.

RFMicron makes no representation or warranty, and assumes no liability, with respect to infringement of patents and/or the rights of third parties, which may result from any assistance provided by RFMicron or from the use of RFMicron products in Customer's applications or products.

RFMicron represents and Customer acknowledges that this product is neither designed nor intended for use in life support appliances, devices, or systems where malfunction can reasonably be expected to result in personal injury.

This product is covered by U.S. patents 7586385 and 8081043; other patents pending. Chameleon® and Magnus® are trademarks of RFMicron.