VIANCE
Validate Your World

PureRF ASK API User's Guide

Version 1.0

Important Notice

Vuance reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to Vuance's terms and conditions of sale supplied at the time of order acknowledgment.

Vuance warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with Vuance's standard warranty. Testing and other quality control techniques are used to the extent Vuance deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

Vuance assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using Vuance components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

Vuance does not warrant or represent that any license, either express or implied, is granted under any Vuance patent right, copyright, mask work right, or other Vuance intellectual property right relating to any combination, machine, or process in which Vuance products or services are used. Information published by Vuance regarding third-party products or services does not constitute a license from Vuance to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from Vuance under the patents or other intellectual property of Vuance.

Resale of Vuance products or services with statements different from or beyond the parameters stated by Vuance for that product or service voids all express and any implied warranties for the associated Vuance product or service and is an unfair and deceptive business practice. Vuance is not responsible or liable for any such statements.

All company and brand products and service names are trademarks or registered trademarks of their respective holders.

About This Guide

This API user guide is intended for integrators and programmers of PureRF ASK (Amplitude Shift Keying) systems. It describes how to use the PureRF API software to interact with PureRF tags and receivers.

This guide contains the following chapters:

- **Chapter 1, Introduction,** page 1, introduces the PureRF system and its components. It then describes how to get started using the PureRF API.
- **Chapter 2, PureRF.Receiver Class,** page 13, provides a reference describing each of the objects in the PureRF.Receiver Class, which is the primary class used to interact with the PureRF system.
- **Chapter 3, PureRF.Receiver.Tag Class,** page 23, provides the data structure that contains a single tag message that was sent by a tag, read by a receiver and transmitted to the PureRF server.

The following chapters provide data structures for interacting with a PureRF receiver:

- **Chapter 4, PureRF.Receiver.ActivateRelayParameter Class,** page 25
- **Chapter 5, PureRF.Receiver.Power_Control Class,** page 26
- **Chapter 6, PureRF.Receiver.AllUnitInfo Class,** page 27
- **Chapter 7, PureRF.Receiver.UnitInfo Class,** page 28
- **Chapter 8, PureRF.Receiver.UnitNameParameter Class,** page 30
- **Chapter 9, PureRF.Receiver.UnitStatus Class,** page 31

The following chapters provide functions for handling PureRF loops:

- **Chapter 10, PureRF.ReceiversManager Class,** page 32
- **Chapter 11, PureRF.ReceiversManager.Loop Class,** page 36

Table of Contents

Safety Precautions

IMPORTANT:

The equipment contains communication devices. Any changes or modifications made to the equipment without the written consent of Vuance, and its resellers or distributors, can nullify the user's authority to operate this equipment.

The user assumes all risks associated with the use and handling of the equipment, and specifically acknowledges that Vuance, and its resellers or distributors, will not be liable for any damages of any kind, including personal injury or property damages resulting from use of the equipment.

IMPORTANT:

Carefully read the safety information contained in this section, and throughout this user guide, before installing, operating, or performing any maintenance task on the equipment.

IMPORTANT:

Operations not performed as per the instructions in this user guide are done at the user's own risk and liability.

IMPORTANT:

Only trained, authorized personnel should install, maintain and repair the equipment.

IMPORTANT:

Once you have thoroughly reviewed this user guide, if you have any questions, please contact your reseller.

General

- Follow all warnings and cautions provided in this User's Guide.
- Comply with all approved and established precautions for operating the equipment.
- Never install equipment that is damaged.

This page was intentionally left blank.

Chapter 1

Introduction

About This Chapter

This chapter introduces the PureRF system, its components and the PureRF API. It then describes how to get started using the PureRF API.

This chapter contains the following sections:

What is PureRF?

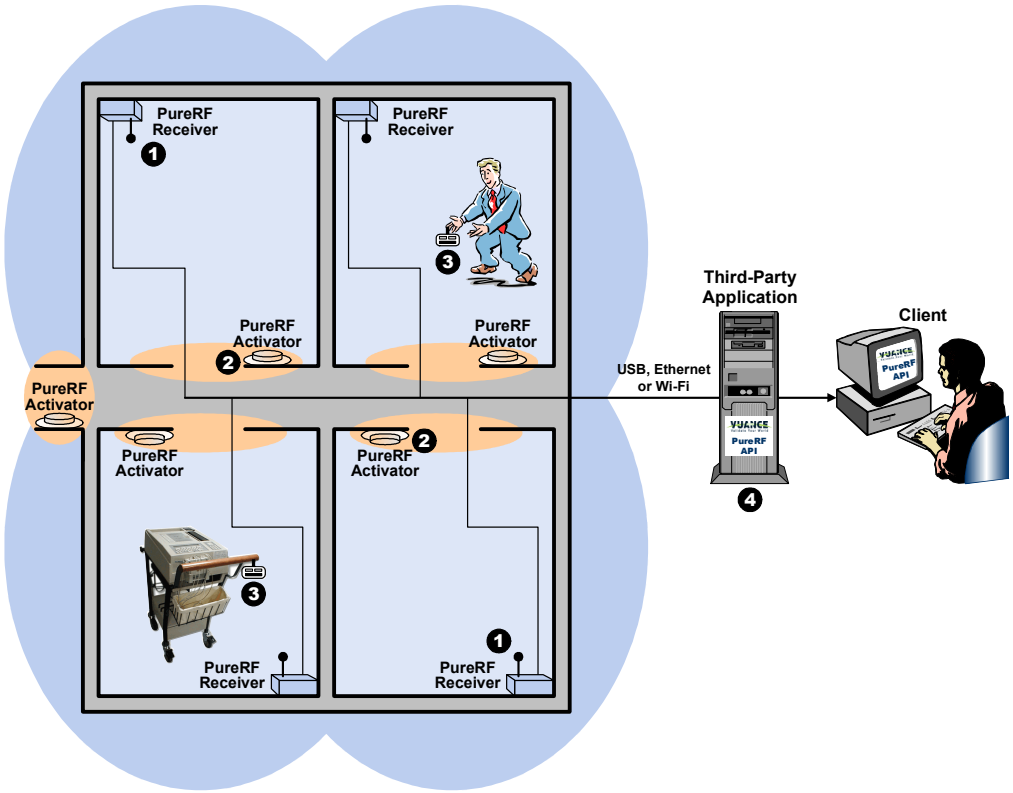
Vuance's innovative solutions enable organizations to monitor, track, locate, secure and manage multiple physical assets (key objects and personnel) that are either in motion or static. Vuance helps organizations alleviate wasted time and resources and provides the ultimate solution for authentication, validation, identification, location and real-time monitoring of its assets.

PureRF suite is an RFID hardware-software asset-tracking management platform, which streamlines asset management scenarios through the introduction of an integrated platform of realtime Identification, Movement Detection and Real Time Locating System (RTLS). It consists of Receivers, Activators, RFID tags and a variety of accessories and software tools for setup, management, monitoring and maintenance. PureRF can be used for access control, security, availability, inventory and incident management systems, as well as RFID solutions for public safety for the commercial and government sectors.

PureRF is operated through a secure, knowledge-based, interactive, user-friendly interface.

PureRF System Architecture

The following shows the architecture of a PureRF system.



The following describes the signal flow of a PureRF system:

1 **PureRF receivers** are deployed throughout the monitored space in order to pick up the RF signals that are transmitted by PureRF tags that are installed on monitored assets (for example, in the drawing above, a person and a copy machine). The size of the monitored space can vary from a single door to an entire campus covering wide areas both indoors and outdoors. In the drawing above, the blue area designates the combined coverage of the four installed receivers. The receiver that picks up a PureRF tag indicates the tag's location because a PureRF tag must be in the vicinity of a receiver in order for the receiver to pick up its signal.

When covering a wide area, the receivers may overlap and therefore, several receivers may receive the same signal. In this case, the location of the tag is determined by the receiver that receives the highest signal strength (highest RSSI). The receiver that picks up the transmission adds its own unique ID to the tag's message that represents this tag's (asset's) current location. Additionally, the receiver can perform conversions between various commonly used interfaces/protocols. More information is provided in the *What is a PureRF Receiver?* section on page 8.

2 **PureRF activators** are used for improving the accuracy of locating assets compared to what is provided by the receiver ID. For this purpose, PureRF activators are deployed throughout the monitored space wherever improved tag location measurement is required. The PureRF activators continually transmit a short-range uniquely identifying Low Frequency (LF) signal. Tags can read this signal when they are close to the activator (up to about 2 meters/6 feet). In the drawing above the pink area designates the activators' range. The activator ID that a tag reads is added to the message that the tag transmits to the receiver. An activator's ID indicates the location of a PureRF tag at that time because the tag must be physically close to this activator in order pick up its signal. More information is provided in the *What is a PureRF Activator?* section on page 9.

3 **PureRF RFID tags** repeatedly (according to a preset period) transmit their unique ID and their status (Low Battery, Motion or Tamper/Panic). Whenever a tag is close enough to a PureRF activator to pick up its signal, the PureRF RFID tag adds the received activator's ID to its periodically transmitted message. These transmissions are picked up by the nearby receivers. More information is provided in the *What is a PureRF Tag?* section on page 5.

4 The application that manages and controls the assets/personnel runs on a standard PC. This application can communicate with the connected receivers through the PureRF API (SDK) via the following interfaces/protocols: RS232, RS485, Wiegand, Ethernet and Wi-Fi. The application periodically collects the tags status messages that the receivers collect and records them in its database. Each tag message received contains the unique ID of the receiver that picked up its signal. The application can also analyze the database periodically to generate additional events based on status combinations.

NOTE:

The **PureRF Initializer** is a device used to configure tags and to set up their operational parameters, such as: Tag On/Off, Tag Type, Sensors On/Off, transmissions timings, alarms and so on. The Initializer is the only device that can perform two-way communication with tags using a combination of the RF and LF channels that are incorporated in each tag.

What is a PureRF Tag?

The PureRF Tag is Vuance's RFID tag. RFID stands for Radio Frequency Identification. An RFID tag is an electronic PCB combined with an antenna in a compact package. The packaging is structured to enable the RFID tag to be attached to an object or a person to be tracked. It can be attached to or incorporated into a product, animal or person for the purpose of identification and location detection using radio waves. Tags can be detected from varying distances depending on the surroundings influence on radio waves propagation.

After a PureRF tag is activated by a PureRF Initializer, it periodically broadcasts a signal that contains its unique identifier that is picked up by one or more PureRF receivers. The current position of the tag is determined by the receiver that picks up the strongest RSSI (signal strength) value. More information is provided in the *What is a PureRF Receiver?* section on page 8.

PureRF tags also contain a short-range LF receiver that can pick up signals transmitted by PureRF activators and Initializers. PureRF activators are deployed throughout the monitored space and continually transmit a short-range uniquely identifying signal.

When an activator's ID is picked up by a PureRF tag, the activator's ID is included in the PureRF tag's message transmissions. This indicates the tag's location with more precise accuracy (compared to the RSSI method described earlier) because the PureRF tag must be physically very close (approximately within 2 meters/6 feet) to this activator in order to pick up the activator's signal.

PureRF Tag Events

A tag can broadcast a variety of types of PureRF messages which are picked up by the receivers, which transfers them to the user's application in response to its request. These messages are broadcasted by the tag repeatedly at the intervals defined during setup, such as:

NOTE:

The location of a tag can be determined according to the receiver(s) that picks up the strongest signal.

- **Periodic Transmissions:** The tag transmits periodic broadcasts to indicate its ID and status. This message is transmitted periodically. The frequency (meaning how often) this message is transmitted is defined during the tag's setup. The definition of these periodic transmissions may vary, as described in this guide.
- **Tag Activator:** Whenever a tag enters into the range of an activator, the tag picks up the activator's unique ID and adds it to the periodically transmitted message. This indicates that the tag is very close to that activator (approximately within 2 meters/ 6 feet).
- **Tag Sensors:** The tag can incorporate up to three types of internal sensors for indicating its status. Each sensor controls one bit in the transmitted message. Such sensors include:
  - **Low Battery:** Indicating that the tag's battery voltage is under a predefined threshold meaning that the tag must soon be replaced.
  - **Tag Motion:** The tag turns on the in-motion bit to indicate that it is in motion (being moved). These transmissions can be at regular periodic frequencies (timings) or at an accelerated rate as alarm transmissions.
  - **Tag Tamper/Panic:** The tag turns on the tamper/panic bit to indicate that its tamper/panic switch has been activated.

Typically, a tag can be set with two transmission frequencies (timings), one set of frequency for the periodic transmissions and another frequency (usually which transmits more often) for an alarm status, which is activated by a sensor. For example, the tag may be set to transmit every four minutes, when it is static, and every one minute, if it is moving. The duration of alarm transmissions may vary from a limited period of time, to the actual alarm period or may be transmitted indefinitely until turned off by an Initializer.

Tag Permission Events

In addition to the above, other types of events can be triggered for a tag as a result of the third-party user application that can be programmed to periodically analyze the received messages according to a set of defined rules in order to detect interesting combination events such as the following:

- **Tag Inside Authorized Zone:** The tag's signal was detected in an authorized zone.
- **Tag Outside Authorized Zone:** The tag's signal was detected in an unauthorized zone.
- **Tag Inside Homereceiver:** The tag's signal was detected inside its homereceiver.
- **Tag Outside Homereceiver:** The tag's signal was detected outside of its homereceiver.
- **Tag Authorized Timeout:** The tag's signal was detected in an authorized zone, but it is no longer transmitting.
- **Tag Homereceiver Timeout:** The tag's signal was detected in its home receiver zone, but it is no longer transmitting.
- **Tag Missing:** When no signal is detected for a defined period of time from a tag that was recently transmitting.
- **Tag Shock:** A tag that is undergoing a significant amount of movement.

What is a PureRF Receiver?

Each PureRF receiver picks up the messages transmitted by the PureRF tags in its vicinity and transfers tag and receiver events to the user application. The receiver(s) pick up each tag's message (which contains its ID) and adds to it the received signal strength and its own (receiver's) ID. Thus enabling the PureRF system to determine the location of the tag (and the asset to which it is attached).

Each PureRF receiver has a defined coverage area and the receivers of a system must be deployed to cover the entire area in which tags are to be tracked.

Receivers store the messages of the received tags in a buffer, which they download periodically to the user application.

Receiver Events

The user application can generate the following receiver events:

- **Receiver OK:** The receiver is functioning normally.
- **Receiver Timeout:** A receiver that was transmitting is now not transmitting any signal at all. This may occur, for instance, when the receiver has no power.
- **Receiver Error:** The receiver is connected and powered up but is indicating an erroneous state.
- **Receiver in Upload:** This event is generated and displayed while new firmware is being uploaded to the receiver.

What is a PureRF Activator?

A PureRF activator is a device used for improving the accuracy of location measurement of tags as provided by receivers only and is used mainly for entrances and exits.

An RFID activator contains an electronic PCB combined with an antenna and it continuously transmits a unique identifier on a Low Frequency (LF) channel.

PureRF activators are deployed throughout the monitored space at key locations where the improved location measurement accuracy of assets is required. Whenever a PureRF tag enters in range of an activator (approximately within 2 meters/6 feet), the tag receives the activator's signal and adds the activator ID to the transmitted message indicating that the tag is located very close to that activator. Thus, a transmitted activator's ID indicates the location of a PureRF tag at a specific point in time because the tag must have been physically close to this activator in order pick up its signal.

The activator signal is received by the LF receiver incorporated in the tag. Such channel separation enables the tag to receive (activator ID) and transmit (to the receiver) simultaneously.

The activator is an independent device that does not need to communicate with the third-party application. The activator only needs power to operate.

What is a PureRF Initializer?

A PureRF Initializer is a device that integrates an LF transmitter and an RF receiver into the same device. This enables the Initializer to perform bi-directional communication with tags.

The Initializer is used for controlling a tag's mode of operation (on/off) and for setting or modifying a tag's operational parameters, such as: transmission frequency (timing), activated sensors and so on.

Using the PureRF API

The PureRF API (Application Protocol Interface) provides a simple and straightforward object-oriented interface for accessing information collected by the PureRF receivers from PureRF tags and controlling their settings.

The PureRF API can be integrated into a variety of types of applications, such as those intended for access control, security, incident management systems and so on.

There are two types of functions that are used most frequently in the PureRF API: those that retrieve tag messages from the receiver and those that configure the receiver. Mostly you will use the functions of the **PureRF.Receiver Class,** such as the following:

- **GetAllTags:** page 16, Gets the current list of tag messages read by a specified receiver.

NOTE:

GetAllTags retrieves an array of objects of tag messages type. Each message in the array has the structure of *PureRF.Receiver.Tag Class*, as described on page 23.

- **GetReceiverStatus:** page 19, Gets all the information of a specified receiver, such as: unit information, noise level, antenna sensitivity, time etc.
- **ActivateRelay:** page 17, Activates a receiver's relay.
- **SetBootloaderMode:** page 20, Sets a receiver to bootloader mode. In this mode, the receiver loads new firmware and verifies its validity.

Getting Started – PureRF API?

Minimal Requirements

- A standard PC with Windows XP
- .Net 2.0
- RAM – 1GB or higher
- Interfaces for PureRF Receivers:
  - TCP/IP
  - USB
  - RS232
- RS485 to USB converter (including a driver)

Installation

Use the provided CD for installation. This installs three files in the Program Files folder in a folder called Vuance/PureRF:

- The API/SDK DLL
- The Com Port control DLL
- An .exe file that runs a basic GUI for communicating with the receivers. This GUI enables you to perform various basic functions.

The installation also creates an icon for this GUI on the PC's desktop.

System Integration

In order to activate the API for developing a user application, you must set up a basic system of at least one receiver and several tags. The receiver can be connected via one of the interfaces specified above.

User Application Development

For developing a user application, Visual Studio 2005 or higher is required.

Class Reference Table

Class	Description	Page
PureRF.Receiver Class	Handles the interface with a PureRF receiver.	13
PureRF.Receiver.Tag Class	The data structure that describes a single tag message received from a tag by a receiver.	23
PureRF.Receiver.ActivateRelay Parameter Class	Controls the **ActivateRelay** function of a receiver.	25
PureRF.Receiver.Power_Control Class	Controls the power settings of the receiver.	26
PureRF.Receiver.AllUnitInfo Class	Holds the complete receiver status (Info, Status, Noise level, and Power). Used as a return value from the Receiver.GetReceiverStatus function.	27
PureRF.Receiver.UnitInfo Class	A data structure that contains information received from the function **GetUnitInfo**.	28
PureRF.Receiver.UnitName Parameter Class	A data structure that contains the name of a receiver unit.	30
PureRF.Receiver.UnitStatus Class	A data structure that contains information received from the function **GetUnitStatus**.	31
PureRF.ReceiversManager Class	A class for managing network(s) of receivers. It handles multiple loops, with multiple receivers on each loop and can send commands and receive replies in both synchronous and asynchronous modes.	32
PureRF.ReceiversManager.Loop Class	Handles a single loop (physical serial connection).	36

Chapter 2

PureRF.Receiver Class

Overview

Description: The **Receiver** class handles the interface with a **PureRF** receiver. A receiver is the device that receives transmissions from the tags. More information describing a receiver is provided in the *What is a PureRF Receiver?* section on page 8.

Public Types

TagMsg enumerator: List of messages that can be received from a tag.

- 1 LowBattery: Low Battery alert.
- 2 MotionSensor: Motion Sensor alert.
- 4 Tamper: Tamper/Panic alert.
- 8 ACTIVATOR: Near Activator indication. The unique identifier of the activator that is close to the tag and can be retrieved from the Public Attribute **activatorNum** in the PureRF.Receiver.Tag Class, as described in *Chapter 3, PureRF.Receiver.Tag Class* on page 23.

Public Member Functions

- **GetUnitInfo:** page 15, Gets the information about the receiver unit itself.
- **GetUnitStatus:** page 15, Gets the receiver's unit status.
- **GetAllTags:** page 16, Gets the current tags read by the receiver. This function (GetAllTags) calls the Receiver.GetTags function once for up to every eight messages stored in the receiver's buffer. This means that it calls the Receiver.GetTags function in a loop until no more tag messages are in the receiver's buffer. This action empties the receiver's buffer.
- **GetNoiseLevel:** page 16, Gets the measured noise level in the vicinity of the receiver.
- **ActivateRelay:** page 17, Activates a receiver's relay.
- **SetUnitName:** page 17, Sets the name of the receiver unit.
- **GetUnitName:** page 18, Gets the name of the receiver unit.
- **FlushTagBuffer:** page 18, Clears all the tag messages that are in a receiver's buffer.
- **GetReceiverStatus:** page 19, Gets all the information of a specified receiver, such as: unit information, unit status, noise level and unit power.
- **GetMode:** page 20, Gets the current receiver mode, which is one of the following:
  - **Firmware mode:** This is the normal mode when the receiver software is running.
  - **Bootloader mode:** In this mode, the receiver loads new firmware and verifies its validity.
- **SetBootloaderMode:** page 20, Sets a receiver to bootloader mode. In this mode, the receiver loads new firmware and verifies its validity.
- **SetMainMode:** page 21, Sets a receiver into firmware mode. This is the normal mode when the receiver software is running.
- **Download:** page 21, Sends a single firmware packet to the specified receiver. This function must only be used after using the SetBootloaderMode function.
- **FirmwareChecksum:** page 22, Gets the current state of the receiver's firmware and checks its validity.

GetUnitInfo

Description: Gets the information about the receiver unit itself.

```
ReceiverRetVal PureRF.Receiver.GetUnitInfo
                        ( byte           unitID,
                          out UnitInfo   unitInfo
                        )
```

Parameters:

unitID	Receiver unit ID from which to retrieve the unit information.
unitInfo	[output] Returns the UnitInfo class, which is a data structure that contains all the information about the receiver unit, as described in *PureRF.Receiver.UnitInfo Class* on page 28.

Returns: A standard **PureRF.ReceiverRetVal**

GetUnitStatus

Description: Gets the receiver's unit status.

```
ReceiverRetVal PureRF.Receiver.GetUnitStatus
                        ( byte           unitID,
                          out UnitStatus unitStatus
                        )
```

Parameters:

unitID	Receiver unit ID from which to retrieve the status.
unitStatus	[output] Returns the UnitStatus class, which is a data structure that contains all the information about the receiver unit, as described in *PureRF.Receiver.UnitStatus Class* on page 31.

Returns: A standard **PureRF.ReceiverRetVal**

GetAllTags

Description: Gets the current tags read by the receiver. This function (GetAllTags) calls the Receiver.GetTags function once for up to every eight messages stored in the receiver's buffer. This means that it calls the Receiver.GetTags function in a loop until no more tag messages are in the receiver's buffer. This action empties the receiver's buffer.

```
ReceiverRetVal PureRF.Receiver.GetAllTags
                        ( byte          unitID,
                          out Tag[]     tags
                        )
```

Parameters:

unitID	The unit ID of the receiver from which to retrieve messages.
allTags	[output] An array of tag messages. Each message in the array has the structure of *PureRF.Receiver.Tag Class*, as described on page 23.

GetNoiseLevel

Description: Gets the measured noise level in the vicinity of the receiver.

```
ReceiverRetVal PureRF.Receiver.GetNoiseLevel
                        ( byte          unitID,
                          out ushort    noiseLevel
                        )
```

Parameters:

unitID	Receiver unit ID from which to retrieve the noise level.
noiseLevel	[output] The noise level in the vicinity of the receiver.

Returns: A standard **PureRF.ReceiverRetVal**

ActivateRelay

Description: Activates a receiver's relay.

```
ReceiverRetVal PureRF.Receiver.ActivateRelay
                ( byte                    unitID,
                  ActivateRelayParameter
                RelayParameter
                )
```

Parameters:		
	unitID	Unit ID of the receiver whose relay to activate.
	RelayParameter	The receiver's relay to activate and the interval at which to activate it, as described in the data structure *PureRF.Receiver.ActivateRelayParameter Class* on page 25.

SetUnitName

Description: Sets the name of the receiver unit.

```
ReceiverRetVal PureRF.Receiver.SetUnitName
                ( byte                    unitID,
                  ref UnitNameParameter   aName
                )
```

Parameters:		
	UnitID	Unit ID of the receiver whose name to set.
	UnitNameParameter	The name to which to set the receiver unit, as described in the data structure *PureRF.Receiver.UnitNameParameter Class* on page 30.

GetUnitName

Description: Gets the name of the receiver unit.

```
ReceiverRetVal PureRF.Receiver.GetUnitName
                ( byte                  unitID,
                  out UnitNameParameter  aName
                )
```

Parameters:

UnitID	Unit ID of the receiver whose name to get.
UnitNameParameter	[output] The name retrieved from the receiver unit, as described in the data structure *PureRF.Receiver.UnitNameParameter Class* on page 30.

FlushTagBuffer

Description: Clears all the tag messages that are in a receiver's buffer.

```
ReceiverRetVal PureRF.Receiver.FlushTagBuffer
                        ( byte          unitID   )
```

Parameters:

unitID	The unit ID of the receiver whose buffer is to be emptied.

Returns: A standard **PureRF.ReceiverRetVal**.
If a broadcast was received, GOT_BROADCAST is returned.
If no broadcast was received, NO_BROADCAST is returned.

GetReceiverStatus

Description: Gets all the information of a specified receiver, such as: unit information, unit status, noise level and power.

Activating this function is the same as activating all of the following functions:

- **GetUnitInfo,** as described on page 15
- **GetUnitStatus,** as described on page 15
- **GetNoiseLevel,** as described on page 16

```
ReceiverRetVal PureRF.Receiver.GetReceiverStatus
                      ( byte                 unitID,
                        out ReceiverStatus  status
                      )
```

Parameters:

unitID Unit ID of the receiver whose information is to be retrieved.

status [output] The following information is returned into the classes listed below:

- **UnitInfo class,** which is a data structure that contains all the information about the receiver unit, as described in *PureRF.Receiver.UnitInfo Class* on page 28.
- **UnitStatus class,** which is a data structure that contains all the information about the receiver unit, as described in *PureRF.Receiver.UnitStatus Class* on page 31
- The noise level near the receiver.

Returns: A standard **PureRF.ReceiverRetVal**.
Note that this function only succeeds if all the operations described above have succeeded.

GetMode

Description: Gets the current receiver mode, which is one of the following:

- **Firmware mode:** This is the normal mode when the receiver software is running.
- **Bootloader mode:** In this mode, the receiver loads new firmware and verifies its validity.

```
ReceiverRetVal PureRF.Receiver.GetMode
                        ( byte          unitID   )
```

Parameters: unitID — Unit ID of the receiver whose mode is to be retrieved.

Returns: A standard **PureRF.ReceiverRetVal**.

If the receiver is in bootloader mode, then MODE_BOOTLOADER is returned.

If the receiver is in firmware (the normal) mode, then MODE_FIRMWARE is returned.

SetBootloaderMode

Description: Sets a receiver to bootloader mode. In this mode, the receiver loads new firmware and verifies its validity.

```
ReceiverRetVal PureRF.Receiver.SetBootloaderMode
                        ( byte          unitID   )
```

Parameters: unitID — Unit ID of the receiver to be set to bootloader mode.

Returns: A standard **PureRF.ReceiverRetVal**

SetMainMode

Description: Sets a receiver into firmware mode. This is the normal mode when the receiver software is running.

```
ReceiverRetVal PureRF.Receiver.SetMainMode
                        ( byte          unitID   )
```

Parameters:

unitID	Unit ID of the receiver to be set to firmware mode.

Returns: A standard **PureRF.ReceiverRetVal**

Download

Description: Sends a single firmware packet to the specified receiver. This function must only be used after using the SetBootloaderMode function, as described on page 20.

```
ReceiverRetVal PureRF.Receiver.Download
                        ( byte          unitID,
                          ushort        packet_num,
                          byte[]        firmwarePacket
                        )
```

Parameters:

unitID	Unit ID of the receiver to which to download a firmware packet.
packet_num	Firmware packet number. This is a sequential number.
firmwarePacket	The packet must contain exactly Receiver.FIRMWARE_PACKET_LEN bytes. Receiver.FIRMWARE_PACKET_LEN is a constant defining the number of bytes in each packet.

Returns: A standard **PureRF.ReceiverRetVal**.

FirmwareChecksum

Description: Gets the current state of the receiver's firmware and checks its validity.

```
ReceiverRetVal PureRF.Receiver.FirmwareChecksum
                        ( byte           unitID,
                          out ushort     flash_cksum
                        )
```

Parameters:	unitID	Unit ID of the receiver whose firmware to verify.
	flash_cksum	[output] The checksum of the receiver's firmware.

Returns: A standard **PureRF.ReceiverRetVal**.
Returns FLASH_OK or FLASH_DAMAGED according to the firmware's state.

Public Attributes

- **const byte FIRMWARE_PACKET_LEN = 16**

  Length of each firmware payload packet in a download command.

- **const byte MAX_TAGS_TO_FETCH = 8**

  Number of tags to fetch in each request in the GetAllTags commands.

- **const int BOOTLOADER_BAUDRATE = 57600**

  The baud rate of the receiver when in Bootloader mode.

- **const int FIRMWARE_BAUDRATE = 57600**

  The baud rate of the receiver when in routine operational mode, meaning while running the firmware.

Chapter 3

PureRF.Receiver.Tag Class

Overview

Description: The data structure that describes a single tag message received from a tag by a receiver. This is the data structure that is picked up by receivers from the tags transmitting in their vicinity. This information is retrieved using the *GetAllTags* function of the PureRF.Receiver Class, as described on page 16.

Public Attributes

- **TagID** **tagID**

  The unique identifier of a Tag ID. This ID uniquely identifies the person or asset to which this tag is attached.

- **byte** **transmissionIndex**

  The tag's transmission index. A messages ID that helps identify duplicate messages, it recognizes the following two cases:

  - Two receptions of the same message by the same receiver.
  - The reception of the same message in two (or more) receivers.

- **TagMsg** **tagMsg**

  The actual message of the tag.

  enum List of messages that can be received from a tag.

  - 1 LowBattery: Low Battery alert ON/OFF.
  - 2 MotionSensor: Motion Sensor alert ON/OFF.
  - 4 Tamper: Tamper/Panic alert ON/OFF.
  - 8 ACTIVATOR: Near Activator True/False. The unique identifier of the activator that the tag is near can be retrieved from the Public Attribute **activatorNum** in the PureRF.Receiver.Tag Class, as described in *PureRF.Receiver.Tag Class* on page 23.

- **byte** **activatorNum**

  The unique identifier of the activator that is currently near the tag. You may refer to the *What is a PureRF Activator?* section on page 9, for a description of an activator. The Activator ID holds until the tag is out of range.

- **ushort** **RSSI**

  RSSI representing the signal strength of the received tag.

- **ushort** **NoiseLevel**

  Noise level as measured in the vicinity of the receiver.

- **Time** **ts**

  Currently, the tag does not have a realtime clock so that a time stamp is added by the DLL when a tag message is retrieved from the receiver. The DLL uses the clock of the PC. In the future, when a realtime clock is added to the receiver, the time stamp will be added to each tag message by the receiver when the message is received.

Chapter 4

PureRF.Receiver.ActivateRelayParameter Class

Overview

Description: This data structure contains the parameters that are applied when the *ActivateRelay* function of the PureRF.Receiver Class is called, as described on page 17. It defines which relay to activate on a receiver and for how long.

Public Attributes

- **int** **Relay**

  Specifies the relay to activate out of a choice of three: 0, 1 or 2.

- **int** **Interval**

  Specifies the interval for the relay activation (for how long) with the following parameters:

  - 0x00: Relay Off
  - 0xFF: Relay continuously On
  - 0x01 to 0xFE (1 to 254): Interval time in 100msec multipliers

Chapter 5

PureRF.Receiver.Power_Control Class

Overview

Description: Controls the power settings of the receiver.

Public Attributes

- **byte** **Power_Mode:** 1 RF 0FF
  2 RF ON
  3 Reset
  4 Inquiry data request check status
- **int** **Input_Voltage:** Measured input voltage multiplied by 100mV. Normal value is 120 (equivalent to 12V). Permitted tolerance is between 104 and 144 (10.4V to 14.4V).

Chapter 6

PureRF.Receiver.AllUnitInfo Class

Overview

Description: Holds the complete receiver status (Info, Status, Noise level, and Power). Used as a return value from the Receiver.GetReceiverStatus function.

Chapter 7

PureRF.Receiver.UnitInfo Class

Overview

Description: A data structure that contains information that describes a specific receiver unit. This information is retrieved using the *GetUnitInfo* function of the PureRF.Receiver Class, as described on page 15.

Public Member Function

override string

- **ToString ():** Get receiver Unit Information in text format.

Public Attributes

- **byte** **protocol_version**

  The byte representation of the receiver's protocol version (BCD encoded protocol version).

  For example: 1 for version 0.01 and 255 for version 2.55

- **byte** **Device_Class**

  Receiver's class is 0x01. F/W encoded.

- **byte** **Device_SubClass**

  Receiver's sub-class is 0x01. F/W encoded.

- **byte** **Firmware_Version**

  BCD encoded firmware version. F/W encoded.

- **string** **Unit_Name**

  Set by user. ASCII string of 20 characters.

- **int** **Unit_Serial_Number**

  A unique factory number set for each receiver.

Chapter 8

PureRF.Receiver.UnitName Parameter Class

Overview

Description: A data structure that contains the name of a receiver unit. This information is set using the *SetUnitName* function of the PureRF.Receiver Class, as described on page 17, and is retrieved using the *GetUnitName* function of the PureRF.Receiver Class, as described on page 18.

Public Attributes

- **string** **UnitName: Unit name string (limited to text only).**

Chapter 9

PureRF.Receiver.UnitStatus Class

Overview

Description: A data structure that contains information that describes a specific receiver's status. This information is retrieved using the *GetUnitStatus* function of the PureRF.Receiver Class, as described on page 15.

Public Attributes

- **byte** **Sub_mode**

  The receiver unit's mode of operation: firmware (the normal mode) or bootloader.

- **int** **Uptime**

  The amount of time that this receiver has been up (in seconds) since the last time it was reset.

- **byte** **Average_Processor_Workload**

  Not applicable.

- **byte** **Number_of_Resets**

- **byte** **Tag_messages_in_buffer**

  Number of tag messages in receiver's buffer. You may refer to *Chapter 3, PureRF.Receiver.Tag Class* on page 23 for a description of a tag message.

Chapter 10

PureRF.ReceiversManager Class

Overview

Description: This class manages networks of receivers.

All the receivers connected to a single RS485-to-USB adapter, which is connected to a USB port, belong to the same Connection Loop. Alternatively, receivers can be connected via WiFi or the Ethernet to comprise a Connection Loop.

The class can handle multiple Connection Loops, with multiple receivers on each loop. It can be used to communicate with multiple Connection Loops/receivers in order to send commands and receive replies in both synchronous and asynchronous modes.

Public Attributes

- **const int** **DEFAULT_TIMEOUT** = 1500

  The default timeout for serial communication in case no timeout is specified at the constructor.

- **ArrayList** **LoopsList**

  List of Connection Loops managed by this class.

- **ArrayList** **ReceiversList**

  List of receivers managed by this class.

Public Member Functions

- **AddLoop:** page 33, Adds a loop based on an existing loop.
- **RemoveLoop:** page 34, Removes a Connection Loop from the list of loops handled by this class.
- **AddReceiver:** page 34, Adds a new receiver to be handled by this class.
- **RemoveReceiver:** page 34, Removes a receiver from the list of handled receivers.
- **WaitForRequestCompletion:** page 35, Blocks new activities until current request has fully completed.
- **GetAllResults:** page 35, Gets the complete result set, meaning all the tag messages from all the receivers.
- **GetResult:** page 35, Gets a single result for a specified receiver.
- **AbortRequest:** page 35, Stops the current request.

AddLoop

Description: Adds a loop based on an existing loop.

```
RetVal PureRF.ReceiversManager.AddLoop
                          ( Loop           loop,
                            bool           needOpen
                          )
```

Parameters:

loop	Existing loop, to be added loop.
needOpen	Specifies whether the loop must be open and active.

Returns: **ReceiversManager.RetVal**

RemoveLoop

Description: Removes a Connection Loop from the list of loops handled by this class.

Note: Removing a Connection Loop also removes all the receivers residing on that loop.

```
RetVal PureRF.ReceiversManager.RemoveLoop
                          ( string         Name   )
```

Parameters: Name The name of the Connection Loop to be removed.

Returns: **ReceiversManager.RetVal**

AddReceiver

Description: Adds a new receiver to be handled by this class.

```
RetVal PureRF.ReceiversManager.AddReceiver
                          ( string         Name,
                            string         LoopName,
                            byte           UnitID
                          )
```

Parameters: Name Unique name for the receiver.

LoopName Loop name on which the receiver resides.

UnitID Unit ID of the receiver.

Returns: **ReceiversManager.RetVal**

RemoveReceiver

Description: Removes a receiver from the list of handled receivers.

```
RetVal PureRF.ReceiversManager.RemoveReceiver
                          ( string         Name   )
```

Parameters: Name Name of receiver to be removed.

Returns: **ReceiversManager.RetVal**

WaitForRequestCompletion

Description: Blocks new activities until current request has fully completed.

```
RetVal PureRF.ReceiversManager.WaitForRequestCompletion
                          (                  )
```

Returns: **ReceiversManager.RetVal**

GetAllResults

Description: Gets the complete result set, meaning all the tag messages from all the receivers.

```
RetVal PureRF.ReceiversManager.GetAllResults
                          ( out ResultSet  Results   )
```

Parameters: Results [output] The complete result set (all results from all receivers).

Returns: **ReceiversManager.RetVal**

GetResult

Description: Gets a single result for a specified receiver.

```
RetVal PureRF.ReceiversManager.GetResult  ( string
                          receiverName,
                            out ReceiverResult  Result
                          )
```

Parameters: receiverName Receiver from which to get the result.

Result [output] A tag message of the specified receiver.

Returns: **ReceiversManager.RetVal**

AbortRequest

Description: Stops the current request.

```
RetVal PureRF.ReceiversManager.AbortRequest  (  )
```

Returns: **ReceiversManager.RetVal**

Chapter 11

PureRF.ReceiversManager.Loop Class

Overview

Description: This class handles a single loop, meaning a single physical serial connection of receivers.

Public Member Functions

- **ConnectSerial:** page 37
- **ConnectIP:** page 37
- **ConnectBus:** page 37
- **OpenPort:** page 37, Opens the serial port.
- **ClosePort:** page 37, Closes the serial port.
- **SetPortBaudrate:** page 38, Modifies the loop's baud rate.

Public Attributes

- **string Name**
  Unique identifier of this loop.
- **ReceiverBusConnection Conn**
  Bus on which the loop resides.
- **Receiver ReceiverInterface**
  An interface for interacting with the receivers on this loop.

ConnectSerial

```
void PureRF.ReceiversManager.Loop.ConnectSerial
                          ( string         PortName,
                            int            BaudRate,
                            int            Timeout
                          )
```

ConnectIP

```
void PureRF.ReceiversManager.Loop.ConnectIP
                          ( string         Host,
                            int            Port,
                            int            Timeout
                          )
```

ConnectBus

```
void PureRF.ReceiversManager.Loop.ConnectBus
                          ( ReceiverBusConnection  conn
                           )
```

OpenPort

Description: Opens the serial port.

```
bool PureRF.ReceiversManager.Loop.OpenPort  (  )
```

Returns: **true** on success, **false** otherwise.

ClosePort

Description: Closes the serial port.

```
bool PureRF.ReceiversManager.Loop.ClosePort  (  )
```

Returns: **true** on success, **false** otherwise.

SetPortBaudrate

Description: Modifies the loop's baud rate.

```
void PureRF.ReceiversManager.Loop.SetPortBaudrate  (
                          int               BaudRate   )
```

Parameters: BaudRate New baud rate of the loop.