rACM Quick Start Guide

On-Ramp Wireless Incorporated
10920 Via Frontera, Suite 200
San Diego, CA 92127
U.S.A.

rACM Quick Start Guide

010-0020-00 Rev. D

January 28, 2013

Contents

Figures

Tables

Revision History

Revision	Release Date	Change Description
A	June 21, 2012	Initial release.
B	July 20, 2012	Updated for changes in the naming conventions used for build targets and support scripts. Also included key-value pair for Reliable Host Transfer (RHT) protocol support on the UART Host Interface.
C	September 13, 2012	Updated for the racm_0.7.0 release.
D	January 25, 2013	Updated the rACM Key-Value Pairs table. Added two new appendices providing Operating and OTA Payload examples.

1 Introduction

1.1 Purpose

This document describes the necessary steps to build, download, and test the reference Application Communication Module (rACM) software executing on the On-Ramp Wireless' battery-powered reference host application. It is a reference for external partners in the development of a sensor application on the reference host platform using On-Ramp Total Reach wireless technology.

By observing the rACM application as it joins the On-Ramp Total Reach Network and generates user-payload for various input stimuli, the end-user can better understand the battery-powered software architecture and implementation considerations required to integrate and develop third-party sensor applications.

NOTE: On-Ramp Wireless is in the process of updating product/application names. During this process, the following names will be used interchangeably within this document:

Former Product Name	New Product Name
ULP	On-Ramp Total Reach
CIMA	On-Ramp Total View

1.2 Scope

This quick start guide focuses on the build tools and scripts necessary to:

- Download and execute the rACM software application
- Observe the network join process
- Demonstrate the Over-the-Air (OTA) capabilities of the application.

Detailed information on the reference host software architecture, On-Ramp Total Reach Network components (e.g., Access Point, Gateway, back-end components) and the Total Reach air-link itself, are beyond the scope of this document.

1.3 References

1. *rACM Software High Level Design document (013-0008-00)*
   This document discusses in detail the design requirements, software architecture, and software modules executing on the application host.
2. *Reference Application Developer Guide (010-0021-00)*
   This document provides a comprehensive overview of the reference host software design, state machines, APIs, and data types that make up the software architecture.
3. Reference Platform Schematic Prints (409-0038-00 Schematic Prints, Sep 2011)

2 Build Environment

2.1 Supported Tool Chains

The rACM software supports the following build environments:

- GNU Compiler Collection (GCC) tools on Linux platform using Makefile. The cross compiler used on rACM were extracted from Sourcery Codebench source (version 4.6.1). The Makefile looks for the GCC source in the *opt/CodeSourcery/arm-2011.09* path.
- IAR WARM IDE (version 6.30) running natively on a Windows platform.
- IAR ARM Compiler tools using Makefile a Windows platform using Cygwin.

The choice of which of these three build environments is left up to the user. However, for integrators who do not have previous experience with GCC and Linux-based debug capabilities (e.g., OpenOCD), we recommend that the IAR Workbench IDE be used. The rACM project file for the IAR EWARM contains all of the necessary configuration settings to quickly build, download and debug the rACM application via the Segger J-LINK JTAG Emulator.

Regardless of the actual choice of tool chain, the build environment also requires the use of Python to successfully build the rACM project. In addition, the scripts used to send and monitor message exchanges over the MCM Host Interface UART are also Python-based. The following Python sources must be installed on the build platform:

- matplotlib-0.99.1.win32-py2.6.exe
- numpy-1.3.0-win32-superpack-python2.6.exe
- pyserial-2.5.win32.exe
- python-2.6.3.msi
- pywin32-214.win32-py2.6.exe
- wxPython2.8-win32-unicode-2.8.10.1-py26.exe

NOTE: The installers listed above are for 32-bit Windows OS only. If you are developing on a Linux-based platform or 64-bit Windows OS, you must obtain the appropriate installers available on the internet.

2.2 Supported Processor Targets

The rACM supports the following Kinetis Processor targets:

- K20 80-pin LQFP 72MHz package (K20D7)
- K20 80-pin LQFP 100Mhz package (K20DZ10)

The important differentiation (other than the maximum processor speed) that is not obvious is the Silicon die revision (1.x vs. 2.0). The K20DZ10 is an older die version that uses the 1.x die mask and the K20D7 is instantiated using the newer 2.0 die mask. Due to register changes affecting the PLL clock initialization, code compiled for the wrong target silicon die will NOT

execute properly. As a failsafe, if the boot sequence indicates that it is executing on an incompatible target processor (as identified by the Kinetis Silicon ID registers), it will disable watchdog, turn on the status LED, and remain in a software trap until the correct firmware can be downloaded via JTAG.

NOTE: It is recommended that if the end-user ports the reference application software on to a target MCU other than the two listed packages (e.g., a K10 variant), Freescale and On-Ramp Wireless field engineering should be consulted for a compatible target build. On a related note, the Kinetis processor target under consideration to run the rACM firmware MUST support the FlexNVM option in the second bank of flash.

2.3 Project Directory Structure

After the rACM source files have been extracted onto a local machine, the directory structure is organized as follows:

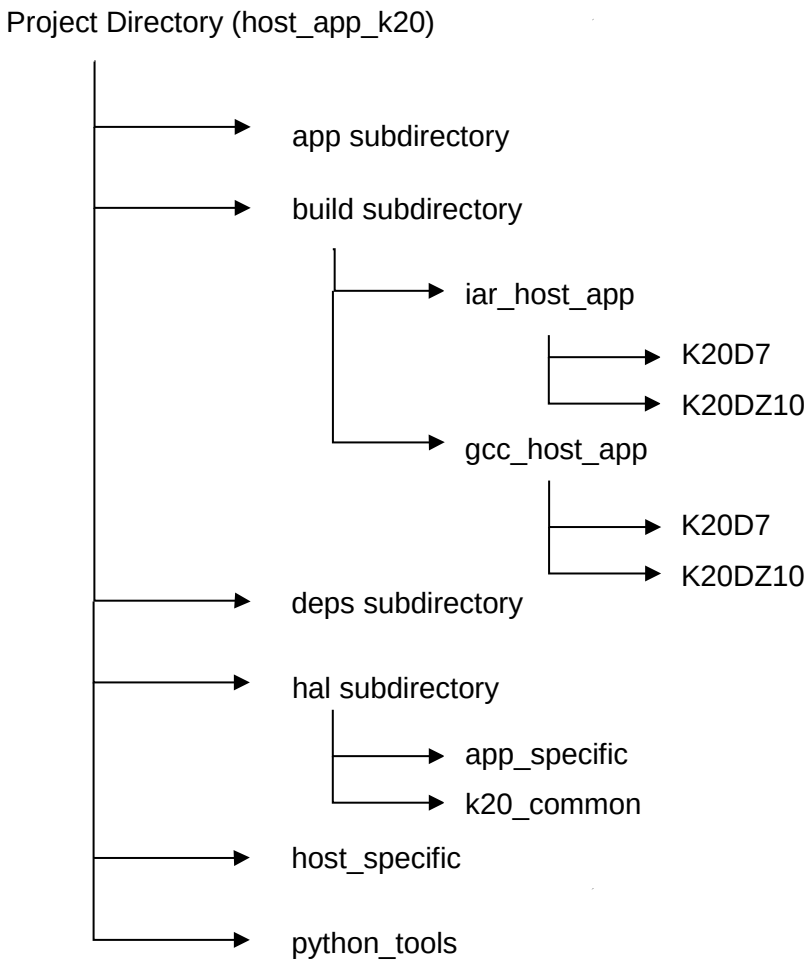


Figure 1. Reference Host Software Directory Structure

The subdirectories are summarized as follows:

- **app** – The set of application source code for the target build which implements the state handlers, message parsing/handling, and the necessary logging that supports the application type (e.g., water meter, pressure sensor, remote monitoring, etc.). These files do not interface directly with the hardware interfaces and, as such, are designed to be portable across different hardware platforms that are designed to perform the same functionality (e.g., a water meter application that can run on either a K10-based platform or a K20-based platform).
- **build** – This contains the project and/or makefiles used to compile and link the desired application and platform hardware abstraction layer. As noted, the reference host software currently supports both the IAR and GCC toolchains. The *gcc_host_app* and *iar_host_app* subdirectories within build also contain the firmware binaries, list files, and dictionary output from the two different build tools (see the next section).
- **deps** – The external dependencies directory contains the shared/common source code used by On-Ramp Total Reach hardware platforms. For the reference host, this includes all of the Host-to-Node DSPI management routines as well as common message definitions. These are typically the software elements that make up the HOST_CMN library (see reference [1])
- **hal –** The Hardware Abstraction Layer (HAL) support functions/drivers that serve as the interface layer between the upper-level application functions and the actual hardware interfaces. These interface files are further divided into two subdirectories – *app_specific* contains interface drivers that are customized and/or unique to the rACM application. *k20_common* contains interface drivers that are common for any application executing on the Kinetis microcontroller unit (MCU) (e.g., flash drivers, board support functionality, etc.).
- **host_specific –** The host specific folder contains message descriptors, both for the C-source and Python scripts, which are specific to the rACM configuration and control commands sent over the Host Interface UART (e.g., Device Status Request, Log Retrieval command, etc.).
- **python_tools** – This subdirectory contains all of the Python scripts necessary to control and configure both the Node and host application through the Host UART. Of particular interest for the Node is the *ulp_node_monitor.py* script which contains all of the necessary GUI tools required to calibrate, configure, and enable the node to successfully track and join the network. For the rACM application, the *host_app_ctrl.py* script is used for configuration and control of the host. In addition, *host_app_logger.py* can also be used to monitor real-time UART logging as it relates to application events. Further details on the rACM control scripts are located in section 6.1).

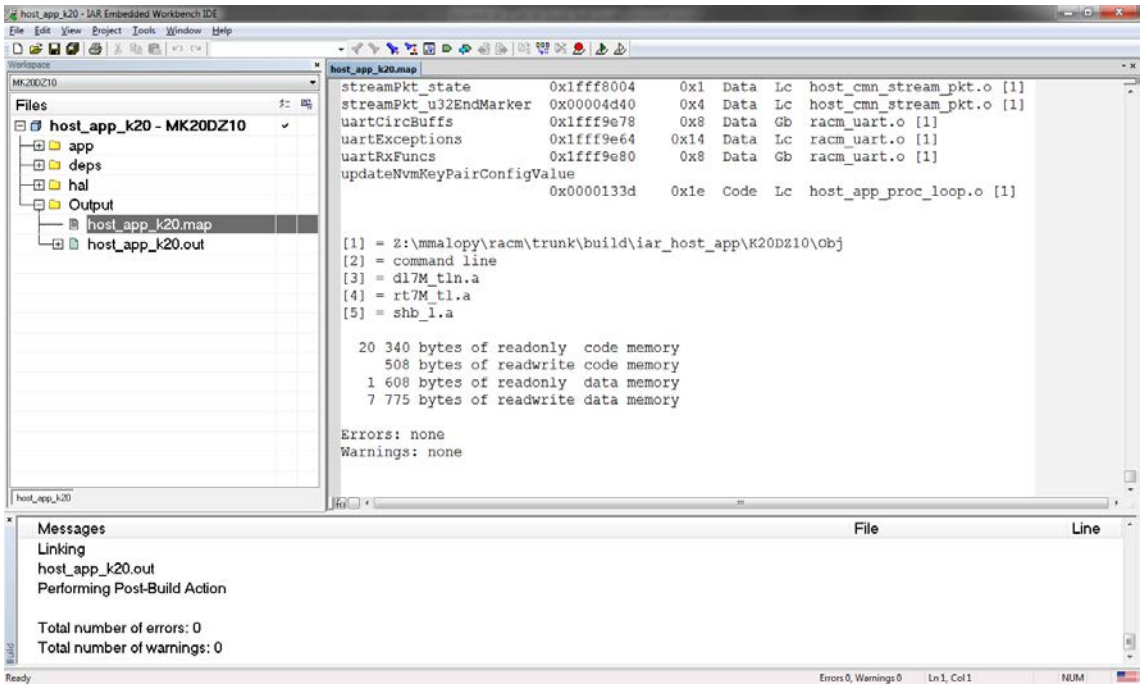
2.4 Building the Application

Although the rACM release already contains the pre-built binaries for both the IAR and GCC tool chains, the application can still be (re)built using the following methods:

2.4.1 IAR Workbench IDE Project Manager Build

When using the IAR IDE, the *build* subdirectory contains the rACM project file *host_app_k20.eww*. After the project is loaded, select the drop-down configuration option for

the appropriate target processor – the rACM platform assembled by On-Ramp Wireless is currently populated with the K20DZ10 option. After that, the 'Project->Build All' tab command performs all of the necessary steps to build the project output (i.e., target binaries, dictionary files, map files). All of the necessary build options and library configurations have previously been set-up and defined for the rACM project.



Details on the IAR Workbench IDE are found in the supporting IDE documentation, help menu, and online from the IAR website at http://www.iar.com/en/Products/IAR-Embedded-Workbench/ARM/.

2.4.2 Makefile Build

The rACM application can also be built from the command line.

- Linux platform – Using the GCC tool chain
- Windows platform – Using Cygwin with the IAR tool chain

Regardless of the platform, the Makefile syntax is the same:

- **make clean** – Clears/deletes the build output folder to force a fresh build.
- **make host_app** – Compiles/links the rACM application

The Makefile definition for each target (i.e., *gnu_host_app.mk or iar_host_app.mk*) contains all of the necessary rules for building the rACM project output.

2.5 Project Output

Regardless of the platform and/or build method, the output of the build process is identical and produces the following output:

- **Dict** – Contains the dictionary output (i.e*., host_app.logDict*) used by the Host application's logging methods. It is essentially a library of ASCII strings that are extracted from the source code during the build and associated with the Host Log Indicators that are broadcasted through the Host Interface UART. This method of logging allows the Host Application to log any user-defined string data and also keep the memory footprint caused by strings to be minimized.
- **Exe** – Contains the binary image (*host_app_k20.out*) and debug image produced by the build process.
- **List** – Contains the pre-processor output of each source file as well as the map file as generated by the build process. The pre-processor output is required to support the generation of the dictionary output.
- **Obj** – The source objects, as generated, during the build process.

It is important to note that the tool chain determines which build subdirectory the project output is piped to:

- *gcc_host_app* – Project output for GCC-based tools on a Linux platform
- *iar_host_app* – Project output for IAR-based tools on a Windows platform (either Cygwin makefile or IAR IDE).

It is further directed into a subdirectory for the currently configured target processor (i.e., K20DZ10 or K20D7 subdirectories).

3 Reference Platform Hardware

The following illustration provides a high level depiction of the rACM hardware platform illustrating the two CPUs (i.e., application host and On-Ramp Wireless Node modem) and external interfaces that are under direct control of the Application Host.

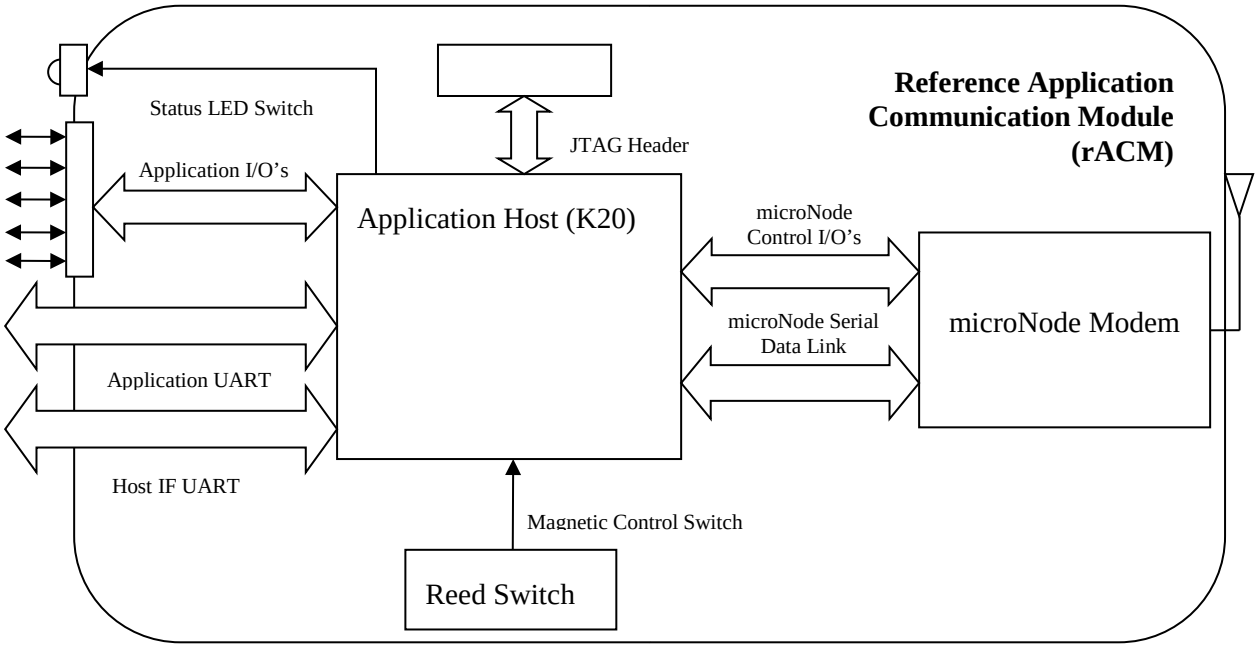


Figure 2. Reference Application Block Diagram

Referring to the rACM hardware schematic, the application interfaces shown in the figure above fall into the following categories:

- **Status LED Switch** (D101) – A GPIO-controlled LED which the application host uses to display status patterns, alarm states, and/or user-feedback. Its exact use is dependent on the application type.
- **Application I/O Header** (J102) – A bank of five external application pins (in addition to ground). The pins are defined as follows:
  - PWR (Kinetis port PTC4)
  - D0 (Kinetis port PTC1)
  - D1 (Kinetis port PTC3)
  - D2 (Kinetis port PTC5)
  - D3 (Kinetis port PTC6)
  - GND

It should be noted that the ports on the Kinetis MCU are multiplexed to support a number of possible functions (even at run-time). Depending on the application type, the listed pins can be configured for a combination of the following:

1. General Purpose Input/Output (GPIO)
2. Kinetis Flex Timer Module (FTM) output (e.g., single-ended or differential clock output, stepped patterns, Pulse-Width Modulation (PWM) signal generation, etc.).
3. Kinetis Analog-to-Digital Converter (ADC) input (single-ended or differential) – used for voltage or current sampling
4. Kinetis Comparator circuit input
5. Kinetis Low-Power Timer (LPT) input – used for pulse counting or clock generation while the application is in deep sleep

The rACM application currently configures and manages D0 (PTC1) as a Low-Power Timer (LPT) input for use in monitoring and reporting pulse counts as user data (see section 5.2).

- **Application UART** (Kinetis ports PTD6 and PTD7) **–** A dedicated 3-wire serial interface for UART communications to an external device (e.g., GPS module, ModBus sensor interface, etc.). This interface is currently unused in the rACM build.
- **Host Interface UART** (J103)**–** A dedicated 3-wire serial interface for UART communications to a Host PC for purposes of factory calibration, field provisioning and/or field debugging.
- **Reed Switch** (SW100 – bottom of reference board on front edge) **–** A dedicated port (configured as a GPIO input) used to monitor a magnetic Reed Switch. Primarily intended for applications they may end up in storage before being deployed in the field where a magnetic provision tool could be used to “activate” the device and initiate the network join process.
- **JTAG Header** (J100) – 10-pin IC Debug port used for on-target debugging or for flashing development builds (more on this interface to follow).
- **VBatt Supply Header** (J101) – 3.3 Volt input used to power the reference platform (battery or wall wart AC adapter). The application host is specified to operate down in the 3.3V to 2.7.

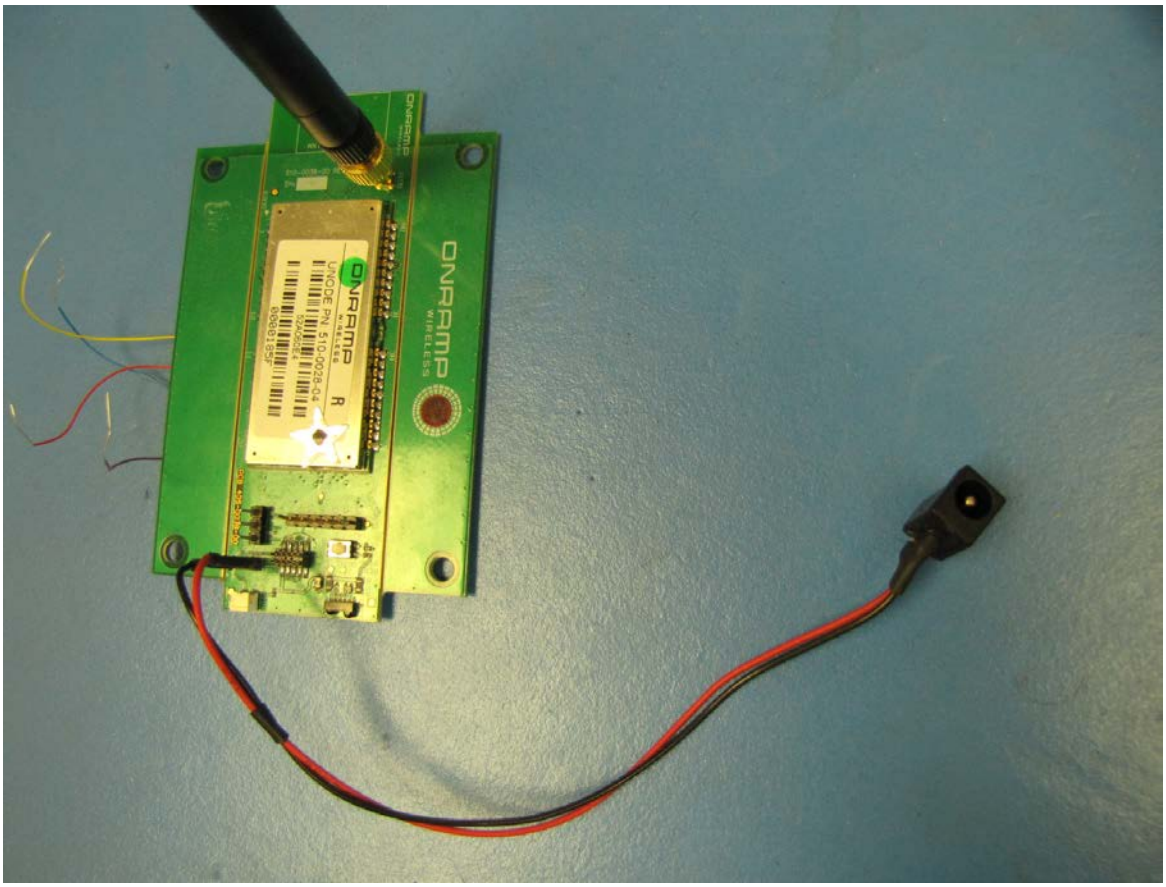
3.1 AC Power

The Reference hardware platform requires a 3.3V DC source on the Vbatt Supply Header (J101). This can come from two different sources:

- 19Ah lithium D cell (Tadiran TLP93311AL).
- 3.3V AC to DC Adaptor using a pigtail cable.

The 3.3V AC to DC adaptor connection is shown in the following photo.

NOTE: Some versions of the Kinetics K20 are prone to latch up if the applied power doesn’t meet a glitch-free ramp profile. Latch up can be avoided by using a quality supply with a controlled profile. If latch up is observed, repeat the application of power until a successful 2 blink LED sequence is observed. During power cycling, all external cabling (UART, sensor) to ensure power isn’t applied through other interfaces.



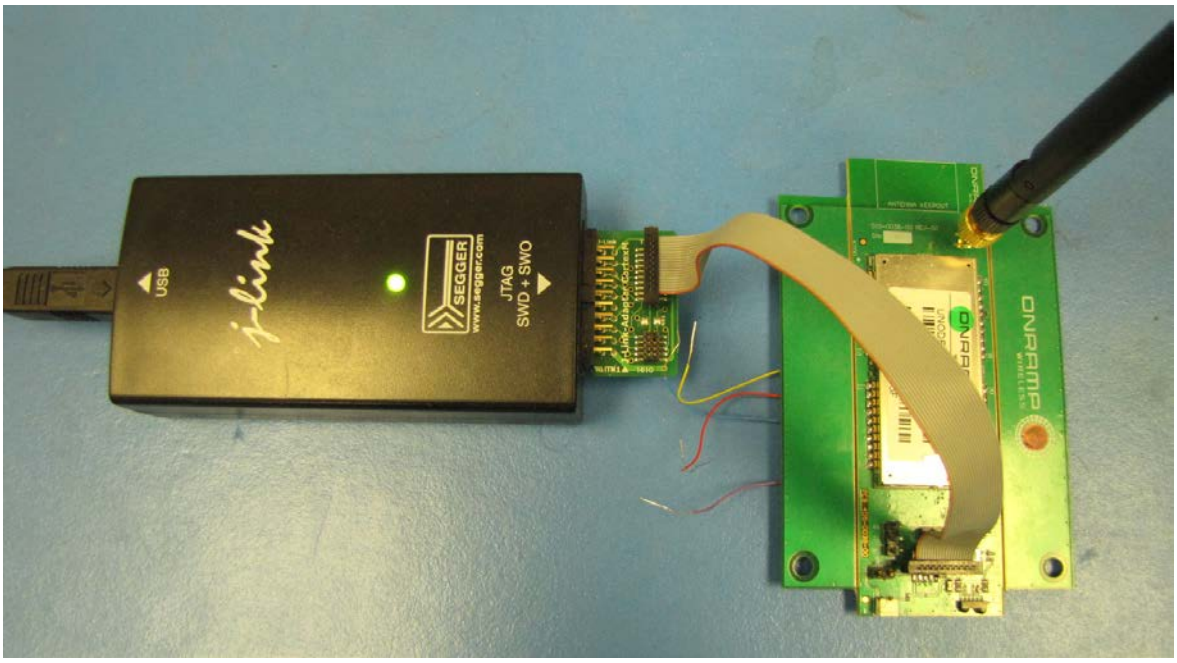
When power is applied to the reference platform (assuming that it has been previously flashed with the rACM firmware), the status LED (D101) flashes initial Power-On Sequence (two half-second on-pulses every five seconds. See section 5.1).

3.2 JTAG Interface

For on-target debugging and/or developer and initial Flash programming of the application MCU on the Reference platform, an external emulator, capable of controlling the Cortex-M4 processor, can be used by connecting though the on-board JTAG header (J100).

If the IAR EWARM IDE is being used with the rACM, it is recommended that the Segger J-Link Emulator be selected for this purpose – the rACM IAR Project is preconfigured to communicate with the rACM platform through the J-Link adaptor. Further details on the Segger J-Link can be found at http://www.segger.com/jlink.html.

The following figure shows the J-Link emulator cabled to the rACM reference platform using the 20-pin cable connector that ships with the Segger unit.



Unfortunately, the JTAG header in the reference platform (J102) is a 10-pin header. Rest assured, that all of the necessary JTAG interface pins can be routed through the incompatible headers with a little bit of overlap as shown by the following figure:

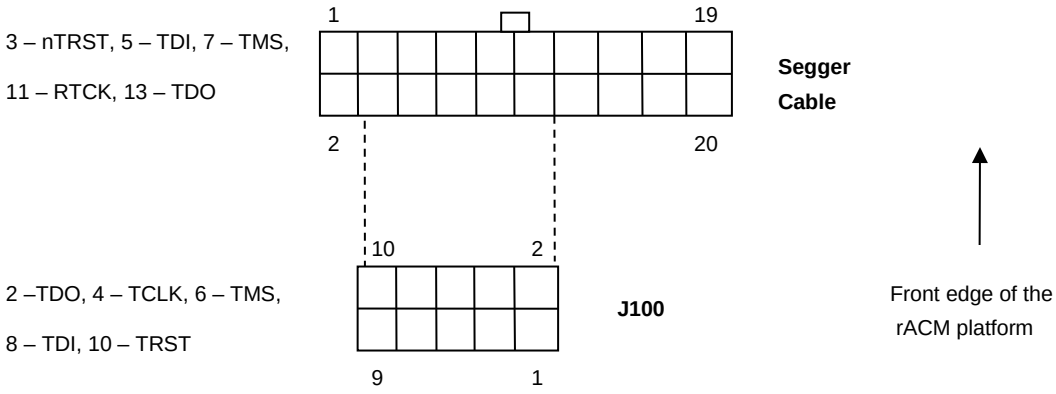


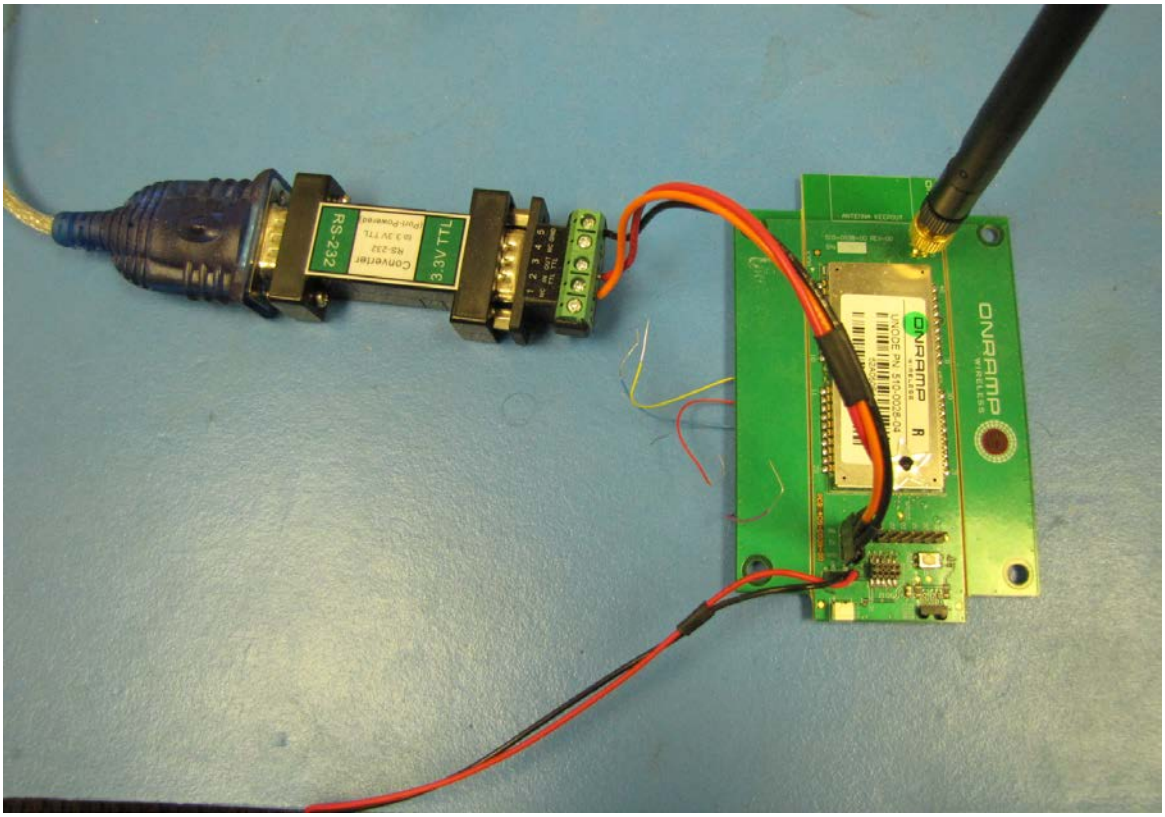
Figure 3. JTAG Header and Interface Pins

When the Segger cable is plugged in, the plastic tab on the connector should face the front edge of the reference platform (i.e., where the status LED and Reed Switch are located). Facing the header from the front edge of the board, Segger pins 1-2 should overlap the left edge of the J100. Alternatively, the 10-pin JTAG ribbon cable can be ordered along with the JTAG emulator to avoid these types of interface issues.

3.3 Host UART Interface

All serial communications with the application host executing on the reference platform are done using an RS-232 serial interface. However, due to the reference platform executing at the 3.3V power domain, a TTL convertor is required to safely convert the PC port to the levels that are safe for the application host. By default, the Host UART 3-wire serial interface routed through the J103 header is configured to use Kinetis ports PTB10/PTB11 for Rx/Tx UART data.

The following figure shows the TTL Converter connected to the reference platform.



3.4 Antenna Connection

The antenna power is located on the back edge of the reference platform (i.e., opposite of the status LED and Reed Switch) and uses a standard micro-miniature coaxial RF connection (MMCX) on port J105.

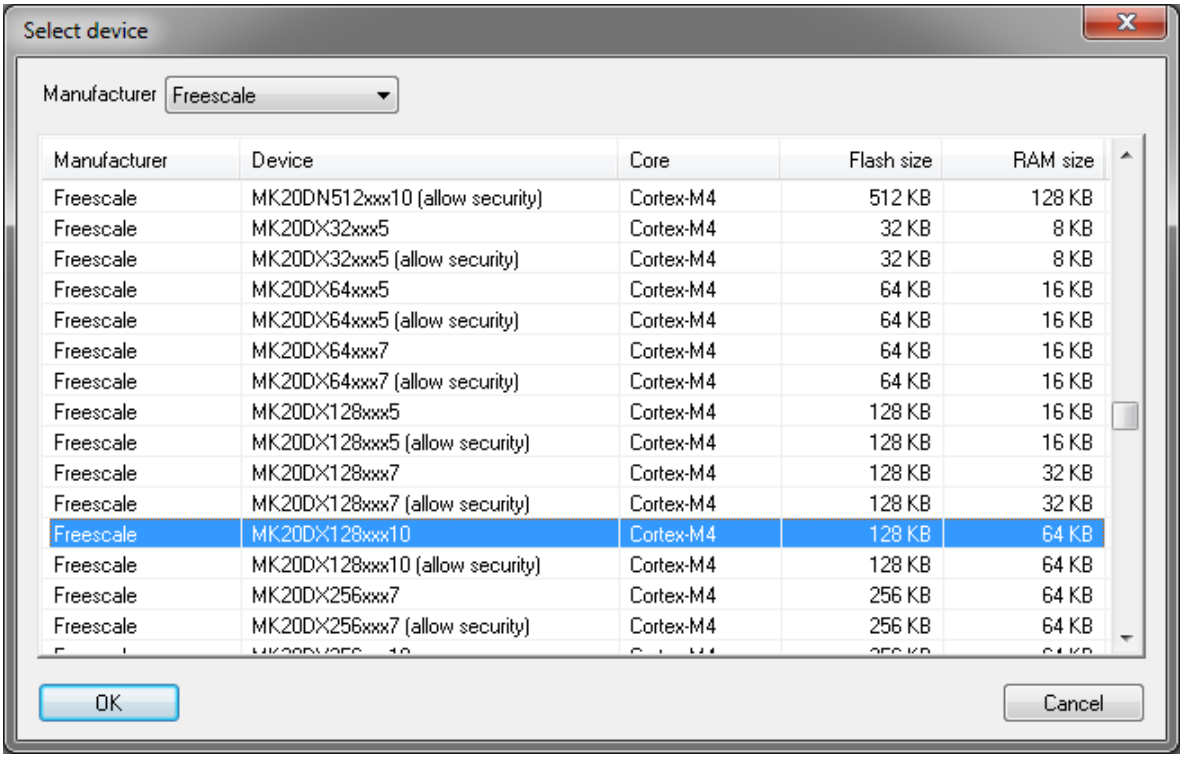
4 JTAG Code Download and Debug

During manufacturing test or application development, it is more efficient to download the application host software through the JTAG interface. The procedure for downloading and storing the rACM image to Flash varies based on the build method.

4.1 IAR EWARM Project

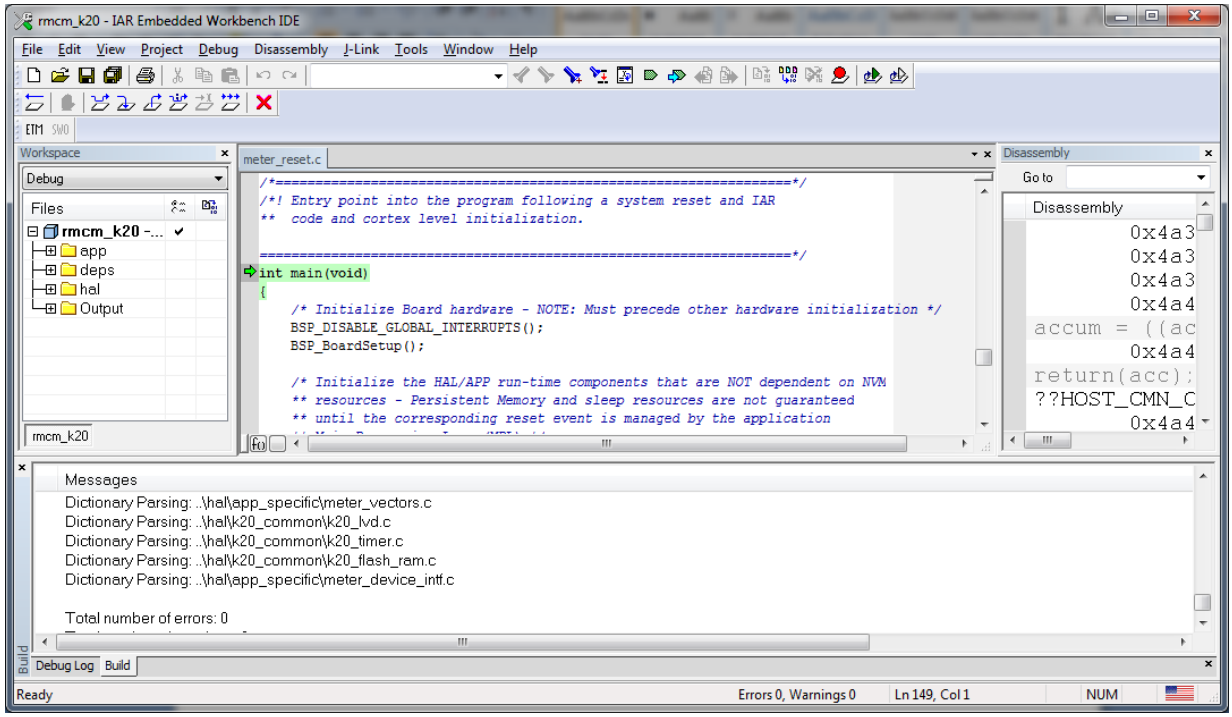
Perhaps the most straightforward method of downloading and debugging the rACM application is to use the rACM project which has been preconfigured with the necessary Flash loaders and J-Link interface settings necessary to perform this operation.

After downloading the project file (see section 2.4.1) and applying power to the reference platform, the 'Options->Download and Debug' command can be used to kick off the JTAG flashloader process (alternatively, the green-arrow command icon can be selected). When J-Link is invoked for the first time, it prompts the user to confirm the target processor as shown in the following figure.



Ensure that the MK20DX128xxx10 option is selected before proceeding with the download process. The rACM binary is less than 20K bytes. The actual download through the J-Link happens within a second at the configured clock rate of 48 MHz.

After the image is downloaded, the JTAG halts the processor at main() as the following figure shows:



NOTE: If the image downloaded through the JTAG is being done for development purposes (i.e., after code changes), we recommend that the application undergo a Power-On-Reset (POR) by removing the 3.3V source voltage following the successful upload. The ACM POR handler performs all of the necessary (re)initialization of the run-time variables. These may change as a result of software modifications. Otherwise, the ACM application host treats the JTAG reset as a recoverable scenario and attempts to resume steady-state operations. If the tracking variables have changed, this may lead to undefined behavior.

The IAR IDE contains all of the standard debug features (i.e., breakpoints, call stack trace, register display, memory display, etc.). Detailed steps for using the IAR Debugger is beyond the scope of this document. The EWARM Debugger manual is available within the IAR source directory or from the IAR website and contains detailed instructions and procedures for using the IAR IDE capabilities.

4.2 IAR Makefile Build

If the rACM application is built using the makefile process, the IAR Debugger can still be used to download and debug the .elf output file. The build directory contains a second project file called *host_app_k20_debug_only.eww.* Instead of source files, this project contains a single build file which can be downloaded in the same manner described for EWARM IDE project builds.

The only difference is that the user may be prompted to find the source file for debugging – depending on where the debugger is halted (i.e., *app*, *hal*, or *host_cmn* source directories).

5 rACM Operational Overview

The default rACM application build, when it first comes out of Power-On-Reset, operates under the following run-time parameters:

- rACM is configured for quick-start, without any user-prompt, the application powers on the Node and initiates the network join process.
- rACM is configured to enable and count pulses over the D0 pin on the Application I/O header (J102) using the Low-Power Timer (LPT) Kinetis block.
- When rACM has joined the network (confirmed by monitoring the Status LED and/or rACM logger display), the application schedules a callback to read the LPT counter every hour on the hour. The value is time-stamped and saved into Non-Volatile Memory (NVM) for later OTA reporting at the Uplink Interval (UI).
- Each Uplink Interval (configured on a per-node bases), the rACM application reports to the backend all undelivered, time-stamped records in NVM in a single Tx Service Data Unit (SDU). If the UI is less than the read interval or if no read records are found in NVM, then the rACM reports the current read value at the time the UI is processed.
- When the rACM has joined the network, a swipe of the provisioning tool generates the asynchronous ALARM0 event. This in turn results in an asynchronous Tx SDU generated to the backend reporting the alarm state change.

  NOTE: All alarm state changes generated by the provisioning tool are subject to a 5-minute hysteresis period before an alarm state change can be generated again – this minimizes the amount of asynchronous SDU transfers generated by the application.
- Deep Sleep and UART Power Management are disabled by default on the rACM build. This is to allow unimpeded Host messaging capabilities that do not require the use of the magnetic provisioning tool to wake the host application from deep sleep.

Further details on the rACM application and its event processing can be found in the rACM Software High Level Design (HLD) (see reference 1).

The following subsections discuss the reference platform interfaces that can be exercised during the course of steady-state operations on the rACM.

5.1 Status LED Patterns

The rACM uses the status LED to provide visual feedback during operational state changes and network join attempts. Distinct sequences are used to identify each state. The sequence consists of the 5-second pattern in the following table, repeated twice.

Table 1. Status LED Patterns

Name	Pattern
SHUTDOWN	1 x (½ sec ON + ½ sec OFF) + 4 sec OFF
PROVISION	2 x (½ sec ON + ½ sec OFF) + 3 sec OFF

Name	Pattern
JOIN	5 x (½ sec ON + ½ sec OFF)
NETWORK_DROP	1 x (2 ½ sec ON + 2 ½ sec OFF)

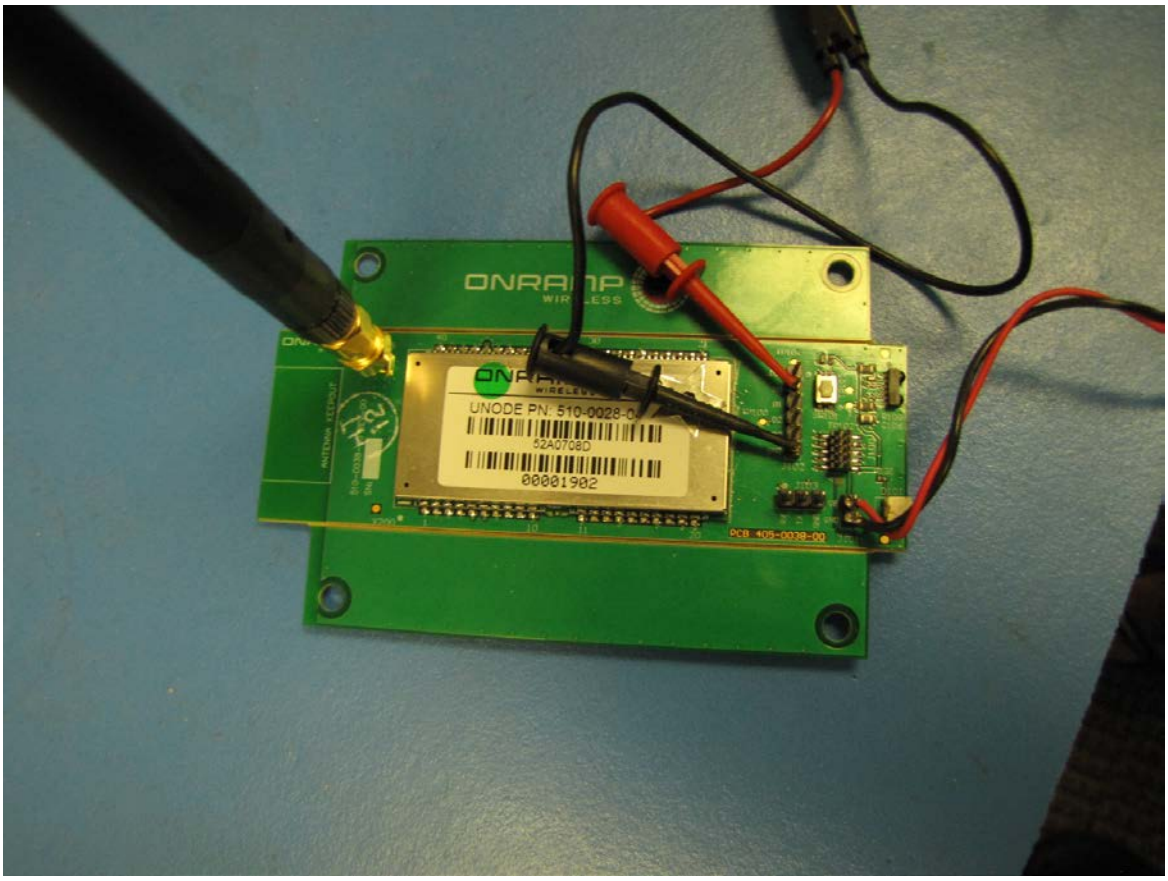
The rACM application autonomously generates the status LED pattern when it first powers on and when the device joins the network for the first time. The current device state is also reported in response to each swipe of the magnetic provisioning tool via the Reed Switch.

5.2 Pulse Counter Inputs

As mentioned in this section overview, the rACM's default sensor interface utilizes the low-powered pulse counter configured to monitor pin D0 on the Application I/O header (J102). At each read interval, the application reads the LPT counter and stores the value into NVM. However, without any periodic pulses applied to this input, the value remains at zero which is not a very interesting demonstration of the rACM logging and reporting capabilities.

The recommended method to generate count data is to use a waveform generator and generate a periodic pulse with active high polarity and amplitude of 3.3V. The width of the pulse is typically expected to be around 100msec. The period of the pulse waveform is at the discretion of the end-user.

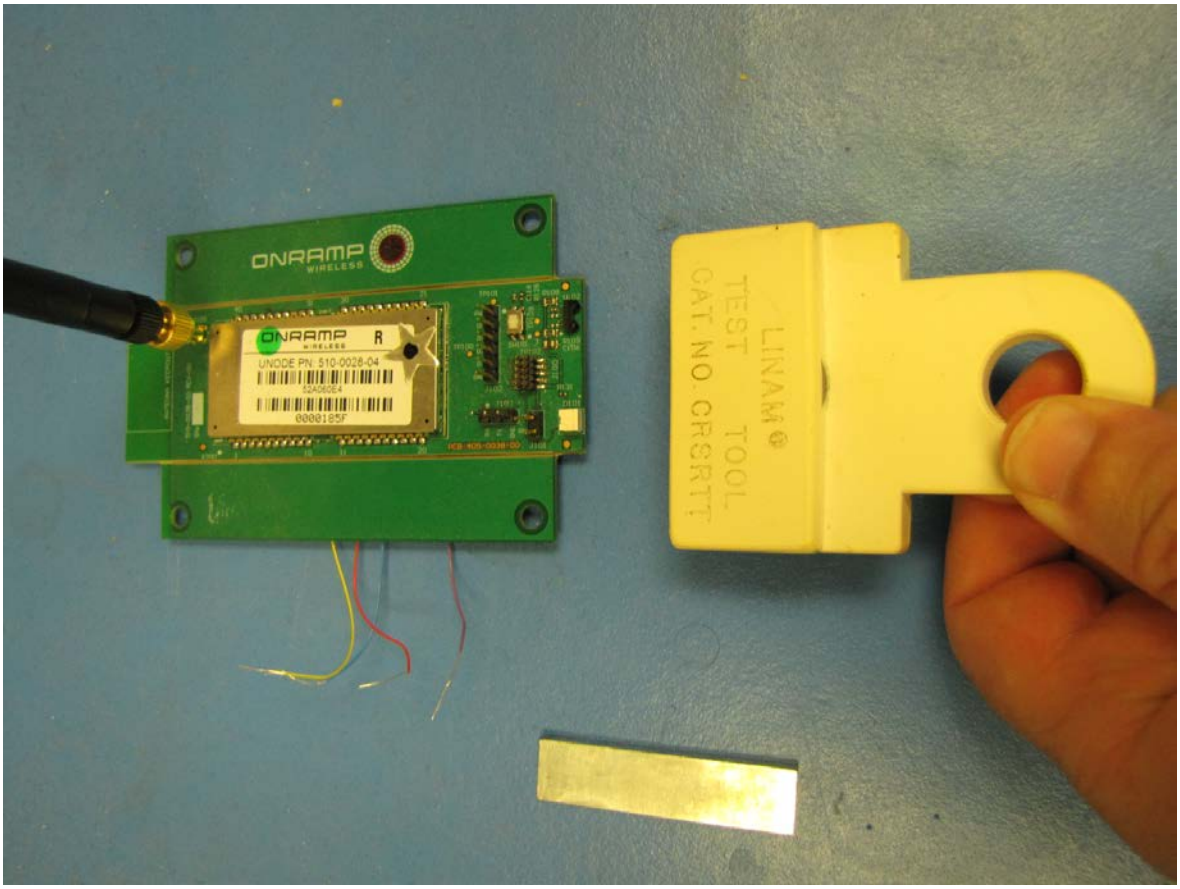
The following figure shows the output of the waveform generator (pulse_in and ground) connected to the reference platforms' Application I/O header:



The current pulse count can be confirmed by using the GET_DEVICE_STATUS command via the Host Interface UART or by post-processing the Tx SDU payload data from the On-Ramp Total View application.

5.3 Alarm Inputs

As mentioned in the overview section of this chapter, asynchronous alarms can be demonstrated through the use of the magnetic provisioning tools with a single “swipe” across the Reed Switch circuit (SW100) mounted on the front edge of the reference platform (i.e. the edge of the platform where the status LED is mounted) as the following figure demonstrates.



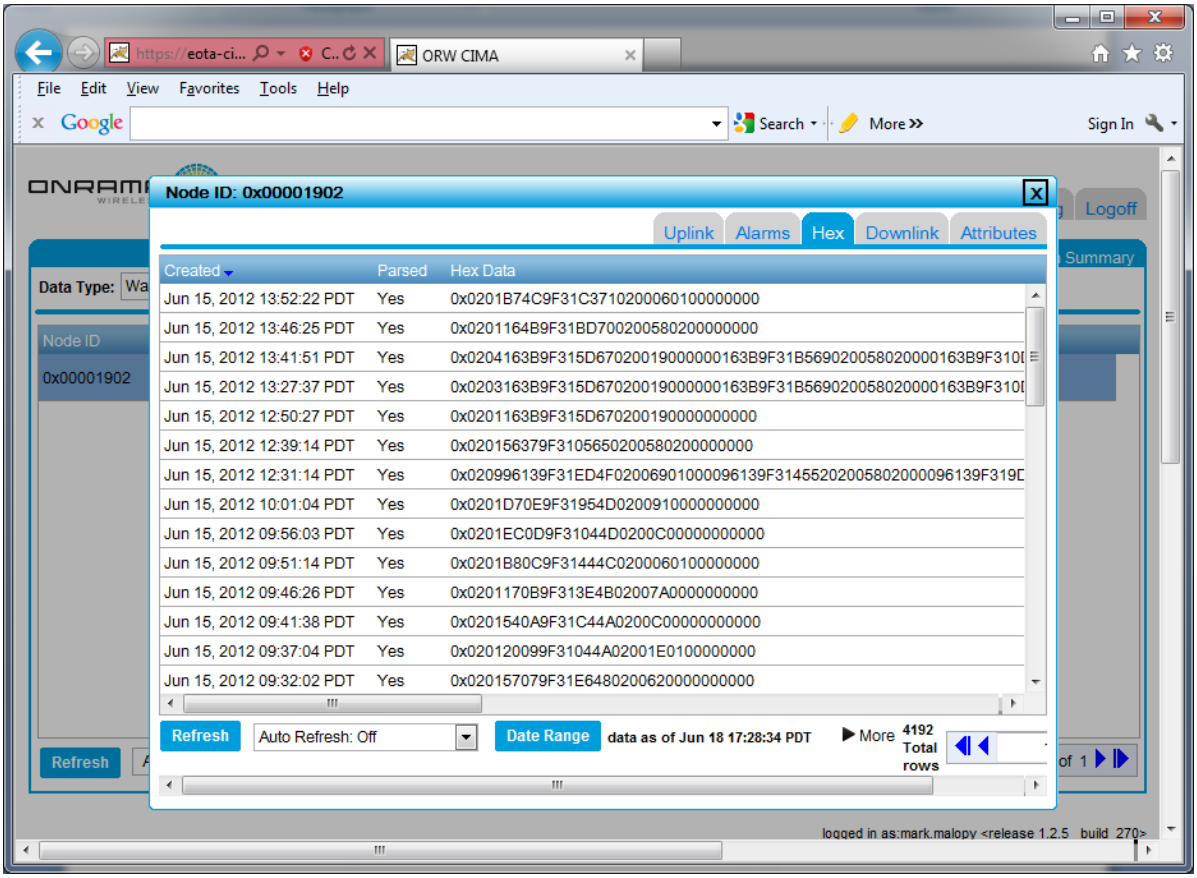
The Reed Switch swipe can be confirmed if the status LED responds with the current state pattern (see section 5.1).

It is recommended that the rACM Logger application (see section 6.2) be monitored when this event occurs as the Reed Switch “swipe” triggers a state change for the ALARM0 type. If the Alarm Hysteresis is not in effect (default period of five minutes), then an asynchronous Tx SDU is generated with payload data that reports the alarm type and state along with a corresponding time stamp. The rACM application also manages conflicts if a Tx SDU is already in progress. It buffers the alarm data and waits for the pending network transaction to finish. By monitoring the rACM Logger, all of these types of application events and/or conflict management are displayed in real-time.

5.4 On-Ramp Total View Payload Verification

All Tx SDUs generated by the rACM application eventually get routed through the backend to the On-Ramp Total View application. It is assumed that, as part of the network configuration process, the Node on the reference platform has been added as a device by the On-Ramp Wireless Element Management System (EMS). Details about the On-Ramp Total Reach backend components are considered beyond the scope of this document.

By clicking on the Node ID associated with the rACM device, all Tx SDU payload data can be verified by monitoring the 'Hex' tab. Each Tx SDU received by the On-Ramp Total View application contains the time-stamped hexadecimal payload data as the following figure illustrates:



The *rACM Software High Level Design document (013-0008-00)* contains all of the message formats necessary to parse the raw hex data (see reference [1]).

6 rACM Host Tools

The *python_tools* directory all of the necessary Python scripts for interfacing with the rACM application through the Host Interface UART. All control scripts over the Host Interface use the following properties:

- Default baud rate of the Host UART is 115200bps. The `-b` option is used to override/set the baud rate to match any changes on the application host.
- Default Com port is COM0 (on a Windows platform) or /dev/ttyS0 (on a Linux platform). The `–d` option is used to override/set the actual port used to communicate with the application host.
- The `–h` or `–help` option displays the usage for the current control script.

6.1 rACM Control Script

The *host_app_ctrl.py* script is the primary utility script for configuring and status reporting of the rACM application. Here is a summary of the support command options.

Table 2. Support Command Options

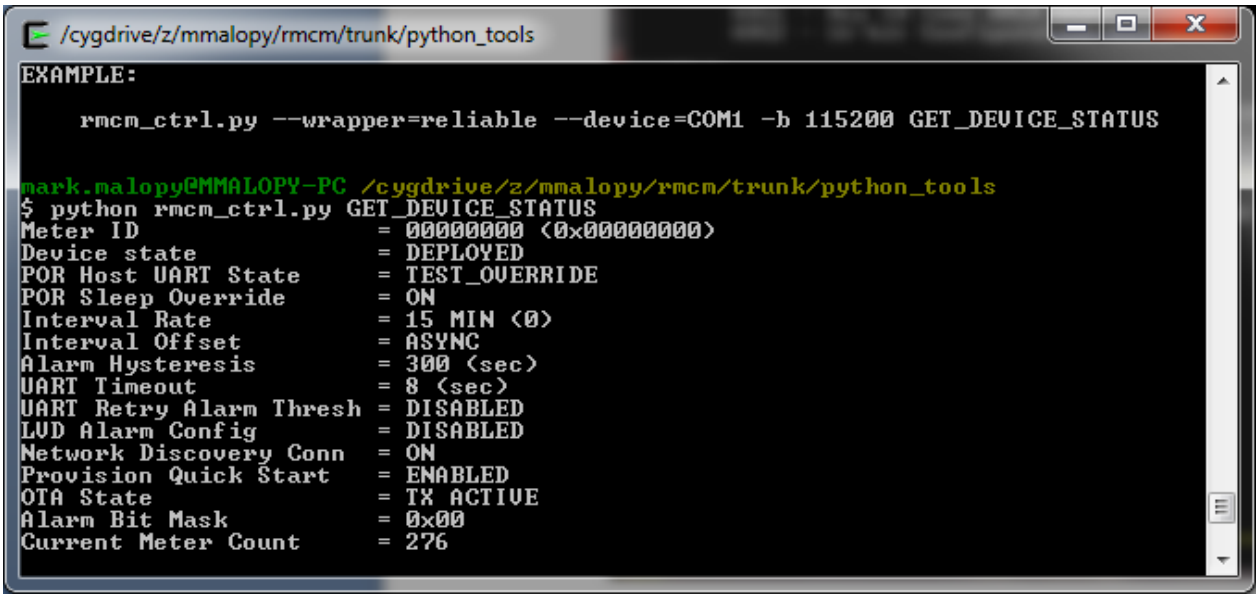
Command	Description
HOST_VERSION	Reports the major, minor, and point release values of the rACM application currently executing on the reference platform.
ENTER_PASSCODE	If the UART security features are enabled (see chapter 7), this is the command is used to unlock the Host UART interface. This command (with a valid passcode) must precede any other rACM commands. In addition, a configurable security lockout feature can be entered into if the passcode is entered incorrectly in a set number of passcode attempts (see reference [1] for more details on the Host Interface security feature).
CONFIG_NODE_ATE	If the device is in the shutdown state, this command can be used to turn on the Node for purposes of network configuration and/or network debugging. If the rACM application is already in an operational state, this command is ignored.
GET_DEVICE_STATUS	The command can be used to report the status of the rACM application, its configuration parameters, and any current sensor read data (i.e., the absolute count value).
SET_ READ_INTERVAL	This command is used to configure the MCM Read Interval rate and time synchronization offset. The rACM Software HLD contains the table of all supported read intervals. It should be noted that the time synchronization feature uses GMT as opposed to local timing. This setting is persistent across system resets and/or power cycles.

Command	Description
SET_ALARM_HYSTERESIS	This command is used to set the delay, in seconds, between generating alarm state changes (the default is 600 seconds – i.e., 5 minutes) that result in the generation of an asynchronous Tx SDU request. This feature helps minimize the amount of transmission spamming that may result from a alarm hardware circuit with transient “glitches”. This setting is persistent across system resets and/or power cycles.
CONFIG_LVD_ALARM	This command is used to configure Low-Voltage Detect (LVD) threshold alarm settings. This setting is persistent across system resets and/or power cycles.
SET_ NETWORK_DISCOVERY	This command is used to enable or disable the unsolicited generation of the Configuration Read Response message (opcode 0x4 – see reference [1]) when the device joins the network for the first time following a Power-On Reset (POR). This is a form of a “birthday boot” mechanism in which the application host’s configuration parameters can be used by the backend for display and/or device capability features. This setting is persistent across system resets and/or power cycles.
SET_UART_TIMEOUT	This command is used to set the inactivity timeout value, in seconds, at which point to power down the Host Interface UART for power savings purposes. This setting is persistent across system resets and/or power cycles.
SET_UART_PASSCODE	This command is used to set the UART Hexadecimal passcode value for Host Interface security purposes. This setting is persistent across system resets and/or power cycles.
SET_UART_PASSCODE_ALARM_THRESH	This command sets the alarm threshold for the Host UART pass-through security lockdown feature for a configurable number of invalid passcode attempts. It should be noted that the only way to unlock the security lockdown is from the backend via a Key-Value Pair. This setting is persistent across system resets and/or power cycles.
SET_DEVICE_ID	This command is used to set the 32-bit Device ID. This setting is persistent across system resets and/or power cycles. **NOTE**: This is not to be confused with the Node ID.
RESET_DEVICE	This command is used to clear and reset any device state parameters effectively reverting to the Shutdown state.
SLEEP_OVERRIDE	This command is used to enable/disable deep sleep functionality on the device. This setting is persistent across system resets and/or power cycles.
HOST_UART_OVERRIDE	This command is used to enable/disable Host UART Power Control. This setting is persistent across system resets and/or power cycles.

Command	Description
SET_PROVISION_QUICK_START	This command is used to enable/disable the provisioned quick start state. When set, it essentially powers on the Node and initiates the network join process following a POR event. If cleared, the device remains in a shutdown state until a “swipe” of the magnetic provisioning tool occurs (i.e., during field deployment). This setting is persistent across system resets and/or power cycles.
SET_RHT_SUPPORT	This command is used to enable/disable the Reliable Host Transfer (RHT) protocol on all Host Interface serial packets (i.e., a retry mechanism to compensate for bit errors on the Host Interface serial link). If enabled, all command line scripts must include the –*wrapper=rht_v2* option for host messaging to properly function.
HOST_SW_UPGRADE	This command is used to do a firmware upgrade, using the rACM binary image, through the Host UART Interface.
HOST_RESET_NODE	This command is used to reset the Node.
DUMP_READ_RECORDS	This command is used to dump all undelivered read records from rACM Non-volatile Memory (NVM) to the PC host.
DUMP_ALARM_RECORDS	This command is used to dump all undelivered alarm records from rACM NVM to the PC host.
DUMP_EXCEPTIONS	This command is used to dump all exception event records from rACM NVM to the PC host (e.g., Watchdog Timeout exceptions, Hard Faults, Logging failures, etc.).
DUMP_NODE_EVENTS	This command is used to dump all Node state changes from rACM NVM to the PC host.
TEST_OTA_ALARM	This test command can be used to generate an asynchronous alarm type and state to the backend. However, Tx SDU generation is subject to the rACM state event handlers (i.e., it may be queued until a current Tx SDU request has completed). Unlike the application alarm events, alarms generated by this command are NOT subject to alarm hysteresis.
SET_KEY_PAIR_VALUE	This command is used to change a single key-value pair (see chapter 7).This setting is persistent across system resets and/or power cycles.

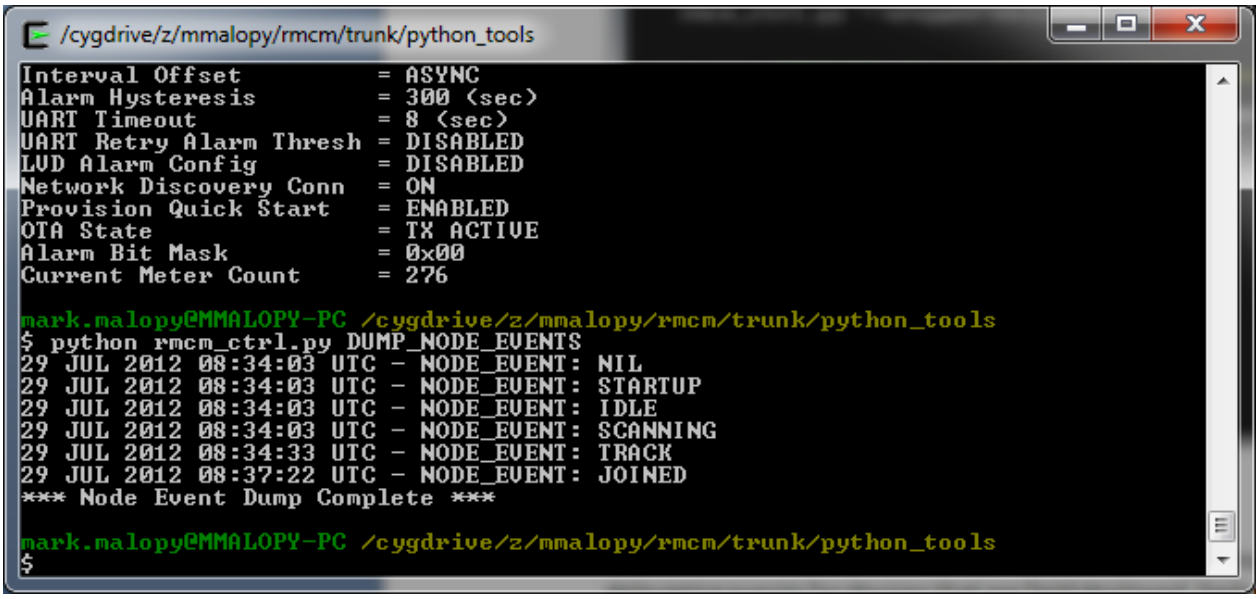
6.1.1 Get Device Status Example

The GET_DEVICE_STATUS command is a useful for reporting the current snapshot of the rACM state variables and configuration settings. The following screenshot shows the typical response to this command:



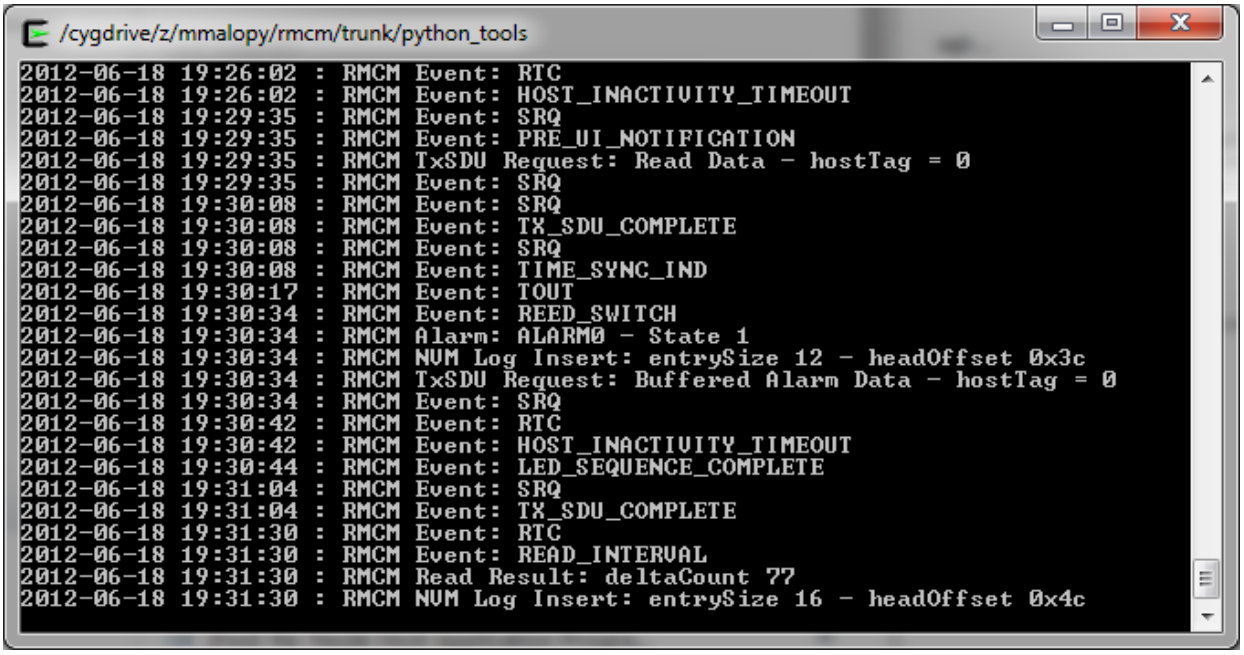
6.1.2 NVM Dump Example

The DUMP commands for read records, alarms, exceptions and node events are also very useful for debugging events for devices that are field deployed. Each record contains the GMT timestamp for when it was logged. The following figure shows an example of the Node events for a rACM device coming out of Power-On Reset:



6.2 rACM Logger Script

The *host_app_logger.py* script is allows for on-target monitoring of all of the rACM application events that occur in real-time through the Host UART. This is an excellent resource for understanding the flow of the rACM event handlers in response to various input stimuli. Each entry is time-stamped as they are received by the PC host to help provide an idea of the timing between the various event processing. The following screen capture shows the log output during an Alarm state change triggered through the Reed Switch.



All of the displayed strings by the logger utility are taken directly from the rACM source code via the Log Dictionary file generated as part of the build process (see section 2.5). By referencing the rACM logger event strings with the source code, one can quickly deduce and reconstruct the flow of the Main Processing Loop implementation.

NOTE 1: To successfully process the rACM logger string output, the dictionary file in the build path MUST match the build that is flashed on target. A mismatch in the dictionary hash stamp that is reported over the Debug Log Indicators results in a parsing exception.

NOTE 2: The rACM Logger application defaults to search for the rACM dictionary file in the project output generated by the iar_build (i.e.*, build/iar_host_app/Dict*). The end-user can choose to override the default Dictionary path/filename by using the `-l` or `--logdict` command line option. For example:

```
--locdict=../build/gcc_host_app/Dict)
```

7 rACM Configuration Options

Once the basic rACM implementation is understood, the end-user can choose to reconfigure the device configuration parameters. Using the same test methods and logging capabilities, the rACM application can be observed under the following conditions:

- rACM event flow with Deep Sleep enabled
- Different read intervals (e.g., 15 minute reads vs. 1 hour).
- Different time synchronization settings (e.g., 1 hour reads synchronized to at the 30 minute offset, 1 daily read to occur at 0:00 GMT, etc.).
- The UART Host Interface security feature can be enabled/tested to prevent unauthorized access.
- Quick start can be disabled to demonstrate the field provisioning process.

All of these configuration options can be read or modified using a designated Key-Value Pair for the rACM application. Each Key-Value pair can also be modified over the Host UART Interface or from the backend via the Write Configuration Command (opcode 0x3). All Key-Value pairs are stored in FlexNVM and protected with a 16-bit CRC.

The following table summarizes the rACM Key-Value Pairs currently implemented.

Table 3. rACM Key-Value Pairs

ORW ID	Configuration	Description
1	UART Timeout	Timeout in seconds Default = 10 seconds
2	UART Password0	Password[15:0]
3	UART Password1	Password[31:16]
4	UART Password2	Password[47:32]
5	UART Password3	Password[63:48]
6	UART Password Disable Threshold	Number of failures 0 = never disable
7	Read Interval Bit 3:0 – Interval Duration Bit 15:4 – Interval Minute Offset (from 0:00 GMT)	Duration Default = 2 (1 hour) 0 = 15 min 5 = 4 hour 1 = 30 min 6 = 6 hour 2 = 1 hour 7 = 8 hour 3 = 2 hour 8 = 12 hour 4 = 3 hour 9 = 24 hour Minute Offset Default = 0x000 (0:00 GMT) **NOTE:** If Minute Offset = 0x3FF, read intervals are asynchronously scheduled for when the rACM first joins the network.

ORW ID	Configuration	Description
8	Low Battery Alarm Threshold	Threshold defined by LVD logic 0 = disabled 1 = 3.0V 2 = 2.9V 3 = 2.8V 4 = 2.7V 5 = 2.56V
9	Device Identification Field0	Device ID[15:0]
10	Device Identification Field1	Device ID[31:16]
11	Alarm Hysteresis	Number of seconds to disable alarm status reporting following an alarm state change. “0” disables feature Default = 300 seconds
12	POR Default Host Interface State	Power-on-Reset Default state for Host UART. 0 = Inactive 2 = Active 3 = App Override
13	POR Sleep Override State	Power-on-Reset Deep Sleep Override State. 0 = OFF 1 = ON
14	Network Discovery Connect	Feature to send all Configuration Key-Value pairs when the network is discovered for the first time: 0 = OFF 1 = ON
15	Provision Quick Start	Feature to skip provision step at Power-on-Reset and immediately initiate the network join process. 0 = OFF 1 = ON
16	RHT Support	Feature to enable the Reliable Host Transfer (RHT) protocol on all Host Interface serialized data. 0 = OFF 1 = ON
0x80	OTA Alarm Set Test	Simulates/generates an OTA Alarm Set record event for the following: 0 – ALARM0 1 – ALARM1 2 – ALARM2 3 – LOW_BATTERY 4 – SECURITY 5 – LOG_NVM 6 – SLEEP_NVM 7 – RUNTIME

ORW ID	Configuration	Description
0x81	OTA Alarm Clear Test	Simulates/generates an OTA Alarm Clear record event for the following: 0 – ALARM0 1 – ALARM1 2 – ALARM2 3 – LOW_BATTERY 4 – SECURITY 5 – LOG_NVM 6 – SLEEP_NVM 7 – RUNTIME
0x82	Security Alarm Clear	Clear the Security Alarm state on the device regardless of data field.

Appendix A Operating Examples

This chapter provides example to illustrate the sequence of steps to:

- Confirm rACM power-up device status using *host_app_ctrl.py*
- Change rACM operating mode to disabled host sleep using *ctrl.py*
- Configure the rACM's AP list using *ulp_node_monitor.py*
- Verify the rACM joins the network using *ulp_node_monitor.py*
- Log host operation using *host_app_logger.py*

A.1 Get Device Status

After a successful power-up signed by the 2-blink LED sequence (see section 5.1), the device status (see section 6.1.1) is read with the following command line example:

```
python host_app_ctrl.py  -d /dev/ttyUSB0 GET_DEVICE_STATUS
```

NOTES:

1. For Windows OS replace /dev/ttyUSB0 with the appropriate COMx parameter.
2. Following a power cycle, the first attempt to connect to the rACM may timeout depending on the state of the serial connection. In this instance, it may be necessary to rerun the command a second time.

A.2 Enable Host Network Operations

Following a successful power-up, the following occurs:

- The host defaults to SLEEP_OVERRIDE
- The host-to-node interface defaults to PASS-THROUGH

This configuration allows direct access to the host and node but prevents any host-initiated OTA network operations.

To enable host-initiated OTA network operations while preventing the host from sleeping, the following command lines are used:

```
python ctrl.py -d /dev/ttyUSB0 HOST_OPERATING_MODE 1
python ctrl.py -d /dev/ttyUSB0 HOST_RESET_NODE 0x2D653723
```

NOTE: Preventing the host from sleeping is desirable during host initiation/node configuration and test because it eliminates the need to wake the host with the magnetic Reed switch before issuing commands. Additional power consumption results from the host operating continuously. Care should be taken not to deploy a battery-powered host with SLEEP_OVERRIDE enabled.

A.3 Configure AP List and Join the Network

Node Monitor is a tool with a graphical user interface that is used to input the AP channel, re-use code, and monitor the rACM as it joins a network AP.

1. To start up the Node Monitor, type the following command at the command line:

   ```
   python ulp_node_monitor.py  -d /dev/ttyUSB0
   ```

2. Click the ***Connect Test PC to eNode*** button located in the center of the screen under the ***eNode Communication*** tab.
3. Change ***Logger Control*** (located at the top and center of the screen) from "unknown**"** to **"**frame stats verbose**.**"
4. Open the ***Flash Config Params*** page by selecting the tab (bottom center).
5. Click the ***Read Configuration from Flash*** button (top center).
6. Verify/edit the following fields:
   - **System ID:** Verify that this value matches the value displayed on the EMS Gateway page.
   - **Channel & Reuse Code:** Start at Candidate #0, fill in both the channel and re-use code for the deployed AP, and leave the unused channels set to 255.
7. Click the ***Write Configuration to Flash*** button (top center) and confirm that the response, "Config File written to flash," is written to the log window (top).
8. Open the ***ORW Graphs*** page by selecting the tab (bottom left).
9. Force a rejoin by changing the ***Over the Air Link*** (upper left) to "OFF" and then to "ON**.**"
10. Verify ***System State*** (upper center) transitions from "SCANNING" to "TRACK" to "JOINED" over the next few minutes. During the process, the following occurs:
    - Ten green bars will briefly be seen in the "Finger Energy" and "Finger Passing AFCs" graphs
    - The "RSSI" and "Spreading Factor" graphs will update accordingly
11. To close the Node Monitor, click on the window frame "X."

    NOTE: ***DO NOT*** use the ***Disconnect Test PC from eNode*** button on the ***Node Communication*** page to exit the Node Monitor. Disconnecting the Test PC from the Node can disable the host-to-node interface leaving the rACM in a non-operational state.

Node Monitor can be used at any time to monitor the OTA network status of the rACM. If the rACM host is not operating in the SLEEP_OVERRIDE mode, it may be necessary to wake the host using the magnetic Reed switch before modifying fields using the Node Monitor.

A.4 Host Logger

The host application logger (see section 6.2) parses the host-to-node traffic and is invoked using the following command line:

```
python –u host_app_logger.py  -d /dev/ttyUSB0 | tee racm_host_log.txt
```

NOTES:

1. The –u option and the UNIX "tee" command echo the logger output to the screen and to the specified file.
2. Depending on the state of the host, several minutes can elapse before any status is logged. A quick test of both the logger and the state of the rACM can be run by doing the following:
   a. Swipe the magnetic wand (see section 5.3) across the Reed switch
   b. Verify that both the logger (and later the EMS/On-Ramp Total Reach web pages; see section 5.4), log the resulting OTA alarm.

   Alarm hysteresis limits the rate of back-to-back OTA alarm generation.

A.5 Useful Commands

Additional command line examples are listed in the following table.

Table 4. Useful Commands

Command	Description
python host_app_ctrl.py -d /dev/ttyUSB0 HOST_VERSION	Get host version
python host_app_ctrl.py -d /dev/ttyUSB0 RESET_DEVICE	Reset rACM to factory defaults
python host_app_ctrl.py -d /dev/ttyUSB0 SET_UART_TIMEOUT 30	Set Wand Timeout to 30 seconds
python host_app_ctrl.py -d /dev/ttyUSB0 SET_ READ_INTERVAL 0 ASYNC ASYNC	Set Read Interval to 15 minutes (enum 0) starting when the rACM joins the network (async)
python host_app_ctrl.py -d /dev/ttyUSB0 SET_ READ_INTERVAL 2 45 0	Set Read Interval to 1 hour (enum 2) at 45 minutes past the hour
python host_app_ctrl.py -d /dev/ttyUSB0 SLEEP_OVERRIDE OFF	Disable SLEEP_OVERRIDE and put host into normal sleep mode
python sw_upgrade.py -d /dev/ttyUSB0 micronode.bin	Upgrade uNode Firmware
python host_app_ctrl.py -d /dev/ttyUSB0 HOST_SW_UPGRADE host_app_k20.bin	Upgrade Host Firmware
python host_app_ctrl.py -d /dev/ttyUSB0 DUMP_EXCEPTIONS	Dump exception buffer

Appendix B OTA Payload Examples

This chapter is provided as a guide to parsing Uplink OTA payload packets as they are received at the backend. The payload examples included in this chapter are:

- Configuration Write (Read Interval)
- Configuration Response (auto-generated by Network Discovery Connect feature)
- Sensor Data History payload for an example Pulse Counter.

NOTES:

1. The Sensor Data History packs message payload fields using Little-Endian byte ordering (i.e., least significant byte first). This is an especially important consideration for multi-byte bit packed fields (e.g., the bit-packed time stamp).
2. All OTA packets used in this example have been captured by an operational rACM unit joined to an On-Ramp Wireless field-engineering test network. Using the On-Ramp Total View application, the raw hex data has been captured for each of the use-cases.
3. For On-Ramp Wireless System Release 1.4, all OTA packets are padded to an 8-byte boundary. Padded bytes are conveniently discarded/ignored in the following parsing examples presented in this appendix.

B.1 Configuration Write Example

In this example, an rACM unit was power-cycled. After acquiring/joining the network, the Downlink tab in the On-Ramp Total View application is used to generate a Configuration Write OTA packet to change the Read Interval (ORWID 7, see Table 3) as the following screen capture from the On-Ramp Total View application shows:

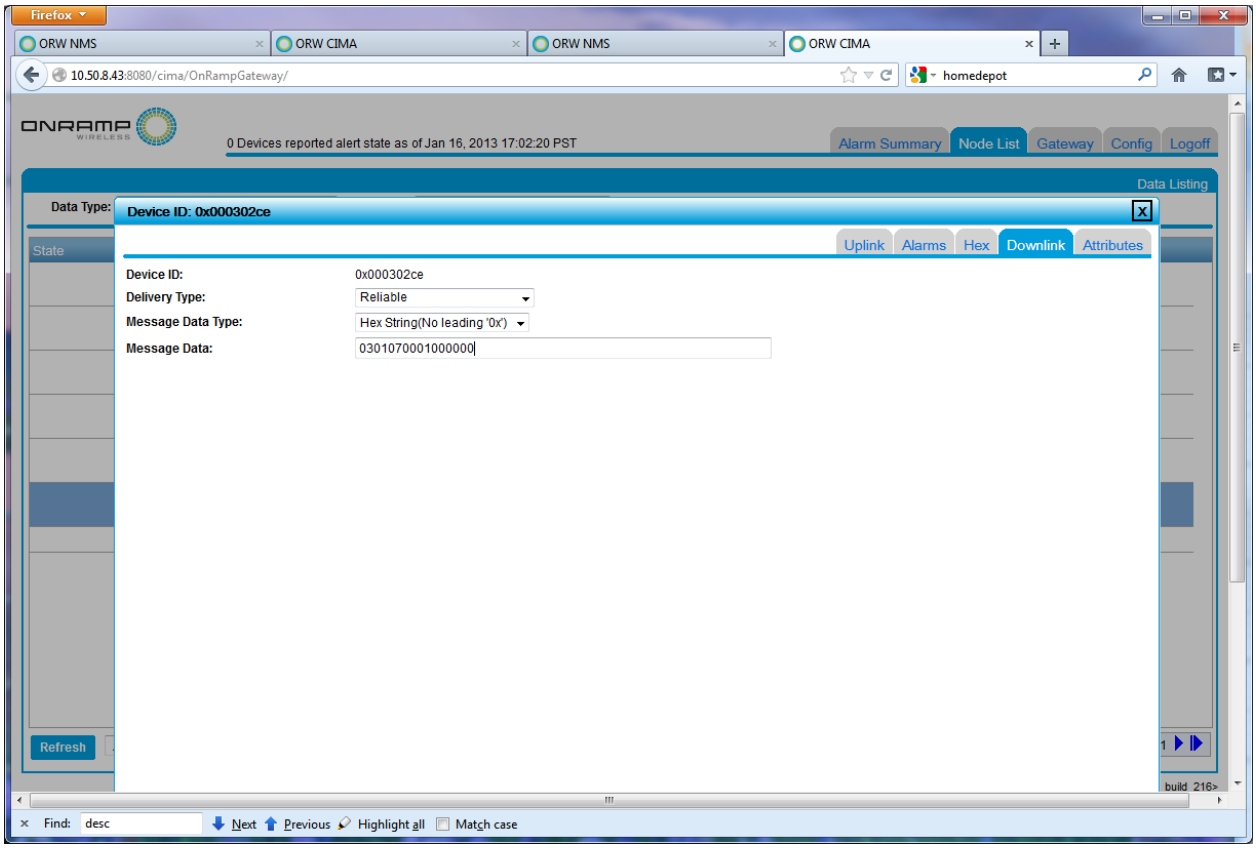
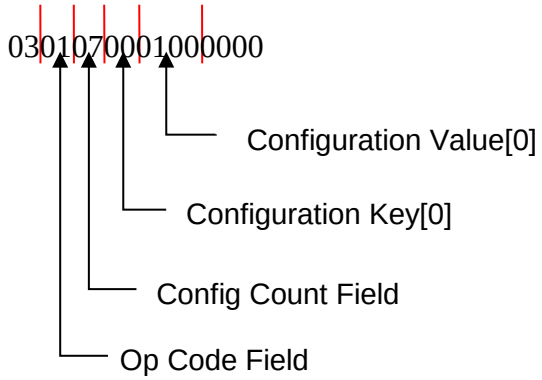


Figure 4. On-Ramp Total View Configuration Write Example

Expanding the highlighted Configuration Write OTA packet, the raw hex data for the Configuration Write OTA payload (see *rACM Software High Level Design document, 013-0008-00*) is broken down into the following data stream:



Taking into consideration the big-endian byte ordering and the Configuration Write Message format (see *rACM Software High Level Design document (013-0008-00)*), the Configuration Write message can be deconstructed into the following fields:

- **Op Code –** 0x03 (Configuration Write Request DL Message)
- **Config Count –** 0x01 (1 Configuration Key-Value Pair)
- **Config Key[0] –** 0x07 (ORW ID#7: Read Interval)
- **Config Value[0] –** 0x0001 (Reads every 30 minutes with zero offset)

NOTE: Configuration write value fields are entered into the On-Ramp Total View Downlink generation box in big-endian byte order while Uplink fields use little-endian byte ordering.

B.2 Configuration Response Example

In this example, a rACM unit, configured with Sleep Override on, was power-cycled. After the acquiring/joining the network, the network discovery connect feature auto-generated to the Configuration Response OTA packet as the following screen capture from the On-Ramp Total View application shows:

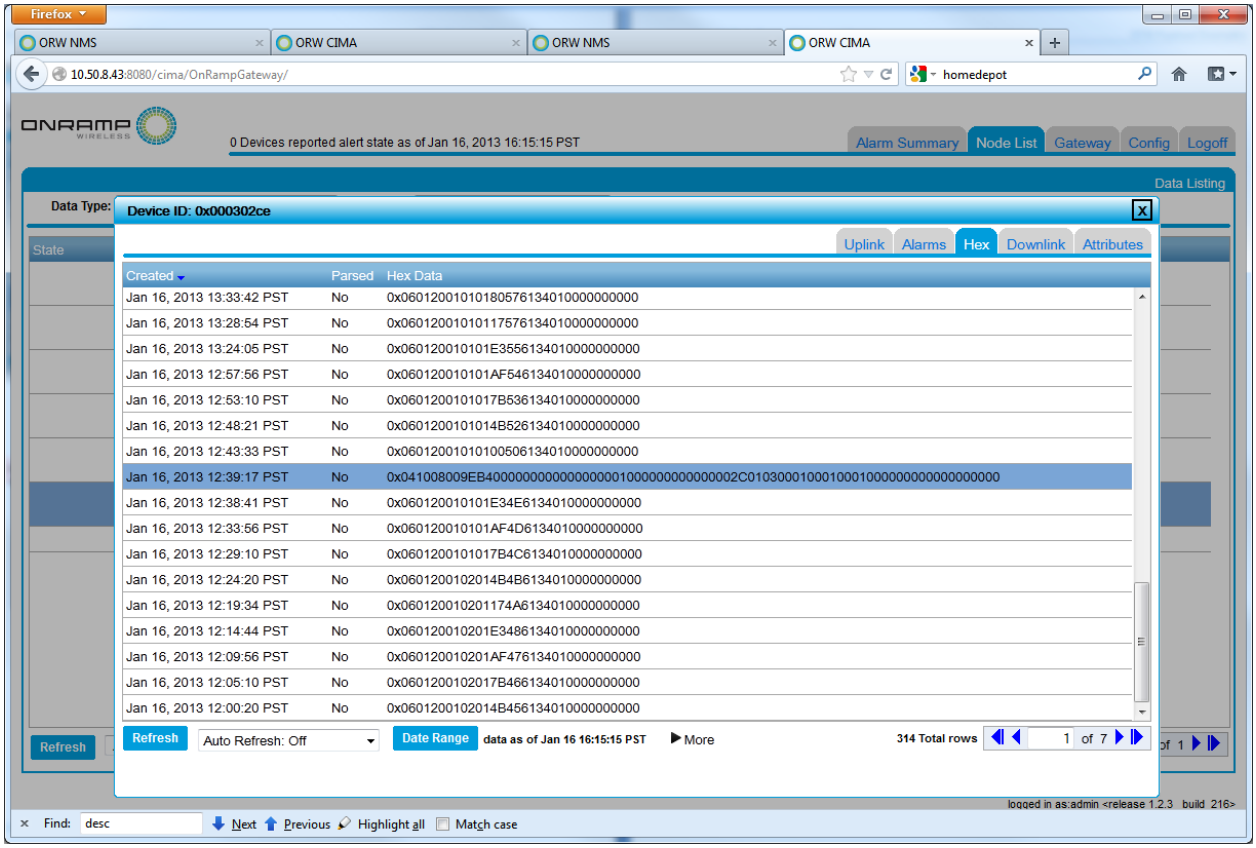
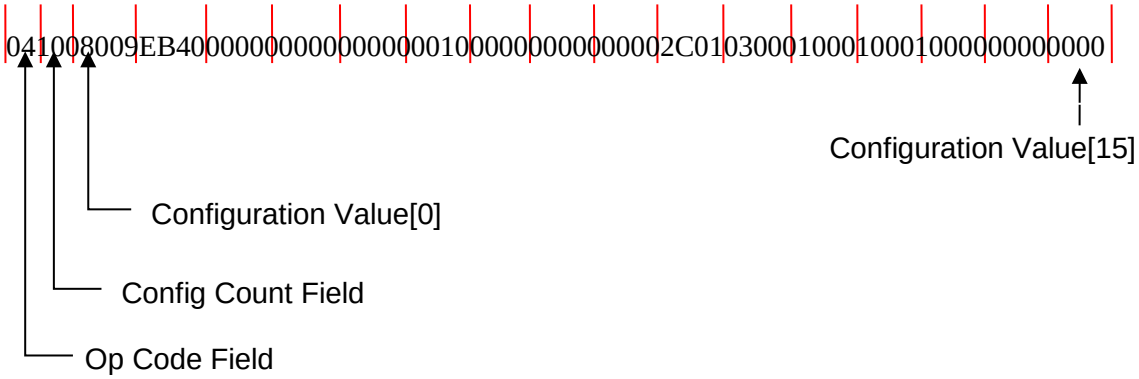


Figure 5. On-Ramp Total View Configuration Response Example

Expanding the highlighted OTA packet, the raw hex data for the Configuration Response OTA payload (see *rACM Software High Level Design document (013-0008-00)*) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Configuration Response Message format (see *rACM Software High Level Design document (013-0008-00)*), the Configuration Response message can be deconstructed into the following fields:

- **Op Code –** 0x04 (Configuration Read Response UL Message)
- **Config Count –** 0x10 (16 Configuration Key-Value Pairs)
- **Config Value[0] –** 0x0008 (ORW ID#1: UART Timeout = 8 seconds)
- **Config Value[1] –** 0xB49E (ORW ID#2: UART Passcode[15:0] = 0xB49E)
- **Config Value[2] –** 0x0000 (ORW ID#3: UART Passcode[31:16] = 0x0000)
- **Config Value[3] –** 0x0000 (ORW ID#4: UART Passcode[47:32] = 0x0000)
- **Config Value[4] –** 0x0000 (ORW ID#5: UART Passcode[63:48] = 0x0000)
- **Config Value[5] –** 0x0000 (ORW ID#6: UART Password Disable Threshold = disabled)
- **Config Value[6] –** 0x0001 (ORW ID#7: Read Interval = Reads every 1 hour with zero offset)
- **Config Value[7] –** 0x0000 (ORW ID#8: Low Battery Alarm Threshold = disabled)
- **Config Value[8] –** 0x0000 (ORW ID#9: Device ID[15:0] = 0x0000)
- **Config Value[9] –** 0x0000 (ORW ID#10: Device ID[31:16] = 0x0000)
- **Config Value[10] –** 0x012C (ORW ID#11: Alarm Hysteresis = 300 seconds)
- **Config Value[11] –** 0x0003 (ORW ID#12: POR Default Host Interface State = App Override)
- **Config Value[12] –** 0x0001 (ORW ID#13: POR Sleep Override State = On)
- **Config Value[13] –** 0x0001 (ORW ID#14: Network Discovery Connect = On)
- **Config Value[14] –** 0x0001 (ORW ID#15: Provision Quick Start = On)
- **Config Value[15] –** 0x0000 (ORW ID#16: RHT Support = Off)

B.3 Pulse Counter Data Example

In this example, an rACM unit, configured to monitor a pulse counter digital input, was power-cycled and allowed to acquire/join the network. Once joined, the unit entered steady-state operations taking pulse measurements every 15-minutes and generating OTA read records at a 2-hour Uplink Interval as the following screen capture from the On-Ramp Total View application demonstrates:

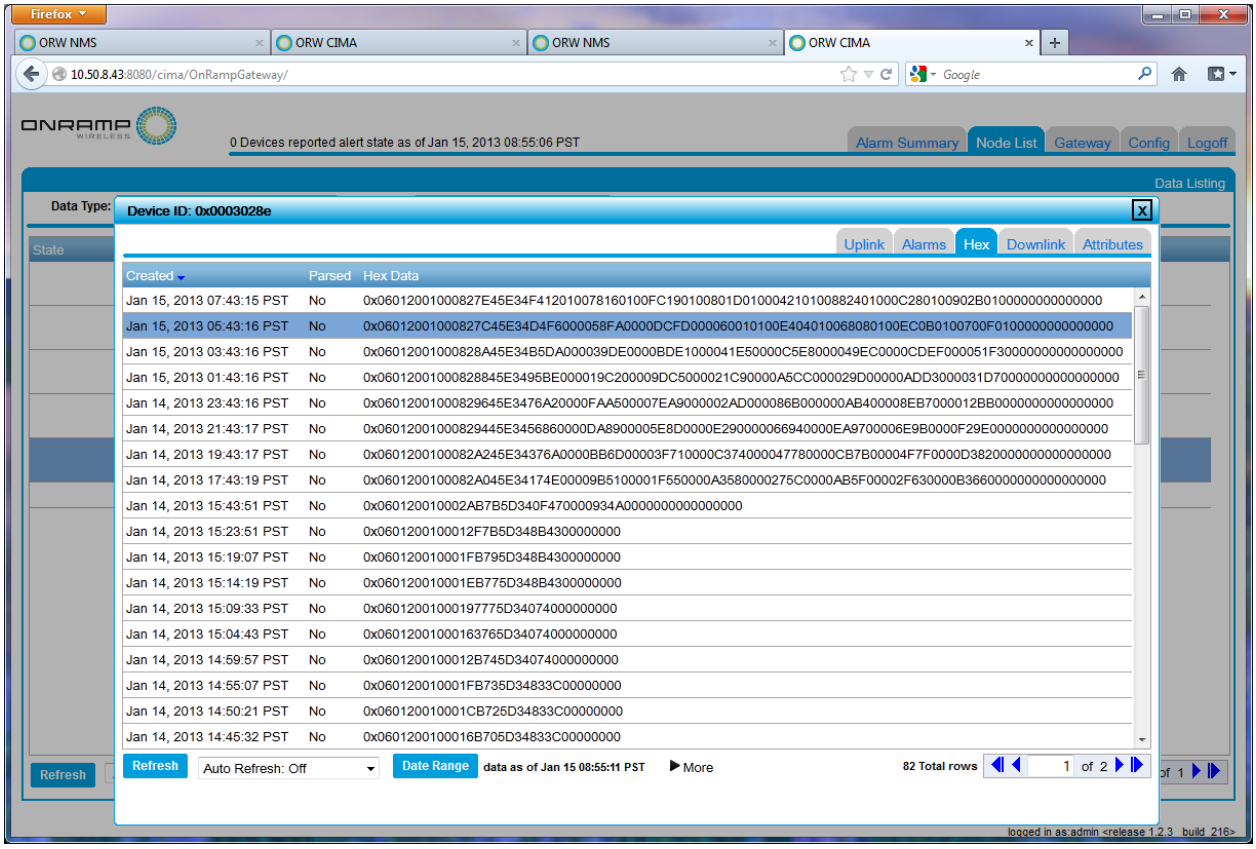
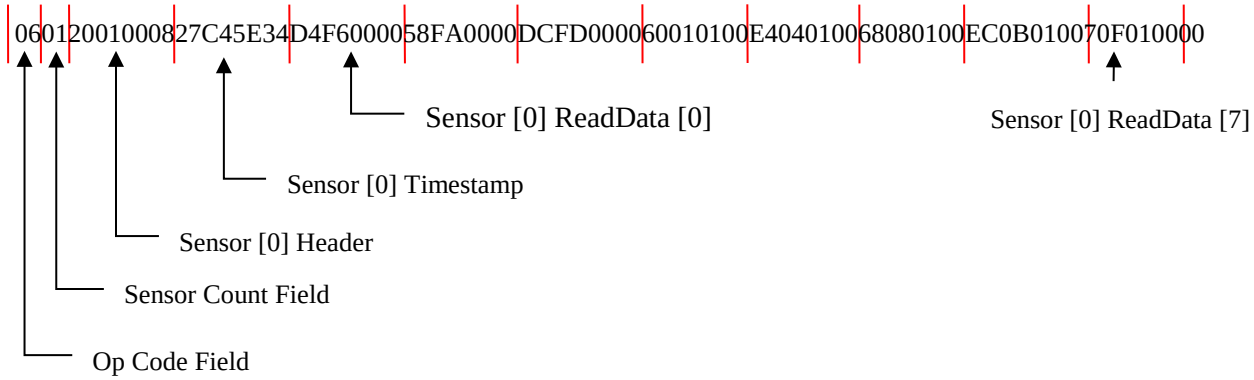


Figure 6. On-Ramp Total View Pulse Count Sensor Data Example

Expanding the highlighted OTA packet, the raw hex data for the Sensor Data OTA payload (see *rACM Software High Level Design document, 013-0008-00*) is broken down into the following data stream:



Taking into consideration the little-endian byte ordering and the Sensor Data Message format (see *rACM Software High Level Design document, 013-0008-00*), the Sensor Data OTA payload can be deconstructed into the following fields:

- **Op Code –** 0x06 (Configuration Read Response UL Message)
- **Sensor Count –** 0x01 (A single sensor record)
- **Sensor[0] Header** - 0x08000120 (bit packed Header field)
  - Sensor Interface ID (see Table 17) - 0x0 (Sensor Index)
  - Sensor Interface Type (see Table 18) - 0x012 (32-bit unsigned pulse counts)
  - Read Interval – 0x00 (15-minute reads)
  - Read Count - 0x08 (8 chained read records)
- **Sensor[0] Timestamp** - 0x345EC427 (bit packed GMT timestamp for first read record):
  - Seconds - 0x27 (39 sec)
  - Minutes – 0x10 (16 min)
  - Hour – 0x0C (12 hour)
  - Day – 0x0F (15th day)
  - Month – 0x01 (January)
  - Year Offset from 2000 – 0x0D (2013)
- Sensor[0] ReadData[0] - 0x0000F6D4 (63188 pulses)
- Sensor[0] ReadData[1] – 0x0000FA58 (64088 pulses)
- Sensor[0] ReadData[2] – 0x0000FDDC (64988 pulses)
- Sensor[0] ReadData[3] – 0x00010160 (65888 pulses)
- Sensor[0] ReadData[4] – 0x000104E4 (66788 pulses)
- Sensor[0] ReadData[5] – 0x00010868 (67688 pulses)
- Sensor[0] ReadData[6] – 0x00010BEC (68588 pulses)
- Sensor[0] ReadData[7] – 0x00010F70 (69488 pulses)

Extrapolating further, the deconstructed sensor data reported to the backend consists of a chained record of 8 pulse count measurements spaced 15-minutes apart starting at Jan 15th, 2013 12:16:39 GMT.

Appendix C Abbreviations and Terms

Abbreviation/Term	Definition
ACM	Application Communication Module
AP	Access Point. The On-Ramp Total Reach network component geographically deployed over a territory.
CIMA	Critical Infrastructure Monitoring Application. The network component that passes data from the Gateway to the associated upstream databases. This product name is transitioning to On-Ramp Total View.
EMS	Element Management System. The network component that provides a concise view of controls and alarms on the On-Ramp Total Reach network.
FTM	Flex Timer Module. A timer block on the Kinetis MCU.
GPIO	General Purpose Input/Output. A pin on the Kinetis MCU.
HAL	Hardware Abstraction Layer. A software functional block on the ACM used to implement drivers for all node interfaces.
HLD	High Level Design
MCU	Microcontroller Unit
microNode	A second generation, small form factor, wireless network module developed by On-Ramp Wireless that works in combination with various devices and sensors and communicates data to an Access Point. Also referred to as uNode.
Node	The generic term used interchangeably with an end point On-Ramp Wireless device.
NVM	Non-Volatile Memory. In the context of the Kinetis, memory that is persistent across the different Low Leakage power modes.
On-Ramp Total View	The network component that passes data from the Gateway to the associated upstream databases. Formerly known as CIMA.
On-Ramp Total Reach	The On-Ramp Wireless' proprietary wireless communication technology and network.
ORW	On-Ramp Wireless
OTA	Over-the-Air
POR	Power-on Reset
PWM	Pulse Width Modulation. A waveform with a configurable duty cycle.
rACM	reference Application Communication Module
RHT	Reliable Host Transfer. An optional retry protocol used on all serial data transfers over the Host Interface.
Rx	Receive/Receiver
SDU	Service Data Unit
Tx	Transmit/Transmitter
UART	Universal Asynchronous Receiver/Transmitter
UI	Uplink Interval. The scheduled uplink interval where the rACM periodically sends accumulated sensor read data.
uNode	Shorthand for microNode. The microNode is a second generation, small form factor, wireless network module developed by On-Ramp Wireless that works in combination with various devices and sensors and communicates data to an Access Point.