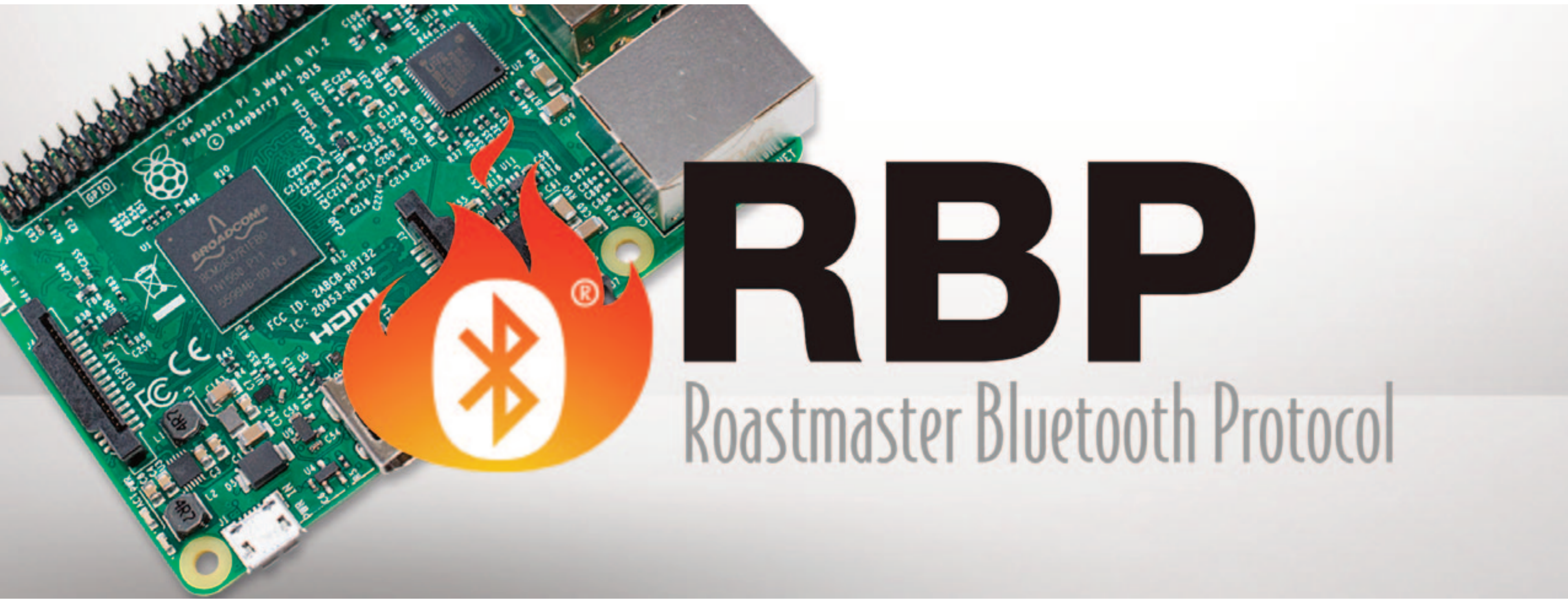


Roastmaster Bluetooth LE Protocol

Datasheet

RBP 2.0
Roastmaster iOS 14.0
3/30/21

Introduction to RBP and Server Devices

What is Roastmaster Bluetooth LE Protocol (RBP)?

RBP is a protocol we created specifically for Roastmaster that allows any circuit board equipped with a CPU and Bluetooth LE hardware to collect and send data from on-board sensors directly to Roastmaster iOS.

Historically, Roastmaster has relied on third party manufacturers to produce both the hardware, as well as the software API that Roastmaster would use to communicate with that hardware. If those APIs contained bugs, or didn't function correctly, there was nothing that could be done with Roastmaster to resolve it.

RBP was designed from the ground up for coffee roasters, and tailored for Roastmaster's feature set, and is now built directly into Roastmaster's code. Anyone can implement RBP on a variety of hardware. This removes the reliance on third party vendors, and lets anyone design and build a probe server for very little cost that can easily communicate with Roastmaster using this flexible protocol.

What is a Protocol?

A protocol is really nothing more than a language two entities agree to "speak" in order to communicate with each other. If you speak English, you can understand the information in this datasheet because you understand the English language. English is, essentially, the "protocol" of this datasheet.

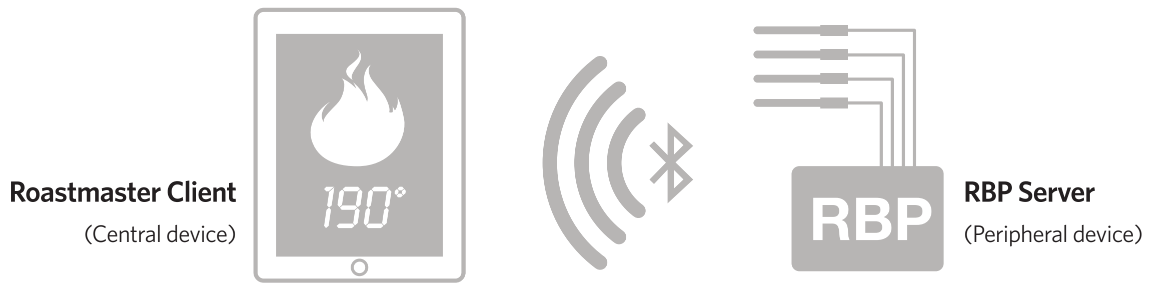
The same is true with software. As long as two applications or devices speak the same "language", they can talk to each other and share information. Since Roastmaster now "speaks" RBP, any circuit board with a CPU and Bluetooth LE capabilities can be programmed to talk directly to Roastmaster using the language of RBP.

What is an RBP Server Device?

An RBP device (called the **Server**[1]) is used to collect and send readings to Roastmaster (the **Client**[1]). A server device is comprised of the hardware required to gather the data, running software to interpret and make this information available via its Bluetooth LE GATT to Roastmaster.

A server can be a DIY project, based on a popular SBC (Single Board Computer), such as Arduino, Raspberry Pi, or others. Or, it can be a professionally designed circuit board unit, either standalone or factory fitted into a roaster, that is coded by the manufacturer to provide out-of-the-box support for Roastmaster iOS via RBP.

Whatever the specifics, as long as it can run software, read a sensor, and update those readings to a Bluetooth LE GATT, Roastmaster iOS can receive those readings, and interact with the device wirelessly.



[1]Bluetooth LE nomeclature typically describes this Client/Server relationship as one of a Peripheral/Central–where the Client is the Central device, and the Server is the Peripheral.

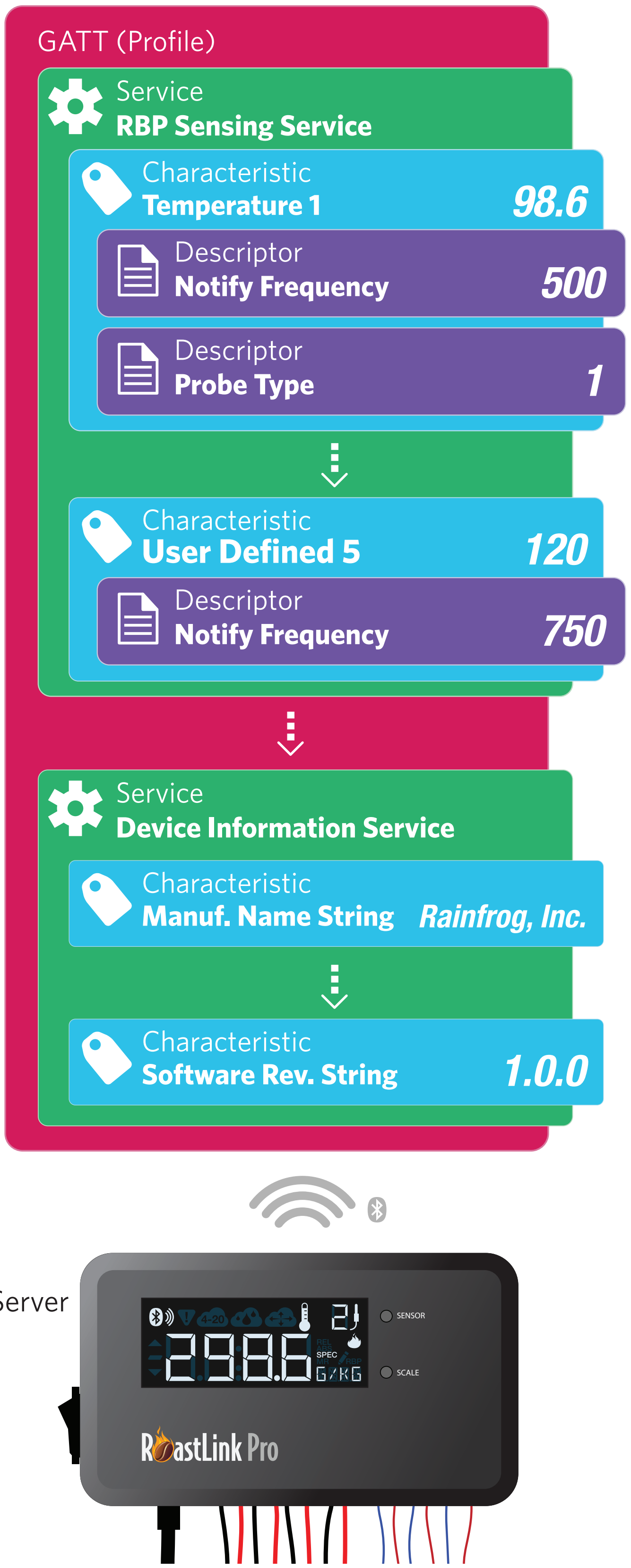
Bluetooth LE GATT

A Bluetooth LE server device contains and manages a structured table of hierarchical data that it exposes to client devices that connect to it. This table of data, and the ways in which it can be used by client devices, are collectively referred to by the term Generic Attribute Profile, or **GATT** for short.

The profile itself is an abstract concept. It refers simply to the series of data objects that reside at the top level of the data hierarchy, called Services. Each Service contained in the GATT encapsulates nested details about functionality that is specific to part of the physical device, or behavior that is similar in logical function, called Characteristics.

Characteristics manage state information about the Service, such as sensor data. Each Characteristic, in turn, has the capability to own and manage a set of Descriptors, which further annotate Characteristic data, or allow clients to change certain functionality of the Characteristic.

Collectively, these three data types compose the GATT framework, and define the procedures and functionality of Services and their data, including how connected clients that are interested in this data can discover, read, write, observe and interact with them.



Services, Characteristics and Descriptors

GATT Services

A Service is a container for data elements that are logically related in functionality, each of which is held in a nested container called a Characteristic. A Service can be thought of as the owner of the Characteristics inside it, each of which is also the owner of their respective data items. Often a Service represents a particular feature of a device, like buttons or a sensor. Other times, it groups a collection of associated functions into one logical container.

The latter is the arrangement of the chief Service of RBP, called the **RBP Sensing Service**. This Service, designed by Rainfrog, Inc. for coffee roasters, encompasses all of the sensing related functionality pertaining to coffee roasting defined within the RBP Protocol. It contains a large number of Characteristics for a wide array of sensing types, each of which holds individual sensor data. In addition, these Characteristics each hold their respective Descriptors, which allow Roastmaster to interact with the owning Characteristic, and instruct it on how to interpret and send that data.

Another example is a Service defined by the Bluetooth Special Interest Group (Bluetooth SIG), called the **Device Information Service**, which contains various items of information about the server hardware itself, such as its manufacturer and serial number. RBP implements the Device Information Service in order to communicate hardware and software details to Roastmaster. In addition, this allows Roastmaster to have direct access to the device serial number, which enables Roastmaster to differentiate among similar Bluetooth servers, and safely use data from multiple RBP servers simultaneously.

GATT Characteristics

Characteristics are contained and owned by a Service, and are items of data which relate to a particular internal state of the device, or, in the case of the RBP Sensing Service, the current reading of a particular sensor.

Characteristics that support notification can be subscribed to by a client. This is the way in which the RBP Sensing Service operates. Once subscribed to a Characteristic, the client will be notified by the server whenever the value of that Characteristic changes.

RBP provides a host of Characteristics, including Temperature, Air Pressure, Humidity, 4-20 mA sensors and user defined Characteristics. Each of these can store the current reading of their respective sensor in the appropriate scale units. For example, the Temperature 1 Characteristic of the RBP Sensing Service holds the current temperature reading of the first temperature sensor, which is stored and transmitted in degrees Celsius. Whenever a new value is published to the GATT, Roastmaster will be notified if it is currently subscribed to this Characteristic. The Temperature 1 Characteristic also provides Descriptors to allow you to customize this behavior via Roastmaster.

GATT Descriptors

Descriptors are contained and owned by a Characteristic, and contain metadata which either augments certain details relating to the owning Characteristic, or allows configuration of one or more behaviors of that Characteristic.

An example of an RBP Descriptor is the Notify Frequency Descriptor of any Characteristic contained in the RBP Sensing Service. The value of this Descriptor can be changed from within Roastmaster to control how often the sensor reading for that particular Characteristic should be broadcast to Roastmaster via GATT updates.

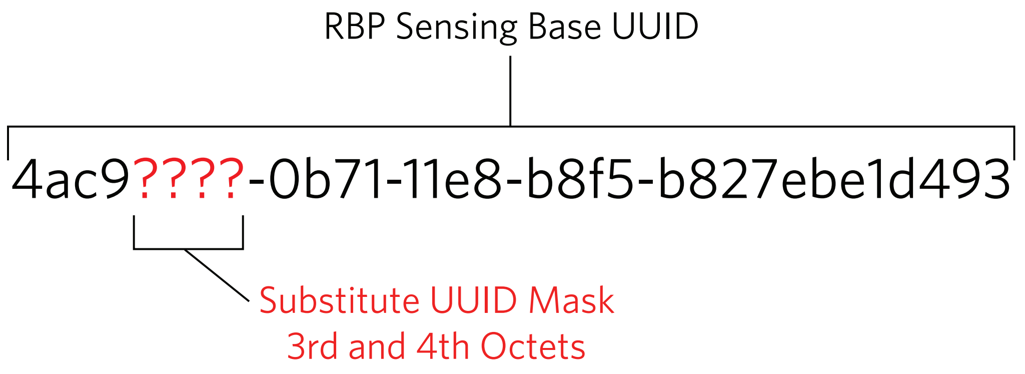
Another example is the Probe Type Descriptor of any of the Temperature Characteristics of the RBP Sensing Service. The value of this Descriptor can be changed from within Roastmaster to tell the server what type of probe you have attached to the applicable hardware input–effectively telling it how to interpret those probe readings, e.g. K Type thermocouple, J Type thermocouple, etc.

Bluetooth LE UUIDs

Services, Characteristics and Descriptors are identified and addressed by their Universally Unique IDentifier (UUID). A UUID is a 128 bit number expressed in hexadecimal notation in the format below, where each x is a hex digit:

xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

The UUIDs of all data objects contained in the RBP Sensing Service share a common base UUID: 4ac9????-0b71-11e8-b8f5-b827ebe1d493. The string "????" represents the four digits that should be replaced by the "**UUID Mask**" that is associated with the Service, Characteristic or Descriptor that you are trying to address. You can find UUID Mask values in the "RBP GATT Data Reference" chart.



For example, if you want to address the Temperature 1 Characteristic, you would find the UUID mask 0001, and apply it to the base UUID, resulting in 4ac90001-0b71-11e8-b8f5-b827ebe1d493.

Note: *If you have done any past GATT programming, it's important to note that RBP Sensing Service UUIDs must ALWAYS be accessed by their full 128-bit UUID. The Bluetooth SIG has defined 16 bit shortcuts for its core UUIDs, however, these are only usable for items defined in the Bluetooth SIG core set. All other custom items, including the RBP Sensing Service and its contained hierarchy, must be accessed with the full 128 bit UUID.*

RBP GATT Data Reference

Name	UUID Mask	GATT Perm.	GATT Data Type	Raw Data Type	Encoding Method	Units
RBP Sensing Service	**0000**					
Temperature 1	**0001**	**N**	**Int32**	**Double**	**Float×10^2**	**°C**
Probe Type	001a	RW	UInt8	UInt8	-	Probe Type Enum
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Temperature 2	**0002**	**N**	**Int32**	**Double**	**Float×10^2**	**°C**
Probe Type	001a	RW	UInt8	UInt8	-	Probe Type Enum
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Temperature 3	**0003**	**N**	**Int32**	**Double**	**Float×10^2**	**°C**
Probe Type	001a	RW	UInt8	UInt8	-	Probe Type Enum
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Temperature 4	**0004**	**N**	**Int32**	**Double**	**Float×10^2**	**°C**
Probe Type	001a	RW	UInt8	UInt8	-	Probe Type Enum
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Temperature 5	**0005**	**N**	**Int32**	**Double**	**Float×10^2**	**°C**
Probe Type	001a	RW	UInt8	UInt8	-	Probe Type Enum
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Pressure 1	**0006**	**N**	**Int32**	**Double**	**Float×10^2**	**ATM**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Pressure 2	**0007**	**N**	**Int32**	**Double**	**Float×10^2**	**ATM**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Pressure 3	**0008**	**N**	**Int32**	**Double**	**Float×10^2**	**ATM**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Pressure 4	**0009**	**N**	**Int32**	**Double**	**Float×10^2**	**ATM**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Pressure 5	**000a**	**N**	**Int32**	**Double**	**Float×10^2**	**ATM**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Humidity 1	**000b**	**N**	**Int32**	**Double**	**Float×10^2**	**(Hum. Scale Descr.)**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Humidity Scale	001c	RW	UInt8	UInt8	-	Humidity Type Enum
Humidity 2	**000c**	**N**	**Int32**	**Double**	**Float×10^2**	**(Hum. Scale Descr.)**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Humidity Scale	001c	RW	UInt8	UInt8	-	Humidity Type Enum
Humidity 3	**000d**	**N**	**Int32**	**Double**	**Float×10^2**	**(Hum. Scale Descr.)**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Humidity Scale	001c	RW	UInt8	UInt8	-	Humidity Type Enum
Humidity 4	**000e**	**N**	**Int32**	**Double**	**Float×10^2**	**(Hum. Scale Descr.)**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Humidity Scale	001c	RW	UInt8	UInt8	-	Humidity Type Enum
Humidity 5	**000f**	**N**	**Int32**	**Double**	**Float×10^2**	**(Hum. Scale Descr.)**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10^2	Seconds
Humidity Scale	001c	RW	UInt8	UInt8	-	Humidity Type Enum

RBP GATT Data Reference Continued

Name	UUID Mask	GATT Perm.	GATT Data Type	Raw Data Type	Encoding Method	Units
420 mA Sensor 1	**0010**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
Min Value	001d	RW	Int32	Double	Float×10²	-
Max Value	001e	RW	Int32	Double	Float×10²	-
Quantization	001f	RW	UInt32	Double≥0	Float×10²	-
420 mA Sensor 2	**0011**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
Min Value	001d	RW	Int32	Double	Float×10²	-
Max Value	001e	RW	Int32	Double	Float×10²	-
Quantization	001f	RW	UInt32	Double≥0	Float×10²	-
420 mA Sensor 3	**0012**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
Min Value	001d	RW	Int32	Double	Float×10²	-
Max Value	001e	RW	Int32	Double	Float×10²	-
Quantization	001f	RW	UInt32	Double≥0	Float×10²	-
420 mA Sensor 4	**0013**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
Min Value	001d	RW	Int32	Double	Float×10²	-
Max Value	001e	RW	Int32	Double	Float×10²	-
Quantization	001f	RW	UInt32	Double≥0	Float×10²	-
420 mA Sensor 5	**0014**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
Min Value	001d	RW	Int32	Double	Float×10²	-
Max Value	001e	RW	Int32	Double	Float×10²	-
Quantization	001f	RW	UInt32	Double≥0	Float×10²	-
User Defined 1	**0015**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
User Defined 2	**0016**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
User Defined 3	**0017**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
User Defined 4	**0018**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
User Defined 5	**0019**	**N**	**Int32**	**Double**	**Float×10²**	**-**
Notify Frequency	001b	RW	UInt32	Double≥0	Float×10²	Seconds
Device Info. Service	**180a**					
Manuf. Name String	**2a29**	**R**	**UTF8**			
Model Number String	**2a24**	**R**	**UTF8**			
Serial Number String	**2a25**	**R**	**UTF8**			
Hardware Rev. String	**2a27**	**R**	**UTF8**			
Firmware Rev. String	**2a26**	**R**	**UTF8**			
Software Rev. String	**2a28**	**R**	**UTF8**			

N = Notify, R = Read, W = Write

NOTE: See "Data Formats" for details about encoding and decoding data between Raw and GATT data types.

Data Formats

GATT Data Types

Following are the data types for GATT data, i.e. data that will reside in the GATT, and be transferred back and forth between the server and client.

Int32:	Data represents a 32 bit signed integer, e.g. int32_t in C. (See notes about encoding and Byte Order)
UInt32:	Data represents a 32 bit unsigned integer, e.g. uint32_t in C. (See notes about encoding and Byte Order)
UInt8:	Data represents an 8 bit unsigned integer, e.g. uint8_t in C

Raw Data Types

Following are the data types for raw data, i.e. real world data, such a sensor reading, or a descriptor value that you work with in code.

Double	Data represents a signed decimal number, e.g. double in C
Double≥0	Data represents a decimal number that is greater than or equal to 0
UInt8:	Data represents an 8 bit unsigned integer, e.g. uint8_t in C

Encoding Raw Data to GATT Data and Vice Versa

If the Raw Data Type and GATT Data Type in the "RBP GATT Data Reference" do not match for a particular entity, it will be necessary to encode raw data from your server to the correct type of GATT data according to its "Encoding Method" before publishing it to your GATT. Conversely, when reading data from the GATT, you will need to decode that data to the raw data value to work with in your code.

For example, since the concept of transferring floating point decimal numbers from one platform to another is risky at best, RBP uses an integer of some type in its GATT to represent that decimal number, which is encoded via the "Floatx10^2" encoding method.

Float×10^2 Encoding

This type of encoding shifts the decimal point two places to the right, rounding the result to an integer. This allows the value to be transmitted as an integer and still retain two decimal places of precision. This is a platform-independent and safe way to transmit data in a server-client relationship.

For example, a sensor reading of 123.456 would be multiplied by 100 to become 12345.6, and rounded to the nearest integer value of 12346. This integer would be written to the server's GATT. In Roastmaster, the value of 12346 would be read from the remote Server's GATT, and would be divided by 100.0 to yield 123.46.

Formula example: For an entity whose GATT data type is Int32, this encoding would be accomplished with the C code: **(int32_t)round(a_double * 100.0)**. This code multiplies the raw data value by 100, rounds the result to the nearest integer, and casts it to an int32_t type.

Byte Order

The Bluetooth SIG specifies that multibyte data values within the GATT table are encoded as Least Significant Byte first (LSB), i.e. Little Endian. RBP conforms to this convention, and Roastmaster will interpret all multibyte values obtained through the GATT as Little Endian data.

Constant Enums

The following enumerations illustrate the UInt8 constants that are used to represent real world values or settings in RBP.

Temperature Probe Type Constants

Type K Thermocouple	1
Type J Thermocouple	2
Type N Thermocouple	3
Type R Thermocouple	4
Type S Thermocouple	5
Type T Thermocouple	6
Type E Thermocouple	7
Type B Thermocouple	8

Humidity Type Constants

Relative %	1
Absolute g/m3	2
Specific g/kg	3
Mixing Ratio g/kg	4

Uniquing Your RBP Server via the Device Information Service

The Device Information Service exposes manufacturer and/or vendor information about a server. With the exception of the Serial Number String Characteristic, this information is purely informational, and is read by Roastmaster and reported in a corresponding Probe Details view:

- Manufacturer Name String
- Model Number String
- Serial Number String
- Hardware Revision String
- Software Revision String
- Firmware Revision String

Serial Number String Characteristic

Roastmaster can connect to an unlimited number of RBP Servers at the same time, and use any data they provide concurrently in one or more roasts.

To equip users with the ability to do this, a unique serial number should be available to Roastmaster to differentiate among similar devices. Roastmaster CANNOT differentiate devices by MAC address - this functionality can only be achieved in one of two ways

Uniquing via the Advertised Device GAP Name

You can embed a unique serial number in the advertised GAP device name. This method works fine, if the combined string can be kept to the common limit of 14 bytes or shorter. Since this doesn't provide much flexibility for long serial numbers, it is not advisable.

Uniquing via the Device Information Service

Instead, you can publish a unique serial number in the Serial Number String Characteristic of the Device Information Service. This is the preferred method. When Roastmaster encounters an RBP server device, it first retrieves its advertised GAP name. It then looks to see if the device advertises the Serial Number String Characteristic of the Device Information Service. If so, it briefly connects to it, retrieves the unique serial number, and appends it to the advertised GAP name–forming the full name by which it will be identified.

For example, consider an RBP Server with an Advertised name of "My Server". If this device also advertises and exposes the Serial Number String Characteristic of the Device Information Service, Roastmaster will briefly connect to it once to retrieve this information. If the Serial Number String is "12345", Roastmaster will thereafter consider the device to be named "My Server 12345", which will unique this particular device independent of similar devices. So Roastmaster can simultaneously connect to and concurrently read and write data to and from several similar devices, e.g. "My Server 12345" and "My Server 23456", even if they are identical in terms of hardware.

Descriptor Functions

RBP Descriptors provide the ability for you to control how the server processes sensor readings and communicates these readings to Roastmaster.

Notify Frequency

This Descriptor controls how often the RBP Server device should update its internal GATT table with the sensor reading of the Descriptor's owning Characteristic, thereby notifying Roastmaster of the change. This can be used as a means of reducing unnecessary traffic on a server with many sensors.

Server Action: When a Server registers that a change has been written to its GATT for this Descriptor from Roastmaster, it should use the new value to control how many seconds transpire between GATT updates for the owning Characteristic.

Probe Type

This Descriptor controls how the probe sensor readings should be interpreted for the owning Characteristic.

Server Action: When a Server registers that a change has been written to its GATT for this Descriptor from Roastmaster, it should use the new value to translate subsequent probe readings for the owning Characteristic.

Humidity Scale

This Descriptor controls what scale is used for publishing humidity sensor readings in a server's GATT for the owning Humidity Characteristic. Roastmaster and RBP both support several humidity scales that are reliant on not only air moisture, but also air pressure and air temperature, e.g. Specific Humidity and Mixing Ratio. Since these values cannot be inferred by Roastmaster for scale conversion, a humidity sensor reading must be converted server-side to the desired scale before it is transmitted to Roastmaster. This Descriptor dictates the scale that should be applied to sensor readings before being written to the GATT.

Server Action: When a Server registers that a change has been written to its GATT for this Descriptor from Roastmaster, it should convert subsequent humidity readings to the new scale when publishing them to its GATT for the owning Characteristic.

Min Value

Characteristics designed to accept input from a 4-20 mA sensors need to know what value to map the minimum input to. For example, a 4-20 mA device that measures air pressure between .5 and 10 ATM should have its Min Value Descriptor set to .5. This tells the RBP server to map 0% (4 mA) to 0.5. Together with the Max Value, the RBP server can now map the entire range of values from .5 to 10 correctly.

Server Action: When a Server registers that a change has been written to its GATT for this Descriptor from Roastmaster, it should use the new low value (4 mA) when translating 4-20 mA device readings to publish to its GATT for the owning Characteristic.

Max Value

Characteristics designed to accept input from a 4-20 mA sensors need to know what value to map the maximum input to. For example, a 4-20 mA device that measures air pressure between .5 and 10 ATM should have its Max Value Descriptor set to 10. This tells the RBP server to map 100% (20 mA) to 10. Together with the Min Value, the RBP server can now map the entire range of values from .5 to 10 correctly.

Server Action: When a Server registers that a change has been written to its GATT for this Descriptor from Roastmaster, it should use the new high value (20 mA) when translating 4-20 mA device readings to publish to its GATT for the owning Characteristic.

Quantization

Characteristics designed to accept input from a 4-20 mA sensor are often used to supply data to a control curve in Roastmaster. Since Roastmaster alerts users to changes in control curves used as reference in a Profile during a roast, it is advantageous to keep the number of nodes to a minimum to simplify the roasting workflow.

For this reason, RBP provides a Quantization Descriptor for each 4-20 mA Characteristic. It instructs the server to lock values for the owning Characteristic to a grid of sorts, preventing miniscule changes from propagating unnecessary nodes in control curves.

Server Action: When a Server registers that a change has been written to its GATT for this Descriptor from Roastmaster, it should use this to quantize the actual value of the 4-20 mA sensor of the owning Characteristic.

Example: Setting this to a value of 0.5 would instruct the server to quantize (round) every value to .5 increments, having the following effects:

Sensor Reading	Quantized
3.4	3.5
3.74	3.5
4.43	4.5
4.51	4.5
5.23	5
5.49	5.5

Implementation Checklist

Required

The following items are required on your RBP server for interoperability with Roastmaster, iOS.

- Advertise a GAP server name
- Create, advertise and expose the RBP Sensing Service in your server's GATT
- Create, advertise and expose one or more Characteristics of the RBP Sensing Service in your server's GATT, with subscribing/notifying permissions on each.
- Update GATT value on exposed sensor Characteristics when (and only when) sensor readings change.
- Ensure that you send the correct data type according to the GATT Data chart (UInt32, UInt8, etc) for each item in your GATT, and that you have encoded this value correctly if applicable (Float×10^2, etc.). See RBP GATT Data Reference chart.
- If working with humidity sensors, advertise and expose the Humidity Scale Type Descriptor with at least read capabilities. This allows Roastmaster to report the scale in its Probe details views.

Optional

The following items are optional, but greatly augment the user's experience in Roastmaster, iOS

- Advertise the Serial Number String Characteristic of the Device Information Service, and expose a serial number that is unique for every server. This allows Roastmaster to concurrently access multiple devices of the same hardware type safely, and avoids potential conflicts with servers from other vendors.
- Advertise and expose the Notify Frequency Descriptor with read/write capabilities for exposed Characteristics, and respond to write changes. Use the Descriptor value to determine how many seconds should transpire between GATT updates–updating only when the appropriate amount of time has transpired since the last update AND the sensor has changed.
- If working with temperature sensors, advertise and expose the Probe Type Descriptor with read/write capabilities. Use the Descriptor value to determine how to interpret temperature probe readings.
- If working with humidity sensors, advertise and expose the Humidity Scale Type Descriptor with write capabilities in addition to the required read capabilities. Use the Descriptor value to determine the appropriate scale to use for GATT updates.
- If working with 4-20 mA sensors, advertise and expose the Min Value and Max Value Descriptors with read/write capabilities. Use the Descriptor value to interpret 4-20 sensor readings.
- If working with 4-20 mA sensors, advertise and expose the Quantize Descriptor with read/write capabilities. Use the Descriptor value to quantize 4-20 sensor readings.

Support

We are always willing to help. Please contact us at support@rainfroginc.com if you have any questions, or need clarification on any points in this datasheet.

Terms of Use

Using, transmitting, receiving, compiling, developing, sharing, distributing, selling, promoting, or any other action involving the Roastmaster Bluetooth Protocol (RBP), and associated software and documentation, whether personal or professional, must be done in accordance with one of the licenses outlined below.

Licensing

GNU General Public License v3 (GPL)

Permission to use the entirety of the Roastmaster Bluetooth Protocol (RBP), Services, Characteristics, Descriptors, as well as associated software and documentation, is granted under the GNU General Public License, providing that all obligations of said license are met. This includes, but is not limited to, licensing your software or device under GNU GPL as well.

GNU Lesser General Public License v3 (LGPL)

In situations where the GNU GPL license obligations cannot be met, permission to use the Temperature sensing Characteristics 1–5 of the Roastmaster Bluetooth Protocol (RBP), and any Service and Descriptors related to and used in their sole specific context, as well as associated software and documentation, is granted under the GNU Lesser General Public License, providing that all obligations of said license are met. This allows more flexible licensing options for your particular software or device.

Note: This licensing scheme expressly **precludes** the use of the remaining Characteristics (Pressure, Humidity, 4-20mA, and User Defined), and any Descriptors related to and used in their specific context.

Commercial License

In situations where the preceding license obligations cannot be met, permission to use the entirety of the Roastmaster Bluetooth Protocol (RBP), Services, Characteristics, Descriptors, as well as associated software and documentation, may be granted under a custom commercial license provided by Rainfrog, Inc., and provides the most flexible options for licensing your software or device.

Please contact support@rainfroginc.com for more information.

Special Thanks

Endless thanks for the enthusiasm and input from countless Roastmaster users that keep it growing!

Roastmaster Bluetooth Protocol (RBP)
Copyright (c) 2017 • Rainfrog, Inc.
Written by Danny Hall, for Rainfrog, Inc.