Sandia National Laboratories

Interface Specifications for RAdiation Portal Technology Enhancement & Replacement (RAPTER) Modules

January 2023
SAND2023-xxxx

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Contents

List of Abbreviations

ARDIS Automated Radiation Portal Monitor (RPM) Data Integration System.

ARM Advanced RISC Machine.

ASCII American Standard Code for Information Interchange.

CAN Controller Area Network.

CPU Central Processing Unit.

DHCP Dynamic Host Configuration Protocol.

DHS Department of Homeland Security.

DNDO Domestic Nuclear Detection Office.

FPGA Field Programmable Gate Array.

GUI Graphical User Interface.

HID Human interface device.

HTTP Hypertext Transfer Protocol.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

int integer.

IP Internet Protocol.

JSON JavaScript Object Notation.

NaN Not a Number.

NIC Network Interface Card.

NTP Network Time Protocol.

PKI Public Key Infrastructure.

PPS Pulse Per Second.

PRIDE Port-Radiation Inspection, Detection, & Evaluation.

PTP Precision Time Protocol.

RAPTER RAdiation Portal Technology Enhancement & Replacement.

RDE Radiation detection equipment.

RFC Request for Comments, specifically by the IETF.

RFC request for comments.

RPM Radiation Portal Monitor.

RSP Radiation Sensor Panel.

RSP radiation sensor panel.

SSDP Simple Service Discovery Protocol.

ssh Secure Shell.

TCP Transmission Control Protocol.

TCP/IP Transmission Control Protocol/Internet Protocol.

TLS Transport Layer Security.

TTL transistor–transistor logic.

UDP User Datagram Protocol.

uint unsigned integer.

UPnP Universal Plug and Play.

UPS Uninterruptible Power Supply.

URL Uniform Resource Locator.

USB Universal Serial Bus.

UTC Coordinated Universal Time.

UTF-8 Unicode Transformation Format 8-bit.

UUID universally unique identifier.

VNC Virtual Network Computing.

WSS WebSocketSecure.

XML eXtensible Markup Language.

YAML YAML Ain't Markup Language.

List of Terms

Advanced RISC Machine A computer CPU architecture commonly used for single board computers, embedded systems, microprocessors, or custom electronics.

American Standard Code for Information Interchange A character encoding for electronic communications for the representation of typographic symbols common in the United States, such as alphanumeric characters and punctuation.

analysis message group The set of messages that relate to input of data to, or the output of analysis results from, the analysis module. Analysis messages transit between the control module and the analysis module.

analysis module The physical device in a RAPTER portal that performs the analysis of radiation and vehicle presence data in order to determine the need for further inspection of the conveyance. Connects to the control module via the portal's Ethernet network and implements the core message group, the data-out message group, and the analysis message group. See Section 1.3.4.

background In the context of radiation measurements, the ambient radiation that exists even when no item of interest is present, for which no further inspection is warranted.

boolean A data type with two possible values: true or false.

command device A device in a RAPTER system that uses the command message group, in addition to the core and data out message groups in its communication with the control module. Command devices may communicate with another device in the system using the command module as a proxy by using relay command messages; see relay command message.

command message group A set of messages for effecting portal-wide changes in operating mode or measurement type, or for configuring settings describing the position of detectors. The group also includes the relay command message; see relay command message..

condition data See parameter.

control module The only physical component within a RAPTER portal that hosts the network, and that communicates with and controls all other modules on the network. Implements all message groups. The physical component that implements the roles and logic described in Section 1.3.1.

core message group A set of messages for controlling device-level operations and functions, and reporting device information. Every device connected to the control module must be able to receive and reply to every type of core message.

data Digital information. Unless otherwise described, data refers to measurement results produced by a detector, of which examples include detected gamma or neutron radiation data, a measured vehicle position, a vehicle presence determination, or an image.

data collection mode See data mode.

data mode A device operating condition for the collection of data from detectors in the device. See also mode.

data type Classification of how to interpret a a specified amount of digital bits. Data types include integers, floating point numbers, Booleans, and strings.

data-out device A device that communicates with the control module using the data-out message group and the core message group.

data-out message group The set of messages that convey information generated within a RAPTER portal to data-out. The group also includes messages for requesting data.

denormal Floating point numbers with leading zeros in the significand. Also sometimes referred to as subnormal numbers.

detection zone A region of space within a portal of which measurements are made by detectors.

detector A device that senses physical quantities that are analyzed to determine the need for further inspection of a conveyance. These devices include gamma, neutron, and vehicle presence detectors. Devices that sense quantities not directly related to the determination of the need for further inspection, such as temperature, voltage, humidity, etc., are not considered detectors for the purpose of this document.

device A physical component that is connected to the control module through an Ethernet connection in a manner specified by this document. Implements core messages and other message groups according to the functionality of the device plural.

dual-homed A device with more than one network interface that are each attached to separate Ethernet networks. Among reasons devices may be dual-homed is to bridge networks or act as a proxy between separate networks. .

Dynamic Host Configuration Protocol Standardized network protocol used on Internet Protocol (IP) networks to dynamically distribute network configuration parameters, such as IP addresses, for interfaces and services.

energy calibration The mapping of the detected electronic signal of a detection gamma radiation quanta, to the actual energy deposited in the detection medium. The usually takes the form of an equation that maps multichannel analyzer channels to to an energy scale determined by gamma radiation sources with known emissions.

health data See parameter.

Inf ±Infinity. A special value in floating point representation of numbers indicating positive or negative infinity.

interface A boundary definition that allows two devices to communicate. Interface may also refer to a physical interface, for example an Ethernet interface, or network interface card.

interlock A feature used to enforce the mutual dependence of two mechanisms or functionalities. Interlocks are commonly used for safety purposes to prevent potentially hazardous operations if required prerequisite are not met. An example interlock might be a switch that prevents a high voltage power supply from turning on if the enclosure has not been properly closed.

item In the context of radiation measurements, radiation data from an item of interest, or simply item, is analyzed in order to determine the need for further inspection. See also background.

JTAG connector A digital electronics connector typically included for development purposes.

list mode For radiation and vehicle presence detectors, a data recording mode in which each detection event, including one or more attributes of the event and the precise time of occurrence, is reported sequentially. For vehicle presence detectors such as break beams, each detection event and its time of occurrence is likewise reported sequentially.

live time Time during which a radiation detector is capable of processing an input radiation pulse.

live-time dwell mode A data recording mode of a radiation detector for accumulating data during a single interval of live time. See also mode.

may As defined in Reference [1], this word means that an item is optional and an implementer or device is free to choose whether to implement or use the item. If the option is chosen, the item must be implemented as specified.

mechanism specifically, **messaging mechanism** A set of messages within a message group that addresses a particular functionality. Examples are the parameter mechanism for managing parameters, and the firmware upgrade mechanism for managing upgrades, both within the core message group.

message A packet of binary data with interpretation as defined in this document. See Section 4 for definitions of messages.

message group A set of RAPTER messages that are related by general functionality.

message type A specific, defined RAPTER message within a message group.

mode a selectable condition for hardware or software. See also operating mode and data mode (including list mode, live-time dwell mode, real time interval mode, real-time dwell mode).

module A physical component of a RAPTER portal that connects to the Ethernet of the portal, specifically any of the following modules defined by this document: analysis module, control module, radiation detection module, power management module, or vehicle presence module. See also device.

multicast A data transmission that is addressed to a group of destination devices simultaneously.

must Means that the statement is an absolute requirement of the specification, as in Reference [1].

Network Interface Card A physical adapter that connects a device to a network.

Not a Number A special value in floating point representation of numbers indicating an invalid value.

occupancy The presence of a conveyance within the detection zone of a portal.

operating Adjective relating to portal or device operation.

operating mode A selectable operating condition of a device. Options are Standby, Ready, and Operating. See also mode.

operation The use of a device or system in its intended function.

operational Adjective relating to operations.

operations The full range of activities involved with the production or use of portal data for purposes of scanning, inspecting, and interdicting conveyances.

parameter A settable, measurable, or indicative (or a combination of these) quantity communicated using specific messages. Devices or the control module define parameters relevant to themselves. Measurable parameters are used, for example, to relay the state of health or condition of devices, or relay status information such as CPU temperature or device power usage. Indicative parameters report existing stored values, such as control settings on devices. Settable parameters are those with values that can be changed in response to a received message, such as new values of control settings on devices. See Section 2.1.1 and Section B for more information.

parameter mechanism The messaging procedure for reporting of a parameter update by a device to the control module, outputting the updated parameter by the control module to data-out messages, and the facility to query or set the current parameter value. (See also Section 2.1.1. or 4.1.78.

portal See RAPTER portal.

power management message group The set of messages that relate to the control and monitoring of the power management module. The included content may be proxied by the control module to the control and data-out message group.

power management module A physical device that provides control and monitoring of the portals power supply, with functionality similar to that of an Uninterruptible Power Supply and is interfaces to the control module using the power management message group.

Radiation detection equipment Radiation data or vehicle presence data or both.

radiation detector data Data, from a measurement of neutron or gamma radiation, produced by a radiation detector module and included in a message.

radiation detector message group The set of messages that relate to the detection and reporting of data by a radiation detector module. Radiation detector messages transit between the control module and the radiation detector module(s). The included content may be forwarded by the control module using the analysis message group or the data-out message group.

radiation detector module A physical device that detects gamma radiation, neutron radiation, or both, and provides radiation data to the control module via the Ethernet network. The radiation detector module implements core messages and radiation detector messages. See Section 1.3.2.

Radiation Portal Monitor A system for performing radiation monitoring of a portal with functionalities and components as defined by a user. See also RAPTER system.

radiation sensor panel Radiation portal monitors are often constructed using multiple panels that each contain some number of gamma and/or neutron detectors. Combining multiple RSPs together allows constructing portals with geometry and detection sensitivity desired.

RAPTER Interface Specification The communications, architectures, assumption, and requirements specified in this document.

RAPTER portal A portal consisting of a RAPTER control module and the following RAPTER modules connected thereto on a dedicated Ethernet network: an analysis module and all radiation detector datas and vehicle presence modules.

RAPTER system A system, as defined in the RAPTER Interface Specification, consisting of a control module and all devices connected directly to the control module, whether the connection is by Ethernet (see also module, device) or by a physical interface (see Section 1.2).

real time Time, true time, or wall-clock time, irrespective of the state of a detector or device.

real time interval mode A data collection mode of a radiation detector for accumulating data during repeated, contiguous intervals of real time. See also mode, real-time dwell mode.

real-time dwell mode A data collection mode of a radiation detector for accumulating data during a single interval of real time. See also mode, real time interval mode.

reference mark Physical mark, or marks, on the outside of a sub-device used to position it, or a measuring device, at a point where the conventionally true value of a quantity is to be measured, unless the position is clearly identifiable from the construction of the sub-device.

relay command message A message type of the command message group that a command device may send to the control module. The relay command message is a wrapper for a request message to be sent to a specific target device from the originating command device; the control module treats the relay command message as a request (which it may deny) and proxies and controls all communication between the originating device and the target device. The payload message may be of any request type the control module could send to the target device. The request may, for example, change settings or state, or obtain information with respect to the target device.

request for comments Formal document from the Internet Engineering Task Force that typically is used to define protocols, standards, or definitions.

requirement A necessary or compulsory condition.

Secure Shell A network protocol to securely facilitate remote network services over an unsecured network.

setting A value that is settable by a message. A settable parameter.

shall As defined in Reference [1], indicates an absolute requirement of this specification.

should As defined in Reference [1], this word means there may exist may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

significand Part of floating-point numbers consisting of its significant digits. Also referred to as mantissa or coefficient.

sub-device A major physical component within a device for which location information may be specified and queried. For example, multiple radiation sensors as sub-devices within a radiation detector module.

system A set of connected things or parts that operate together[1]. See RAPTER system.

transistor–transistor logic Digital signal that can represent either an off (0) or on (1) state. Devices may use TTL inputs, for example, for inputs by external switches, or interrupts.

unicast A one-to-one transmission from one device on the network to another device, each with a unique network address.

Unicode Transformation Format 8-bit A character encoding which is a superset of ASCII, and can encode all of Unicode.

Uninterruptible Power Supply Provides power to a load when mains supply power fails, is disrupted, or outside of tolerances. Typically also protects against voltage surges or sags..

Universal Serial Bus A standard that defines cables, connectors and communications protocols for connection and communication between computers and devices.

UNIX A family of operating systems, including Linux, BSD, and macOS.

User Datagram Protocol A basic network message that provides for addressing and data integrity checks, but does not account for unreliability in the network (e.g., unbeknown to the sending device, a sent UDP message may never arrive at the destination).

vehicle presence data Digitized data produced by a vehicle presence module and included in a message that provides information relating to the presence, position, or motion of a vehicle, or other attributes of the vehicle.

[1] https://dictionary.cambridge.org/us/dictionary/english/system.

vehicle presence message group The set of RAPTER messages that relate to the detection and reporting of the occupancy of the portal by a vehicle. Vehicle presence messages transit between the control module and the vehicle presence module(s).

vehicle presence module A physical device of a RAPTER portal that detects vehicle presence and provides vehicle presence data to the control module. The vehicle presence module implements core messages and vehicle presence messages. See Section 1.3.3.

Virtual Network Computing A graphical desktop sharing system used to remotely control another computer.

WebSocket A persistent bi-directional, message based protocol defined in Reference [2].

Section 1

Overview

1.1 Overview

Radiation Portal Monitors (RPMs) were deployed throughout the port and border infrastructure of the United States (U.S.) beginning in 2003 to monitor for the possible presence of uncontrolled radiological and nuclear materials. Since that time, the U.S. Government (USG) has learned much about the operational challenges faced in the field. Principal among the shortcomings has been the lack of flexibility afforded the USG when all Internet Protocol (IP) rights and interfaces of the system are owned by the Original Equipment Manufacturer (OEM).

In this document, USG has designed an open architecture with defined interfaces within a portal for use in future deployed portal systems. This document defines the RAdiation Portal Technology Enhancement & Replacement (RAPTER) interface that will make hardware and software upgrades easier for the USG to manage. Any RAPTER module within an RPM system can easily be replaced without concern about the interfaces. The open architecture will also enable the USG to conduct studies of special-purpose algorithms and performance enhancements in real-world operational conditions.

RAPTER provides a modular design with well-defined modules and interfaces that OEMs will build their systems around. There are five modules types within a RAPTER system – radiation (gamma-ray and/or neutron) detection, vehicle presence, analysis, power management, and control. While there may be multiple radiation detector module, vehicle presence modules, and power management modules in a system, there is only one operating analysis module and one control module. Inherent in the architecture is emphasis on state-of-health information that will enable the user to monitor equipment health for reliability and systematically reduce maintenance costs over the life of the system. A standard Ethernet network is specified between the modules to facilitate network connectivity, which is essential to support remote diagnostics, software/firmware updates, and performance enhancements.

One of the primary system architecture changes compared to existing RPM systems is that the control module will be physically separate from the analysis module that contains the detection and alarm algorithms. The RAPTER system will also need to enable remote operations, external data archiving, and developmental analysis capabilities.

This document is a USG-controlled Interface Control Document (ICD) that defines all interfaces within a RAPTER portal as well as the interface to external networks. Example source code for these interfaces may be made available to OEMs in order to ease the engineering task of

implementing the new open system architecture.

1.2 Introduction

This document presents a USG specification that defines a modular architecture for a portal used for monitoring radiation, the allowed communications to and from the modules of the portal, and the communication between the portal and the external world. A RAPTER portal, constructed from the interfaced modules, may be interfaced with further devices as specified herein. Compliant modules and devices may be upgraded or replaced without replacing the entire system. The functionalities and dependencies of modules and devices are defined, as are the allowed communications between them. Functionalities include alarming, archiving, and device health monitoring.

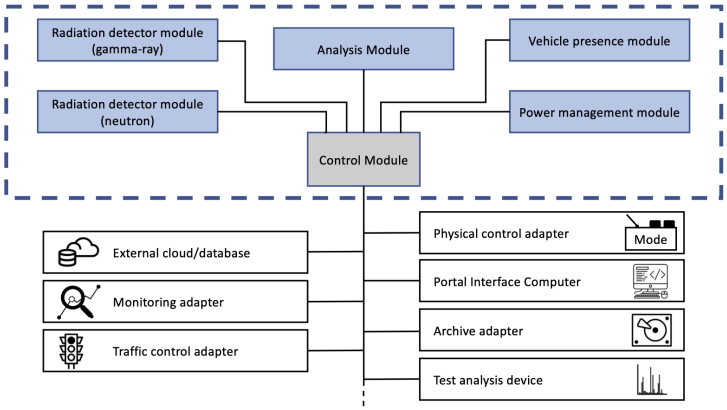

Figure 1.1: Diagram of a potential RAPTER portal (within the dashed line) with connected notional data-out device and command devices (outside the dashed line). Modules reside within the portal. All communications over the diagrammed interconnections are specified RAPTER messages that start or end at the control module and transmitted over Ethernet networking. Data-out device and command devices functionalities will be site and location specific. Non-networked interfaces and controls, such as manual power switches, are not diagrammed.

The set of interface specifications defined herein is referred to as the RAPTER Interface Specification. The specification defines messages that provide full access to digital data generated by the RAPTER system, including data from radiation measurements, vehicle presence data, analysis results, measures of system health, and numerous parameters relating to the status, operation, health, maintenance, configuration, and sustainment of the system.

This document does not specify system performance or how any modules or devices internally work.

A diagram of a potential RAPTER portal can be seen in Figure 1.1. The architecture defined by this Interface Specification includes five types of RAPTER modules (analysis module, control module, radiation detector module, vehicle presence module, and power management module) that each perform a concrete function in the portal. All communications over the diagrammed network interconnections are specified RAPTER binary messages that start or end at the control module. Each component that connects directly to the control module by Ethernet is referred to as a device. Devices do not communicate with each other, except by Ethernet through the control module. The term module is reserved for the five types of RAPTER modules, and other devices

are not referred to as modules, regardless of their actual modularity. Devices required for stand-alone portal operations are connected to the control modules Ethernet network. A RAPTER system comprises the control module and the directly connected devices.

A radiation detector module is a physical device that detects gamma radiation, neutron radiation, or both, and provides radiation data to the control module. A vehicle presence module is a physical device that detects vehicle presence and provides vehicle presence data to the control module. The power management module is a physical device that provides common Uninterruptible Power Supply (UPS) capabilities and monitoring to the control module. The portal may have more multiple radiation detector modules, vehicle presence modules and power management modules. The portal has a single analysis module, a physical device that performs the analysis of radiation and vehicle presence data in order to determine the operational need for further inspection of the conveyance.

Communications are conducted with binary messages that are defined in this document. Message definitions are organized in message groups that are tailored for different types of devices. For example, portal data are reported from the control module in data-out messages. Certain devices are able to issue command messages that influence system operations; these devices are referred to as command devices. Command devices may issue commands to other devices by relay through the control module, which exercises control over all messages. The RAPTER modules, other than the control module, are not command devices. These and other message groups are introduced in Section 3.1, expanded further in Section 3, and defined in Section 4.

There are two broad types of non-module devices, according to whether they communicate with the control module with command messages and data-out messages (command devices), or only data-out messages (data-out devices).

The devices in the diagram that are outside the RAPTER portal are notional. These devices make use of data provided by defined RAPTER messages transmitted by the control module of the portal. The notional devices are suggestive of how a RAPTER portal can be integrated into a user's site operations. For example, Figure 1.1 illustrates notional devices that interface to the CBP network. As a further example, a Human interface device (HID) (not diagrammed) could be constructed to include a monitoring adapter (for display of portal information), a physical control adapter (for execution of actions), and an integrated graphical user interface (GUI) that complies with the user's standard procedures.This sort of custom integration into a user's site operations can be carried out without modifying the RAPTER portal. Figure 1.2 shows an alternative scenario where a dedicated command device, referred to as a Portal Interface Computer, sits between the control module and the site network and portal controls.This document does not explore or specify the notional device themselves, but instead focuses on the messages that such devices are able to receive and send in communication with the control module.

All RAPTER messages are sent by twisted-pair, Cat 5 or higher, Ethernet connection to or from the control module.

The possibility that specialty, non-Ethernet, physical interfaces may be required by a user's site operations is addressed in Section 1.5.4.

A RAPTER portal may be operated as a stand-alone system, or it may be integrated into a site having multiple portals with additional tie-ins to site operations. An example network architecture for such a site is diagrammed in Figure 1.2.

Encryption and device authentication mechanisms are outside the scope of this interface specification, but it is worth noting the technologies used within RAPTER are commonly used with

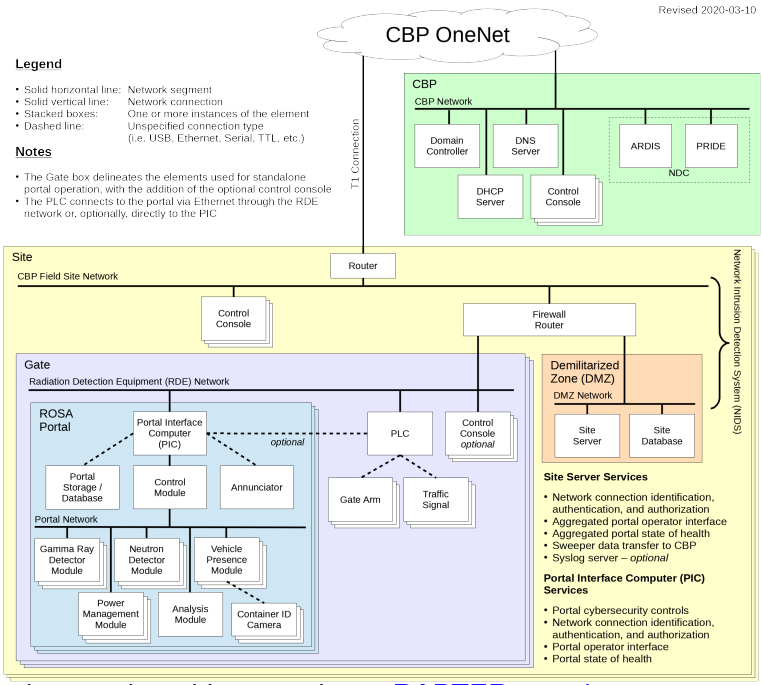

Figure 1.2: Notional network architecture that a RAPTER portal may operate in. The RAPTER interface only applies to the RAPTER portal in the blue container, and only the interfaces between the control module and the other devices it connects to.

well established encryption and attestation infrastructures (i.e., Transport Layer Security (TLS), Public Key Infrastructure (PKI), etc.)

The following overview subsections discuss the RAPTER modules, the RAPTER message group, device networking, and physical interfaces.

1.3 RAPTER Modules

This subsection gives general descriptions of the modules. Messages are transmitted between the control module and other RAPTER modules over a Ethernet network. The possible use of physical interfaces is discussed in Section 1.5.4. Any monitored condition that interferes with operations or prevents use of a device must be reported via the parameter mechanism or the 4.1.13 mechanism.

1.3.1 Control Module

The control module acts as a central point to which all devices (i.e., radiation detector modules, vehicle presence modules, power management modules, the analysis module, data-out devices, and command devices) connect to in order to send and receive data, commands, and responses to commands. Devices may only communicate with the control module and not with each other. As such, the control module functions as a gatekeeper for which devices can connect, and determines what information they can send or receive. It acts to coherently package information from modules and send it to the analysis module, the data-out devices, and the command devices; it also requests interim and final analysis results from the analysis module and relays them to the data-out devices and command devices. The control module processes command messages for

setting the position and location configurations of detectors within devices (see Section 2.1.1.1). The control module implements all message groups.

1.3.1.1 Hardware for Control Module

The control module hosts the Ethernet-based network; e.g., provides Dynamic Host Configuration Protocol (DHCP), network switching, etc. The control module must also act as, or make available, an IEEE 1588 Precision Time Protocol (PTP) [3] grandmaster clock to the portal devices, as explained in Section 1.5.3.

> Example: The control module, for example, could be implemented by using a standard computer with it's Network Interface Card (NIC) connected to a networking switch servicing the portal devices, the computer providing the DHCP service, and the switch also acting as, or is connected to, a stratum 1 PTP clock for the portal devices.

> Example: The control module could also be implemented as a single integrated device with all networking and portal operation logic controlled by the same computing device. The standard system clock could be used as the PTP grandmaster source, which in turn could be set using Network Time Protocol (NTP) via an external or dedicated connection.

The possible use of physical interfaces is discussed in Section 1.5.4. The physical implementation used for the control module is not further defined.

1.3.1.2 Logic for Control Module

The control module determines which devices may connect, and how the system will react to differing request messages, device failures, or changing environmental or operational conditions. The control module implements the operating mode selected by the user, or changes mode in response to changing conditions. The control module controls the output of all portal data.

The control module may disconnect devices that attempt communications outside of the RAPTER Interface Specification, and similarly, other devices may disconnect if the control module uses disallowed communications. (For more information see Section 3.5.1.)

The messages which the control module may issue or to which it must respond, and their meanings, are defined in this document (see Section 4). All parameters of any module or device shall be network accessible through command messaging. Further, all settable parameters shall be settable through command messaging. Updates of firmware, executables, and software libraries for the control module proceed through core messages wrapped in CmdDeviceRelayRequest messages (see section 1.3.1.3). Further messages are defined herein.

The control module can receive user commands through an Ethernet connected device for physical control of the portal, and output portal information to a connected storage device. Portal information includes data generated by the modules, parameters, and any other data generated by devices or by the control module itself. Physical control and output of portal information may be combined in a connected Graphical User Interface (GUI) device that functions both as a data-out device and as a command device.

The control module receives data from the radiation detector modules and vehicle presence modules, determines occupancy state, and determines whether to group the radiation detector data as background, occupancy, or "other." Non-occupancy radiation data that are known to the control module to be unacceptable for use as background data, are grouped as other. The control module transmits data-out message group messages containing vehicle presence data and radiation detector data to the analysis module for processing. For each occupancy, upon request from the control module, the analysis module analyzes the provided vehicle presence data and radiation detector data, and returns results to the control module, including both the results of the analysis, as well as information necessary to reconstruct the results (e.g., backgrounds used, threshold settings, etc.) contained within a N42 2012 file (see Reference [4]). The control module then transmits this information to data-out devices and command devices.

The control module is the component within the portal whose implementation is most natural to customize to fit the requirements of a specific deployment without modifying other modules.

1.3.1.3 Relay command messages

It is strictly required that each device connected to the control module network communicate with only the control module, and only on the network. Since, at times, it is still useful to manually send an individual device a command (e.g., for maintenance or troubleshooting purposes, to upgrade firmware, or to set a parameter), a relay command mechanism has been provided that allows command devices to send the control module a relay command message, which contains a target device universally unique identifier (UUID) and device-specific RAPTER request message. The control module can then act as a proxy and send the device-specific RAPTER request message to the target device; the command device relays the device's response message back to the originally requesting command device. This mechanism is primarily intended to allow adjusting or debugging of individual devices and is not typically to be used during operation. The control module can be implemented to reject any relay command for any operational reason, but may not use information not relevant to operations.

1.3.2 Radiation Detector Modules

Gamma and neutron detectors implement radiation detector message group in addition to the core message group. The internal workings and detection sensitivities of gamma or neutron detector modules are not specified by the RAPTER Interface Specification. A radiation detector module shall be completely self-contained and detect gammas or neutrons or both. Each radiation detector module in a RAPTER portal shall operate completely independent of the other non-control devices in the system. Communication over the network between modules shall not occur. The operations of one device shall not influence the operations of another device over the Ethernet network, except possibly through the logic implemented in the control module.

However, it may be possible for devices to physically influence each other (not through the network). For example, a gamma detector containing a normally sequestered radioactive check source could expose that source to perform an energy calibration, thereby altering the physical environment for the other detectors in the portal. This may only happen when the control module explicitly says it can (see ActiveMaintenance).

Radiation detector modules NIC (Ethernet adapter) must support hardware-based time stamping of PTP packets. Sub-millisecond time synchronization between devices is required for operating

consistency of a RAPTER system. The control module shall support one-microsecond synchronization, which some anticipated implementations may need between detectors.

Radiation detector modules may internally contain multiple sub-device radiation sensing elements, whose data shall be reported separately from each other. All radiation sensing elements within a module must operate in the same data collection mode. There are four data collection modes that detectors may support:

1. List mode (each detection event is individually read out and time stamped),
2. Real-time interval (radiation data is output at repeated intervals of real time),
3. Live-time dwell(accumulated radiation data is output at completion of a single interval of live time), and
4. Real-time dwell (accumulated radiation data is output at completion of a single interval of real time).

Modules that support neutron detection must support list mode, while gamma detecting modules must support real time interval mode. Support for all four modes is encouraged but not required. Individual detectors must also support reporting their energy calibration.Radiation detector modules should also report health status and sufficient measured values (e.g., parameters) with which to monitor hardware condition and health.

1.3.3 Vehicle Presence Module

When a vehicle is occupying the portal, radiation data are analyzed to determine whether further inspection of the vehicle is needed. A vehicle presence module provides information in real time that may be used by the control module to determine whether the portal is currently occupied by a vehicle and if so, the speed of the vehicle. The vehicle presence module may also may provide the location or changes in speed of the vehicle in real time or other information about the vehicle. Vehicle presence may be verified with digital imagery of the vehicle that is acquired by camera sensors that are sub-devices of the vehicle presence module.

A vehicle presence module implements the vehicle presence message group in addition to the core message group. The RAPTER specifications do not require the use of any particular sensing technology. A basic vehicle presence module may consist of multiple infrared binary status break beam sensors located at both the entrance and exit of the detection zone of the portal, so that the control module can determine if a vehicle is present, as well as use timing information from the break beams to determine the vehicle speed. A more advanced vehicle presence module may use, for example, ultrasonic or lidar based distance sensors to allow tracking the vehicle's progress while it is approaching, in, and leaving the portal in order to provide better positioning information for any radiation detected.

The vehicle presence module may provide digital pictures of the vehicle for use in reading shipping container numbers, or for other purposes. The module may capture pictures based on its own criteria (for example when break beam sensors are initially interrupted by a vehicle), as well as when prompted.

A single vehicle presence module may have any number of binary status detectors (such as break beams), distance sensors, and/or cameras. The portal must include a vehicle presence module that is able to provide occupancy information sufficient to determine vehicle speed.

Vehicle presence modules NIC must have hardware support for PTP timing (see Section 1.5.3), and must report all monitored values or external inputs using the parameter mechanism. See Section 2.1.1 for more information.

A vehicle presence module operates in a single data collection mode (see Section 2.1.2.2): it sends data when a sensor's value changes. For example, whenever any of the beams of an IR break-beam system changes state, the vehicle presence module sends a message, containing the time stamp of the change and the new detector's state, to the control module. Other examples of this mode are a distance sensor that may send each new measured value as it becomes available, or a picture that may be sent as soon as the vehicle enters or leaves the portal.

1.3.4 Analysis Module

It is the job of the control module to package radiation detector data and vehicle presence data, and other information from relevant devices into a coherent message, and send it to the analysis module. The control module can ask for a preliminary analysis result, perhaps while a vehicle is still in the portal, or a final analysis result for the most recent item of interest (i.e., vehicle). The analysis module must implement the data-out message group, and the analysis message group in addition to the core message group. For more information on how the vehicle occupancy state is determined, see Section 2.1.2.4.

The analysis module is a stand-alone computing device. There shall be at most one analysis module. If additional analysis modules attempt to connect to the control module, the control module shall terminate the WebSocket connection of the newly connecting analysis module, and issue a `DataOutMiscNotificationPush` message to all data-out device and command devices.

Thresholds, radiation analysis findings on which to alarm, and other adjustable pieces of information can be specified through setting values using the parameter mechanism. Additional information, more detailed than that facilitated by the parameter mechanism or the device location mechanism (described in Section 2.1.1.1) may also need to be specified in order for the analysis module to operate with the set of devices in the portal. The RAPTER Interface Specification provides the core message group to do this via firmware upgrades. For example, if a new model of radiation detector module is added to the system, a firmware upgrade may be necessary to provide the detector response function and detailed setup configuration of the module and sub-device to the analysis algorithm.

Final analysis results for an occupancy are provided by the analysis module to the control module through the `AnalysisItemFinalResultsRequest` message. This message contains fields for the alarm status, the alarm type, a measure of confidence, a list of isotopes identified, and a few other fields, including one to hold the contents of a N42.42-2012 [4] file. The N42 file contains the data, including any backgrounds, other radiation data, or vehicle presence data used to perform the analysis, as well as the analysis results themselves. The N42 file contents shall provide sufficient information to allow reproduction of the analysis results by a re-analysis replay tool.

1.3.5 Power Management Module

Power management modules provide typical UPS services with associated control and monitoring using the power management message group and core message groups. The primary role of these modules is to provide battery-based backup power when the main power fails or becomes out of tolerance; they also typically protect against spikes, surges, and sags in voltage. They report their current status, estimated remaining battery levels, and self test results through the power management message group to the control module. The power management module will likely provide electrical protection and battery backup continuously, not just when placed into operating mode via RAPTER messaging.

1.4 Introduction to Message Groups

This section gives a general introduction to the RAPTER message group: analysis, command, core, data-out, radiation detector, power management, and vehicle presence. The groups and the message types within each group are discussed further in Section 3.1 and specified in detail in Section 4.

The groups used by RAPTER modules are shown in Table 1.2, and those used by other types of devices are listed in Table 1.1.

Communication to and from devices is conducted by message sets that allow implementations with functionality tailored to the application. All connections are made to the control module, and communication between devices, if any, must pass through the control module. Communication between devices outside the RAPTER Interface Specification is not allowed.

Communications are conducted using messages that are grouped as presented in Table 1.1.

Message Group	Devices That Implement
analysis	analysis module
command	command devices
core	all devices
data out	analysis module, command devices, data-out devices
power management	power management module
radiation detector	radiation detector module
vehicle presence	vehicle presence module

Table 1.1: **RAPTER Message Groups.** Listing of message groups defined by this document, and which devices must implement the message group. The control module must implement all message groups, and devices may not only partially implement a given message group.

The core message group is a set of messages, each one of which every device connected to the control module must be able to receive or send as a reply, or “push”, as the message type dictates. Updates of firmware, executables, and software libraries for the control module and devices proceed through core messages. Command messages relate to command and control of devices and the system. The analysis message group is the set of messages that relate to the request for, and the output of analysis results from, the analysis module. Data-out messages convey information generated by the RAPTER system. Radiation detector messages relate to

Module	Message groups
Analysis	core, analysis, data-out
Control	core, analysis, command, data-out, radiation detector, vehicle presence
Power Management	core, power management
Radiation Detector	core, radiation detector
Vehicle Presence	core, vehicle presence

Table 1.2: **Message Groups used by RAPTER Modules.**

the detection and reporting of data by a radiation detector data. Vehicle presence messages relate to the detection and reporting of the occupancy of the portal by a vehicle, and power management messages relate the control and reporting of the power management module to the control module.

Message Group	Request	Reply	Push	Acknowledge
analysis	control module	device	control module	device
command	device	control module	control module	device
core	control module	device	device	control module
data out	device	control module	control module	device
power management	control module	device	device	control module
radiation detector	control module	device	device	control module
vehicle presence	control module	device	device	control module

Table 1.3: Originating source for each class of messages in each message group.

Within a message group, individual messages request information or action from a device or the control module, which then will reply. Similarly the control module or device will "push" messages to the other device on the connection, which then may acknowledge them. Further, within a single message group, all requests will only originate from either the control module or the device, and the replies will all go the other direction, and similarly for pushes and acknowledges. Table 1.3 summarizes the directions of requests, replies, pushes, and acknowledges in each message group.

1.4.1 Analysis Message Group

A RAPTER portal has a single analysis module. The analysis message group contains messages with which the control module requests and receives analysis results from the analysis module. The data-out message group, which the analysis module also receives, contains messages that provide the analysis module with radiation detection data and vehicle presence data used to perform its analysis. If additional “test” analysis modules are desired to be run in parallel, they can be configured as data-out device to receive all the relevant data in real time, but they will not report their analysis results through RAPTER messaging (e.g., will need to store their results on a USB drive, or some other medium). See Section 4.5.

1.4.2 Command Message Group

Devices using the command message group also receive, and can send, the same messages as data out devices. Command devices can issue commands that influence the operating status of the portal, like starting or stopping of data taking, or even issuing requests (through the control module) to individual devices (using relay "request" messages, see Section 2.2). Command devices could be used, for example, to perform remote debugging and maintenance operations, or control the portal via a GUI program, website, or a dedicated human interaction device. See Section 4.7.

1.4.3 Core Message Group

Every device must implement the core message group, a large group of messages for controlling device-level operations, settings, updates, and information reporting. See Section 4.1.

The core message group, for example, facilitates changing the operating state of individual devices, reporting of parameters (e.g., high voltage, Central Processing Unit (CPU), temperature, etc.) and hence state of health, negotiating version of the messaging protocol to use, telling the control module basic information or capabilities of the device, recovering buffered data after network interruptions, firmware upgrades, and other functionality common to all devices. Even if a device does not support an optional feature, like data buffering, devices are expected to respond to those messages, although the response could be "not supported." See Section 4.1 for a full list of messages.

1.4.4 Data-Out Message Group

Data-out devices receive all radiation data, vehicle presence data, analysis results, parameter updates, status of each device, and device connection and disconnection notifications. Requests sent through command messages are partially mirrored to data-out devices via the DataOutDataFrame::received_requests message field, and the results of those requests (device operating state changes, energy calibrations, parameter value changes, etc.) are sent to data-out devices in the other fields of DataOutDataFrame that is in the DataOutDataPacketPush message. Data-out devices may perform actions based on the content of the data received, for example to display portal status on a GUI, drive a traffic light or gate arm, or archive data as it is received. See Section 4.6.

The flow of information is strictly from the control module to data-out devices, with the following exceptions in which the data-out devices provide a information:

- Upon request by control module to a data-out device for device information (DeviceInfoRequest message).
- The device shall send parameter updates to the control module, which includes device health information (ParameterUpdatePush messages).
- NotificationPush messages from the device
- Responses to core message requests from the control module.

1.4.5 Radiation Detector Message Group

The radiation detector message group is used by the control module for managing the radiation detector modules and for obtaining interpretable radiation data from them. The radiation detector modules use this message group to identify device features and report radiation data. Energy calibration is managed and reported with this message group.

1.4.6 Power Management Message Group

The power management message group is used by the control module for managing the power management modules and for obtaining status of the mains power (the input power to the module), the output power (i.e., that powers the rest of the portal), and the backup battery status. When mains power is lost and the batteries are nearly exhausted the control module can power down the portal in a safe manner, or for debugging purposes trigger the power off, and then power back on of the portal.

1.4.7 Vehicle Presence Message Group

The vehicle presence message group is used by the control module for managing the vehicle presence modules and for obtaining interpretable vehicle presence data from them. The vehicle presence modules use this message group to identify device features and report vehicle presence data.

1.5 Device Networking

During operation all communications between devices and the control module take place by transmission of binary messages over a WebSocket connection sitting on top of a standard Transmission Control Protocol/Internet Protocol (TCP/IP) networking stack, using IPv6. The binary message formats are defined in Section 3.3.3, as interpreted by the rules given in Section 3.3.3. Devices of the system communicate with the control module but do not communicate with each other. Only the defined RAPTER binary messages may be used for communications. Similarly, connections to the control module from the outside world take place using a WebSocket connection and the pre-defined message formats in this specification.

Interruptions to the WebSocket connection are exceptional events such as network interruptions, device power cycles, or to isolate non-compliant devices.

RAPTER systems communicate using standard networking infrastructure (RFC 894 [5] and successors and extensions), TCP/IP software stack (RFC 791 [6], RFC 2460 [7], RFC 2710 [8]) and WebSockets (RFC 6455 [2]) connection - that is, using the standard network technologies which are nearly ubiquitous in general purpose computer networking, the internet, and in many embedded devices. This interface specification defines how devices communicate over Ethernet-based [9] networks using twisted pair links, Cat 5 or higher cabling with connectors electrically equivalent to 8P8C.

Since at the application level, the physical medium used to facilitate the networking may be abstracted on many computing platforms, it is worth noting other physical communication mech-

anisms such as WiFi (IEEE 802.11 [10]), fiber-optic, power over Ethernet [11], or other technologies could be candidates for inclusion in future versions of the RAPTER Interface Specification. However, since all testing and reference implementations of this RAPTER Interface Specification used Ethernet networks, with no consideration or testing of special requirements (e.g., reliability, time synchronization, potential interference, latency effects, etc.) that these other physical communication mechanisms may impose on the system, this current RAPTER Interface Specification requires Ethernet-based network communications.

Under the RAPTER Interface Specification, the control module facilitates all network functionality for the portal. The control module acts as the network hub that performs DHCP services (RFC 3315 [12] and updates), performs network switching, and provides time synchronization. The control module deals with all aspects of the network, including managing network congestion, isolating devices from each other, network routing, and any other related responsibilities. The control module also acts as a WebSockets server to which each of the other devices create connections; this connection mechanism is explained in Section 1.5.1.

To implement communications from outside of the portal to the control module (e.g., remotely control or monitor the portal), a data-out device or command device, that is dual-homed would have to be used. Command devices can facilitate interacting with individual devices in the portal (e.g., debugging, upgrading firmware, or issuing specific commands) via the relay command messages, which ask the control module to relay the requests and responses of a command devices. This is the mechanism that would be used to remotely configure or debug a specific device.

The motivation for requiring the portal's network to be separate from any other network includes:

- The portal's operations are not dependent upon the operating conditions of an external network. For example, if an external network goes down, or is degraded, the portal can continue running, potentially unaffected.
- The control module can enforce network isolation of devices from each other. For example, it can ensure a gamma detector's network messages cannot influence a neutron detector's operation.
- The control module retains the ability to know exactly which modules are connected to it, and to take any actions it deems necessary (e.g., disconnecting a "noisy" device from its network) to ensure good system operation.
- Configuration ambiguities are reduced. That is, as far as a device connected to the portal is concerned, there can only be one control module, and similarly, a control module has exclusive control over any of the devices on its network.
- Some degree of isolation of the network and the outside world is achieved. This helps to reduce packet collisions or other performance degrading conditions, as well as shield devices from errant, but non-malicious network traffic. Note that separation of networks through dual-homed computers is not a recommended cyber-security protection measure; see NIST 800-82 [13].

Device authentication and encryption of portal network traffic is not specified by this standard since each installation or deploying organization is likely to have their own unique infrastructure in place. The RAPTER Interface Specification, however, is compatible with, and has been tested

using both TLS [14] and IPSec [15] technologies to authenticate and encrypt information (using pinned certificates).

Settings and updates of operating systems may either proceed outside the RAPTER Interface Specification, or by using the FirmwareUpgradeRequest and related messages.

1.5.1 Establishing Device Connections

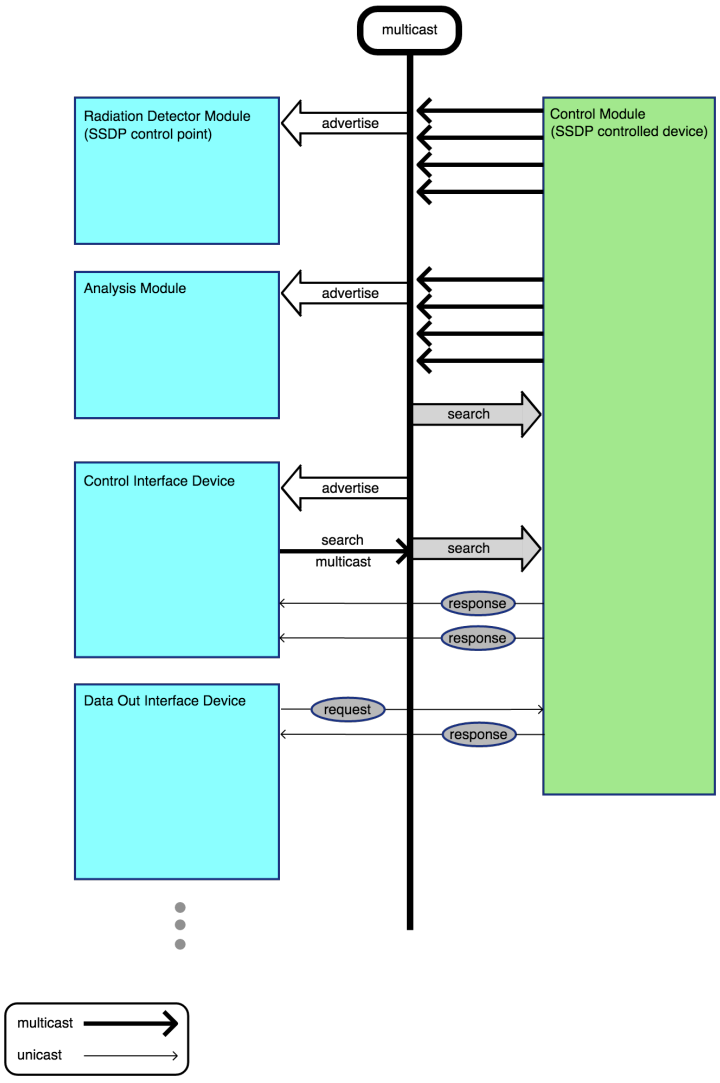

Figure 1.3: Schematic representation of the SSDP broadcast mechanism illustrating some examples of the sequence of message exchanges that may occur in order for devices to get the URL to which they should create a WebSocket connection to.

Radiation detector modules, vehicle presence modules, power management modules, the analysis module, as well as command devices and data-out devices are required to create a WebSocket connection to the control module. In order to know what IP address and port to connect to, a slightly amended version of *Step 1* of Universal Plug and Play (UPnP)/2.0 [16] is used to communicate to devices what address and port to connect to. *Step 1* of UPnP/2.0 is also referred to as the *"Discovery"* phase of UPnP/2.0, or separately as the *Simple Service Discovery*

Protocol (SSDP), which is how it will be referred to in this document. No other part of UPnP, outside of SSDP, is used, or allowed to be used on RAPTER devices.

SSDP defines a mechanism that multicasts Hypertext Transfer Protocol (HTTP) 1.1 headers as User Datagram Protocol (UDP) packets for devices to announce their presence on the network and provide some basic information. SSDP also includes a mechanism for devices to search for other devices on the network. SSDP was chosen because it is lightweight, has been widely used and tested, and because there are libraries available to implement it for most common programming languages and platforms. Implementing the necessary parts of the service from scratch in C, C++, and Python were also found to be minor tasks relative to other aspects of device development.

The control module acts as SSDP "controlled devices", which function in the role of a server, responding to requests from the other devices to send them the IP address and port on the control module to create WebSocket connections. Non-control module devices act as SSDP "control point" devices. Control point devices broadcast their presence and availability on the network, as well as perform "searches" for the control module; they also listen for responses or broadcasts from the control module.

This RAPTER Interface Specification adds the following customizations to the standard SSDP mechanism:

1. SSDP messages sent from the control module to other devices ***must*** include the HTTP header field "`WS_PORT.RAPTER.CWMD.DHS.GOV`" which provides the network port number the control module is accepting RAPTER WebSocket connections on. Devices will use this port number, and the IP address the packet originated from to create a WebSocket connection to the control module for all of the devices RAPTER network traffic to flow through.

2. A required header in SSDP messages is "`LOCATION`", which normally describes the URL on the device that presents an eXtensible Markup Language (XML) file containing a UPnP description of the device and its services. The value of this field must be blank. This requirement is driven by RAPTER devices not using the rest of the UPnP mechanism so this file is not needed, and making it available would add additional work, networking protocols to implement, and reduce the isolation of portal devices. Relevant device information is instead exchanged through the RAPTER interface.

3. The control module must announce and respond to searches for three types of SSDP-devices (the values "`NT`" and "`ST`" headers of SSDP messages)

   (a) upnp:rootdevice

   (b) urn:cwmd-dhs-gov:device:rapter-control-module:1

   (c) urn:cwmd-dhs-gov:service:rapter-control-module:1

4. The control module may optionally isolate devices on the network so that broadcasts from devices will not be received by any other device on the network, besides the control module; this is part of the control module's ability to ensure devices connected to it do not influence each other's behavior.

5. The UUID used for SSDP messages must be the same UUID used within the RAPTER interface to uniquely identify a device.

6. Once a RAPTER WebSocket connection is established between a device and the control module, there is no longer a need for the SSDP services, so all SSDP messaging can be stopped by either of the devices.

Devices on the network must use the IPv6 IP address FF02::C and port 1900 for link-local multicasting. The control module must send to, and listen on FF02::C, port 1900.

An example exchange of SSDP messages might be:

1. Once a radiation detector module is powered on and becomes ready, it might broadcast multiple (more than one due to the unreliable nature of UDP) messages similar to the following:

```
M-SEARCH * HTTP/1.1
HOST: [FF02::C]:1900
MAN: "ssdp:discover"
MX: 1
ST: urn:cwmd-dhs-gov:service:rapter-control-module:1
USER-AGENT: Unix/5.1 UPnP/2.0 RAPTER/1.0
CPFN.UPNP.ORG: Gamma Detector
CPUUID.UPNP.ORG: bbcc4467-e89b-adc4-567e-123412341234
```

   Where the `ST` field could have also had the value `urn:cwmd-dhs-gov:service:rapter-control-module:1`.

   The `CPUUID.UPNP.ORG` field contains the UUID of the radiation detector module, and the `CPFN.UPNP.ORG` is the "friendly" name of device. The `USER-AGENT` field value is that defined by HTTP/1.1, and the `MX` field specifies a maximum wait time (in seconds) for a response and should have a value between 1 and 5. The response line (first line of message) and `HOST` fields values are fixed. See reference [16] for further details on each field; line endings are given by the carriage return and line feed characters (`\r\n`) and the final line is blank.

2. The control module would respond with a unicast UDP response to the source IP address and port, similar to:

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=1800
DATE: Mon Oct 16 17:34:13 2017
EXT:
LOCATION:
SERVER: Unix/5.1 UPnP/2.0 RAPTER/1.0
ST: urn:cwmd-dhs-gov:device:rapter-control-module:1
USN: uuid:111e4337-e89b-12d3-1285-123412341234::urn:cwmd-
          dhs-gov:device:rapter-control-module:1
BOOTID.UPNP.ORG: 1508200453
WS\_PORT.RAPTER.CWMD.DHS.GOV: 7878
```

Which indicates the radiation detector module should create a WebSocket connection to *ws://[fe80::13d8:c37a:13aa:2012]:7878*.

In principle, the control module should wait a random amount of time between 0 and `MX` seconds before sending its response to the above `M-SEARCH`. However, the UPnP/2.0 specifications also allow the control module to adjust this value based on network congestion heuristics meaning the control module has the freedom to reduce or eliminate this delay if network conditions support it. Also, if the `M-SEARCH` request had contained the optional `TCPPORT.UPNP.ORG` field, then the response would have instead been by Transmission Control Protocol (TCP) (instead of UDP) to the port indicated.

1.5.2 WebSocket Connection

Once a device knows the address and port to connect to on the control module, as described in Section 1.5.1, the device should then initiate a WebSocket connection to the control module. The WebSocket connection is a standard connection as described in RFC 6455 [2], and is typically resolved via a URL similar to 'ws://ip.address'. Once a WebSocket connection has been established, the control module will start the RAPTER handshake, as described in Section 3.5 and the connection will persist for the duration of operations. If the WebSocket connection is interrupted, for example by one of the participating devices restarting or a network cable becoming temporarily disconnected, then the SSDP mechanism described in Section 1.5.1 should be used to re-establish the WebSocket connection. If a WebSocket connection is terminated or interrupted for any reason, devices should transition to stand by state (see Section 2.1.2) unless buffering has been negotiated as described in Section 3.5.2, in which case devices should continue in their current state of operation until the WebSocket connection has been re-initiated; see Section 3.5.2 for more information. Extensions to WebSockets, like compression (RFC 7692 [17]) may optionally be supported by RAPTER devices, but support is not mandatory, and usage will be negotiated during the standard WebSocket connection process.

1.5.3 Network based Precision Time Protocol

In order for the independent modules to operate coherently together and allow consistent interpretation of the produced data, the modules within a portal must be time-synchronized. To facilitate the time synchronization the control module must make a IEEE 1588 Precision Time Protocol (PTP) [3] grand-master clock available. This may be accomplished in a number of ways, including, for example:

- Using a dedicated PTP grand-master clock network device which in turn synchronizes off of GPS.
- Running PTP server software on the control module system which uses the computers real-time clock as a source; the real-time clock in turn may be set using an external source.
- Some network switches contain the ability to act as a PTP server.

The control module, radiation detector modules, and vehicle presence modules must support

hardware PTP packet time stamping[1] to allow achieving microsecond level synchronization. For general portal operations precise time synchronization is not as important for the analysis module, power management modules, command devices, and data-out devices, so hardware timestamping support is not required, as millisecond level synchronization can still be achieved using software timestamping. These requirements ensure that all the detectors and the control module are synchronized approximately to the microsecond level (network conditions, CPU loads, time since connection, time changes to the grandmaster clock, etc., can effect synchronization performance), while all devices are synchronized approximately to the millisecond level for consistent operations. If devices are not capable of achieving an appropriate level of synchronization, the control module may disconnect them and not allow them to participate in portal operations.

1.5.4 Physical interfaces

If a device or the control module implements a non-Ethernet, physical interface to supply an input, such as a button to acknowledge an alarm, or a manual push button to force an occupancy, then that input must be reported through the normal parameter mechanism (which includes a ParameterUpdatePush message (see Section 4.1.78), followed by the control module posting a data-out message, as well as that inputs status being available for query through ParameterInfoRequest message. Similarly, if the control module implements a non-Ethernet, physical output (e.g., electrical signals) or implements non-Ethernet, direct control of a component (e.g., a gate-arm for traffic control, an indication light, a physical indication to operators), then a change in such an output must be reported by the control module in a data-out message, with status available for query through ParameterInfoRequest message.

For example, any interlocks monitored by a device (e.g., turn off high voltage if enclosure is opened, or shut down if water is present, etc.) or monitored conditions (e.g., ambient or circuit temperatures, input voltages, high voltage, input current usage, etc.), must be reported to the control module by the device through the parameter mechanism. See Section 2.1.1 for more information. If a condition may affect the health of the device, or make it non-operable, it should be reported as a parameter.

[1] At the time of writing this there are a number of IEEE-1588 compliant Ethernet transceivers available in the single unit $10USD price range, embedded single chip systems with hardware IEEE-1588 Ethernet support and enough computational resources to potentially base RAPTER devices on for around $20USD, some single board computers in the $100USD range which contain powerful CPUs, Field Programmable Gate Arrays (FPGAs) and other advanced busses and capabilities.

Section 2

Portal and Device Operations

Each individual device within a portal is controlled by the control module using digital messaging. The control module exercises its oversight of settings and configuration by digital messaging using messages that are part of the parameter mechanism (for simple settings such as individual values, see Section 2.1.1) and the upgrade mechanism (for more complex settings such as detector response functions, see Section 2.1.1.2). The control module also oversees operation, fault handling, modes, and status of all connected devices, as well as all behaviors of the portal.

Relay command messages (part of command message group) allow command devices to request the control module to change or retrieve parameters of, or alter the behavior of, other devices. The logic by which the control module ascertains that such a request can or should be fulfilled, and how it is fulfilled, are not specified.

2.1 Portal Configuration

The initial setup of a portal or a device includes the process of establishing the settings to be used during operation. The value of any setting that can impact operations must be retrievable by the control module through the parameter mechanism (see Section 2.1.1), and any setting value that is not fixed must be settable through the parameter mechanism. Settings may be retrieved and reported, for example, in support of configuration management or the conduct of an inventory.

The firmware upgrade mechanism (in the core message group, Section 2.1.1.2) may be used to enter or update more extensive sets of values, such as a detector response function (Section 1.3.4). However, software and firmware version numbers (see the DeviceInfoReply message) shall be updated whenever a firmware update takes place.

A few settings require user entry, such as those that express a relationship between one device and another, for example the location of a detector element within the portal (see the CmdDeviceSetReferenceInfoRequest message). User entry may be accomplished through a human-machine-interface command device (not specified herein) that is able to communicate using the command message group with the control module.

The control module must store its own settings, as well as (sub-)device locations, names (as the devices will be referred to in N42.42 files), and other information in the CmdDeviceSetReferenceInfoRequest in non-volatile memory for retrieval and use when needed. Devices are also

responsible for persisting their own parameter settings in non-volatile memory across power cycles.

2.1.1 Parameter Mechanism – Settings and Status

Parameters are managed using a set of messages - the parameter mechanism - in the core message group, each of which contains "`Parameter`" in the message name.

Under the RAPTER Interface Specification, parameters are used to convey a wide array of operating information. Parameters can be used to record or retrieve settings, such as target high voltages, algorithm-specific thresholds, or enabling or disabling specific features. The range of measured[1] values that are considered healthy may be allowed to be set (at the option of the device), as well as the change necessary in measured quantities before the device should report to the control module the new value. They can be used to report quantities (other than radiation detector data or vehicle presence data) that are useful to monitor status and health, such as temperature, measured high voltages, power draw, CPU utilization of the past minute, free disk space or ram, or other device-specific quantities.

As specified by ParameterValueDataType, parameters may be boolean, signed 32-bit integers, 32-bit floating point numbers, or Unicode Transformation Format 8-bit (UTF-8) encoded string values, a list of floating point values, or a list of integer values, and may consist of a set value, a measured value, the lower and upper ranges of health (if a measured parameter), how much of a change is required before reporting a new value (measured parameters), the lower and upper values that it can be set to (if the parameter is settable), and an indication of the impact the parameter is having on the health of device operations. See ParameterInfo for complete details of information a parameter may supply.

Devices may be supplied with fixed settings that are determined and entered by the manufacturer, which can be used to communicate information such as date of manufacture, scintillator or He^3-tube volumes, etc. A parameter that provides a UTF-8 value could for instance be used to indicate errors/issues not indicated elsewhere, for example: storage medium (e.g., hard drive) failure, trouble writing to an internal bus (e.g., I2C or SPI busses) or component, or other potential errors that are not expected to occur frequently enough to justify creating a dedicated parameter. The values of settable parameters can be changed by the control module by sending a SetParameterRequest message to the device.

A list of parameter names can be requested from a device with the ParameterNamesRequest message, and then full information about each named parameter can be requested using ParameterInfoRequest messages. The full information about a parameter is specified in the ParameterInfoReply message sent by the device. When a device updates a measured parameter, the update is sent from the device using a ParameterUpdatePush message. See Appendix B for example C source code for encoding and decoding parameter messages. Devices implementing the data-out message group can request parameters of a given device using the DataOutDeviceParametersRequestValue message.

A summary of the ParameterPropertiesFlags that a device can use to define a parameter is:

ValueFixed Defines the "`value`" field of the parameter as not be alterable through SetParameterRequest requests. This can be useful to compare measured value against, for example

[1] measured is used to indicate both physically measured quantities, such as temperature and voltage, as well as monitored conditions such as hard drive space, data bus congestion, or a resources initialization status.

supply voltages. Can not be combined with the `Settable` option.

Settable The "`value`" field can be altered using SetParameterRequest requests. Note that the device may reject specified values for any number of reasons, including them being out of range, cant be set while operating, or being incompatible with other parameter values. Cannot be combined with the `ValueFixed` option.

Measured The value is measured, in some way, by the device. For example, CPU temperature, high voltage current, free ram, status of intializing harware resources, operating system errors, data bus congestion, etc.

HealthyValueRanged There is a range defined in order for the measured value to be considered healthy. The `Measured` option must also be set if this option set.

SetValueRanged There is a range defined that values may be set within. Not applicable to string or boolean valued parameters. The `Settable` option must also be set if this option set.

ReportOnChange Indicates the device will send updates when the `Measured` value for this parameter changes, using ParameterUpdatePush messages. The `Measured` option must also be set if this option is set.

ReportOnChangeDeltaSettable Indicates the change, or delta, in the measured value that is required before a parameter update should be sent, can be set by the user. Not applicable to string valued parameters. For boolean parameters setting the delta to false indicates the device should report on change, while true indicates do not report[2]. The `ReportOnChange` option must also be set if this option is set.

HealthyLimitsSettable Indicates the healthy range of measured values may be set through SetParameterRequest messages. The `Measured` and `ReportOnChange` options must also be set if this option set. Not applicable to string valued parameters.

EffectsHealthWithoutLimits Indicates that even though healthy limits for the measured parameter are not defined, the measured value of this parameter may effect device health. The `Measured` option must also be set if this option set, and the `HealthyValueRanged` option must not be set.

NotSettableWhileOperating Indicates that this parameters value can not be set while the device is in active operation using SetParameterRequest messages. An example use case would be if the high voltage could not be dynamically set, but required turning the voltage off and then back on to the new value. The `Settable` option must also be set if this option set.

SettableWithPredefinedValuesOnly Indicates that only certain pre-defined values will be accepted when setting the parameter value. The `Settable` option must also be set if this option set, and the parameter description should indicate valid values.

[2]The logic for the boolean case falls out from boolean values typically being represented as integers internally in computers, with '0' as false, and non-zero values as true

2.1.1.1 Device Position Settings

The initial setup of a radiation detector data or a vehicle presence module includes entering the positions (and names) of detection elements (sub-devices). The positions of portal detection elements may influence the analysis or use of portal data. Position information can be used, for example, by the analysis module to determine how far apart vehicle presence break-beam sensors are, in order to use break-beam timing information to approximate the vehicles speed. The analysis module may use the relative locations of Radiation Sensor Panel (RSP) or their sub-devices within the portal to locate a radiological source within a conveyance.

Positions shall be specified using a standard right-hand Cartesian coordinate system with the z-axis parallel to the direction of vehicle travel, with the origin on the center line of the portal lane at ground level, at the midpoint of the detection zone along the z axis. The positive z direction is fixed and is normally chosen as the preferred direction of vehicle traffic through the portal; such that the vehicle enters the portal from the negative z and exits at positive z. Once fixed, the z direction does not change with changes in direction of vehicle traffic. The negative x axis points toward the vehicle driver's side of the lane (i.e., the left hand side when driving in the positive z direction), and the positive y axis gives the distance above ground level. Orientations and device geometry are not included in this position information. Such a simple geometry system was chosen to allow easy setup and use.

The location of a sub-device may be specified with respect to a manufacturer-provided physical reference mark on the item. In the absence of a pre-defined reference mark, the detection element's 3D center point shall be determined or estimated (e.g., the center of a He3 tube, or for IR recievers or cameras the location of the photo-sensitive silicon device), while for other devices the 3D center of the device is used.

The orientation of the device is not recorded.

Positions can be specified to the control module using CmdDeviceSetReferenceInfoRequest messages. The control module is responsible for persisting the specified position data. Data-out devices, including the analysis module, can query the position of devices and sub-devices using the DataOutDevicePositionRequestMsg.

2.1.1.2 Update Mechanism

Software, firmware, and extensive parameters may be updated by using the update mechanism of the core message group, which is exercised by the FirmwareUpgradeRequest and FirmwareUpgradeReply messages for upgradeable devices. Upgrades can be initiated by a relay command message (see Section 2.2).

Upgrades may be used to change quantities which otherwise can not be changed throught the RAPTER Interface Specification, such as device UUID (see the DeviceInfoReply message) or the cumulative operating time (see the DeviceUsageStatisticsReply message). An upgrade may necessitate the restart of the upgraded device, including disconnection (see DeviceDisconnectReason enumerated value DueToFirmwareUpgradeValue).

2.1.2 Device Operating State

This section describes the operating states of the portal and individual devices. Operating states include operating modes, data modes, and measurement characteristics.

Each device in the system (i.e., not the control module) may receive the ChangeDeviceStateRequest by which the control module informs it of how to operate. In addition to requested transitions, transitions may be necessary due to hardware errors or other operating conditions; devices indicate this to the control module using DeviceStatusPush messages.

ChangeDeviceStateRequest messages include a field requested_transition_time that specifies when the device should transition (as microseconds after the UNIX epoch) to the desired state. If this value is zero, or anytime previous to when a device receives the message, the device should treat the transition request as being wanted immediately. If a future time is specified, the device should make an attempt to finish the transition at that time. At most, one transition can be queued up (e.g., if there is a transition scheduled in the future already, sending out another transition request cancels the first one). See requested_transition_time for more details.

The ChangeDeviceStateRequest message contains the following additional pieces of information: device operating mode, device data collection mode (measurement) mode, and measurement characteristics.

2.1.2.1 Device Operating Mode

There are three device operating modes. All devices must support all three. It is up to the control module to determine which operating mode to ask a device to be in. The device operating modes are:

- **StandBy**: The device will respond to RAPTER WebSocket communications, but resource intensive sub-systems of the device, potentially hazardous, or other optional sub-systems should be powered down.
- **Ready**: The device is not currently performing its intended functionality like taking data, performing analysis, or operating sub-components such as gate-arms. However, all the resources that are necessary to perform the intended device functionality are ready to be utilized with minimal notice. For radiation detector modules, this might mean that high voltages have been stabilized, selfchecks have been performed, acquisition hardware has been initialized, and so on. For a data-out device that records data and other information, this might mean that a connection to the database has been established, and blocking tasks like operating system upgrades have been finished.
- **Operating**: The device is performing its intended function, whether it is collecting radiation data, measuring vehicle location, performing analysis, recording information to a database, or some other function.

2.1.2.2 Data Collection Mode

The DataCollectionModes enumeration provides the possible values of how radiation data is aggregated and sent to the control module when the device is operating. The operating_mode

field of the ChangeDeviceStateRequest message specifies if the detector should send a spectrum at regular time intervals, once after a specified dwell duration, or in listmode as data is detected. For other devices in the system, or for OperatingMode values other than Operating, this field must have values specified below. Devices must specify which DataCollectionModes they support in the data_collection_modes field of the DeviceInfoReply message.

- **NoOriginate**: For radiation detector modules, vehicle presence modules, and power management modules:
  - This is the value that must be specified for any OperatingMode mode other than Operating.
  - This value must not be specified when OperatingMode is equal to Operating.

  For all other devices:
  - This is the value that must be specified. In particular, for the analysis module, data-out devices, and command devices, the value of DataCollectionModes must always be NoOriginate.
- **ClockTimeInterval**: The device collects information and sends it to the control module at regular intervals of wall clock time. That is, if a sensor collects data at 0.1 second intervals, after starting data collection (at either the time specified in the requested_transition_time field, or if it has a value before the current time, as soon as possible) it will collect information until the next wall time that is a multiple of 0.1 second. Immediately upon concluding one interval, it begins a new collection interval of the same duration, and so on (i.e., at x.1 seconds, then at x.2 seconds, x.3 seconds, etc.). Except for possibly the first interval of data collection, each time interval starts at a time such that the modulo of the start wall time with the interval length is zero. To continue on with the 0.1 second interval example: a possible sequence of collection start time is 03:24:02.34 (first interval), 03:24:02.40, 03:24:02.50, 03:24:02.60, etc. The sequence 03:24:02.34, 03:24:02.44, 03:24:02.54, etc., would not be correct. Requiring collection intervals (other than the first) to start at wall time multiples of the interval duration introduces the limitation that intervals must evenly divide a second, but then allows straightforward synchronization of multiple devices. The maximum time interval is 1000 ms.

  Radiation detector modules which detect gammas must support this mode, and radiation detector modules which detect only neutrons may optionally support this mode. Vehicle presence modules must not support this mode.

  In this mode, data collection continues until the device is instructed otherwise.
- **OnEvent**: The device sends information as soon as an event is detected. For gamma and neutron detectors this is list mode data acquisition, and for vehicle presence modules this is whenever a beam state changes, a distance measurement becomes available, or picture is triggered to be taken. For power management modules this is pushing information to the control module whenever an event happens (although note that the power management modules will likely always be providing electrical protection and battery backup, even when not in the RAPTER operating state). If an interval is specified (the collection_interval_ms field), the device must also send a HeartbeatPush message to the control module at the wall times that are multiples of that interval; i.e. at the same times that define the start of collection intervals as defined in the ClockTimeInterval bullet.

Vehicle presence modules and power management modules must support this mode. Radiation detector modules which detect neutrons must support this mode, and radiation detector modules that only detect gamma radiation may optionally support this mode.

In this mode data collection continues until the device is instructed otherwise.

- **RealTimeDwell**: The device acquires and aggregates data for a single specified time interval, sending it to the control module at the end of collection. Once done with collection the device returns to the Ready state unless instructed to do otherwise.

  Radiation detector modules can optionally support this data collection mode. Vehicle presence modules must not support this data collection mode since it is unclear how the results would be interpreted.

  In this mode data collection continues for the specified collection interval.

- **LiveTimeDwell**: Similar to RealTimeDwell, but instead of the time period corresponding to wall-clock time, the time interval specifies the time the detector is available to take radiation data. That is, for many gamma and neutron detectors after a radiation quantum is detected there is a small amount of time in which the detector is not available for processing another detection event either because of physical or electrical limitations. This is often referred to as "dead time" and does not count towards the detection interval. In contrast, the time during which the system is available to detect incoming radiation is referred to as "live time."

  Radiation detector modules can optionally support this data collection mode. Vehicle presence modules must not support this collection mode since it is unclear how the results would be interpreted.

  In this mode data collection continues until the specified amount of live time has accumulated.

2.1.2.3 Measurement Collection Interval

- For ClockTimeInterval collection mode, this is the interval of time, in milliseconds, at which the detector will send radiation data to the control module. The sending of data at this interval continues until the collection mode changes. The value specified must in the list: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000.
- For OnEvent collection mode, if the value specified is non-zero, this is the interval of time, in milliseconds, at whichHeartBeatPushs will be sent to the control module. The sending of these messages continue until the collection mode changes. The value specified must either be 0 (zero), in which case HeartBeatPush messages will not be sent, or in the list: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000.
- For RealTimeDwell, and LiveTimeDwell collection modes this is the period of time, in milliseconds, to accumulate radiation data for before sending to the control module.
- For the NoOriginate collection mode: when the operating mode is Operating, this value has the same meaning and consequences as for the OnEvent collection mode, and for other operating modes must be zero.

Devices can be queried for their supported collection intervals using the SupportedDataCollectionIntervalsRequest message.

2.1.2.4 Measurement Type

The MeasurementType enumeration provides possible values to indicate what is being measured by the portal. This information can be conveyed both at the beginning of a measurement by the ChangeDeviceStateRequest message, or at any subsequent time with the MeasurementTypeChangeRequest message. Any of the MeasurementType values can be specified to any of the devices. The possible values are:

- **NotSpecified**: When sent to an device it must be interpreted as *what* is being measured is unknown to the control module.
- **Item**: Until further specified, an item of interest is being measured.
- **Background**: There is no item of interest present and conditions are believed to be steady-state background.
- **PossibleInterferingSourceValue**: There may not be an item of interest being measured, but there is perhaps some activity happening that could contaminate the radiation or vehicle presence data. Examples include: there is an alarming vehicle in a neighboring RPM; there is calibration being performed nearby; there is work being done in the lane that may erroneously trigger the vehicle presence sensors. One potential use case for this value is to ensure that devices that take advantage of background aggregation do not accidentally contaminate their sample.
- **ActiveMaintenance**: Indicates to the device that it can spend time performing maintenance, which may impact other systems of the portal. One example of a usage might be if a radiation detector module requires a period of active energy calibration with a source not normally present. Data taken during this interval should still be sent to the control module if possible, but it should not be interpreted as being useful for analysis, adjudications, or other tasks.

  Indicates to the device that it can spend time performing maintenance, which may impact other devices of the portal (but only if the DeviceUsesActiveMaintenanceFlag flag is set in the device_features field of the DeviceInfoReply message). One example of a usage might be if a radiation detector module requires a period of active energy calibration with a source not normally present. Data taken during this interval should still be sent to the control module if possible, but it should not be interpreted as being useful for analysis, adjudications, or other tasks. When done performing active maintenance, the device should transition back to a OperatingMode state of Ready (and notify the control module via a DeviceStatusPush message).

2.1.3 System Operating State

The operating state of the portal system can either be automatically determined by the control module or be changed using a CmdSystemStateChangeRequest message from a command device. This message is defined similar to the ChangeDeviceStateRequest message of Section 2.1.2. The CmdSystemStateChangeRequest message has a single OperatingMode field, a single data collection time period field, and optionally a transition time that are applied to all devices in the portal. However, three separate DataCollectionModes are specified. One applies

to radiation detector modules that detect gamma radiation, including modules that detect both gamma and neutron radiation. One applies to all neutron-only radiation detector modules. And the third DataCollectionModes applies to all vehicle presence modules. The control module will always determine the operating state and data collection mode for the analysis module, power management modules, data-out devices, and command devices. When a operating mode of Operating is specified, but a data collection mode of NoOriginate for a class of devices is specified, then those devices will be placed in the Ready mode by this request instead of Operating. See CmdSystemStateChangeRequest message for further details.

It is the responsibility of the control module to get individual devices in the system into the desired states, and there is no guarantee that the control module will request the respective devices to go into a requested state. Reasons for leaving this up to the control module implementation include: hardware issues, incompatible DataCollectionModes (e.g., a particular gamma detector does not support the requested list mode acquisition), active maintenance being needed or in progress, or problem devices. Should the control module reject a request, it must attach a Notification to the reply message (see HasNotificationAttachedFalg of Section 3.3.2.1) providing a reason.

The CmdSystemStateChangeRequest message is intended to cover typical portal operation scenarios, however, finer grained control of devices within the system can be achieved using using "relay commands" (discussed in Section 2.2) to issue ChangeDeviceStateRequest messages from command devices to individual devices within the portal. The system's operations state may also be decided by logic implemented by the control module. For example, upon being turned on, the control module may automatically transition to a pre-defined operating mode that accommodates the portal's typical duties of detection and analysis.

2.1.3.1 System Measurement Type

While devices are operating, it can be important for them to know what is currently being measured. For example, gamma detectors may accumulate background from which to perform energy calibration. The analysis module may need to know whether the received radiation data is from background or from an item of interest (e.g., a vehicle) for which an analysis result will be requested. In many RPMs the vehicle presence subsystem, such as break-beam style sensors, determines if background or an item of interest is being measured. In a RAPTER portal, the vehicle presence system plays a similar role in terms of detecting a vehicle's presence, however it is ultimately up to the control module to determine what is being measured, and then inform the other devices of this information.

However, the CmdMeasurementTypeChangeRequest message can be used to override the current measurement type. When a value other than NotSpecified is given, the control module will change to that measurement type until another CmdMeasurementTypeChangeRequest message, or a CmdSystemStateChangeRequest message is received. A value of NotSpecified indicates to the control module to go back to determining the measurement type itself, likely through use of the vehicle presence module data. This functionality can be useful, for example, if it is wanted to manually trigger occupancies or backgrounds.

2.2 Relay Commands

Although command devices can only communicate with the control module, and not other devices in the system, it is still useful for the command devices to be able to issue requests to other individual devices. Some examples include changing parameter values, sending firmware upgrades, or changing an individual device's state for debugging. To facilitate this a CmdDeviceRelayRequest message is used. The command device sends a CmdDeviceRelayRequest message to the control module; the message specifies the UUID of the target device, and contains the binary encoded message intended for the target device. The control module is free to decide if it will pass the message on to the target device, and if it does, it will take care of translating the message version to the version used by the device (which may be different than that used by the originatin command device) before sending the message to the target device. Any replies from the target device will be relayed through the control module back to the command device, again translating as necessary. Command devices may only do this for "request" type messages as it would make no sense for "push," "reply," and "acknowledge" messages.

In addition to enabling communications with general devices within the portal, the CmdDeviceRelayRequest message is also how parameter values can be set or log files requested from the control module itself, which is done by specifying the UUID of the control module.

Section 3

RAPTER Messaging

RAPTER communications are defined in terms of binary messages with rigid structure. Communications other than with valid messages, including every field having a valid value, may be grounds for the control module to terminate connections to a device, or for the device to disconnect from the control module. This section provides a description of general message features and sequences, while the structure of individual message types is given in Section 4.

3.1 Message Groups

All devices must be able to recognize and respond to all messages that the control module might send to them. To reduce the complexity of implementing specific devices, RAPTER messages are divided into seven groups, which limits the set of messages that a given type of device must be able to process. The control module uses all message groups. A general introduction to message groups was presented in Section 1.4.

Group	Description	Device Type
Core	Messages in this group cover common functionality and must be understood by all devices.	All
Radiation Detector	Covers functionality and information specific to radiation detector modules.	Radiation detector modules
Vehicle Presence	Covers functionality and information specific to detecting a vehicle's presence, such as reporting or requesting data from break-beam style sensors, vehicle position sensors, or digital cameras.	Vehicle presence modules
Power Management	Covers functionality and information specific to power management modules which provide UPS capabilities to the portal	Power management modules
Data Out	Covers functionality for retrieving information about and from the connected devices, as well as how data which is packaged together and output from the control module.	Data-out devices, Command devices, Analysis module
Analysis	Messages to allow the control module to request and receive analysis results.	Analysis module
Command	Messages to allow changing the operating status of the system, or to interact with specific devices through relay messages.	Command devices

In addition to all devices implementing the core message group, radiation detector modules must implement the radiation detector message group, the vehicle presence modules must implement the vehicle presence message group, and power management modules must implement power management message group. Data-out devices, the analysis module, and command devices must all implement the data-out message group; the analysis module must also implement the analysis message group; and command devices must implement the command message group.

The data-out message group primarily contains information from all the devices packaged together in a coherent manner. Some information such as connected device summaries, or which devices are connected, can be requested by data-out devices (and consequently by the analysis module and command devices). But other information that may be generated dynamically and delivered to the control module, such as spectra, vehicle presence data, or parameter changes, are "pushed" from the control module to data-out devices, to the analysis module, and to command devices. The analysis message group enables the control module to request interim or final analysis results for an occupancy. The command message group messages enable, for example, changing the portal's mode, overriding the portal's occupancy status, or sending commands to individual devices.

This categorization of messages not only maps to the functionality of devices comprising the system, it allows future changes to one message group without necessarily affecting another (see Section 3.2), and it also allows a device to selectively decode or understand only those messages relevant to that device (see Appendix A).

The DataOutDataPacketPush message, which the data-out devices, command devices, and the analysis module receive from the control module, is more or less a container message for information from core, radiation, power management, and vehicle presence message groups. A result of this is that if you want to extract, for example, radiation data from this message, then

you will need to implement decoding at least a sub-set of radiation detector message group messages. To keep receivers of this message from having to decode all message groups, the DataOutDataPacketPush message provides the device_frame_uuids and device_frame_offsets fields which can be used to locate information from specific devices within the message, allowing you to ignore device types you are not interested in.

The format for encoding messages is given in Section 3.3, with an example C language encoding of messages given in Appendix A. The control module must implement encoding and decoding all message groups, but individual devices may only have to implement a subset of the message groups.

The messages that carry information about device settings or state (e.g., high voltage, settings, temperatures, etc.), namely ParameterInfoReply, SetParameterRequest, and ParameterUpdatePush messages obey the same encoding rules as other messages, but the values for the parameters are represented as strings within the message, instead of as binary like all the other messages. The reason for this is that parameters can be booleans, integer, floating point numbers, strings, or lists of integers or floats, so to keep encoding consistent, no matter the value type of the parameter, it is always represented as a string within the message. See the ParameterValueDataType enumeration and ParameterInfoReply message for details, and example of encoding and decoding them in the C language in Appendix B.

3.2 Message Group Versioning

Devices in a single portal are not all required to use or support the same versions of messages, and the control module may simultaneously support multiple versions of a message group. A single version number is used by a given device for each message group (core, radiation detector, vehicle presence, power management, data out, and command). A handshake mechanism is employed to allow negotiating message group versions. When the control module relays data or other information from one device to another device (e.g., sends the radiation detector data to the analysis module), if the version for that message group was negotiated to a different value for each device, the control module is responsible for translating the message format when transferring it between devices. The motivation for versioning each group of messages separately is to allow changes or improvements to be made to one message group, or to a single device type's capabilities, while the other types of devices, except the control module, can remain oblivious to the changes[1]. Future changes to the RAPTER Interface Specification will either have to make changes in a compatible manner (e.g., if a new field is added to a message, a possible value for it might be "not available") or there may be inherent incompatibilities between devices (e.g., if a newer version of an analysis module requires information not available from an older radiation detector module, then the devices may not be able to be used together, or may be usable with reduced capability). Devices in or connected to a single portal are not all required to use or support the same versions of messages, as the control module may simultaneously support multiple versions of a message group. See the handshake mechanism section (Section 3.5) for more information about message version negotiation during the handshake.

Currently all message group version numbers are zero.

[1] There may be a loose coupling between the versioning numbers in that for message A, version X may require core message group version of at least Y because any version of the core message group Y may not support certain functionality required for message A, version X.

3.3 Message Contents

RAPTER messages are transmitted in a binary format that is intended to allow encoding or decoding messages with a minimal amount of overhead on resource constrained devices. With some care about memory alignment, most messages, except notably DataOutDataPacket messages, can be easily mapped on common computational platforms (but not all architectures, operating systems, or compilers) to C programming language struct or primitive.

Each RAPTER message, other than a DataOutDataPacket message, is built up from an ordered combination of primitive data types. The meaning and relative location (i.e., number of bytes offset from the beginning of the message) of each field in a message is given in Section 4 for all messages, but the general rules for interpreting a field's value is given in the following section.

3.3.1 Primitive Data Types Used

RAPTER messages are built up from a series of primitive data types, whose positions (i.e., offsets from message beginning) within the message are uniquely determined. The primitive types used are:

- All integer values within messages are communicated in twos-complement (if signed) little-endian format (i.e., native x86, and most, but not all, operating systems on ARM-based computers).
- The type and size of integers is specified using C/C++ sized types. That is `uint8_t` is a one-byte unsigned integer, `int8_t` is a signed one-byte integer, `uint16_t` is an unsigned two-byte integer, `int16_t` is a signed two-byte integer, and similarly for 32 and 64 bit variants.
- For fields that can take on a discrete number of specified values with specific meanings associated with the values, integer enumerations are used to represent this information. The integer enumerations representation is defined (e.x., `uint8_t`, `uint16_t`, etc), and the meaning associated with each valid integer value is specified by this documentation. An example is MessageGroup which is the first byte of all RAPTER messages, and determines the message group (0→Core, 1→Command, 2→Data Out, 4→Analysis, 8→Radiation Detector, or 16→Vehicle Presence) of the message. These fields may not take on any value not defined by the enumeration.
- Messages may also have bitfields represented by an unsigned integer (`uint8_t`, `uint16_t`, `uint32_t`, or `uint64_t`), whose individual bits are defined in an enumeration (see previous bullet) where each enumerated value has exactly one bit set, and a corresponding meaning defined if that bit is set in the bitfield. An example is the device_features field of the DeviceInfoReply message, whose bits are defined by the DeviceFeaturesFlags enumeration. The bitfields may have zero bits set, one bit set, or multiple bits set; multiple bits indicate multiple values defined by the enumeration are manifest. For many of the defined bitfields any combination of the defined bits may be set so as to appropriately indicate the relevant information, but some enumerations specify constraints on which enumeration values may or may not be specified together. A bitfield with no bits set indicates none of the defined options are set. All bits in a bitfield not defined by the enumeration, must be zero.

- All floating point values are transmitted in little-endian IEEE 754-2008 (see reference [18]) format; with the additional requirement that the representation is required to be normal (i.e., represented without leading zeros in its significand and not denormal), and values of Infs or Not a Numbers (NaNs) are not allowed.

- Strings must be encoded as UTF-8. Strings are encoded by first specifying the string byte length as a `uint32_t`, which is then followed by the number of bytes given by the string length, and then between 1 and 4 zero-valued bytes trailing so that the total byte count is a multiple of four. Note that an empty string will be 8-bytes long (all zeros). Strings are designated by the types: `ShortString`, `MediumString`, and `LargeString`. These string types have a maximum of 255 (short), 65,535 (medium), or 4,294,967,295 (large) bytes of character content. Note that the length specifies the number of bytes in the string, not the number of characters.

- `Struct`s are a named grouping of fields (integers, floats, enumerations, strings, arrays, and other structs) used to organize mixed types of information. For each struct defined, the data type and name of each field is given, with their position within the struct given by the same rules as in Section 3.3.3.

- Arrays provide the ability to encode a variable-number of elements that have a homogeneous type. The array is encoded as the element count n (an unsigned 32-bit integer) followed by the encoding of each of the array's elements, starting with the first element, and progressing to the last. Arrays are designated as `ShortArray`, `MediumArray`, and `LargeArray`, that can hold a maximum of 255 (short), 65,535 (medium), or 4,294,967,295 (large) elements. Arrays can hold primitive types such as `integers`, `floats`, or `enumerations`, as well as `strings`, and `structs`. An example of an array is the parameter names contained in a ParameterNamesReply message.

- `Uuid128` values are 128bit fields representing a UUID of the device. See the `device_uuid` field of `DeviceInfoReply` for more information.

3.3.2 Message Header Format

The initial eight bytes of all RAPTER WebSockets messages follow a standardized format. The first byte of every message defines which message group the message belongs to, as defined by MessageGroup (e.g., core, radiation detector, analysis, vehicle presence, data out, or command). The second byte determines the type of message within that message group. The combination of the first and second bytes uniquely determines how to interpret the message. (For detailed descriptions of specific message types, see CoreMsgType, AnalysisMsgType, CommandMsgType, DataOutMsgType, VehiclePresenceMsgType, PowerManagementMsgType, and RadDetectorMsgType.) Each message type within a message group will have a unique value within that message group, but could have the same value as a message in another message group.

The third byte is an unsigned 8-bit integer giving the version of the message group; this value must be equal to the value negotiated during the handshake. The fourth byte indicates modifiers to the message, as specified by the MsgFlags enumeration, and further explained in Section 3.3.2.1.

The next four bytes are a message ID that facilitates the asynchronous nature of network communications. All “replies” or “acknowledges” to a message will contain the same ID value as the original message allowing the determination of what the reply or acknowledge is in reference to. The message ID should not be interpreted by the device receiving a “request” or “push” message as having a meaning, because the selection of ID values is implementation dependent. However, the ID should be unique relative to previous or future “request” and “push” messages from the originating device to the control module, or vice versa, with a repetition cycle (number of messages between re-use of an ID) of at least 2^{28} = 268,435,456 messages[2]. Note that the ID’s uniqueness is direction dependent, meaning a device only has to ensure uniqueness of IDs it originates, and does not have to consider values of IDs it has received when generating IDs[3].

All bytes after the ID are the payload of the message; length and format is message type dependent (as determined by the first two bytes of the message). An overview of the message header can be seen in Figure 3.1.

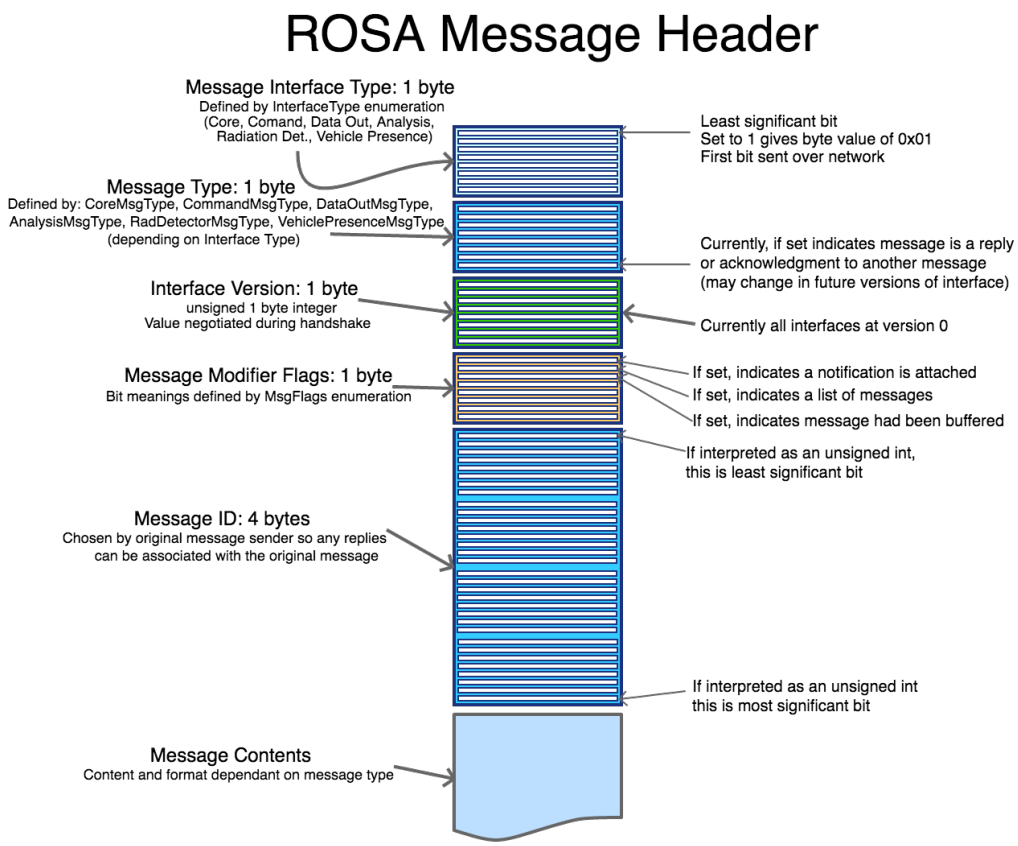

Figure 3.1: The leading 8 bytes of every RAPTER message.

[2]This provides for approximately three hours of unique IDs assuming list mode data and a count rate of 20kcps.

[3]The TCP/IP mechanism indicates the source of incoming messages, meaning a control module may have to combine the TCP/IP source information with the message ID, and message type (“request”/ “push” vs “reply”/“acknowledge”) to resolve message context.

3.3.2.1 Message Modifier Flags

The fourth byte of every RAPTER message indicates modifiers to how the message should be interpreted, depending on which bits are set (e.g., is a bitfield); the meaning of each bit is defined by the MsgFlags enumeration, and options are described bellow.

MessageHasBeenBuffered (value 0x04) Indicates this message has been buffered, and is not being sent at the time of the originating event. The generating event for this message may have happened while the network connection was not active. Or, if buffer recovery was serial, this message may have been generated while the sending of previously buffered messages was still occurring, so it could not be sent in real time.

This bit being set does not alter the byte-format of how this message should be decoded.

3.3.3 Message Body Format

Directly after the eight-byte header, the message body begins. The binary format of messages is inspired from the memory layout and required memory alignment used in a number of common computing architectures and platforms, as well as widely used data transfer standards such as External Data Representation Standard (XDR) [19], Structured Data Exchange Format (SDXF) [20], and other commonly used protocols for network transfer. More verbose, or human readable formats like XML, JavaScript Object Notation (JSON), or YAML Ain't Markup Language (YAML) were found to add an unacceptable overhead for larger portal installations, as well as exclude micro-controller based RAPTER devices.

The encoding and decoding of all messages can be performed using the following set of rules:

- Fields all occur in the same order as given in this document (see tables defining messages in Section 4), however there may be padding bytes needed between fields because each data type has a specified alignment of where it may be placed; fields must be placed an even multiple of its data types alignment from the beginning of the message. For example, a 32-bit integer has an alignment of four, meaning it can only be placed at locations within the message that are a multiple of four bytes from the beginning. Sequential fields in a message are placed as close as possible together, such that this alignment is obeyed (i.e., the fewest number of required padding bytes are used). Padding bytes must have a value of zero.
- Integers, floats, bitfields, and enumerations, all have an alignment equal to the byte-width of their data type. That is a `uint8_t` or char have alignments of one, and can be placed immediately after the previous field, but a `uint16_t` has an alignment of two, so must be placed at a multiple of two bytes from the beginning of the message; `float`s and 32-bit integers have alignments of four, and `int64_t`s at alignments of eight.
  - All integer values communicated in twos-complement (if signed) little-endian format (i.e., native x86, and most, but not all, ARM-based platforms).
  - All floating point values are little-endian IEEE 754-2008 [18] format, with the additional requirement that the representation is required to be normal (i.e., represented without leading zeros in its significand. i.e., not denormal), and values of Inf or NaN are not allowed.

  - Enumerations are represented as the type of integer specified in the enumeration definition in this document.
  - Time stamps are represented by `int64_t`s, that give the number of microseconds since the UNIX epoch (00:00:00 Jan 1, 1970), not counting leap seconds, and are in Coordinated Universal Time (UTC). Their alignment is the same as `int64_t` (i.e., eight).

- UUIDs have an alignment of eight; i.e., must start at a byte location that is a multiple of eight from the message beginning.

- `String`s have an alignment of four byte length. String contents start immediately after the length specifier. After the contents, zero-valued bytes (bytes which if interpreted as a `uint8_t`, values would be zero) are appended so that there is at least one zero-valued byte and until a multiple of four bytes from the message beginning (i.e., between one and four padding bytes).

  - The `string` contents byte length must not be longer than 255, or 65,535, or 4,294,967,295 for `ShortString`, `MediumString`, and `LargeString` respectively (padding bytes are not included in string content length).
    * A `string` with no contents requires eight bytes to encode: four bytes for the length field, and four zero-valued bytes.
    * The total `string` field length will always be a multiple of four bytes.
    * Every `string`s content is terminated with between one to four zero valued bytes.
  - All strings are UTF-8 encoded, and some fields impose additional restrictions such as only American Standard Code for Information Interchange (ASCII) or alphanumeric characters.
  - Note that even though `string` contents will always be terminated with at least one zero-byte, message decoders should always verify this assumption on incoming messages.

- `Array`s must start at a multiple of four bytes from the beginning of the message (e.g., have an alignment of four), where a `uint32_t` is used to specify the number of *elements* in the array, potentially followed by zero or four bytes of padding, then the `array` elements, and then from zero to three bytes of zero-padding to ensure the overall length of the array is a multiple of four. How the elements are written is dependent on the data type of the `array`, with there being three applicable categories:

  - **Integers, floats, bitfields, enumerations, UUIDs**: The first element of the array is located starting at a multiple of the data types alignment (i.e., `int64_t`, `uint64_t`, and `UUID`s may need four bytes of padding after the array length field, but other types will not need padding). The remaining elements are then located immediately adjacent to each other.
  - **Strings**: The first `string` in the `array` is located starting immediately after the `array` element length specification. Each remaining `string` is located immediately after the previous one (strings are a multiple of four bytes, so no additional padding is needed). Each `string` element is represented as described in the previous major bullet.

    Note that:

    - An empty string will take up eight bytes of space.
    - To locate the n^{th} string, you will need to iterate over the $n-1$ preceding strings, using their specified lengths to determine the starting location of the next string.
- **Struct**s: Each array element must start at a multiple of eight bytes from the beginning of the message, and if necessary, have padding added after the structs fields so that each element is a multiple of eight-bytes long. This also means that there are either zero or four bytes of zero-padding between the array length, and the first element. Some examples are:
    - A message containing 36 bytes of content before the start of an array of structs which each take up 10 bytes of space: The first 36 bytes of content would be written, followed by the array length written at an offset of 36, and the first array element at an offset of 40, then six bytes of zero-padding after the structs contents, and then the second element written, and so on for each element, including the padding after the final element.
    - A message contains 48 bytes of content before the start of an array of structs: After the first 48 bytes of content the array offset would be written at an offset of 48, four bytes of zero-padding would be written (at an offset of 52), and then the first struct would be written at an offset of 56 bytes, followed by padding if needed, and then subsequent elements of the array.
    - A message contains 14 bytes of content before the start of an array of structs with a length of zero (i.e., no entries in the array), and then followed by a `ShortString` field: The initial 14 bytes of content is written, followed by two bytes of zero-padding, followed by a `uint32_t` (with value zero) for the array length, and then followed directly by the string (written as a `uint32_t` string length followed by string contents); no padding is needed between the array length and string length.

All messages can be encoded/decoded for sending across the network using the above rules; an example C language program for encoding and decoding a general message is given in Appendix A and a ParameterInfoReply message in Appendix B. With the exception of the DataOutDataPacketPush message, messages can generally be encoded and decoded in a straightforward, sequential manner to streams, sockets, buffers, or similar, depending on the programming languages paradigm. The DataOutDataPacketPush can be decoded in the same straightforward sequential manner, but when encoding (which only happens on the control module), the values in the device_frame_uuids, device_frame_offsets, and device_frames fields are dependent on each other, so depending on the programming language, it may be necessary to encode the device_frames field before the other two fields. This choice was made to potentially ease implementing data-out devices that may only care about looking at data from a certain device type (e.g., only looking for energy calibration changes), so they wouldn't need to implement decoding structures applicable to devices not of interest. See the DataOutDataPacketPush message definition for further details.

3.4 Message Transaction Model

Messages with the "request" or "push" suffix in their name (e.x., DeviceStatusRequest or ParameterUpdatePush) initiate a message exchange thread, while messages with "reply" or "acknowl-

edge" (e.x., DeviceStatusReply or ParameterUpdatePushAck) suffixes are respectively always in response to request or push messages of the same base name. The message ID (the fourth through eighth bytes of each message) of replies and acknowledgments must exactly match the request or push message to which they respond.

The convention used in this RAPTER Interface Specification is that request messages are either asking a device for information, or to perform an action, and their respective replies either carry the requested information, or they give the result of the requested operation. All requests must have at least one reply, and multiple replies may be necessary. For example, a ChangeDeviceStateRequest may take some time to complete so the device may send back a number of progress updates during the transition, and then a final reply indicating completion; note that the progress updates and final reply are all ChangeDeviceStateRequest messages. Push messages contain information generated during device operations, such as updates to parameters (e.g., temperature, voltage, etc.), radiation data, or error conditions that occur. Acknowledgments inform the sender of push messages that the message was received, and, if buffering is enabled (see Section 3.5.2), that the message and all previous messages should be removed from the device's buffer.There may be, at most, one acknowledge for every push message. If buffering is not enabled, acknowledgments are optional, and may be omitted to minimize network traffic. If buffering is enabled, then not every push message needs to be acknowledged; see Section 3.5.2 for details.

Replies should be made as soon as possible upon receiving a request to ensure efficient portal operations, however a time limit of five seconds is established so that if the (first) reply is not received, it is considered an issue, and if the requester is the control module, the issue will be reported to data-out devices using a DataOutDeviceResponseIssuePush message. How tolerant devices, particularly the control module, are to timeouts before terminating the connection is not specified, but this may happen after just a single timeout. When the control module disconnects a device it will send out a DataOutDeviceDisconnectedPush message to all connected data-out devices. For further details see the REPLY_TIMEOUT_MS value.

Some "request" messages may get several replies. After the first reply, subsequent replies must be sent at an interval of, at most, every 60 seconds, until the final reply is sent. For further details see the FOLLOWUP_REPLY_TIMEOUT_MS value.

Some "push" messages are expected to be received at regular intervals, like HeartbeatPush or RadChannelDataPush messages when taking data at regular intervals. These messages should normally be received as near when expected as possible, but if they are not received within 2.5 seconds after they are expected, it is an issue and handled as above. See DATA_PUSH_TIMEOUT_MS for further details.

3.5 RAPTER Handshake

Once a WebSocket connection (see Section 1.5.2) has been established, the control module will initiate a series of requests to the device to negotiate the versions of message groups that will be used. All devices must implement the core message group. The control module must negotiate the core message group version by sending the device a SupportedMessageGroupVersionsRequest message, with the message_group field having a value of Core (0x0). The resulting SupportedMessageGroupVersionsReply message will inform the control module of which core message group versions the device supports. Then, before any further communications can pro-

ceed, the control module must inform the device of the version of the core message group that will be used going forward with a UseCoreInterfaceVersionRequest message. At this point communications defined within the core message group can proceed (for example getting further device information, or parameter information, or performing firmware upgrades).

If the control module and the device do not share a common version, then the control module will send a DataOutDeviceDisconnectedPush message to data-out, analysis, and command devices to indicate the issue. The device and control module cannot be used together until a firmware upgrade takes place to allow communications.

The version for any other message groups must still be negotiated using SupportedMessageGroupVersionsRequest/SupportedMessageGroupVersionsReply and UseMessageGroupVersionRequest/UseMessageGroupVersionReply messages before the device can start operations of its intended functionality. Again, if the control module and the device do not share a common version, then the aforementioned messages will be sent and the device and control module cannot be used together until a firmware upgrade takes place to allow communications. For the analysis module and command devices the version of the core message group, data-out message group, and respectively either the analysis message group or command message group are negotiated.

Negotiating the versions of message groups that will be used are the only required steps in the handshake. However, the control module will typically want to know additional information about the device, so it may send a DeviceInfoRequest message, and it may also want to know the health and settings status of the device which can be obtained by first requesting parameter names via a ParameterNamesRequest message, and then requesting information on each parameter using ParameterInfoRequest messages. The control module may also want to know about the devices's current state, by sending a DeviceStatusRequest message. A possible sequence of this handshake is illustrated in Figure 3.2.

If the device supports the message buffering mechanism (as indicated by the DeviceSupportsBufferingFlag bit in DeviceInfoReply::device_features), and the control module supports it, as described in Section 3.5.2, then the control module may also choose to enable message buffering using the BufferingSetOptionRequest message. If the control module does not already know for sure what state the device is in, it may want to check the device's operating status (using a DeviceStatusRequest message), or if it has any buffered messages (using a DeviceBufferStatusRequest request). If there is any buffered messages that can be recovered, recovery can be started using a BufferedMessagesRequest request.

3.5.1 Enforcement of RAPTER Communications

Only specified within this RAPTER Interface Specification is allowed between the control module and RAPTER devices, and no communication between (non-control module) devices is allowed that does not pass through the control module. If communication outside of this happens, then either the device or the control module can terminate the WebSocket connection. Devices are only allowed to communicate with the control module over the network; if communication with other devices is detected by the control module, it may block all network communications between the control module and the offending device(s). The control module may also be implemented in such a way so as to prevent the possibility of other devices communicating with each other.

However, to accommodate repairs or advanced diagnostics, devices may have other physical connections, such as JTAG connector connectors, serial Universal Serial Bus (USB) interfaces,

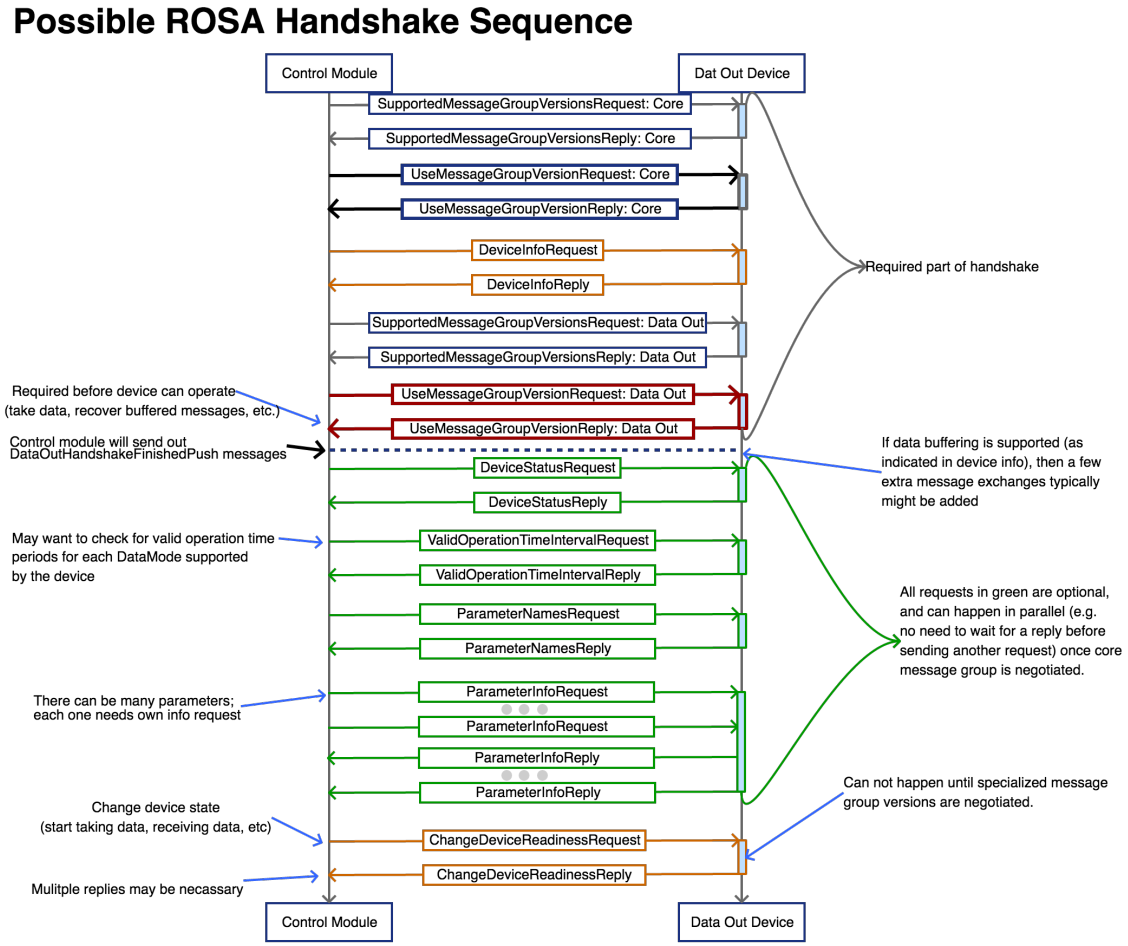

Figure 3.2: Illustration of the sequence of messages that may get exchanged between the control module and a connecting device at the beginning of a connection. Negotiation of the message group versions are the only required exchanges, but the other messages shown may be useful. Also, not shown is the potential exchange of messages to recover messages that may be buffered, for example, if a network cable gets disconnected.

Controller Area Network (CAN) ports, or other data busses to help with this. Connections through the Ethernet network adapter such as Secure Shell (ssh) or Virtual Network Computing (VNC) may also be allowed locally at the portal but may require disconnecting the device from the portals's Ethernet network since the control module hosted network may not allow these communications. If any non-RAPTER debug or troubleshooting interfaces are used on devices, they must not be used to facilitate or contribute to device operations.

3.5.2 Message Buffering

RAPTER devices and the control module may optionally implement buffering of radiation detector data, vehicle presence data, and other information produced during times of network outage. For example, if an Ethernet cable between a radiation detector module and the control module temporarily gets disconnected, then the messages (and hence radiation detector data) that would have been sent during that time can potentially be recovered by the control module, from the radiation detector module, and analyzed by the analysis module. Similarly if the connection between the control module and a data-out device is lost, the control module can buffer the data it would have sent, until a connection is re-established.

Devices specify if they support doing the buffering by the DeviceSupportsBufferingFlag bit in the DeviceInfoReply::device_features field of the DeviceInfoReply message sent to the control module. The BufferingSetOptionRequest message sent to the device from the control module informs the device that the control module may attempt recovery of buffered messages if there is a network interrupt. The control module will send the device a BufferedMessagesRequest message upon re-connection to initiate recovery of buffered messages.

Data-out devices inform the control module to buffer "push" messages it sends to them by sending the control module a DataOutBufferingEnableRequest message to initiate buffering, and a DataOutBufferedMessagesRequest message to initiate recovery of buffered messages upon reconnection.

If buffering is enabled, only "push" messages, such as RadChannelDataPush, or DataOutDataPacketPush messages participate in the buffering mechanism. "Request", "reply", and "acknowledge" messages do not get buffered. When buffering is enabled, the device which receives a push message will send back an acknowledge message[4] so that the sender knows the message, and all messages before it, were received[5]. The acknowledge message does not need to be sent for every message, but an acknowledge message should be sent often enough so that the sender's buffer does not fill up[6,7]. When buffering is active, until an acknowledge message is received by a device, or the device runs out of room in the buffer, the sender should

[4] Even though the TCP/IP protocol includes an acknowledgement mechanism for received packets, the RAPTER Interface Specification implements an acknowledgment at the application level to avoid corner cases that could cause issues such as stale WebSocket connections (frequently caused by loss of network connectivity before session is closed), and also to allow easier device implementation on top of operating system or hardware networking tools and keep separation between the application level logic and the network protocols.

[5] The WebSocket protocol ensures messages are delivered complete, error free, and sequentially in order so that it is known that no WebSocket messages are missed.

[6] The motivation for not requiring acknowledges for every push is to reduce network traffic. If it is known that network congestion will not be an issue (e.g., gamma radiation detector data is not being taken in list mode), then there may no harm in sending an acknowledge for every push message

[7] Even if an acknowledge is sent for every message, because the buffering device may not receive all acknowledgments sent by the push message receiver when an interruption occurs, buffer recovery may result in the re-sending of some messages that were successfully received.

keep the original message so that it may be re-sent, if necessary. If a device's buffer overflows, the oldest data should be removed to allow keeping of the most recent data for potential recovery. The buffer may be volatile memory, and does not need to persist across power cycles. All "push" messages normally sent by devices participate in the buffering mechanism. The buffering performed by the control module works in the same manner.

Radiation detector modules, vehicle presence modules, and power management modules will respectively buffer radiation detector data, vehicle presence data, and power management data, as well as parameters updates, status updates, and any other push messages that they send to the control module. Data-out devices, command devices, and the analysis module buffer parameter and status updates and all other push messages. The control module will buffer sensor data messages (i.e., DataOutDataPacketPush messages) as well as other push message such as DataOutDeviceDisconnectedPush messages to send to them upon re-connection.

If a network connection is re-established and buffering was enabled, the control module can determine the buffer status of devices during the handshake (see Section 3.5) and use a BufferedMessagesRequest message to initiate transfer of the buffered messages. The request to initiate recovery of buffered messages must happen before any ChangeDeviceStateRequest messages are sent to the device; if a device receives a ChangeDeviceStateRequest message before buffer recovery is initiated, the buffering device must empty its buffer and resume sending push messages as they are generated. Until the device receives a ChangeDeviceStateRequest or BufferedMessagesRequest message, it must not send push messages to the control module, but instead buffer them until receiving one of the aforementioned requests. It is worth noting that if the control module elects to not initiate buffer recovery, then any push messages generated by the device since the most recent connection will be lost, meaning any changes to the device's state will not have have been propagated to the control module. For example, if the control module sends the device a ParameterInfoRequest message before sending the ChangeDeviceStateRequest message, but there was an update to one of the parameters within the device and a ParameterUpdatePush was generated, it may never be received by the control module as it may have been placed in the buffer to queue for sending, meaning the control module will not be aware of the change in the parameter if it does not recover the buffered data[8].

There are two possible recovery modes devices must support if they support buffering:

- Serial: With this recovery mode buffered messages are sent to the control module in sequential order according to when they were generated, and while this transfer is taking place all newly generated push messages are placed at the end of the buffer and not transmitted until the messages in the buffer before them have been sent. Once the buffer has been emptied, the "push" messages should resume being sent as they are generated, as well as sending a final BufferedMessagesReply message.

- Parallel: With this recovery mode, newly generated "push" messages should be sent to the control module as they are generated, even while the buffered messages are being sent.

For both recovery modes[9] it is possible, assuming the device's buffer did not overflow, to recover all of the generated messages while the device remains operating, even if there are subsequent

[8] This situation of potentially becoming out of sync can be avoided by the appropriate message sequence choice by the control module.

[9] For both recovery modes, network traffic congestion is regulated by the TCP/IP protocol. Especially for parallel recovery, the sending device should be aware of the network throughput and network stack settings and memory

network interruptions before buffer recovery completes. All messages sent from the buffer, rather than as they are generated, must have the MessageHasBeenBufferedFlag bit set in the `flags` field (the fourth byte) of the message headers.

Buffered message recovery by data-out devices, command devices, and the analysis module from the control module has the same options and proceeds similarly as in the other direction, but the recovery is initiated by the device sending the DataOutBufferedMessagesRequest message to the control module. This request must be sent after the device has sent the control module a DeviceInfoReply message, as well as negotiated the version of data-out message group to use, but before the device has sent acknowledges to any push messages the control module may have sent the device, or the device sends the control module a DataOutBufferingEnableRequest message. When the previously buffered messages are sent from the control module to the device, the MessageHasBeenBufferedFlag bit will be set in the fourth byte of the message header.

3.6 Messages Devices Must Respond to

All non-control module devices must be able to respond to the 19 “request” messages enumerated by CoreMsgType; there are four “push” message devices that may have to generate and send, as well as receive the corresponding four trivial acknowledgment messages. See the CoreMsgType enumeration (Section 4.1.3) for a summary of the messages and content, as well as links to the actual definition of each message.

Radiation detector modules must implement response to three additional “request” messages, with there being another three “push” messages defined the device may send and receive acknowledgments of. See the RadDetectorMsgType enumeration.

Vehicle presence modules must implement responding to an additional four “request” messages, with there being another three “push” messages defined the device may send and receive acknowledgments of. See the VehiclePresenceMsgType enumeration.

Power management modules must implement responding to an additional five “request” messages, with there being another three “push” messages defined the device may send and receive acknowledgments of. See the PowerManagementMsgType enumeration.

On top of the core messages, data-out devices, analysis modules, and command devices must implement the messages given in the DataOutMsgType enumeration (corresponding to the data-out message group), which includes 10 “request” messages they must respond to, and the device may also receive 10 types of ‘push’ messages from the control module, that the device may need to acknowledge, but it is up to the device if it will do anything with the content of the messages. See the DataOutMsgType enumeration.

In addition to the core message group and data-out message group, analysis modules must also respond to two additional “request” messages corresponding to the analysis message group. See the AnalysisMsgType enumeration.

In addition to the core message group and data-out message group, command devices can give the control module five additional “request” messages, that the control module responds to;

levels to avoid situations where the application level logic hands a large amount of buffered RAPTER messages to the networking stack filling up its memory allocations, thus significantly delaying the sending of newly generated push messages, or replies to requests.

however, one of the messages (CmdDeviceRelayRequest) is used to send an individual device within the portal requests the control module would normally send, so the command devices must also implement and understand any additional messages it sends using this message. See the CommandMsgType enumeration for these messages in the command message group.

Section 4

Detailed Message Descriptions

These enumeration, struct, and message definitions will also available as C++ header files and as structured XML files suitable for generating message definition and serialization code for arbitrary programming languages.

4.1 Core Interface

4.1.1 RapterConstants Enumeration

Timing constants used by the control module to help determine when there is a potential connection or communication issue with a device. Devices also use these values to determine if intermediate "in progress" replies should be used.

Underlying integral representation: unsigned int

Enumerated values for `RapterConstants`:

REPLY_TIMEOUT_MS Value 5000

> The timeout value, in milliseconds, that a reply to a request may take, before the sender should assume something went wrong or a device is mis-behaving. In general, a device should respond as quickly as it can to requests, but some requests may involve an operation or measurement that takes some time to execute, such as turning on a high voltage or querying a complex status, so for these cases the reply will contain a status field that allows the device to immediately return a reply indicating the operation is "in progress", and a final response will be sent at a later time. Replies that may take advantage of this have a `CommandReplyStatus`, `DataOutDeviceParametersRequestStatus`, `PwrMngmtTestStatus`, `SystemStateChangeStatus`, or `DataOutDeviceParametersRequestStatus` field in the reply message. For other messages where there is no option to indicate "in progress" in the reply, then the response must be returned by this time limit. If a device times out, the control module will send a `DataOutDeviceResponseIssuePush` message to DataOut devices, at least at the beginning of communications issues.

FOLLOWUP_REPLY_TIMEOUT_MS Value 60000

When a reply indicates a "in progress" status (ex. CommandReplyStatus::InProgressValue), then it must be followed up with either the "in progress" status, or final status before this time period has elapsed, or else it is considered a timeout. More than one "in progress" reply may be sent.

DATA_PUSH_TIMEOUT_MS Value 2500

Some devices are expected to send push messages to the control module at certain times, or at certain intervals, most notably RadChannelDataPush and HeartbeatPush messages. Normally it is expected these messages are delivered to the control module as soon as possible after the scheduled time (e.g., if a portal is operating on 100 ms intervals, then data will almost always be sent within 0.1 seconds of each data interval ending), but this value defines the limit of when a timeout event will occur causing the control module to send a DataOutDeviceResponseIssuePush message to the DataOut devices, at least when delays first start.

4.1.2 MessageGroup Enumeration

Used in the first byte of each message to indicates the message group this message belongs to; the value of the second byte will then define the message type within each message group.

Used in message (other than as first byte): SupportedMessageGroupVersionsRequest

Underlying integral representation: uint8_t

Enumerated values for MessageGroup:

CoreValue Value 0x00

Message group supported by all RAPTER devices.

See also: CoreMsgType

CommandValue Value 0x01

Message group that includes commands that can be sent to the control module to control operations, change settings, debug individual devices in the portal, and provide firmware upgrades. Only command devices communicate these messages with the control module.

See also: CommandMsgType

DataOutValue Value 0x02

Message group allowing devices to receive a copy of all radiation and occupancy data, as well as analysis results, and a copy of all commands and messages sent to or from the individual devices in the portal. Messages in this message group do not effect operation of the portal. Command, DataOut and Analysis devices communicate these message with the control module.

See also: DataOutMsgType

AnalysisValue Value 0x04

Messages specific to the analysis module, allowing the command module to request analysis results, and the analysis module to provide results to the control module.

See also: AnalysisMsgType

RadDetectorValue Value 0x08

Messages communicated between the radiation detectors and the control module for communicating energy calibrations and radiation data.

See also: `RadDetectorMsgType`

VehiclePresenceValue Value 0x10

This interface is for occupancy sensors consisting of binary sensors (ex. IR beams) which indicate if a vehicle is occupying the portal, cameras to provide static images, and/or distance sensors to provide vehicle locations.

See also: `VehiclePresenceMsgType`

PowerManagementValue Value 0x20

This interface provides information from, and control over the uninterruptible power supply (UPS) servicing the portal. The assumption is there is at most one UPS per portal.

See also: `PowerManagementMsgType`

4.1.3 CoreMsgType Enumeration

Values that specify the type of message being conveyed within the core message group. All core message group requests are issued by the control module. All core message group push messages originate at a device and are directed to the control module. The first byte of a core message has a value of `MessageGroup::CoreValue` (0x0). This enumeration defines the value of the second byte of the message so that the receiver can interpret the contents of the message body. All RAPTER devices must implement sending and receiving core messages.

Underlying integral representation: `uint8_t`

Enumerated values for `CoreMsgType`:

SupportedMessageGroupVersionsRequestValue Value 0x01

Message sent from the control module to a device asking it which versions a particular message group the device supports. The message groups are: Command, Core, DataOut, Analysis, RadDetector, and VehiclePresence.

See also: `SupportedMessageGroupVersionsRequest`

SupportedMessageGroupVersionsReplyValue Value 0x81

Reply to a `SupportedMessageGroupVersionsRequest` message containing the versions that the device supports.

See also: `SupportedMessageGroupVersionsReply`

UseMessageGroupVersionRequestValue Value 0x02

Message sent from the control module to a device telling it which version of the specified message group to use.

See also: `UseMessageGroupVersionRequest`

UseMessageGroupVersionReplyValue Value 0x82

Reply to a `UseMessageGroupVersionRequest` message giving the success status of using the specified version of the interface.

See also: `UseMessageGroupVersionReply`

DeviceInfoRequestValue Value 0x05

Message sent from the control module to a device requesting information about the device.

See also: `DeviceInfoRequest`

DeviceInfoReplyValue Value 0x85

Reply to a `DeviceInfoRequest` message containing information about the device.

See also: `DeviceInfoReply`

DeviceStatusRequestValue Value 0x06

Message sent from the control module to a device to request its current operating status.

See also: `DeviceStatusRequest`

DeviceStatusReplyValue Value 0x86

Reply to a `DeviceStatusRequest` message which contains the devices current operating status.

See also: `DeviceStatusReply`

DeviceBufferStatusRequestValue Value 0x07

Message sent from the control module to a device inquiring about its buffer status.

See also: `DeviceBufferStatusRequest`

DeviceBufferStatusReplyValue Value 0x87

Reply to a `DeviceBufferStatusRequest` message containing the status of the devices buffer.

See also: `DeviceBufferStatusReply`

BufferedMessagesRequestValue Value 0x08

Message sent from the control module to a device requesting it to send buffered messages.

See also: `BufferedMessagesRequest`

BufferedMessagesReplyValue Value 0x88

Reply message to a `BufferedMessagesRequest` message providing the status of sending buffered messages.

See also: `BufferedMessagesReply`

BufferingSetOptionRequestValue Value 0x09

Message sent from the control module to a device to indicate to it whether messages should be buffered, or if the buffer should be cleared.

See also: `BufferingSetOptionRequest`

BufferingSetOptionReplyValue Value 0x89

Reply to a BufferingSetOptionRequest message indicating the status of the request.

See also: `BufferingSetOptionReply`

PingRequestValue Value 0x0a

Message sent from the control module to a device asking for a simple response; useful for ensuring devices are responsive and to perform crude checks on latencies or time synchronization.

See also: `PingRequest`

PingReplyValue Value 0x8a

Reply to a `PingRequest` message containing a timestamp of when the device sent the message.

See also: `PingReply`

PowerDownRequestValue Value 0x0b

Message sent from the control module to a device requesting it to either reboot or power off.

See also: `PowerDownRequest`

PowerDownReplyValue Value 0x8b

Reply to a `PowerDownRequest` message giving status of the request.

See also: `PowerDownReply`

SendLogsRequestValue Value 0x0c

Message sent from the control module to a device requesting the contents of its internal log file for a specified time period; maintaining a log file is optional for devices to support.

See also: `SendLogsRequest`

SendLogsReplyValue Value 0x8c

Reply to a `SendLogsRequest` message containing the status of being able to fulfill the request, and if so the log file contents.

See also: `SendLogsReply`

DeviceTimeStatisticsRequestValue Value 0x0d

Message sent from the control module to a device requesting basic time statistics such as total time the device has operated and its current time being powered on.

See also: `DeviceTimeStatisticsRequest`

DeviceTimeStatisticsReplyValue Value 0x8d

Reply to a `DeviceTimeStatisticsRequest` message containing basic time statistics of the device.

See also: `DeviceTimeStatisticsReply`

DeviceOperabilityCheckRequestValue Value 0x0e

Message sent from the control module to a device asking if there is anything that would prevent it from transitioning to the `OperatingMode::OperatingValue` state.

See also: `DeviceOperabilityCheckRequest`

DeviceOperabilityCheckReplyValue Value 0x8e

Reply to a `DeviceOperabilityCheckRequest` message that indicates if the device is currently capable of operating.

See also: `DeviceOperabilityCheckReply`

SupportedDataCollectionIntervalsRequestValue Value 0x0f

Message sent from the control module to a device asking for timing intervals supported by the device for a given `DataCollectionModes`.

See also: `SupportedDataCollectionIntervalsRequest`

SupportedDataCollectionIntervalsReplyValue Value 0x8f

Reply to a `SupportedDataCollectionIntervalsRequest` message that specifies valid time intervals for a specific `DataCollectionModes`.

See also: `SupportedDataCollectionIntervalsReply`

ChangeDeviceStateRequestValue Value 0x10

Message sent from the control module to a device telling it to change its operating mode (e.g., start, stop, or get ready to take data).

See also: `ChangeDeviceStateRequest`

ChangeDeviceStateReplyValue Value 0x90

Reply to a `ChangeDeviceStateRequest` message providing the status of the state change. There may be multiple replies for a single request.

See also: `ChangeDeviceStateReply`

FirmwareUpgradeRequestValue Value 0x11

Message sent from the control module to a device which contains a firmware upgrade package.

See also: `FirmwareUpgradeRequest`

FirmwareUpgradeReplyValue Value 0x91

Reply to a `FirmwareUpgradeRequest` message providing the status of the firmware upgrade. There may be multiple replies for a single request.

See also: `FirmwareUpgradeReply`

ParameterNamesRequestValue Value 0x12

Message sent from the control module to a device asking for the names of all the parameters the device defines. See `ParameterInfoRequest` and `ParameterInfoReply` for further information about parameters.

See also: `ParameterNamesRequest`

ParameterNamesReplyValue Value 0x92

Reply to a `ParameterNamesRequest` message containing a list of names for all of the parameters of the device.

See also: `ParameterNamesReply`

ParameterInfoRequestValue Value 0x13

Message sent from the control module to a device asking for information about a single parameter.

See also: `ParameterInfoRequest`

ParameterInfoReplyValue Value 0x93

Reply to a `ParameterInfoRequest` message containing information on the requested parameter, including most recent measured values, if applicable.

See also: `ParameterInfoReply`

SetParameterRequestValue Value 0x14

Message sent from the control module to a device instructing it to change a parameter's set value; only applicable to parameters that may be set.

See also: `SetParameterRequest`

SetParameterReplyValue Value 0x94

Reply to a `SetParameterRequest` message indicating status of setting the specified value.

See also: `SetParameterReply`

MeasurementTypeChangeRequestValue Value 0x15

Message sent from the control module to a device telling it that the `MeasurementType` (what is being measured) has changed.

See also: `MeasurementTypeChangeRequest`

MeasurementTypeChangeReplyValue Value 0x95

Reply to a `MeasurementTypeChangeRequest` message indicating when the change was processed.

See also: `MeasurementTypeChangeReply`

NotificationPushValue Value 0x16

Message sent from a device to the control module notifying it of an error, for example, if it received a message that could not be decoded, etc.

See also: `NotificationPush`

NotificationPushAckValue Value 0x96

Acknowledgement of receiving a `NotificationPush` message.

See also: `NotificationPushAck`

ParameterUpdatePushValue Value 0x18

Message sent from a device to the control module providing updated information about the parameter, for example, a newly measured value, or a change in the state of health with respect to the parameter.

See also: `ParameterUpdatePush`

ParameterUpdatePushAckValue Value 0x98

Acknowledgement of receiving a `ParameterUpdatePush` message.

See also: `ParameterUpdatePushAck`

DeviceStatusPushValue Value 0x19

Message sent from a device to the control module containing an updated status of the device.

See also: `DeviceStatusPush`

DeviceStatusPushAckValue Value 0x99

Acknowledgement of receiving a `DeviceStatusPush` message.

See also: `DeviceStatusPushAck`

HeartbeatPushValue Value 0x1b

Message sent from a device to the control module at regular intervals when the device is in the `OperatingMode::OperatingValue` with a data collection mode of `DataCollectionModes::NoOriginateValue` or `DataCollectionModes::OnEventVa` with a heartbeat interval specified.

See also: `HeartbeatPush`

HeartbeatPushAckValue Value 0x9b

Acknowledgement of receiving a `HeartbeatPush` message.

See also: `HeartbeatPushAck`

4.1.4 MsgFlags Enumeration

Flags that are part of the 4th byte of each message to indicate further information about this message.

Underlying integral representation: `uint8_t`

Enumerated values for `MsgFlags`:

MessageHasBeenBufferedFlag Value 0x01

Indicates this message has been buffered, and is not being sent at the time of the originating event. The generating event for this message may have happened while the network connection was not active, or if buffer recovery was `BufferRecoveryMode::SerialValue` this message may have been generated while the sending of previously buffered messages was still occurring, so it could not be

sent in real time. This bit being set does not alter the byte-format of how this message should be decoded.

See also: BufferRecoveryMode , BufferStatusFlags::DeviceHasBufferedDataFlag , BufferedMessagesRequest

4.1.5 SupportedMessageGroupVersionsRequest Message

This message is sent from the control module to connecting devices to request the versions of the RAPTER message groups that the device supports.

This will be used by the control module to negotiate the version of messages that will be sent, and will be done during the handshake portion of setting up the connection. Once the control module determines the version to use, it will send the device an UseMessageGroupVersionRequest message so that the device knows what version of the message group to use. If a common message version can not be identified, the control module will send a DataOutDeviceDisconnectedPush message with a value set indicating an invalid message group version. The format of the present message will not change, even if the version of the core interface is incremented, because a common version of the core message group has not yet been negotiated when this message is received. No other messages may be sent until a version of the core message group has been negotiated.

See Also: SupportedMessageGroupVersionsReply , UseMessageGroupVersionRequest

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x01
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
message_group_to_negotiate	uint8_t enumerated by MessageGroup
Field Description: The message group for which the supported versions are requested to be listed in the reply message.	

4.1.6 SupportedMessageGroupVersionsReply Message

This message is a reply to an SupportedMessageGroupVersionsRequest and carries with it a list of versions that the device supports.

The format of this message will not change, even if the version of the core interface is incremented, because a common version of the core interface has not yet been negotiated when this message is received. No other messages may be sent until a version of the core message group has been negotiated.

See Also: SupportedMessageGroupVersionsRequest , UseMessageGroupVersionRequest

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x81
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
message_group_being_negotiated	uint8_t enumerated by MessageGroup
Field Description: The message group for which the supported versions are requested.	
message_group_versions_supported	Short Array of uint8_t
Field Description: The supported versions of the requested message group.	

4.1.7 UseMessageGroupVersionRequest Message

Message from the control module to a client device, instructing the device what version of the interface to use when sending messages or replies to messages.

Until the version of the RAPTER Core message group has been defined, additional communications cannot and should not be exchanged. An attempt at communication before this is invalid and makes the WebSocket connection subject to disconnect at the device or control modules discretion. The format of this message will not change, even if the version of the core interface is incremented, because a common version of the core message group has not yet been negotiated when this message is received. No other messages may be sent until a version of the core message group has been negotiated. The version of a message group used may not be re-negotiated after the control module sends this message until the WebSocket connection has been terminated and the connection process starts again.

See Also: SupportedMessageGroupVersionsRequest , SupportedMessageGroupVersions

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x02
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
message_group_being_negotiated	uint8_t enumerated by MessageGroup
Field Description: The message group for which a version is being specified to be used.	
message_group_version_to_use	uint8_t
Field Description: Version of the message group to use when communicating with the control module.	

4.1.8 UseMessageGroupVersionStatus Enumeration

Enumeration to specified the status of a UseMessageGroupVersionRequest message.

See also: UseMessageGroupVersionReply

Used in message: UseMessageGroupVersionReply

Underlying integral representation: uint8_t

Enumerated values for UseMessageGroupVersionStatus:

UseMessageGroupVersionAcceptedValue Value 0x00

The requested version of the message group was accepted, and all further communications will use this version.

UseMessageGroupAlreadyUsingRequestedVersionValue Value 0x01

This was a duplicate request; no action taken.

UseMessageGroupUnsupportedVersionValue Value 0x02

The requested version was not one of the supported versions.

UseMessageGroupUnsupportedMessageGroupValue Value 0x03

The message group specified isnt supported by this device.

UseMessageGroupVersionAlreadySetValue Value 0x04

The message group version had already been set, and its not the requested version; changing the version after setting it is not allowed.

4.1.9 UseMessageGroupVersionReply Message

A reply to an `UseMessageGroupVersionRequest` message, which acknowledges the command to the control module, and gives the timestamp of when it became effective (if successful).

See Also: `UseMessageGroupVersionRequest` , `UseMessageGroupVersionStatus`

Message Contents:

Field Name	Data Type
`message_group` *Field Description*: The RAPTER `MessageGroup` this message corresponds too.	`uint8_t` with value 0x00
`message_type` *Field Description*: The type of message within the specified message_group, that this message corresponds to.	`uint8_t` with value 0x82
`message_group_version` *Field Description*: The version of the message_group being used.	`uint8_t`
`message_flags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	`uint8_t` bits defined by `MsgFlags`
`message_id` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	`uint32_t`
`effective_timestamp` *Field Description*: Microseconds since the UNIX epoch.	UtcTimePoint
`message_group_being_negotiated` *Field Description*: The `MessageGroup` being negotiated.	`uint8_t` enumerated by `MessageGroup`
`use_message_group_status` *Field Description*: The status of responding to a request to comply with an assigned message group version.	`uint8_t` enumerated by `UseMessageGroupVersionStatus`

4.1.10 NotificationSeverity Enumeration

Indicates the severity of the issue reported by this notification.

Used in message: `NotificationPush`

Underlying integral representation: `uint16_t`

Enumerated values for `NotificationSeverity`:

InfoValue Value 0x00

For the device, module, or system being reported, this notification provides information about a condition that does not prevent operation.

WarningValue Value 0x01

The reported issue should be addressed, or has the indicated potential to compromise device operation, but device operation continues for the time being.

CriticalValue Value 0x02

For message caused by a failure that partially impairs the device operations, or which may cause data being produced or function being performed to be compromised. The device will keep attempting to work, but the results are suspect.

InoperableValue Value 0x03

The reported issue has resulted in a fatal condition for the device, module, or system being reported, preventing continued operation.

4.1.11 NotificationCause Enumeration

Enumeration to indicate details on error message cause; may help to provide 'why' an error message has occurred.

This is necessarily a non-exhaustive list, so if no other value fits, choose a value of `NotificationCause::OtherCauseValue`.

Used in message: `NotificationPush`

Underlying integral representation: `uint8_t`

Enumerated values for `NotificationCause`:

IllFormattedMsgValue Value 0x00

Indicates that a received message was an invalid type, jumbled, or an otherwise invalid format relative to what was expected, i.e. a message whose length is too short for the type of message it is.

IllegalFieldEntryValue Value 0x01

Indicates that a value expressed as (part of) a message is an invalid value, for example, if the message type value is not one defined by this specification.

InvalidMessageContextValue Value 0x02

Indicates a received message is being used in either the wrong context, or is not supported by the receiving device, for example if the wrong type of message is replied, i.e. an `PingReply` is sent in response to a version request.

InvalidStateForRequestValue Value 0x03

Indicates a received message was not valid for the current device state, for example, if a heartbeat message is received from the device while the device is in the Standby mode.

HardwareErrorOperationNotStartedValue Value 0x04

Indicates a hardware error prevented an operation from starting, for example, the high voltage could not be turned on when requested due to a tripped interlock.

HardwareErrorOperationNotCompletedValue Value 0x05

Indicates a hardware error prevented the completion of an operation, for example, a fuse blew while turning on the high voltage.

HardwareErrorOccurredValue Value 0x06

Indicates a hardware error occurred during operation, for example, the fuse blew during data acquisition.

OperationAnomalyValue Value 0x07

Indicates a non-standard event occurred during operation, for example, a significant voltage drop followed by recovery.

SoftwareIssueValue Value 0x08

Indicates a software issue that caused a disruption.

DebugValue Value 0x09

Indicates a message utilized for debug purposes only.

InvalidStringValue Value 0x0a

Indicates a string which was passed as part of a message is invalid. This could be due to it being ill formed (i.e. not UTF-8), not being a valid name (e.g. a parameter name which doesn't exist), or it being too short or too long.

InvalidTypeValue Value 0x0b

Indicates that a wrong type is specified, for example, a parameter value is indicated as a string when the expected parameter is an integer.

IncompatibleVersionValue Value 0x0c

Indicates a common version of a given protocol could not be found.

InvalidMessageTypeValue Value 0x0d

A value of MessageType was specified for which no message is defined for this message group.

InvalidStateRequestValue Value 0x0e

A request to go into an invalid state was received.

FeatureNotSupportedValue Value 0x0f

A request was made which requires a feature the device does not support.

InvalidPropertyValue Value 0x10

Indicates that an attempt is made to access or set a field or property which can not be accessed or set, e.g. trying to set the value of a parameter when the parameter is fixed.

ValueOutOfRangeValue Value 0x11

Indicates that the value attempting to be set is out of the allowed range.

ValueTimeoutValue Value 0x12

Indicates a timeout occurred, or something is taking longer than expected.

UnexpectedDeviceRestart Value 0x13

Indicates device had an unexpected restart or crash, and this message provides information on what happened.

OtherCauseValue Value 0xff

Indicates that the message does not fit into any of the previous error causes.

4.1.12 Notification Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct used to hold a warning, debug message, abnormal response, error, or condition warranting an explanation. This contents of this struct is used within `NotificationPush` messages for devices to report errors to the control module.

Issues prompting a notification include, but are not limited to, invalid message, un-executable message, execution error, debug information, hardware issue, and the like.

See Also: `DataOutDataFrame` , `NotificationPush` , `DataOutSystemStateChangePush`

Message Contents:

Field Name	Data Type
`timestamp`	UtcTimePoint
Field Description: The timestamp when the error actually happened, not when it was sent on the network. If the time is not applicable to this particular message, this integer should have the value of zero.	
`reference_message_id`	`uint32_t`
Field Description: If this notification is being sent with regards to another message, this field provides the 'message_id' of that message. If this notification is not in regards to another message, then this field must be zero. An example of when a notification may reference another message would be if `CmdSystemStateChangeRequest` can not be successfully completed, in addition to the device sending a `CmdSystemStateChangeReply` indicating the request could not be completed, the device could also send a notification with an English description with further device specific information that may help to diagnose the issue. If non-zero, this message_id field must correspond to a message sent within the time specified by `RapterConstants::DATA_PUSH_TIMEOUT_MS`.	
`severity`	`uint16_t` enumerated by `NotificationSeverity`
Field Description: Indicates the severity of the issue reported by this notification.	
`cause`	`uint8_t` enumerated by `NotificationCause`
Field Description: The cause of the notification; should be considered to be indicative or descriptive, and not an absolute or definitive form of information, since there will likely be errors or issues that are not enumerated in the `NotificationCause` notification types.	
`description`	`Large String`
Field Description: Human readable, UTF-8 encoded text which describes the error or issue in a manner appropriate to display to a technician to provide information or to help diagnose the problem.	

4.1.13 NotificationPush Message

Message sent from a device to the control module notifying it of an error, warning, or providing debug information. This message may also be used to provide additional information when a request can not successfully be completed. This message will also be sent upon new RAPTER connections if there are pending issues (see DeviceStatusFlags and DeviceOperabilityStatus) or the device had to restart due to an issue. Examples of when a device might send a NotificationPush to the control module include: hardware error, invalidly formatted messages, not being able to successfully complete a request, when an unintended operation state change occurs due to hardware issues, or after new RAPTER connections are formed and there are pending maintenance issues or errors, or there was an unintended restart of the device.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x16
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
timestamp	UtcTimePoint
Field Description: The timestamp when the error actually happened, not when it was sent on the network. If the time is not applicable to this particular message, this integer should have the value of zero.	
reference_message_id	uint32_t
Field Description: If this notification is being sent with regards to another message, this field provides the 'message_id' of that message. If this notification is not in regards to another message, then this field must be zero. An example of when a notification may reference another message would be if CmdSystemStateChangeRequest can not be successfully completed, in addition to the device sending a CmdSystemStateChangeReply indicating the request could not be completed, the device could also send a notification with an English description with further device specific information that may help to diagnose the issue. If non-zero, this message_id field must correspond to a message sent within the time specified by RapterConstants::DATA_PUSH_TIMEOUT_MS.	
severity	uint16_t enumerated by NotificationSeverity
Field Description: Indicates the severity of the issue reported by this notification.	
cause	uint8_t enumerated by NotificationCause

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The cause of the notification; should be considered to be indicative or descriptive, and not an absolute or definitive form of information, since there will likely be errors or issues that are not enumerated in the NotificationCause notification types.	
description	Large String
Field Description: Human readable, UTF-8 encoded text which describes the error or issue in a manner appropriate to display to a technician to provide information or to help diagnose the problem.	

4.1.14 NotificationPushAck Message

A message acknowledging the receipt by the control module of a NotificationPush message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x96
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.1.15 DeviceInfoRequest Message

Message sent by the control module to a device requesting information about the device. This includes information about which message group the device supports, and device specific details such as serial number or hardware/firmware versions.

See Also: DeviceInfoReply , DeviceInfoRequest , DataCollectionModes

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_type	uint8_t with value 0x05
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.1.16 DataCollectionModes Enumeration

Flags that may be bitwise OR'd to indicate one or more conditions regarding data collection modes. Specifies the data collection modes supported by a device, or a specific mode for the device. A single flagged bit indicates to the mode in which the detector should (or does) collect data. A device may be capable of multiple data collection modes. Bitwise flags are used to communicate to the control module which data collection modes the device is capable of exercising.

See also:

- DeviceInfoReply
- ChangeDeviceStateRequest

Used in message: DeviceStatusReply

Underlying integral representation: uint8_t

Enumerated values for DataCollectionModes:

NoOriginateValue Value 0x00

This value is used for devices that do not collect data (i.e. not detectors), or for messages where the operating mode is specified as OperatingMode::StandByValue. If the device supports DataCollectionModes::NoOriginateValue mode (as specified the DeviceInfoReply message), it can not support any other operating mode. DataOut, Command, and Analysis devices must support this data collection mode, and only this data collection mode. If this DataCollectionModes is specified along with OperatingMode::OperatingValue, then a DeviceStatus::collection_interval_m can still be specified but it must be in the list: 0, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, or 1000 (milliseconds). (These intervals evenly divide a second.) If this value is non-zero, then the device must send a HeartbeatPush message to the control module at the specified millisecond multiple, of each second. For example, if a value of 100 is specified, then at the beginning of each absolute clock second (e.g., the time determined via PTP) a HeartbeatPush message will be sent, and then 0.1 seconds later, another one, and so on.

ClockTimeIntervalValue Value 0x01

The device collects information and sends it to the control module at integral divisors of system clock time. The collection time period (specified by a ChangeDeviceStateRequest message) must be equal to or less than 1000 ms. The integral divisors limitation means that intervals must evenly divide a second; this then allows straightforward synchronization of multiple detector modules and devices. As an integral divisor, the interval must be in the list {1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000} milliseconds, and meet the requirements specified by the device via a SupportedDataCollectionIntervalsReply message. If a sensor collects data at 0.1 second intervals, it will start collection when the system clock time second begins (time x), and then at 0.1 seconds after the second begins, i.e. at x.1 seconds, then at x.2 seconds, x.3 seconds, etc. Gamma, neutron, and occupancy detectors may support this mode of data collection. If a heartbeat is specified for non-sensor devices, the same timing rules apply.

OnEventValue Value 0x02

The device sends information as soon as an event is detected. For gamma and neutron detectors this is list mode data acquisition. Vehicle presence sensors send information whenever a beam state changes. Gamma and neutron detectors may support this mode of data collection; vehicle presence and power management modules must support this mode, and only this mode. The DeviceStatus::collection_interval_ms specifies when and if HeartbeatPush messages must be sent by the device, with a value in the list: 0, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, or 1000 (milliseconds). A value of zero specifies HeartbeatPush messages should not be sent. The timing of HeartbeatPush messages follow the same rules as for DataCollectionModes::ClockTimeIntervalValue.

LiveTimeDwellValue Value 0x04

Specifies live time dwell mode data acquisition; may only be supported by gamma and neutron radiation detectors. The length of the measurement dwell time is specified in the ChangeDeviceStateReply command, given as the number of milliseconds to acquire. If no time is specified, data will be acquired until receipt of a ChangeDeviceStateRequest command, which causes a transition. When acquisition is finished, data will be sent to the control module, and then the device will transition to the OperatingMode::ReadyValue and send a DeviceStatusPush to the control module to indicate the device's new state. This mode of data is useful to allow longer acquisition, such as a 5 minute gamma-ray spectral acquisition. In the case that a state transition out of OperatingMode::OperatingValue is scheduled (by a ChangeDeviceStateRequest message) after data acquisition is started, but before the dwell time specified is reached, then the scheduled transition should be obeyed and data sent as normal. If no transition is specified and the dwell time requested is able to be fulfilled, the device should transition to the ready state from the operating state upon completion of the dwell. Currently no modules are required to support this mode of data acquisition, but support in both gamma and neutron detectors is encouraged.

RealTimeDwellValue Value 0x08

Specifies real time dwell mode data acquisition; may only be supported by gamma and neutron radiation detectors. Similar to DataCollectionModes::LiveTimeDwellValue, but acquisition time is limited by the real (system clock) time, instead of live time.

4.1.17 DeviceFeaturesFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more features common to all RAPTER modules or devices.

Devices further indicate capabilities unique to the device type in `RadSubDetectorInformationReply`, `VehiclePresenceSubDetectorInformationReply`, or `PwrMngmtInformationReply` messages.

See also:

- `DeviceInfoReply`
- `DeviceInfoReply::device_features`

Underlying integral representation: `uint64_t`

Enumerated values for `DeviceFeaturesFlags`:

DeviceSupportsBufferingFlag Value 0x01

Flag to indicate that the device, or control module supports buffering. The device will store push messages in its buffer. If a network connection is dropped, upon re-connection if the `BufferStatusFlags::DeviceHasBufferedDataFlag` bit of `BufferStatus::buffer_status_flags` is set, then the control module can send a request for the buffered data using a `BufferedMessagesRequest` message.

DeviceSupportsFirmwareUpgradesFlag Value 0x10

The device supports firmware upgrades.

See also: `MsgRequestFirmwareUpgrade`

DeviceUsesOperatingBackgroundFlag Value 0x20

Device takes advantage of knowing when the detection system is in the "background" state (e.g. when no vehicle occupies the portal).

DeviceUsesActiveMaintenanceFlag Value 0x40

The device may use the `MeasurementType::ActiveMaintenanceValue` time to perform some sort of calibration, maintenance, or other activity that will block it from other uses or cause reported data to not be as expected.

DeviceActiveMaintenanceMayEffectOtherDevicesFlag Value 0x80

During active maintenance this device may do something that influences the environment in such a way as to potentially affect other devices in the portal. An example would be if a radioactive check source that is not normally exposed, is exposed, which would potentially affect other gamma radiation detection modules. This bit shall only be set if `DeviceFeaturesFlags::DeviceUsesActiveMaintenanceFlag` is also set.

4.1.18 ComponentVersionInformation Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Struct to allow specifying device component versions other than the overall firmware and hardware versions. Examples could include: analysis routine version, nuclide library version, operating system version, high voltage board version, WebSocket library version, a vendor specific library version, etc.

Message Contents:

Field Name	Data Type
`name` *Field Description*:	`Short String`
`value` *Field Description*:	`Short String`
`description` *Field Description*:	`Short String`

4.1.19 DeviceInfoReply Message

Message sent by a device in reply to a `DeviceInfoRequest` from the control module.

Message Contents:

Field Name	Data Type
`message_group` *Field Description*: The RAPTER `MessageGroup` this message corresponds too.	`uint8_t` with value 0x00
`message_type` *Field Description*: The type of message within the specified message_group, that this message corresponds to.	`uint8_t` with value 0x85
`message_group_version` *Field Description*: The version of the message_group being used.	`uint8_t`
`message_flags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	`uint8_t` bits defined by `MsgFlags`
`message_id` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	`uint32_t`
`device_type` *Field Description*: The device type, as determined from the following options (designated by the associated MessageGroup value): command device (Command), analysis device (Analysis), radiation detection device (RadDetector), vehicle presence device (VehiclePresence), or control module (Core). The device type determines the message groups that the device must support. See Chapter 2 for details.	`uint8_t` enumerated by `MessageGroup`
`data_collection_modes` *Field Description*: Bitwise OR of flags representing data collection modes supported by the device.	`uint8_t` bits defined by `DataCollectionModes`
`device_features`	`uint64_t` bits defined by `DeviceFeaturesFlags`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Bitwise OR of flags representing device features.	
buffer_size	uint64_t
Field Description: The size of the buffer for buffering. This size can be an estimate. Should be set to 0 if the device does not support data buffering when the connection is down. When the network connection is down, Push messages should be buffered for retrieval once the connection is back up; see BufferedMessagesRequest and BufferedMessagesReply for details.	
num_subdetectors	uint8_t
Field Description: The number of sub detectors the device contains; applicable only to radiation detector, vehicle presence, and power management modules. For other devices this must be set to 0. As examples, for gamma-ray detectors this might indicate independent crystals (e.g. 4 crystals in one panel or module), for neutron detectors this might be the number of independently read-out He-3 tubes, and for vehicle presence sensors, the number of IR beams plus number of cameras. For power management modules it represents the number of independent output lines. Note that all sub-detectors must operate in the same data collection mode (DataCollectionModes::ClockTimeIntervalValue, DataCollectionModes::OnEventValue, etc) and at the same data collection intervals (10ms, 100ms, etc.). See also: • RadListModeDataPush::subdetector_number • RadChannelDataPush::subdetector_number • RadEnergyCalibrationUpdatePush::subdetector_number • RequestEnergyCalibration::subdetector • VehiclePresenceSubDetectorInformationRequest • PwrMngmtInformationRequest	
device_uuid	Uuid128
Field Description: The universally unique ID for this device. This number must be globally unique, and can be assigned according to the deploying agency conventions; for example assigned as the thumbprint of the devices cryptographic certificate. This value will not change for a given physical device.	
serial_number	Short String
Field Description: The manufacturer specified serial number that is expected to be unique at least with respect to the vendor and model. This value must remain fixed across firmware upgrades, power cycles, setting changes, and if at all possible hardware repairs. No restrictions on format other than length, and valid UTF-8 text.	
manufacturer	Short String
Field Description: Specifies the manufacturer of this device. No restrictions other than length, and valid UTF-8 text.	
model	Short String
Field Description: Specifies the model of this device. No restrictions other than length, and valid UTF-8 text.	
hardware_version	Short String

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Specifies the version of the hardware for the device. No restrictions other than length, and valid UTF-8 text. This should be descriptive and change with respect to hardware changes.	
`firmware_version`	`Short String`
Field Description: Specifies the version of the current firmware for the device. No restrictions other than length, and valid UTF-8 text. This should be descriptive and change with respect to firmware changes or updates.	
`other_component_versions`	`Short Array` of `ComponentVersionInformation`
Field Description: Versions of additional components in the device, that may be informative for compatibility, troubleshooting, or maintenance. Examples include: underlying operating system version, high voltage daughter card version, nuclide library version, analysis library version, etc.	
`specialized_capabilities_description`	`Short String`
Field Description: A free form English language description of the capabilities and hardware specific to the specialized purpose of this device. No restrictions other than length, and valid UTF-8 text. Examples might include: • "3x3 NaI gamma detector, 7% FWHM @ 661 keV, temperature compensated energy calibration, pulse pileup filter, 250MHz waveform sampling" • "Data archiving device to save all health information to a database" • "Graphical user interface device for portal operations. Touchscreen." • "Analysis software based on DHSIsotopeID v13, customized for PVT."	
`computing_platform_description`	`Short String`
Field Description: A free form English language description of the computing resources of the device, such as descriptions of its CPUs, ram memory, hard drive space, Ethernet adapter capabilities, operating system, etc. No restrictions other than length, and valid UTF-8 text. Examples include: • "PC4F1453 MCU, 120 MHz, 256 KB SRAM, 4GB SD-card storage, IEEE-1588 Ethernet transceiver, FreeRTOS 9.0.0" • "Dual core 2 GHz Arm CPU, 1 GB ram, Linux kernel 3.4."	

4.1.20 DeviceStatusRequest Message

Request from the control module for a device's current status.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x06

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.1.21 DeviceStatusFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more potential negative conditions of devices. Any change to the status of one of these flags must trigger a `DeviceStatusPush` being sent to the control module, or if its the control module itself, it needs to provide notice by issuing data out, core, and analysis messages.

Underlying integral representation: `uint32_t`

Enumerated values for `DeviceStatusFlags`:

DeviceHasWarningConditionFlag Value 0x01

The device has a warning condition, but retains the ability to operate, but there is a danger there could be an adverse effect on operations in the future if the condition(s) worsens or persists. If the device is a detector, the data quality is not yet degraded to the point that it is not useable. Examples include: CPU temperature is above a warning threshold, but the device is still able to operate in its full capacity. Supply voltage is outside of expected range, but not currently recognized as impacting performance. The condition causing the warning should be reported via a `ParameterUpdatePush` message as well.

DeviceHasImpactedOperationFlag Value 0x02

The device is still operating, but has an issue impacting performance, or causing it to produce suspect or bad data. Examples include: CPU temperature high enough that processor throttling is being applied. High voltage power supply is unable to reach the desired voltage; detection is still occurring, but data are suspect. The condition impacting performance should be reported via a `ParameterUpdatePush` message as well.

DeviceHasFatalConditionFlag Value 0x04

There is a condition preventing device operation. Examples include: - High voltage power supply is drawing more current than allowed. - A required interrupt is open, like for example the enclosure is open. - A radiation field is too high for a gamma detector. - Error initializing a required resource. If caused by something that can be described by the parameter mechanism, a `ParameterUpdatePush` message will have been sent when the condition started, or the parameters can be queried using the `ParameterInfoRequest`

message. If something like a failed hard drive, corrupt config file, or internal device communications error happens, where it is not practical to represent as a parameter, then a description of the cause must be reported using a NotificationPush message, both when the condition happens, and after new RAPTER connections are made until the condition is fixed.

DeviceNeedsServiceFlag Value 0x08

Indicates device is in need of service, but still operational. A NotificationPush message must also be sent which describes the service type needed, both when the service is initially determined to be needed, and also upon new RAPTER connections are creation until the condition is resolved. Examples include: - A Manual energy calibration is necessary. - A non-essential component (e.g., vibration sensor) has failed. - A cooling fans air filter needs replacing.

DeviceWouldLikeBackgroundTimeFlag Value 0x200

Indicates the device would like dedicated time with no vehicle in the portal, or no item of interest being measured; the device will remain operating during a background period.

DeviceNeedsBackgroundTimeFlag Value 0x400

Indicates the device can not operate without dedicated background time. For example, a gamma detector may be needed to gain-match PMTs, or calibrate off background. For a neutron detector this may be needed to adjust thresholds. For an analysis module to collect more background. For occupancy sensors this may be to check beam alignment.

DeviceWouldLikeActiveMaintenanceTimeFlag Value 0x800

Indicates the device would like dedicated active maintenance time, but can continue to operate with out it.

See also: MeasurementType::ActiveMaintenanceValue

DeviceNeedsActiveMaintenanceTimeFlag Value 0x1000

Indicates the device can not operate until it get some dedicated active maintenance time.

See also: MeasurementType::ActiveMaintenanceValue

4.1.22 OperatingMode Enumeration

Enumeration of operating modes of a device, or system.

Note that these states are used to both control the entire system, as well as each device. Also note that devices may make transitions even when not requested. Examples: when the required dwell duration is over, a detector will ordinarily automatically transition from OperatingMode::OperatingValue to OperatingMode::ReadyValue. A detector might make an un-requested transition when there is a hardware error; this transition may-or-may-not cause the entire system to transition (although a system should try to keep operating if possible).

See also:

- ChangeDeviceStateRequest
- ChangeDeviceStateReply

- CmdSystemStateChangeRequest
- CmdSystemStateChangeReply
- DeviceFeaturesFlags

Used in message: DeviceStatusReply

Underlying integral representation: uint8_t

Enumerated values for OperatingMode:

StandByValue Value 0x01

Default operating mode to which the device should "boot up". If the device has high voltage, bias voltage supplies, or similar potentially hazardous or high energy consuming devices that can be switched on or off, these should be turned OFF in this state. This does not apply to things like HPGe crystal coolers where there is a compelling reason to always have operating. Devices (or systems) will not transition out of this state unless requested. Must be supported by all devices.

ReadyValue Value 0x02

In this operating mode, the device is not collecting data, or otherwise performing its duty, but is ready to. For detectors this means that high voltages are on, and bias voltages applied, and data collection can quickly start when requested. For archive or analysis devices, all necessary resources should be initialized. Devices (or systems) will not transition out of this state unless there is an error, or they are specifically requested to. Must be supported by all devices.

OperatingValue Value 0x04

In this operating mode, the device should be collecting data, or otherwise performing its function. When a ChangeDeviceStateRequest message is sent, to the device from the control module, with this as the operating mode to achieve, a ChangeDeviceStateReply message with a ChangeDeviceStateReply::status_of_tr equal to CommandReplyStatus::CompletedValue (assuming everything went okay and there were no errors) will be sent back from the device to the control module before sending any data that are collected while in this operating mode. If the device is not already in OperatingMode::ReadyValue before receiving the command to transition to OperatingMode::OperatingValue, then it may take a while for the high voltage, or other mechanisms, to stabilize and allow operations, in which case ChangeDeviceStateRequest messages should be sent to update the control module of the progress towards starting operations. Devices should only transition out of this state when requested to, when there is an error, or a dwell (during a DataCollectionModes::LiveTimeDwellValue or DataCollectionModes::RealTimeDwellValue mode collection) finishes. Must be supported by all devices.

4.1.23 MeasurementType Enumeration

Enumeration of the types of measurement that may be performed; applies to both the system level and the module level.

See also:

- MeasurementTypeChangeRequest
- MeasurementTypeChangeReply
- CmdMeasurementTypeChangeRequest
- CmdMeasurementTypeChangeReply

Used in message: DeviceStatusReply

Underlying integral representation: uint8_t

Enumerated values for MeasurementType:

NotSpecifiedValue Value 0x00

In messages originating in the control module, the control module is unable to determine what is being measured, whether from information generated within the portal, or from an external authority. In a message originating in a command device relating to a request for a change of state, no request is being presented for measurement type.

ItemValue Value 0x01

There is an item of interest being measured. For a RPM this might mean that there is a vehicle either in the vicinity or actually in the portal (i.e., an occupancy is in progress) or there is some other external information, as provided by a Command device, that indicates the system should function as if there is a vehicle in it.

BackgroundValue Value 0x02

The device should consider that the background is being measured.

PossibleInterferingSourceValue Value 0x03

There is potentially a source present, like a calibration source, or cross-talk from a known vehicle in a nearby lane, etc, that may be affecting the data in a manner not representative of a vehicle occupying the portal. Basically a way of saying the data shouldn't be considered background and any analysis results should take this into account.

ActiveMaintenanceValue Value 0x04

This indicates that the device is free to perform actions which may affect operation of this device, or of other devices or the system. An example would be a gamma module with a calibration source inside of a Pb pig which, during active maintenance, is removed from the pig to allow calibration. Or, the analysis algorithm can take this time to re-index its database and not report results. If possible, devices should still send data to the control module during this time if they are in the OperatingMode::OperatingValue state, but this isnt a hard requirement since some operations may prevent this. When done performing active maintenance, the device should transition back to OperatingMode::ReadyValue (and notify the control module via a DeviceStatusPush message). If the device's active maintenance might possibly effect operations of other devices within the portal, the DeviceFeaturesFlags::DeviceActiveMaintenanceMayEffectOtherDevicesFlag flag must be set in DeviceInfoReply::device_features. When a device has completed active maintenance, it should transition the MeasurementType and send a DeviceStatusPush indicating the change. When an entire system is put into this mode, it is up to the control module to ensure different modules arent stepping on, or messing up others. An example would be to only have one detector put a calibration source out of their pigs at a time.

4.1.24 DeviceStatusReply Message

Gives the replying device's current operating status in response to a DeviceStatusRequest request

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x86 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
status_timestamp	UtcTimePoint *Field Description*: The timestamp of when this status indicated took effect. That is, the timestamp of the last event which caused the reported status to change. For example, if the last reportable thing to happen was the high voltage turned on, then this field would indicate that time.
device_status_flags	uint32_t bits defined by DeviceStatusFlags *Field Description*: Bitwise OR of flags representing target device status.
operating_mode	uint8_t enumerated by OperatingMode *Field Description*: The operating mode that the device is currently in.
data_collection_mode	uint8_t enumerated by DataCollectionModes *Field Description*: The data collection mode that the device is currently in. Set to DataCollectionModes::NoOriginateValue if the module is in any state other than of DeviceStatusReply::operating_mode OperatingMode::OperatingValue
measurement_type	uint8_t enumerated by MeasurementType *Field Description*: The type of measurement being taken (item of interest, background, not specified, possible interfering source, or active maintenance).
collection_interval_ms	uint32_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The data collection interval is specific to the data collection mode, and zero if not applicable. For information on allowed intervals, see the DataCollectionModes enumeration. For DataCollectionModes::RealTimeDwellValue and DataCollectionModes::LiveTimeDwellValue modes, this interval specifies the entire duration of the measurement, and for the other modes it specifies the interval between sending data or HeartbeatPush (if non-zero) messages.	

4.1.25 DeviceBufferStatusRequest Message

Message from the control module requesting the status of buffering by the device, including the number of buffered messages and availability of space for buffering.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x07
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.1.26 BufferStatusFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more conditions regarding buffered data.

Underlying integral representation: uint8_t

Enumerated values for BufferStatusFlags:

DeviceDoesNotSupportsBufferingFlag Value 0x01

Device does not support buffering. If this bit is set, then DeviceInfoReply::device_features should not have the DeviceFeaturesFlags::DeviceSupportsBufferingFlag bit set. Useful for when a DeviceBufferStatusReply gets sent to a device that does not support buffering.

DeviceHasBufferedDataFlag Value 0x02

Device has not obtained acknowledgment that all data sent was received, so it has some messages in its buffer.

DeviceBufferHasOverflowedFlag Value 0x04

Device had to buffer more data than it had room for, so some messages were lost.

4.1.27 DeviceBufferStatusReply Message

Reply from a device to a DeviceBufferStatusRequest from the control module.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x87
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`timestamp`	UtcTimePoint
Field Description: Timestamp of this reply message.	
`buffer_status_flags`	`uint8_t` bits defined by `BufferStatusFlags`
Field Description: Bitwise OR of BufferStatusFlags.	
`num_buffered_messages`	`uint32_t`
Field Description: The number of (possibly only approximate) buffered messages. If `BufferStatusFlags::DeviceHasBufferedDataFlag` bit in `DeviceBufferStatusReply::buffer_status_flags` is set, then this value should be non-zero.	
`buffer_free_room`	`uint64_t`
Field Description: If the device contains buffering capabilities, this indicates the (possibly only approximate) number of free bytes in the buffer. If the `BufferStatusFlags::DeviceHasBufferedDataFlag` bit is not set in `DeviceBufferStatusReply::buffer_status_flags` then this value should be the same as `DeviceInfoReply::buffer_size`. This must be zero if the device does not have buffering capability.	

4.1.28 DeviceStatusPush Message

Message sent to the control module by a device when there is a non-requested change made by the device to its status flags or operating state.

For example, if a components temperature reaches a pre-defined warning threshold causing the DeviceStatus::device_status_flags to have the DeviceStatusFlags::DeviceHasWarn bit be set. Note that in this case a ParameterUpdatePush message would also be sent corresponding to the temperature change. If there is a hardware, or other error, a NotificationPush message may be attached to the message in order to provide an English description of the issue. When DataCollectionModes::LiveTimeDwellValue or DataCollectionModes::RealTimeDwellValue measurements are finished, this message will also be sent to the control module by the device.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x19 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
status_timestamp	UtcTimePoint *Field Description*: The timestamp of when this status indicated took effect. That is, the timestamp of the last event which caused the reported status to change. For example, if the last reportable thing to happen was the high voltage turned on, then this field would indicate that time.
device_status_flags	uint32_t bits defined by DeviceStatusFlags *Field Description*: Bitwise OR of flags representing target device status.
operating_mode	uint8_t enumerated by OperatingMode *Field Description*: The operating mode that the device is currently in.
data_collection_mode	uint8_t enumerated by DataCollectionModes *Field Description*: The data collection mode that the device is currently in. Set to DataCollectionModes::NoOriginateValue if the module is in any state other than of DeviceStatusPush::operating_mode OperatingMode::OperatingValue
measurement_type	uint8_t enumerated by MeasurementType

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The type of measurement being taken (item of interest, background, not specified, possible interfering source, or active maintenance).	
`collection_interval_ms`	`uint32_t`
Field Description: The data collection interval is specific to the data collection mode, and zero if not applicable. For information on allowed intervals, see the `DataCollectionModes` enumeration. For `DataCollectionModes::RealTimeDwellValue` and `DataCollectionModes::LiveTimeDwellValue` modes, this interval specifies the entire duration of the measurement, and for the other modes it specifies the interval between sending data or `HeartbeatPush` (if non-zero) messages.	

4.1.29 DeviceStatusPushAck Message

Message sent by the control module to the device that sent a `DeviceStatusPush` message, acknowledging receipt of the push message.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x99
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.1.30 DeviceTimeStatisticsRequest Message

A request to get the device time statistics, such as uptime, total use time and similar.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x0d

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.1.31 DeviceTimeStatisticsReply Message

A reply to DeviceTimeStatisticsRequest which contains the time statistics of the device, such as current uptime, current connection duration, and total device on and uptime.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x8d
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`timestamp`	UtcTimePoint
Field Description: The time at which these statistics were calculated. Microseconds since the UNIX epoch.	
`current_uptime_seconds`	`uint32_t`
Field Description: The time, in seconds, since the device has been started or restarted. Similar to 'uptime' command in Linux and BSD.	
`current_session_time_seconds`	`uint32_t`
Field Description: For non-control modules, the number of seconds the current RAPTER WebSocket session has been established for the device. For the control module, the amount of time, in seconds, continuously available, up to the present, to accept WebSocket connections.	

Continued on next page

Table continued from previous page

Field Name	Data Type
current_session_time_operating_seconds	uint32_t
Field Description: For non-control modules, the number of seconds the device has been in OperatingMode::OperatingValue during the current RAPTER WebSocket session. Must be no greater than DeviceTimeStatisticsReply::current_session_time_seconds, i.e., even if the device was operating when session began, because buffering was enabled, this value should start accumulating at the beginning of the current session, not when data acquisition actually started. For the control module, then this should be the longest uninterrupted time that any connected device has been in OperatingMode::OperatingValue.	
current_operating_mode_time_seconds	uint32_t
Field Description: For devices, the number of seconds since the most recent change of the device's OperatingMode. May be larger than DeviceTimeStatisticsReply::current_session_time_seconds if not a control module and the device was previously in the current state before the RAPTER WebSocket connection was started. For the control modules, the time since DataOutSystemStateChangePush::requested_system_operating_mode has been changed, or if it hasn't been changed, the time since system start up.	
cumulative_time_on_seconds	uint32_t
Field Description: The number of seconds this device has been powered on. This value should monotonically accumulate across power cycles, firmware upgrades, and if at all possible, hardware upgrades.	
cumulative_time_operating_seconds	uint32_t
Field Description: The number of seconds this device has been powered on and in the OperatingMode::OperatingValue. This value should monotonically accumulate across power cycles, firmware upgrades, mode changes, and if at all possible, hardware upgrades.	
number_restarts	uint32_t
Field Description: Number of times, at the application level, the device has restarted. That is, if the device is implemented as a software program application running within an operating system, this number would be incremented each time the application is started by the operating system, wether due to it crashing, or the device power cycling. If the device is implemented as a real time operating system, Unikernel, or dedicated hardware, this number would increment each power cycle. This value should monotonically accumulate across power cycles, firmware upgrades, and if at all possible, hardware component upgrades.	
number_websocket_connections_initiated	uint32_t
Field Description: If a device: cumulative number of WebSocket connection it has attempted to initiate over its lifetime, whether the connection was successful or not. If a control module: cumulative number of WebSocket connections devices have attempted to make to it, whether the connection was successful or not. This value should monotonically accumulate across power cycles, firmware upgrades, and if at all possible, hardware component upgrades.	

Continued on next page

Table continued from previous page

Field Name	Data Type
number_completed_handshakes	uint32_t
Field Description: The number of times a WebSocket connection resulted in at least negotiating the message group. This value should monotonically accumulate across power cycles, firmware upgrades, and if at all possible, hardware component upgrades.	

4.1.32 HeartbeatPacket Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Struct to hold information about elapsed real and live time for a measurement interval. See `HeartbeatPush` for further information.

Message Contents:

Field Name	Data Type
start_timestamp	UtcTimePoint
Field Description: Timestamp indicating the start of the interval this heartbeat is for. If part of a `HeartbeatPush` message, must be equal to `HeartbeatPush::start_timestamp`.	
end_timestamp	UtcTimePoint
Field Description: Timestamp indicating the point in time all data (radiation, vehicle presence, parameter measurements, etc) has been transferred up through; if part of a `HeartbeatPush` message, must be equal to `HeartbeatPush::end_timestamp`. The value will nominally be the last microsecond of the time-interval, and so a listmode events sent before this heartbeat will have timestamps less than or equal to this timestamp, and listmode events sent after this heartbeat will have timestamps greater than this value.	
subdetector_number	uint8_t
Field Description: The sub-detector this information corresponds to. For Analysis, DataOut, and Command devices this must have a value of zero (e.g., to represent the entire device).	
interval_clock_time_seconds	float
Field Description: The elapsed clock time, in seconds, covered between the previous `HeartbeatPush` and this one. If this is the first `HeartbeatPush` message of the current state, then this is the time since the change in operating mode (e.g., since the effective time of `ChangeDeviceStateReply` or `DeviceStatusPush`).	
interval_live_time_seconds	float

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Similar to HeartbeatPacket::interval_clock_time_seconds, except instead of the clock time, this field gives the time the device has been available to perform its primary function. An example of when this field may not match HeartbeatPacket::interval_clock_time_seconds is for gamma detectors, there is often a time period after detecting a gamma event, where another gamma event can not be detected (often referred to as 'dead time'), this period would not contribute to HeartbeatPush::interval_live_time_seconds. For devices such as analysis modules, archiving systems, or human interaction devices that are available to perform their duty 100% of the time, then this field will match HeartbeatPacket::interval_clock_time_seconds if there are no system clock updates.	

4.1.33 HeartbeatPush Message

A heartbeat sent at regular intervals, by devices in their operational state OperatingMode::OperatingVa and in DataCollectionModes::OnEventValue or DataCollectionModes::NoOriginateVa modes, if the control module instructed the device to send these heartbeats (see ChangeDeviceStateRequest::collection_interval_ms) at regular intervals.

This message indicates all previously generated 'push' messages have been sent. The purpose of this message is to allow synchronizing of data more easily from instruments collecting list-mode data with those collecting in the DataCollectionModes::ClockTimeIntervalValue mode, so data can be packaged by the control module and sent off to the Analysis/DataOut/-Command devices. For example, if gamma sensors collect data every 100ms, and neutron detectors collect list mode data with a low count rate (1 count every 10 seconds), without this heartbeat the control module would have to wait a while (there may be a network issue or congestion) to be sure the neutron detector wasn't going to send the data, before it could forward the data to the other devices. With this heartbeat, it will usually not have to wait to know there were zero detected counts (unless there really is a network congestion/issue).

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x1b
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`start_timestamp`	UtcTimePoint
Field Description: Timestamp indicating the start of the interval this heartbeat is for.	
`end_timestamp`	UtcTimePoint
Field Description: Timestamp indicating the point in time all data has been transferred up through; may be slightly larger than time scheduled for the heartbeat to be sent (maybe due to CPU congestion), but not smaller.	
`infos`	`Short Array` of `HeartbeatPacket`
Field Description: Must contain one `HeartbeatPacket` for each sub-detector (see `DeviceInfoReply::num_subdetectors`). Note that DataOut and Command devices, as well as the analysis module, will have exactly zero entries, since they must have `DeviceInfoReply::num_subdetectors` set to zero. For radiation detectors, vehicle presence modules, and power management modules there will be an entry for each detection element.	

4.1.34 HeartbeatPushAck Message

A message acknowledging the receipt by the control module of a `HeartbeatPush` message. This message is optional to send if buffering is not enabled.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x9b
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.1.35 BufferRecoveryMode Enumeration

Enumeration to represent how recovery should happen; either Serial (transmit all messages in the buffer then start transferring live data), or Parallel (transmit live data in pseudo-realtime while also transmitting buffered messages).

Used in message: BufferedMessagesRequest

Underlying integral representation: uint8_t

Enumerated values for BufferRecoveryMode:

SerialValue Value 0x00

First transmit all of the requested buffered messages, then start transmitting live messages; this may involve buffering the live data while transferring the buffered data.

ParallelValue Value 0x01

Transmit the live data while also transmitting the buffered data. It is up to the sending device to monitor the data transmit speed (which may be limited due to the TCP congestion resolution) and insure the "live" data isnt sitting in a queue (the websockets libraries queue, the OSes TCP queue, etc) behind a bunch of the buffered data. That is, sending current data should be prioritized ahead of sending the buffered data. Also note, if the connection fails before the full retrieval of the buffered data, the data taken after failing will be appended to the queue, with no delineation of which data was before which disconnect (although see BufferedMessagesReply::newest_recovered_msg_id).

4.1.36 BufferedMessagesRequest Message

Request sent to a device for buffered data, for example to recover after a dropped connection.

Note that messages that have been buffered should have the MsgFlags::MessageHasBeenBufferedF flag set in the message header. When a connection is dropped during data collection, the sensor device should keep in its current state (e.g., continue collecting data), and upon re-connection, it should not send data messages (or any message with an Ack reply) until the control module either sends a ChangeDeviceStateRequest message (at which point the device will clear its buffer but can then send new data in real time, even if the message didn't actually change the operational state), or a BufferedMessagesRequest message requesting to send the buffered messages.

See Also: BufferStatusFlags::DeviceHasBufferedDataFlag, MsgFlags::MessageHasBe

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x08
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
recovery_mode	uint8_t enumerated by BufferRecoveryMode
Field Description: Whether live messages should be sent serially after the buffered messages, or in parallel.	

4.1.37 BufferedDataRequestStatus Enumeration

Enumeration to represent the possible outcomes of buffered data requests from a device or module.

Used in message: BufferedMessagesReply

Underlying integral representation: uint8_t

Enumerated values for BufferedDataRequestStatus:

NotSupportedValue Value 0x00

Buffering is not supported.

NoBufferedContentsValue Value 0x01

There are no buffered messages to be sent.

WillSendValue Value 0x02

Buffered messages are present, but remain to be sent.

SendFailedValue Value 0x03

An attempt was made to send buffered messages, but failed

SendAlreadyInProgressValue Value 0x04

Buffered messages are in the process of being transmitted, but the last buffered message has not been sent.

SendCompletedSuccessfullyValue Value 0x05

The last buffered message has been transmitted to the receiving device.

4.1.38 BufferedMessagesReply Message

Message sent in response to an BufferedMessagesRequest request; multiple instances of these messages will be sent in response to the original request message. One of these messages will be sent before re-sending the buffered data, and one of these messages will be sent once sending of the buffered data is complete. However there may be other of these messages sent, if, for instance, there is a failure in re-sending the data, the original message will be followed up to indicate the failure.

If the device is able to provide the buffered data, it should return this message followed by the buffered data. If recovery mode is `BufferRecoveryMode::SerialValue`, then data currently being taken should not be sent after the first `BufferedMessagesReply` message until the last currently buffered message is sent; this applies to any "push" message. Processing of commands which do not fall into the buffering scheme (so do not have a "Push"/"Ack" in the message or its reply) can proceed even while the recovery (i.e. the communications 'catch up') is still happening. Note that if any buffer overflow (causing discarded messages) happens during this catch up an `DeviceStatusPush` update message should be sent to indicate this. Upon successful completion of the buffer transfer a `BufferedMessagesReply` message should be sent with the `BufferedMessagesReply::buffered_status` field set to `BufferedDataRequestStatus::SendCompletedSuccessfullyValue`. Note that if the detector is currently taking data, in Serial mode the buffered data will precede current data (which will follow), while in Parallel mode the current data will begin to be sent immediately. Note: upon reconnection occurring during data taking, the data currently being acquired and the buffered data will not be sent until requested; or, if a change of state is requested, the buffer will be cleared.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x88
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`buffered_status`	`uint8_t` enumerated by `BufferedDataRequestStatus`
Field Description: Status of the buffering operation.	
`oldest_recovered_msg_id`	`uint32_t`
Field Description: The message ID of the oldest message that will be, or was sent. If `BufferedMessagesReply::buffered_status` is `BufferedDataRequestStatus::NotSupportedValue`, `BufferedDataRequestStatus::NoBufferedContentsValue`, or `BufferedDataRequestStatus::SendAlreadyInProgressValue`, then this variable should be ignored.	
`newest_recovered_msg_id`	`uint32_t`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: When BufferedMessagesReply::buffered_status is BufferedDataRequestStatus::WillSendValue, then this is the most recent message in the buffer. When BufferedMessagesReply::buffered_status is BufferedDataRequestStatus::SendFailedValue, this is the most recent message to enter the buffer, or zero if none. If status is BufferedDataRequestStatus::SendCompletedSuccessfullyValue then this was the last message sent from the buffer. For all other BufferedMessagesReply::buffered_status values this variable should have a value of zero and be ignored. Note that if BufferRecoveryMode::SerialValue option was selected, there may be subsequent messages after this ID that are marked with the MsgFlags::MessageHasBeenBufferedFlag bit, and the final BufferedMessagesReply sent after buffer has been emptied will have a BufferedMessagesReply::newest_recovered_msg_id indicating the last message transferred before the buffer was emptied (so might be data taken after the buffer recovery was begun).	
number_failed_messages	uint32_t
Field Description: Indicates the number of buffered messages that either will not, or were not recovered. When BufferedMessagesReply::buffered_status is BufferedDataRequestStatus::WillSendValue, this field indicates the number of messages that are known to not be able to be recovered, for example because the buffer overflowed. When BufferedMessagesReply::buffered_status has a value BufferedDataRequestStatus::SendFailedValue or BufferedDataRequestStatus::SendCompletedSuccessfullyValue this variable indicates the number of message that were not able to be sent. For other values of BufferedDataRequestStatus, this value must be zero.	

4.1.39 BufferingSetOptionRequest Message

Message sent from the control module to tell a device to either not bother buffering data during disconnects, or to buffering back on; there is also an option to clear the current buffer.

Note that if a device supports buffering, it should default to not having the buffering mechanism on, until told otherwise.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x09
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`set_buffering_enabled`	`uint8_t`
Field Description: If nonzero, set buffering to enabled. If zero, disable buffering.	
`clear_buffer`	`uint8_t`
Field Description: A non-zero value indicates a request to clear any pending messages in the buffer. Note that if recovery of buffered messages is under way, and this is set to true, then the behavior is undefined.	

4.1.40 BufferingSetOptionsStatus Enumeration

Enumeration to represent results of a request to set buffering options.

Used in message: `BufferingSetOptionReply`

Underlying integral representation: `uint8_t`

Enumerated values for `BufferingSetOptionsStatus`:

SuccessValue Value 0x00

The requested operation(s) were successful.

BufferingNotSupportedValue Value 0x01

Device doesn't support buffering.

CouldNotClearBufferValue Value 0x02

Buffer couldn't be cleared; Possible causes include: buffered data is being transferred currently, network conditions are causing new events to be put into the buffer, or something else.

OtherErrorValue Value 0x03

Some other error occurred.

4.1.41 BufferingSetOptionReply Message

Message acknowledging, and giving timestamp of message buffer options being set.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_type	uint8_t with value 0x89
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
status	uint8_t enumerated by BufferingSetOptionsStatus
Field Description: Status of the request	
timestamp	UtcTimePoint
Field Description: The time at which the data buffer was either turned on, off, or emptied.	
num_messages	uint64_t
Field Description: The number of messages in the buffer that are currently in the buffer, or were cleared (depending on request, and success). If accurately counting the messages is impractical, this may be an estimate; a value of zero should indicate exactly zero messages were buffered.	
num_bytes	uint64_t
Field Description: The number of bytes in the buffer currently, or before it was cleared, depending on request and success. If accurately counting the number of bytes is impractical, this may be an estimate; a value of zero should indicate exactly zero bytes were buffered.	

4.1.42 PingRequest Message

A message sent from the control module to a device to check connection status, and possibly gain timing delay information.

Note that there is no requirement that the system should process this command in any given order (either before or after previous commands).

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x0a
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
timestamp	UtcTimePoint
Field Description: Microseconds since the Unix epoch. This timestamp should be calculated at the application level, immediately before passing the message off to the networking stack, so that the receiver of the message may use it to gain information regarding timing delays present. A new timestamp will be provided on the reply from the original receiver as well.	

4.1.43 PingReply Message

Reply to a PingRequest.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x8a
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
timestamp	UtcTimePoint
Field Description: This timestamp should be taken immediately before passing the message off to the networking stack since the control module may use this message pair as a coarse estimate of time offset accuracy.	

4.1.44 PowerDownOption Enumeration

Enumeration to indicate options for the behavior of a PowerDownRequest.

Used in message: PowerDownRequest

Underlying integral representation: uint8_t

Enumerated values for PowerDownOption:

RebootSystemValue Value 0x01

Restart the the system.

PowerOffValue Value 0x02

Power off the system. E.g. make it safe to work on or unplug the device.

4.1.45 PowerDownRequest Message

A request to a device to PowerDown.

Useful to attempt to resolve transient issues. The device should start acting as soon as it receives this command. This message must be replied to before the WebSocket connection disconnects; note that the WebSocket close code (see RFC 6455 7.4.1) should also be properly selected.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00 *Field Description*: The RAPTER `MessageGroup` this message corresponds too.
`message_type`	`uint8_t` with value 0x0b *Field Description*: The type of message within the specified message_group, that this message corresponds to.
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
`power_option`	`uint8_t` enumerated by `PowerDownOption` *Field Description*: The option to reboot or power off.

4.1.46 PowerDownStatus Enumeration

Enumeration to indicate potential outcomes of a `PowerDownRequest`.

Used in message: `PowerDownReply`

Underlying integral representation: `uint8_t`

Enumerated values for `PowerDownStatus`:

PowerDownProceedingValue Value 0x00

PowerDown is proceeding; you should expect to lose connection momentarily.

PowerDownFailedValue Value 0x01

Something is preventing a PowerDown; a notification should be attached describing the issue.

4.1.47 PowerDownReply Message

The reply to the PowerDownRequest message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x8b
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
power_status	uint8_t enumerated by PowerDownStatus
Field Description: The power status that the device is about to implement.	

4.1.48 SendLogsRequest Message

A request for a device to send its internal log to the control module, if the device has some form of internal log. The format, information content, or existence of a log is not specified and would be expected to be device specific. This is likely to be utilized for technician maintenance or diagnosis of device behavior.

See Also: LogReplyStatus , SendLogsReply

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x0c
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`start_time`	UtcTimePoint
Field Description: The desired start time of the log to be returned. Microseconds since the UNIX epoch. The device can use or not use this time at its discretion, so it should be treated as a preference of the requester and not absolute. A value of zero indicates that the device should send log information as far back as is possible.	
`end_time`	UtcTimePoint
Field Description: The desired end time of the log to be returned. Microseconds since the UNIX epoch. The device can use or not use this time at its discretion, so it should be treated as a preference of the requester and not absolute. A value of zero indicates the device should return log information up to the current time.	
`max_log_size_bytes`	`uint32_t`
Field Description: The maximum size of the log that should be transferred, in bytes. This is used to avoid sending large volumes, e.g. a 10 TB database, while in operation, which could potentially hamper necessary network traffic. A value of zero indicates no limit, however the device may choose to impose a limitation on log size.	

4.1.49 LogReplyStatus Enumeration

Enumeration of values to represent the status of retrieving a device's internal log.

See also:

- `SendLogsRequest`
- `SendLogsReply`

Used in message: `SendLogsReply`

Underlying integral representation: `uint8_t`

Enumerated values for `LogReplyStatus`:

LogRetrievedSuccessValue Value 0x00

The log is being returned successfully.

LogDoesNotExistValue Value 0x01

The device does not keep a log.

LogTimePeriodSpecifiedInvalidValue Value 0x02

An invalid time period was specified, e.g., end_time before start_time.

LogNoLogForSpecifiedPeriodValue Value 0x03

A log for the requested time period is not defined.

LogTruncatedValue Value 0x04

The log is truncated due to the maximum log size.

4.1.50 SendLogsReply Message

The response to an SendLogsRequest request that includes the device's internal log if supported.

See Also: SendLogsRequest , LogReplyStatus

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x8c
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
reply_status	uint8_t enumerated by LogReplyStatus
Field Description: The success status of request.	
start_time	UtcTimePoint
Field Description: The start time of the returned log, if it is known. Microseconds since the UNIX epoch. This does not have to be the same as the request specified. A value of zero indicates that the log start time is unknown.	
end_time	UtcTimePoint
Field Description: The end time of the returned log, if it is known. Microseconds since the UNIX epoch. This does not have to be the same as the request specified. A value of zero indicates that the log end time is unknown; it is likely that this would be the current time.	
log_data	Large Array of uint8_t
Field Description: The log data from the device - intended to be displayed in text editor, or similar type of display. This is intended to be the equivalent of a text file, so if a specific encoding is used, the data may begin with the appropriate byte order mark (BOMs).	

4.1.51 DeviceOperabilityCheckRequest Message

A request from the control module for a device to report its ability to transition to the Operating mode, or if there is something that would block the transition. This is not a request to actually make the transition.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x0e
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.1.52 DeviceOperabilityStatus Enumeration

Description of a devices operability status.

Used in message: DeviceOperabilityCheckReply

Underlying integral representation: uint8_t

Enumerated values for DeviceOperabilityStatus:

DeviceIsOperating Value 0x00

Device is currently operating.

DeviceIsPreparingToOperate Value 0x01

Device is preparing to operate. For example if if the device is still initializing resources after starting up, or performing some self-tests.

DeviceIsNotOperatingButCan Value 0x02

The device is not currently in the operating state, but there are no known issues that will prevent it from operating if asked.

DeviceCanNotOperate Value 0x03

There is an issue and the device can not operate.

4.1.53 DeviceOperabilityCheckReply Message

A device's reply informing the control module of its ability to transition to the DeviceOperabilityStatus: mode. If there is something that would block the transition, the device may also send a NotificationPush message (referencing this message id) to explain more details.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x8e
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
operability_status	uint8_t enumerated by DeviceOperabilityStatus
Field Description: Indication of if the device is in, or can be brought to the operational state.	
estimate_time_to_operating_ms	uint32_t
Field Description: Estimated time to resolve any issues that are preventing operability, or to transition to operating, if known; zero if the system is already operating, or no estimate is available.	

4.1.54 SupportedDataCollectionIntervalsRequest Message

Request to check which data collection intervals a device supports. A use case for this message is if the control module wants to make sure a detector supports a 100ms interval for taking DataCollectionModes::ClockTimeIntervalValue data. If the control module a-priori knows the valid intervals of a certain device, it is not required to check with the device. Also if a device is requested to transition to the operation mode with an invalid interval, the device will refuse to do this and will be indicated as such in the ChangeDeviceStateReply returned to the device. For data taking modes such as DataCollectionModes::OnEventValue (listmode data) where there is no natural interval for the data, this message checks for the intervals at which heartbeats can be sent.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x0f
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`data_collection_mode`	`uint8_t` enumerated by `DataCollectionModes`
Field Description: Data collection mode being inquired about.	

4.1.55 SupportedDataCollectionIntervalsType Enumeration

Enumeration to convey the different ways a detector may list its valid intervals for a given data collection mode.

Used in message: `SupportedDataCollectionIntervalsReply`

Underlying integral representation: `uint8_t`

Enumerated values for `SupportedDataCollectionIntervalsType`:

NoIntervalRestrictionsValue Value 0x00

The device can use any interval allowed by the RAPTER Interface Specification. There are no further limitations imposed by the device. For example this for a data collection mode of `DataCollectionModes::ClockTimeIntervalValue`, the device would support 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, and 1000 milli-second intervals. For `DataCollectionModes::LiveTimeDwellValue` or `DataCollectionModes::RealTimeDwellValue` this option would imply the device supports anything between 1 ms and 2^{}32-1 milliseconds (a `uint32_t` is used to specify time interval period). This option should be specified for `DataCollectionModes::OnEventValue` and `DataCollectionModes::NoOriginateV`

ListOfIntervalsValue Value 0x01

A list of valid intervals, in ms, will be returned by the device.

MinMaxIntervalValue Value 0x02

Two values will be returned that specify the minimum and maximum time intervals (inclusive) that can be used.

InvalidDataCollectionModeValue Value 0x03

The data collection mode being queried isn't supported.

4.1.56 SupportedDataCollectionIntervalsReply Message

Message that specifies valid intervals for data collection that the device supports. For information on allowed intervals, see the DataCollectionModes enumeration.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x8f *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
data_collection_mode	uint8_t enumerated by DataCollectionModes *Field Description*: Data collection mode these limits are applicable to.
time_interval_type	uint8_t enumerated by SupportedDataCollectionIntervalsType *Field Description*: How the limitation is specified. If SupportedDataCollectionIntervalsReply::data_collection_mode is not supported by the device, then this field must be set to SupportedDataCollectionIntervalsType::InvalidDataCollectionModeValue, and no time periods returned. DataCollectionModes::NoOriginateValue must specify SupportedDataCollectionIntervalsType::NoIntervalRestrictionsV
time_intervals	Short Array of uint32_t *Field Description*: Time in milli-seconds of a data collection interval that the device supports. If SupportedDataCollectionIntervalsReply::time_interval_type has values SupportedDataCollectionIntervalsType::NoIntervalRestrictionsVa or SupportedDataCollectionIntervalsType::InvalidDataCollectionModeValue then no values will be provided. If SupportedDataCollectionIntervalsReply::time_inte has value SupportedDataCollectionIntervalsType::ListOfIntervalsValue then this field will be a list of supported intervals. If SupportedDataCollectionIntervalsReply::time_interval_type has value SupportedDataCollectionIntervalsType::MinMaxIntervalValue, then this field will have two entries. If SupportedDataCollectionIntervalsReply::data_c is DataCollectionModes::ClockTimeIntervalValue or DataCollectionModes::OnEventValue,

4.1.57 ChangeDeviceStateRequest Message

Message from the control module to a device requesting a change in the operating state of a the device.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x10
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
operating_mode	uint8_t enumerated by OperatingMode
Field Description: The requested new operating mode.	
data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: The requested new mode for collecting data. If transitioning to an operating mode other than OperatingMode::OperatingValue this value must be DataCollectionModes::NoOriginateValue (integer value 0). For transitions to OperatingMode::OperatingValue, the mode specified must be one the device supports, as specified by DeviceInfoReply::data_collection_modes or else the request will fail. Note that when a dwell measurement finishes (and OperatingMode automatically changes to OperatingMode::ReadyValue), the device will send a DeviceStatusPush to the control module, after it sends the data, to inform the control module of the device's new state.	
measurement_type	uint8_t enumerated by MeasurementType
Field Description: The type of measurement to perform or being performed (item of interest, background, not specified, possible interfering source, or active maintenance).	
collection_interval_ms	uint32_t
Field Description: Data collection interval. Specifies interval to send data (DataCollectionModes::ClockTimeIntervalValue, DataCollectionModes::LiveTimeDwellValue, or DataCollectionModes::RealTimeDwellValue), or HeartbeatPush messages (DataCollectionModes::OnEventValue or DataCollectionModes::NoOriginateValue, if value is non-zero). If transitioning to an operating mode other than OperatingMode::OperatingValue this value must be 0.	
requested_transition_time	UtcTimePoint

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Target time for the transition to occur, if non-zero. A value of zero, or any value less than the present indicates to perform the transition now. Otherwise, this is the target time for the transition to become complete, with the exception of transitioning out of `OperatingMode::OperatingValue`, in which case it is the time to stop operating and start transitioning to the state specified. When this transition time is specified, and it is in the future (relative to when the message is received), at least two `ChangeDeviceStateReply` messages will be sent back to the control module by the device. One to acknowledge the transition can occur), and the other to indicate completion or failure of transition. There may be more intermediate `ChangeDeviceStateReply` messages if the transition estimate needs updating for some reason (ex will take longer to stabilize the high voltage than is available when transitioning to operation). If you are taking a dwell, and you specify a time after the dwell ends, then the dwell measurement will not be effected; if you specify a time before the dwell ends, then the dwell will be cut short. This transition time should be considered a target time, as there may be limitations in hardware (ex. cooling down, bringing H.V. up, or timer accuracy) that prevent this time from being exact. If there is a currently pending request for a transition at a given time, and then another request for a transition is specified, the original pending transition is canceled. If a transition has been specified for the future, to cancel it, specify a transition to the current state, to which you will get a reply `CommandReplyStatus::CompletedValue` and whose `ChangeDeviceStateReply::previous_device_operating_mode` and `ChangeDeviceStateReply::requested_device_operating_mode` will be the current state. Note that the original transition message should still get a `ChangeDeviceStateReply` message with the `ChangeDeviceStateReply::status_of_transition` set to `CommandReplyStatus::CanceledValue`; this message should be sent before any replies to the second message (that does the canceling) is sent. If the value specified by `ChangeDeviceStateRequest::measurement_type` is different then the previous value, then it should be assumed that this change takes place at the transition time as well. When transitioning out of `OperatingMode::OperatingValue`, the `ChangeDeviceStateReply` message should be sent out before the (final) data.

4.1.58 CommandReplyStatus Enumeration

Indicates the status for a number of types of reply messages. Messages whose replies use this status may receive more than one reply; for things like turning a high voltage on which could take a while may first send a reply with status of `CommandReplyStatus::InProgressValue` to prevent a communications timeout, and then once the voltage is on and stabilized, another response to the original message will be sent with a status of `CommandReplyStatus::CompletedValue`.

Used in message: `ChangeDeviceStateReply`

Underlying integral representation: `uint8_t`

Enumerated values for `CommandReplyStatus`:

CompletedValue Value 0x01

Indicates that the requested action has been completed with no issues.

RedundantValue Value 0x02

Indicates that the requested action has no effect. An example would be if an ChangeDeviceStateRequest request with a corresponding OperatingMode::OperatingVa message was sent, but the device was already in the process of initializing or conducting data collection.

InProgressValue Value 0x04

Indicates that the request was received and is currently in the process of being completed. This must be sent when commands will maybe take longer than RapterConstants::REPLY_TIMEOUT_MS, so that the sender will not assume that the device has become un-operational, or is ignoring the request. If the final reply will take longer than an additional RapterConstants::FOLLOWUP_REPLY_TIMEOUT_MS milliseconds, then you must send further replies with a status of CommandReplyStatus::InProgre at least within RapterConstants::FOLLOWUP_REPLY_TIMEOUT_MS milliseconds or the control module will assume a timeout. An estimate of the time remaining may, or may not be included in the message to indicate when the device will be ready. Zero, one, or multiple messages with this status may be sent for each request.

CanceledValue Value 0x08

Indicates that the command which was in progress, or which was set to be done in the future, was canceled or superseded by the explicit dispatch of an interfering command. For example, an ChangeDeviceStateRequest command is sent with the operating mode field set to OperatingMode::ReadyValue, which will take 50 seconds to complete; during this 50 second window, another ChangeDeviceStateRequest message is sent with corresponding field set to OperatingMode::StandByValue. The first command will therefore be replied to with an ChangeDeviceStateReply message with the status field set to CommandReplyStatus::CanceledValue; this message may be sent before or after any replies to the second command. The second command will be sent replies as normal.

CompletedWithIssueValue Value 0x10

Indicates that the requested action has been completed, but that there may have been issues. A NotificationPush message must also be sent to provide additional information on the issue.

See also: NotificationPush

FailedValue Value 0x20

Indicates that the requested action could not be completed. A NotificationPush message must also be sent to provide additional information on the issue.

See also: NotificationPush

4.1.59 ChangeDeviceStateReply Message

Reply from device to a ChangeDeviceStateRequest, which updates the control module on the status of a requested transition, or any errors encountered.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x90
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
status_of_transition	uint8_t enumerated by CommandReplyStatus
Field Description: Status of the transition. Note that the device must send a message with status of CommandReplyStatus::CompletedValue or CommandReplyStatus::CompletedWithIssueValue after all data of the previous state have been sent and before any data of this new state are sent.	
previous_device_operating_mode	uint8_t enumerated by OperatingMode
Field Description: The operating mode of the device at the time that the request to change state was received.	
previous_device_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: The data collection mode of the device at the time that the request to change state was received.	
previous_device_collection_interval_ms	uint32_t
Field Description: The data collection interval of the device at the time that the request to change state was received.	
previous_measurement_type	uint8_t enumerated by MeasurementType
Field Description: The measurement type in effect on the device at the time that the request to change state was received.	
requested_device_operating_mode	uint8_t enumerated by OperatingMode
Field Description: The requested new operating mode for the device.	
requested_device_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: The requested new data collection mode for the device.	
requested_device_collection_interval_ms	uint32_t
Field Description: The requested new data collection interval for the device.	
requested_measurement_type	uint8_t enumerated by MeasurementType
Field Description: The requested new measurement type to be applied by the device.	
new_state_effective_timestamp	UtcTimePoint

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Time that the state change was completed (`CommandReplyStatus::CompletedValue` or `CommandReplyStatus::CompletedWithIssueValue`), or is predicted to be completed (`CommandReplyStatus::InProgressValue`), or failed (`CommandReplyStatus::FailedValue`), or was cancelled (`CommandReplyStatus::CanceledValue`), or a redundant request (`CommandReplyStatus::RedundantValue`) was received. If the status is `CommandReplyStatus::InProgressValue`, then this value may be zero if there is no estimate available for when the transition will finish.	

4.1.60 MeasurementTypeChangeRequest Message

Message sent from the control module to each connected device to indicate what is currently being measured. This is intended as a hint for the device on how to treat the data internally in case the device calibrates off data, or something similar.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x15
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`effective_timestamp`	UtcTimePoint
Field Description: Time when new `MeasurementTypeChangeRequest::measurement_type` went into effect. Microseconds since the Unix epoch. This must not be a future time, or a time before indicated by any previous `MeasurementTypeChangeRequest`. Any messages from the old `MeasurementType` must be sent before the `DeviceStatusPush` message is sent that reports compliance with the requested change of `MeasurementType`. Any messages originating from the new state must not precede this `DeviceStatusPush` message. If the control module is unable to specify the time (for example, when the system first starts up, or a module is just coming online and the control module implementation didn't keep the timestamp of the last transition) then this field may be zero (but this is discouraged) where the receiving device should interpret it as 'now'; if a future value is specified, the transition should be interpreted as 'now'.	

Continued on next page

Table continued from previous page

Field Name	Data Type
measurement_type	uint8_t enumerated by MeasurementType *Field Description*: Message sent by control module to change to a new measurement type.

4.1.61 MeasurementTypeChangeReply Message

An acknowledgement of receiving a MeasurementTypeChangeRequest.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x95 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
processed_timestamp	UtcTimePoint *Field Description*: Timestamp of when the MeasurementTypeChangeRequest was processed.

4.1.62 FirmwareUpgradeRequest Message

A request sent to a device to upgrade its firmware with the supplied blob.

The blob can be hashed and signed using the manufacturer choice of hashing scheme, as well as detached signature; these fields are optional and not using them does not necessarily imply insecurity as the blob may have a signature attached to the blob that both validates its integrity and origin. Should only be sent by devices that have the DeviceFeaturesFlags::DeviceSupportsFirmwareUpgradesFlag bit of DeviceInfoReply::device_features set.

The actual mechanism behind the upgrade is not specified. One potential example mechanism that could be used is: The blob could be a zip file that once unzipped contains a script that gets executed, as well as any files necessary for the script to do its job of upgrading the firmware. PGP could be used to sign the blob, with the appropriate public key pre-installed on the device, and MDA5 used to create the hash (these could also be contained within the zip file). There could be two separate OS volumes used such that the previous operating system or software is not re-written, but instead the newly upgraded firmware can be selected to be used. There could

also be a counting boot-loader, and other failure detection protections, that would roll-back the upgrade if the device fails to boot up more than X times, or fails to connect more than Y times after restarting - or it could restore the device back to the factory state (erasing all data).

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00 *Field Description*: The RAPTER `MessageGroup` this message corresponds too.
`message_type`	`uint8_t` with value 0x11 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
`blob`	`Large Array` of `uint8_t` *Field Description*: The actual binary data for the firmware upgrade. The mechanism by which upgrade takes place is not specified.
`blob_hash`	`Short Array` of `uint8_t` *Field Description*: Hash of the `FirmwareUpgradeRequest::blob` field. Ex. if a MDA5 hash is used would be a 16 byte value. The purpose of this field is to verify data integrity. Devices manufacturers can choose whether or not to use this field, but if they do, the hash will need to be supplied with the blob upgrade.
`signature`	`Medium Array` of `uint8_t` *Field Description*: The detached signature for the data to verify source of the update. An example of the use for this field would be to use gpg with a private key (ex: gpg –output blob.sig –detach-sig blob) to generate this signature, which the device could then use the previously installed public key to verify this `FirmwareUpgradeRequest::signature` of `FirmwareUpgradeRequest::blob` was made by an allowable source. Devices manufacturers can choose whether or not to use this field.

4.1.63 FirmwareUpgradeStatus Enumeration

Enumerations to describe the progress of a firmware upgrade.

Used in message: `FirmwareUpgradeReply`

Underlying integral representation: `uint8_t`

Enumerated values for `FirmwareUpgradeStatus`:

FirmwareUpgradeReceivedWillProcessValue Value 0x00

Use this enumeration to prevent a timeout of a firmware upgrade that is proceeding normally but is not yet complete.

FirmwareUpgradeNotSupportedValue Value 0x01

Status to send if the device does not support firmware upgrades.

FirmwareUpgradeInvalidStateToUpgradeValue Value 0x02

Status to send if the device is not in a state that would allow it to upgrade, for example, it is currently taking data. The device itself determines what state it must be in in order to accept updates. There is no mechanism to query the device about its requirements in this regard.

FirmwareUpgradeFailedIntegrityCheckValue Value 0x03

The hash of the upgrade blob failed its check.

FirmwareUpgradeInvalidSignatureValue Value 0x04

The signature was not valid for the given upgrade blob.

FirmwareUpgradeFailedUnpackValue Value 0x05

The upgrade blob had an unexpected format, so could not be unpacked. Ex. the device expected the blob to be a zip file, but it was not.

FirmwareUpgradeInvalidBlobValue Value 0x06

The upgrade was in some way not what the device required. Ex. an expected file was not in the output of unzipping the blob.

FirmwareUpgradeFailedOtherValue Value 0x07

The upgrade failed in some other way. The `FirmwareUpgradeReply::comment` will provide details.

FirmwareUpgradeInProgressValue Value 0x08

Notification that the device is working normally to install the upgrade.

FirmwareUpgradeDoneWillRebootValue Value 0x09

The upgrade has completed successfully, and the device will now disconnect, and then reconnect.

FirmwareUpgradeSuccesfulValue Value 0x0a

The upgrade was successful, can continue on functioning as normal now without disconnecting or rebooting.

4.1.64 FirmwareUpgradeReply Message

Reply from a device to a `FirmwareUpgradeRequest` from the control module.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
	Continued on next page

Table continued from previous page

Field Name	Data Type
`message_type`	`uint8_t` with value 0x91 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
`status`	`uint8_t` enumerated by `FirmwareUpgradeStatus` *Field Description*: The status of the firmware upgrade, whether final, waiting, or an error occurred.
`comment`	`Medium String` *Field Description*: Comments relating to the firmware upgrade.

4.1.65 ParameterNamesRequest Message

A request sent to a device for it to return a list of all parameters of the device has.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00 *Field Description*: The RAPTER `MessageGroup` this message corresponds too.
`message_type`	`uint8_t` with value 0x12 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.1.66 ParameterNameAndSubDetectorNumber Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Message Contents:

Field Name	Data Type
parameter_name *Field Description*:	Short String
subdetector_number *Field Description*:	uint8_t

4.1.67 ParameterNamesReply Message

A reply by a device to a ParameterNamesRequest message. Parameter names must consist of only ascii letters, numbers, underscores, period, dash, and space characters. The possible parameters present can not change after a WebSocket connection is established.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x92 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
parameter_names	Medium Array of ParameterNameAndSubDetectorNumber *Field Description*: Names of the parameters, as well as the sub-detectors they are associated with.

4.1.68 ParameterUpdateOption Enumeration

Option used by ParameterInfoRequest for how the measured value of a parameter should be updated.

Used in message: ParameterInfoRequest

Underlying integral representation: uint8_t

Enumerated values for ParameterUpdateOption:

DefaultValue Value 0x00

Vendor determines if the measured value needs to be read and updated or not before

replying to the ParameterInfoRequest.

MostRecentReadingValue Value 0x01

Device should return the most recent reading of the parameter, but should not bother to update its reading before replying to the ParameterInfoRequest, unless otherwise necessary to get the value.

UpdatedReadingValue Value 0x02

Device should read out the current value, if possible.

4.1.69 ParameterInfoRequest Message

Request to a device from the control module for information on a given parameter of the device.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x13
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
update_option	uint8_t enumerated by ParameterUpdateOption
Field Description: Hint to the device for how the measured value of a parameter should be updated. This option may be ignored for measured parameters, and is ignored for non-measured parameters. Also note that ParameterInfoReply does not have to say if this hint was obeyed or not, but should be inferred from the timestamp.	
parameter_name	Short String
Field Description: UTF-8 encoded name of the parameter.	
subdetector_number	uint8_t
Field Description: Subdetector	

4.1.70 ParameterInfoStatus Enumeration

Enumeration of validity status of a parameter reply message.

Used in message: ParameterInfoReply

Underlying integral representation: `uint8_t`

Enumerated values for `ParameterInfoStatus`:

ValidParameterNameValue Value 0x00

The requested parameter is a valid parameter.

InvalidParameterNameValue Value 0x01

The requested parameter is not a valid parameter.

4.1.71 ParameterValueDataType Enumeration

An enumeration of data types that can be used for parameters; all data types will be represented as a string, but this enum specifies how the string must be formatted.

Used in message: `ParameterInfoReply`

Underlying integral representation: `uint8_t`

Enumerated values for `ParameterValueDataType`:

BooleanValue Value 0x00

A boolean type. Value can only have the following representations: "true", "false", "0", "1"

IntegerValue Value 0x01

An integer, as could be represented by a `int64_t`. Leading zeros are prohibited, and representation is always in base-10. For example: '-1', '0', '2147483647', '+1000', '9223372036854775807'.

FloatValue Value 0x02

A IEEE 754 double precision `float`ing point number represented as text. However, like the rest of this specification, 'INF' and 'NaN' are prohibited. Examples include: '3.14', '-3E4', '1.0', '3.2E-9', '33.823E-1', '-0', '0'. See https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html

Utf8StringValue Value 0x03

A UTF-8 encoded string. Note: may be an empty string.

FloatingPointListValue Value 0x04

A space separated list of IEEE 754 double precision `float`ing point numbers represented as text. 'INF' and 'NaN' values are prohibited. Note: an empty list is valid.

IntegerListValue Value 0x05

A space separated list of 64-bit integers represented as text. Note: an empty list is valid.

4.1.72 HealthSeverityLevel Enumeration

Enumeration of a device's health assessment of a parameter in regards to quality of data and its impact on operation.

Used in message: `ParameterInfoReply`

Underlying integral representation: `uint8_t`

Enumerated values for `HealthSeverityLevel`:

NotApplicableValue Value 0x00

This parameter does not effect operation.

NoProblemValue Value 0x01

Value indicating that the value of this parameter is not impacting operation.

WarningNotAffectingDeviceOperationValue Value 0x02

The value of this parameter is a cause for concern, but not currently affecting quality of the primary operation of the module. Examples include: CPU temperature is higher than expected, but still within its specification, or input voltage is lower than expected, but still high enough to operate normally.

WarningAffectingDeviceOperationValue Value 0x03

The value of this parameter may be causing a degradation of quality in the primary operation of the module. Examples might include: high voltage power supply not reaching target voltage, or a bias current being outside of its calibrated range.

FatalEffectOnDeviceOperationValue Value 0x04

Parameter is abnormal and preventing the device from operating. Examples might include: high voltage power supply to hot, abnormal PMT currents, hard drive free space too low, ambient humidity to high, etc.

IntentionalPreventionOfDeviceOperationValue Value 0x05

The mechanism driving this parameter is doing its intended purpose of preventing the device from operating. Example uses of this might be: enclosure door is open, an external input that prevents data-taking is activated (for example, from an x-ray machine that is operating nearby), or a manual interrupt switch is flipped.

NotAbleToMeasureValue Value 0x06

The parameter is not able to be measured/read-out for some reason. Examples of this might be: hardware failure, garbage data from a humidity sensor, or a failure to initialize appropriate operating system resources to interact with hardware.

HealthNotCurrentlyApplicableValue Value 0x07

This health status of this parameter is not currently applicable most likely due to its operational state. An example of when this value might be used is for high voltage; a healthy range of voltages may be defined, but when the detector is not operating, the voltage will be at zero, which would be outside of the healthy range. Rather than declaring this parameter to be unhealthy, this value would be used to indicate its health status is not currently

applicable. Note that the health status of a parameter changing to or from this value will also case an update push of the parameter to be sent out by the device.

4.1.73 ParameterPropertiesFlags Enumeration

Flags that may be bitwise OR'd to describe one or more properties of a parameter .

Underlying integral representation: `uint32_t`

Enumerated values for `ParameterPropertiesFlags`:

ValueFixedFlag Value 0x01

A parameter that is fixed, can not be altered, and will not change during any WebSocket Connection. Cannot be combined with `ParameterPropertiesFlags::SettableFlag`.

SettableFlag Value 0x02

The parameter can be set by a user. May be combined with all other options other than `ParameterPropertiesFlags::ValueFixedFlag`.

MeasuredFlag Value 0x04

The parameter will report a measured value. May be combined with all other options.

HealthyValueRangedFlag Value 0x08

The parameter has a range of measured values considered to be health or within tolerance. Note, this healthy range may be different than the range the value is allowed to be set, if parameter is settable. May be combined with all other options except `ParameterPropertiesFlags::AffectsHealthWithoutLimitsFlag`. Must also have the `ParameterPropertiesFlags::MeasuredFlag` bit set. Not applicable to parameters of type `ParameterValueDataType::Utf8StringValue`.

SetValueRangedFlag Value 0x10

The values of the parameter are constrained to only be settable in a certain range. May be combined with all other options except `ParameterPropertiesFlags::ValueFixedFlag`. Not applicable to parameters of type `ParameterValueDataType::Utf8StringValue` or `ParameterValueDataType::BooleanValue`. The `ParameterPropertiesFlags::Se` must also be set if this flag is set.

ReportOnChangeFlag Value 0x20

Indicates that the device will notify the control module of changes in a measured values parameter, when it changes by more that a specified value (for ints/bools/floats; changes in string always reported) from the previously reported value, or it has crossed a threshold between healthy value and not healthy, or similar. The `ParameterPropertiesFlags::MeasuredFlag` must also be set.

ReportOnChangeDeltaSettableFlag Value 0x40

Indicates that the delta is settable. Delta is computed as the difference between the previously reported value and the new value. This value should be settable whether device is collecting data or not. The `ParameterPropertiesFlags::ReportOnChangeFlag` and `ParameterPropertiesFlags::MeasuredFlag` bits must also be set.

HealthyLimitsSettableFlag Value 0x80

Indicates that the limits can be set for the healthy range of a parameter. The `ParameterPropertiesFlags::HealthyValueRangedFlag` bit must also be set.

AffectsHealthWithoutLimitsFlag Value 0x100

Indicates that the parameter affects health, however there are no explicit limits that dictate the healthy range. This is useful for string based variables, or numeric variables where the healthy range can not be a priori determined.

NotSettableWhileOperatingFlag Value 0x200

Indicates a settable parameter can not be set while the device is operating. If this bit is set, this parameter can only be set when the device is in the `OperatingMode::StandByValue` or `OperatingMode::ReadyValue` operating modes. If you request a parameter to be changed while in another state, the device can either return a `SetParameterReply` indicating an error, or the device can transition to the `OperatingMode::StandByValue` or `OperatingMode::ReadyValue` operating mode, change the parameter, and then return to its current operating mode (during which each state change needs to be accompanied by a `DeviceStatusPush` message). The `ParameterPropertiesFlags::SettableFlag` bit must also be set.

SettableWithPredefinedValuesOnlyFlag Value 0x400

Settable parameters that can only take on certain discreet values, are indicated by setting this bit. Currently, valid values can only be specified in the description of the parameter. If a change to a value that is not one of the discreet values is allowed by the device, then the device choose to either reject the change indicating an error, or just round to the nearest value. The `ParameterPropertiesFlags::SettableFlag` bit must also be set.

4.1.74 ParameterInfoReply Message

Contains the full information about a parameter. The number of parameters, their names, type (bool/int/float/string/etc), and `ParameterInfoReply::value_properties` of parameters can not change during a single connection to the control module.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x93
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	

Continued on next page

Table continued from previous page

Field Name	Data Type
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`reply_status`	`uint8_t` enumerated by `ParameterInfoStatus`
Field Description: Status of this request. If status is not `ParameterInfoStatus::ValidParameterNameValue`, then `ParameterInfo::parameter_name` and `ParameterInfo::subdetector_number` must match the values requested, but all other integer based fields must have a value of zero, and all string based fields must have a length of zero.	
`value_type`	`uint8_t` enumerated by `ParameterValueDataType`
Field Description: The data type of this parameter. Each applicable field in `ParameterInfoReply::set_value`, `ParameterInfoReply::measured_value`, `ParameterInfoReply::lower_healthy_m` `ParameterInfoReply::upper_healthy_measured_value`, `ParameterInfoReply::lower_settable_value`, `ParameterInfoReply::upper_set` and `ParameterInfoReply::reportable_change_delta` (as can be determined from the `ParameterInfoReply::value_properties` field) must be lexically encoded as this value type (non applicable fields must be empty strings).	
`parameter_health_impact`	`uint8_t` enumerated by `HealthSeverityLevel`
Field Description: The devices assessment as to the impact of the value of this parameter. For both `ParameterPropertiesFlags::HealthyValueRangedFlag` and `ParameterPropertiesFlags::AffectsHealthWithoutLimitsFlag` parameters, it is the responsibility of the sender to mark the health status, regardless of healthy limits or current measured value.	
`subdetector_number`	`uint8_t`
Field Description: The subdetector this Parameter applies to, or zero if not associated with a specific sub-detector.	
`value_properties`	`uint32_t` bits defined by `ParameterPropertiesFlags`
Field Description: A bitwise OR of `ParameterPropertiesFlags` flags.	
`set_timestamp`	UtcTimePoint
Field Description: The timestamp at which this parameter was last set. May be zero for factory values, or other situations where it wouldn't be able to be tracked, or if the parameter is not settable.	
`effective_timestamp`	UtcTimePoint

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The timestamp at which this parameter was last measured, or health status determined. If this parameter does not change, or effect health status this field must be zero. (i.e. zero unless ParameterInfoReply::value_properties has ParameterPropertiesFlags::AffectsHealthWithoutLimitsFlag or ParameterPropertiesFlags::MeasuredFlag bits set)	
parameter_name	Short String
Field Description: Name of the parameter; must not be empty, and use only only ascii letters, numbers, underscore, dash, period, and space characters.	
set_value	Short String
Field Description: Value of the parameter, if ParameterInfoReply::value_properties contains ParameterPropertiesFlags::ValueFixedFlag or ParameterPropertiesFlags::SettableFlag; otherwise the string shall be empty. Value is always represented as a UTF8 string which is a lexical encoding of the type specified by ParameterInfoReply::value_type. Lexical encoding of non-string values is according to the same rules as for XML data types, with the exception that Infs and NaNs are not allowed for floating points; see https://www.w3.org/TR/xmlschema11-2/. If ParameterInfoReply::value_properties is ParameterPropertiesFlags::ValueFixedFlag or ParameterPropertiesFlags::SettableFlag, then this field may only be empty if ParameterInfoReply::value_type has a value of ParameterValueDataType::Utf8StringValue, ParameterValueDataType::FloatingPointListValue, or ParameterValueDataType::IntegerListValue	
measured_value	Short String
Field Description: The measured value of the parameter if, and only if, ParameterInfoReply::value_properties has the ParameterPropertiesFlags::MeasuredFlag bit set, otherwise string must be empty. Value is always represented as a UTF8 string which is a lexical encoding of the type specified by ParameterInfoReply::value_type. Lexical encoding of non-string values is according to the same rules as for XML data types; see https://www.w3.org/TR/xmlschema11-2/, with the exception floats can not be Infs or NaNs.	
lower_healthy_measured_value	Short String
Field Description: The lower limiting value in the range of values over which this parameter is still considered 'healthy'. Used if and only if ParameterInfoReply::value_properties has the ParameterPropertiesFlags::HealthyValueRangedFlag bit set, and otherwise must be an empty string. Value must be lexically encoded according to same rules as XML data types.	
upper_healthy_measured_value	Short String

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The highest value at which this parameter is still considered 'healthy'. Used only if `ParameterInfoReply::value_properties` has the `ParameterPropertiesFlags::HealthyValueRangedFlag` bit set, otherwise must be an empty string. Value must be lexically encoded according to same rules as XML data types.	
`lower_settable_value`	`Short String`
Field Description: The lowest value at which this parameter can be set. If this value is dependent upon a value of another parameter (e.x. both parameters must multiply together and be higher than a given number), then this value lists the absolute lowest value the parameter can take, but the parameter description should give this limitation, and if the parameter is requested to be changed to a value that it cant be set to (partially due to the other dependent parameter), a reply with a status of `CommandReplyStatus::FailedValue` should be given. Used if and only if `ParameterInfoReply::value_properties` field has the `ParameterPropertiesFlags::SetValueRangedFlag` bit set, otherwise this string must be empty. Value must be lexically encoded according to same rules as XML data types.	
`upper_settable_value`	`Short String`
Field Description: The absolutely highest value at which this parameter can be set. If this upper threshold value is dependent upon a value of another parameter (e.x. both parameters must add together and be lower than a given number), then this value should list the absolute highest value the parameter can take, but the parameter description should give this limitation, and if the parameter is requested to be changed to a value that it cant be set to (partially due to the other dependent parameter), a reply with a status of `CommandReplyStatus::FailedValue` should be given. Used if and only if `ParameterInfoReply::value_properties` has the `ParameterPropertiesFlags::SetValueRangedFlag` bit is set, otherwise this string must be empty. Value must be lexically encoded according to same rules as XML data types.	
`reportable_change_delta`	`Short String`
Field Description: The amount a measured value needs to change before an update will be sent. Used only if `ParameterInfoReply::value_properties` has the `ParameterPropertiesFlags::ReportOnChangeFlag` bit set, otherwise this string will be empty. For boolean parameters, setting delta to false indicates report on change (such that a change in value results in a Boolean change from zero to one), while true indicates do not report (for which a change of one or more does not result in a change of Boolean value). Value must be lexically encoded according to same rules as XML data types, with the restriction that for `floating` point types Infs and NaNs are not allowed.	
`parameter_description`	`Medium String`
Field Description: Human readable, UTF-8 description of the parameter.	

4.1.75 ParameterField Enumeration

Describes which field to set when setting a parameter value, or what field changed when notifying of a change. Only one field change can be reported per message.

Used in message: SetParameterRequest

Underlying integral representation: uint8_t

Enumerated values for ParameterField:

SetValueValue Value 0x00

The set value of a parameter.

LowerHealthValue Value 0x01

The lowest healthy value of a parameter.

UpperHealthValue Value 0x02

The uppermost healthy value of a parameter.

ChangeDeltaValue Value 0x03

The reportable change delta (i.e., difference from a previous measurement) for a measured value. For boolean parameters, setting delta to false indicates report on change (such that a change in value results in a Boolean change from zero to one), while true indicates do not report (for which a change of one or more does not result in a change of Boolean value).

MeasuredValueValue Value 0x04

A measured value. Can only be used in ParameterInfoReply, and ParameterUpdatePush messages.

4.1.76 SetParameterRequest Message

Message to set a parameter field. Note that only one field of a parameter can be changed at a time, and changing other fields will require separate SetParameterRequest messages. For example, if a high voltage is controlled with a settable parameter, which also uses upper and lower healthy values to ensure the measured voltage is within a prescribed range, then when changing the set high voltage, you will have to use three separate SetParameterRequest messages in order to change all three fields. Also, the order fields are set may be important so that superfluous warnings are not generated; for example, if increasing the voltage you would first increase the upper healthy value, then the set value, then the lower healthy value.

See Also: SetParameterReply

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x14
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
value_type	uint8_t enumerated by ParameterValueDataType
Field Description: The data type of the value of parameter field.	
parameter_field	uint8_t enumerated by ParameterField
Field Description: Which field of the parameter should be set. Only one parameter field can be set per message. The field being requested to be set should be allowed by this parameters ParameterInfoReply::value_properties value. For example the ParameterPropertiesFlags::ReportOnChangeDeltaSettableFlag bit must be set in order to change the ParameterField::ChangeDeltaValue field.	
subdetector_number	uint8_t
Field Description: The subdetector this Parameter applies to, or zero if not associated with a specific sub-detector.	
parameter_name	Short String
Field Description: UTF-8 encoded name of the parameter.	
value	Short String
Field Description: Data type must match what is specified by SetParameterRequest::value_type, encoded as a string as in https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html	

4.1.77 SetParameterReply Message

A reply to a SetParameterRequest from the control module, indicating the status of the change.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x94
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	

Continued on next page

Table continued from previous page

Field Name	Data Type
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`reply_status`	`uint8_t` enumerated by `CommandReplyStatus`
Field Description: Status of the request, see `CommandReplyStatus`.	
`timestamp`	UtcTimePoint
Field Description: Timestamp of when the parameter setting became effective. Or if status was `CommandReplyStatus::InProgressValue`, than either zero, or an estimate of when it will become effective. Microseconds since the Unix epoch.	

4.1.78 ParameterUpdatePush Message

Message sent by a device to the control module informing of a change to a single parameter of the device.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x00
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x18
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`change_timestamp`	UtcTimePoint
Field Description: Timestamp of when the change became effective, or at least when it was noticed to have become effective. Microseconds since the Unix epoch.	
`value_type`	`uint8_t` enumerated by `ParameterValueDataType`
Field Description: Data type of the value field. The data type cannot change during a connection, once it has been provided.	
`parameter_field`	`uint8_t` enumerated by `ParameterField`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Enumeration identifying which field of the parameter changed. Only one field change can be reported per message. Note that if a measured value changes, which also causes a health status change, this field should report that it was the measured value that changed, while the health status change is reported in the ParameterUpdatePush::parameter_health_impact field, and the new value of the measurement goes in the ParameterUpdate::value field. If the ParameterField::SetValueValue, ParameterField::LowerHealthValue or ParameterField::UpperHealthValue are changed, causing a change to the health status, then it should be reported what was changed, and the new health status should be noted in ParameterUpdatePush::parameter_health_impact.	
parameter_health_impact	uint8_t enumerated by HealthSeverityLevel
Field Description: The impact of the current value of the parameter on health.	
subdetector_number	uint8_t
Field Description: The subdetector this Parameter applies to, or zero if not associated with a specific sub-detector.	
parameter_name	Short String
Field Description: Parameter name; only ascii letters, numbers, underscore, period, dash, and space characters are allowed.	
value	Short String
Field Description: Data type must match what is specified by ParameterUpdatePush::value_type, encoded as a string as in https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html	

4.1.79 ParameterUpdatePushAck Message

A message acknowledging the receipt by the control module of a ParameterUpdatePush

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x00
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x98
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.2 Radiation Detector Interface

4.2.1 RadDetectorMsgType Enumeration

Values to indicate the type of message being sent as part of the `MessageGroup::RadDetectorValue` message group. In this message group, push messages originate at the radiation detector modules, and request messages originate at the control module. The first byte of messages shall have a value of `MessageGroup::RadDetectorValue` (0x08), and the second byte of the message will have a value as indicated by this enum, which will then tell you how to decode the message. For example a second byte value of `RadDetectorMsgType::RadSubDetectorInformationRequestValue` will tell you the message contents are specified by `RadSubDetectorInformationRequest`.

Underlying integral representation: `uint8_t`

Enumerated values for `RadDetectorMsgType`:

RadSubDetectorInformationRequestValue Value 0x61

A request from the control module to the radiation detector module for information about the radiation related capabilities of a given subdetector. See `RadSubDetectorInformationRequest` for message contents and format.

RadSubDetectorInformationReplyValue Value 0xe1

A reply from a radiation detector module to the control module in response to a `RadSubDetectorInformationRequest` message. See `RadSubDetectorInformationRe` for message contents and format.

RadChannelDataPushValue Value 0x62

Message sent from the radiation detector to the control module containing the channel data (spectrum), or gross counts information (i.e., a one channel spectrum) and related information of a measurement interval for a single subdetector. See `RadChannelDataPush` for message contents and format.

RadChannelDataPushAckValue Value 0xe2

Acknowledgment sent from the control module to the radiation detector module that verifies the channel data was received.

RadListModeDataPushValue Value 0x63

Message containing List mode data sent from the radiation detector module to the control module for a single detection event from a single subdetector. See `RadListModeDataPush` for message contents and format.

RadListModeDataPushAckValue Value 0xe3

Acknowledgment sent from the control module to the radiation detector that verifies the listmode data was received. See `RadListModeDataPushAck` for message contents and format.

RadEnergyCalibrationUpdatePushValue Value 0x64

Message sent from a radiation detector module to the control module when the energy calibration has been updated. Note that this message might be sent even in cases when the actual energy calibration parameters have not changed but perhaps were either checked or verified, or the detector's internal scaling changes (e.x., when a detector always linearizes data to a fixed calibration, but its internal scaling periodically gets updated). See `RadEnergyCalibrationUpdatePush` for message contents and format.

RadEnergyCalibrationUpdatePushAckValue Value 0xe4

Acknowledgement sent from the control module to the radiation detector that verifies the energy calibration was received. See `RadEnergyCalibrationUpdatePushAck` for message contents and format.

RadEnergyCalibrationRequestValue Value 0x65

Request sent from the control module to a radiation detector module, for the module to send back the current energy calibration for the specified subdetector. See `RadEnergyCalibrationRequest` for message contents and format.

RadEnergyCalibrationReplyValue Value 0xe5

Reply sent from the radiation detector module to the control module containing the current energy calibration information; a response to a `RadEnergyCalibrationRequest` message. See `RadEnergyCalibrationReply` for message contents and format.

RadUseExternalEnergyCalRequestValue Value 0x66

Energy calibration sent from the control module, to a radiation detector module, requesting that a specified subdetector use the calibration. Note that in order for detectors to accept external calibrations, the `RadDetDeviceFeaturesFlags::AcceptsExternalCalibFlag` bit of `RadSubDetectorInfo::subdetector_features` shall be set in the `RadSubDetectorInfo` information provided by the device. See `RadUseExternalEnergyCalRe` for message contents and format.

RadUseExternalEnergyCalReplyValue Value 0xe6

Reply sent from a radiation detector to the control module in response to a `RadUseExternalEnergyCalRequest` message, informing the control module of the status of using the requested calibration. See `RadUseExternalEnergyCalReply` for message contents and format.

4.2.2 RadSubDetectorInformationRequest Message

Message sent from the control module to the radiation detector module when it would like radiation-detector-specific information about all of the module's subdetectors.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x08
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x61
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.2.3 RadSubDetectorType Enumeration

Specifies the type of detector within a radiation detection module, i.e., if it detects gamma-rays or neutrons.

A detector that detects both gamma rays and neutrons would need to defined as two subdetectors, one of which is devoted to the gamma-ray response, and the other for the neutron response.

Underlying integral representation: `uint16_t`

Enumerated values for `RadSubDetectorType`:

RadDetectorGammaValue Value 0x01

The radiation subdetector detects gamma rays.

RadDetectorNeutronValue Value 0x02

The radiation subdetector detects neutrons.

4.2.4 RadDetDeviceFeaturesFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more features or functionalities of gamma or neutron detectors.

Underlying integral representation: `uint64_t`

Enumerated values for `RadDetDeviceFeaturesFlags`:

NotEnergySensitiveFlag Value 0x01

Indicates the radiation detector only provides gross-count information, not spectroscopic information. This implies that `EnergyCalCoefficientType` will always be `EnergyCalCoefficientType::NotApplicableValue`, as well as `ListModeEvent::energy_channel` always being zero, and `ChannelDataPacket::channe` having exactly one entry.

FixedEnergyCalFlag Value 0x02

The device has a fixed calibration that does not change, and which can not be otherwise set.

PerformsPeriodicSelfCalFlag Value 0x04

The device contains a routine that may periodically run to calibrate itself off of NORM, an internal seed source, an internal LED source, or some other method, resulting in a `RadEnergyCalibrationUpdatePush` being sent to the control module.

CalsOffNormFlag Value 0x08

The device performs calibration off of ambient natural radiation NORM sources. If this flag is set, the `RadDetDeviceFeaturesFlags::PerformsPeriodicSelfCalFlag` must also be set.

ContainsSeedCalSourceFlag Value 0x10

The device contains an internal 'seed' source which is always visible to the detector and is used for calibration. If this flag is set, the `RadDetDeviceFeaturesFlags::PerformsPeriodic` must also be set.

ContainsHousedCalSourceFlag Value 0x20

The device contains an internal source normally shielded (e.g. housed within a lead pig) but which can be made visible to the detector when energy calibration is being performed.

ContainsNonRadCalSourceFlag Value 0x40

The device uses something such as an internal LED source for calibration.

NeedsDedicatedCalTimeFlag Value 0x80

The device performs calibration during dedicated calibration time.

AcceptsExternalCalibFlag Value 0x100

The device accepts external calibration information; calibration information may be sent to the device via an `RadUseExternalEnergyCalRequest` message. Even if the detector contains its own automated energy calibration routine, it is strongly encourage to also support externally setting the energy calibration to help accommodate situations where the built-in calibration algorithms may fail, or for special deployment circumstances.

SupportsPolynomialCalFlag Value 0x200

The device supports `EnergyCalCoefficientType::PolynomialValue` energy calibration; polynomial calibration is utilized in accordance with the N42.42-2012 standard. This feature implies support for spectroscopic data.

SupportsChannelBoundariesEnergyCalFlag Value 0x400

The device supports `EnergyCalCoefficientType::ChannelBoundariesValue` energy calibration; this calibration type specifies the energy boundary of each channel. This feature implies support for spectroscopic data.

SupportsDeviationPairCalFlag Value 0x800

See `RadEnergyCalibrationUpdatePush::deviation_pairs`. The device module supports updated calibration information utilizing deviation pairs. This feature implies support for spectroscopic data.

InternallyLinearizesSpectrumFlag Value 0x1000

Indicates the device performs a linearization of the data before sending it out. Examples of doing this might be: to make the output spectrum always go from 0 keV to 3072 keV in 1024 evenly sized bins (resulting in a polynomial calibration of {0,3,0}), or to apply the detectors intrinsic deviation pairs to the data, but still have polynomial calibration coefficients that are varied to account for temperature or other drifts.

4.2.5 RadDetectorKind Enumeration

The kinds of detection mediums of radiation detectors.

Underlying integral representation: `uint8_t`

Enumerated values for `RadDetectorKind`:

HPGeValue Value 0x01

HPXeValue Value 0x02

NaIValue Value 0x03

LaBr3Value Value 0x04

LaCl3Value Value 0x05

BGOValue Value 0x06

CZTValue Value 0x07

CdTeValue Value 0x08

CsIValue Value 0x09

GMTValue Value 0x0a

GMTWValue Value 0x0b

LiFiberValue Value 0x0c

PVTValue Value 0x0d

PSValue Value 0x0e

He3Value Value 0x0f

He4Value Value 0x10

LiGlassValue Value 0x11

LiIValue Value 0x12

SrI2Value Value 0x13

CLYCValue Value 0x14

CdWO4Value Value 0x15

BF3Value Value 0x16

HgI2Value Value 0x17

CeBr4Value Value 0x18

LiCAFValue Value 0x19

LiZnSValue Value 0x1a

B10GlassValue Value 0x1b

OtherRadDetectorKindValue Value 0xff

4.2.6 RadDetectorGeometry Enumeration

Description of detector geometry.

Underlying integral representation: `uint8_t`

Enumerated values for `RadDetectorGeometry`:

RightCircularCylinderDetectorValue Value 0x01

A right circular cylinder. Examples include a typical 3x3 NaI detector, a He3 filled tube, or mechanically cooled HPGe detector.

RectangularCuboidDetectorValue Value 0x02

A rectangular detector. Examples include most large volume PVT detectors, and vehicle mounted NaI search detectors.

OtherGeometryDetectorValue Value 0xff

Detector is another geometry. A free-form description should be included as part of the `RadSubDetectorInfo::subdetector_description` field.

4.2.7 RadSubDetectorInfo Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold the information about a specific subdetector. `RadSubDetectorInformationReply` messages contain one of these for each of the subdetectors in the system.

Message Contents:

Field Name	Data Type
subdetector_number	uint8_t
Field Description: The radiation subdetector number to which this information corresponds.	
subdetector_type	uint16_t enumerated by RadSubDetectorType
Field Description: The type of radiation detector (gamma or neutron) this subdetector is.	
number_energy_channels	uint32_t
Field Description: The number of energy channels this subdetector reports; e.g., the number of channels in the energy spectrum. Gross count devices such as He3 neutron detectors will report one; no detector may report zero. The highest value allowed is 65536 to be consistent with allowed channel numbers of listmode data or channel data array sizes. If there is a setting (as part of the parameter mechanism) that changes the number of energy channels, then the RAPTER connection must be terminated and re-initiated before this change takes effect.	
subdetector_features	uint64_t bits defined by RadDetDeviceFeaturesFlags
Field Description: Bitwise OR of flags to indicate the features this subdetector supports.	
subdetector_description	Short String
Field Description: A UTF-8 encoded, human readable description of this subdetector. Examples might be: '3x3 NaI 7%', '24cm x 36cm x 10cm PVT two PMT panel', '5 atm He3 detector', etc.	
subdetector_kind	uint8_t enumerated by RadDetectorKind
Field Description: The detection medium type.	
detector_geometry	uint8_t enumerated by RadDetectorGeometry
Field Description: The geometry description of the detection medium.	
detector_dim1_cm	float
Field Description: For RadDetectorGeometry::RightCircularCylinderDetectorValue detectors, this is the diameter of the cylinder. For RadDetectorGeometry::RectangularCuboidDetectorValue detectors this is the width of the detection face.	
detector_dim2_cm	float
Field Description: For RadDetectorGeometry::RightCircularCylinderDetectorValue detectors, this is the length of the cylinder. For RadDetectorGeometry::RectangularCuboidDetectorValue detectors this is the height of the detection face. In units of cm.	
detector_dim3_cm	float
Field Description: For RadDetectorGeometry::RightCircularCylinderDetectorValue detectors, this is value is not used and must be 0.0. For RadDetectorGeometry::RectangularCuboidDetectorValue detectors this is the depth (thickness) of the detector. In units of cm.	
detector_volume_cm3	float
Field Description: The volume of the detection medium in cubic centimeters. This value should not include internal voids or inactive volumes, so may differ from the naive volume computation from the given dimensions.	

4.2.8 RadSubDetectorInformationReply Message

A reply to a RadSubDetectorInformationRequest message, which contains information about all subdetectors in this radiation detection module.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x08 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0xe1 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
informations	Short Array of RadSubDetectorInfo *Field Description*: Information about all subdetectors in this module; must contain exactly one entry for each of the subdetectors. Length should be equal to DeviceInfo::num_subdetectors. Subdetector numbers start with 1 and go to DeviceInfo::num_subdetectors.

4.2.9 RadDataReadoutFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more conditions regarding the status of RadChannelDataPush, or RadListModeDataPush messages.

See also:

- RadChannelDataPush::status_flags
- RadListModeDataPush::status_flags
- OccupancyData::status_flags

Underlying integral representation: uint32_t

Enumerated values for RadDataReadoutFlags:

RadDataReadoutExternallyInterruptedFlag Value 0x01

Indicates that external interrupt (ex. TTL input from a radiography system) was on for at least a portion of the time period that this data represents.

RadDataReadoutSuspectFlag Value 0x02

Indicates that the detector health should be further inspected before using this data.

RadDataReadoutGettingActiveCalSourceReadyFlag Value 0x04

Indicates that for at least a portion of the time period this data represents, the detector's internal active calibration system was in the process of becoming active (led turning on, Th232 coming out of Pb case) and this portion of time was not used to do the calibration.

RadDataReadoutActiveCalibrationDataFlag Value 0x08

Indicates that for at least a portion of the current time period this data represents, the detector's internal active calibration system (ex. scintillator LED on, or Th232 source outside its Pb case) was in effect. For systems where the calibration source is always present, this bit should not be set. Note both this bit, and `RadDataReadoutFlags::RadDataReadoutGettingActiveCalSourceReadyFlag` may be set if the time period this data corresponds too separately had both these conditions.

RadDataReadoutMultipleTimeSamplesFlag Value 0x10

Indicates that there was a readout issue causing the readout to fall behind to the point where combining multiple samples was necessary to catch back up. This should be an exceptional circumstance. The number of combined sample can be figured out by using the real time.

RadDataReadoutDataCouldNotBeCollectedFlag Value 0x20

Data was unable to be collected for this interval.

RadDataReadoutNotAcquiringDataFlag Value 0x40

Indicates that the detector is not taking data, and that this message does not contain valid data (the message must still be in a valid format to allow parsing, but but should be considered to not contain useful radiation data). This bit is useful for when current data of a radiation detector is requested but the detector isn't in operating mode.

4.2.10 RadChannelDataPush Message

This message structure, which is the same as the `RadChannelDataPush` struct, contains the data sent for one time-slice (e.x., every 0.1 seconds), or at the end of a requested dwell, from a gamma detector or neutron detector. If a device has more than one subdetector than it must send one of these messages for each sub detector (i.e. when in `DataCollectionModes::ClockTimeIntervalValue`, `DataCollectionModes::LiveTimeDwellValue`, or `DataCollectionModes::RealTimeDwe` modes); which subdetector this data is for is noted by `RadChannelDataPush::subdetector_number`.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x08
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x62
Field Description: The type of message within the specified message_group, that this message corresponds to.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
start_timestamp	UtcTimePoint *Field Description*: The clock time of the module at the start of the interval to which this channel data corresponds. Microseconds since the UNIX epoch.
end_timestamp	UtcTimePoint *Field Description*: The clock time of the module at the end of the interval to which the channel data corresponds. This value will nominally be the last microsecond in an interval, and not the beginning of the next interval (e.g., it will be one less than the next interval start time). Microseconds since the UNIX epoch. Note that because of time adjustments (via the PTP synchronization mechanisms), or interruptions to data taking, this time might not be ChannelDataPacket::start_timestamp + 1.0E6*clock_time_seconds.
live_time_seconds	float *Field Description*: The time in seconds that the detection system was available to take data during this data taking interval. The difference between RadChannelDataPush::clock_time_seconds and RadChannelDataPush::live_time_seconds is often referred to as "dead time" and can be caused, for example, by limitations in the MCA that prevent it from recording more detection events for a brief time after an event (maybe due to inherent decay time of scintillator, or electronics reset time), or from internal queues filling up, or an external interrupt (e.g., from a nearby radiography system) indicating data should not be taken.
clock_time_seconds	float *Field Description*: The acquisition clock time elapsed, in seconds, to which this data corresponds. A common use for this value, in combination with RadChannelDataPush::live_time_seconds, is to determine the "dead time" of the acquisition electronics to account for pulse-pileup effects. This time is not simply be the difference of the absolute times of the beginning and end of the data taking interval. For example, PTP time synchronizations should not affect this value. Time periods where data acquisition is blocked due to an external interrupt (i.e., from non-intrusive inspection system), or other reason (e.x., data acquisition did not start until part way through the interval) do not contribute to this value.
rad_data_flags	uint32_t bits defined by RadDataReadoutFlags *Field Description*: Bitwise OR of flags to indicate unusual, but important conditions during data taking.
subdetector_number	uint8_t *Field Description*: The subdetector this data is from.
channel_data	Large Array of float

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The channel data as recorded by the detector.	

4.2.11 RadChannelDataPushAck Message

A message sent by the control module to the device acknowledging the receipt of a `RadChannelDataPush` message.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x08
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xe2
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.2.12 ListModeEvent Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct that represents a single listmode detection event. This struct holds a time offset (in microseconds) from `ListModeDataPacket::reference_timestamp`, as well as the detected energy channel.

Message Contents:

Field Name	Data Type
`relative_time`	`uint32_t`
Field Description: Number of microseconds to add to `ListModeDataPacket::reference_timestamp` in order to get the UTC time point of the detection event.	
`energy_channel`	`uint16_t`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The energy channel for the gamma or neutron detection event. Used in conjunction with the energy calibration to determine the energy of this detection event. If the detector does not measure energy in any way (i.e., the `RadDetDeviceFeaturesFlags::NotEnergySensitiveFlag` is set in `RadSubDetectorInfo::subdetector_features`), this value must be zero.	

4.2.13 RadListModeDataPush Message

Message sent by a radiation detection module to convey list mode neutron or gamma data. This message conveys data from a single subdetector, and will contain one or more detection events. This message will only be sent when the device is operating in the `DataCollectionModes::OnEventValue` mode. If the device is operating with a non-zero collection interval (e.g., `ChangeDeviceStateRequest::collection_interval_ms` was non-zero), then the device will also be sending `HeartbeatPush` messages for each interval, and the detection events contained in this message must only be from a single time interval; that is, the contained listmode data will all come either before or after the `HeartbeatPush::end_timestamp`, but not both. However, if no detection events happened during an interval, none of these messages will be sent. Or there may be multiple of these messages sent for a single time interval, for each subdetector. This message can convey multiple detection events for detector that may detect at high rates, so as to allow minimizing the number of separate WebSocket message and Ethernet frames on the network.

See Also: `RadListModeDataPushValue`

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x08
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x63
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`reference_timestamp`	UtcTimePoint
Field Description: The timestamp all contained listmode detection events time offsets (i.e., the `ListModeEvent::relative_time` field) will be added to to get the detection events time. Note, this reference timestamp will come before must be less than or equal to the first detection event.	

Continued on next page

Table continued from previous page

Field Name	Data Type
rad_data_flags	uint32_t bits defined by RadDataReadoutFlags
Field Description: Bitwise OR of the relevant RadDataReadoutFlags. The same flags must be applicable to all contained detection events.	
subdetector_number	uint8_t
Field Description: If a detector contains multiple detection devices (ex. multiple He3 tubes, or multiple NaI blocks) that are individually read out as subdetectors, then the subdetector that detected the event should be specified here. If only a single detection device is present in the module, this value will always be one. See also: • DeviceInfoReply::num_subdetectors • RadChannelDataPush::subdetector_number • RadEnergyCalibrationUpdatePush::subdetector_number	
listmode_events	Medium Array of ListModeEvent
Field Description: The detected listmode events. Events must be in increasing time order (e.g., the earliest detected events of this message come first in the array). Note that if there are more than 65,536 detection events, multiple RadListModeDataPush messages must be sent.	

4.2.14 RadListModeDataPushAck Message

A message sent by the control module to the device acknowledging the receipt of a RadListModeDataPush message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x08
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xe3
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.2.15 EnergyCalCoefficientType Enumeration

Enumeration to indicate how the coefficients of the the energy calibration structure are to be interpreted.

Used in message: RadEnergyCalibrationUpdatePush

Underlying integral representation: uint8_t

Enumerated values for EnergyCalCoefficientType:

NotApplicableValue Value 0x01

The detector does not provide energy information (e.g., He3, Geiger counter, gross-count gamma, etc.). No coefficients or uncertainties should be provided. If the range of energies detected is well defined, you may consider defining a EnergyCalCoefficientType::ChannelBoundariesValue calibration with a single channel.

PolynomialValue Value 0x02

The coefficients are polynomial calibration coefficients, used as specified in the N42.42-2012 standard, i.e. energy_channel_i(keV) = coefficients[0] + i*coefficients[1] + i*i*coefficients[2] + ... There will be two or more coefficients, and the resulting energy of the channels will be monotonically increasing. Note that the N42.42-2012 standard specifies exactly three coefficients, but the use here only requires two or more coefficients, in keeping with the wider use of polynomial energy calibration for gamma radiation detectors. If there are any deviation pairs defined, then these nonlinearity corrections will be applied on top of this polynomial calibration.

ChannelBoundariesValue Value 0x04

The energy boundary of each channel is specified. This means there will be the same number of float RadEnergyCalibrationUpdatePush::coefficients values specified, as there are channels, with optionally one additional value that gives the upper energy of the last channel. The values in RadEnergyCalibrationUpdatePush::coeffici must be strictly monotonically increasing, and no deviation pairs will be defined.

4.2.16 EnergyCalibrationStatus Enumeration

Enumeration of values to describe the status of an energy calibration for a radiation detector. Used by a radiation detector module to inform the control module in a push message, or reply to a request from the control module for status information.

Used in message: RadEnergyCalibrationUpdatePush

Underlying integral representation: uint8_t

Enumerated values for EnergyCalibrationStatus:

EnergyCalibrationGoodValue Value 0x00

The energy calibration presently in use is a good calibration.

EnergyCalibrationQuestionableValue Value 0x01

The energy calibration presently in use is a questionable calibration.

EnergyCalibrationOutOfDateValue Value 0x02

The energy calibration presently in use is out of date.

EnergyCalibrationBadValue Value 0x03

The energy calibration presently in use is a bad calibration.

EnergyCalibrationUnknownStatusValue Value 0x04

The energy calibration presently in use is of undetermined quality.

4.2.17 EnergyCalMethodFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more conditions regarding the sources used and how the energy calibration was derived.

Underlying integral representation: `uint16_t`

Enumerated values for `EnergyCalMethodFlags`:

EnergyCalMethodNoneFlag Value 0x00

Energy calibration is not applicable to this device, e.g., He3 detector or a gross count gamma detector.

EnergyCalFactoryFlag Value 0x01

Energy calibration is fixed at the factory and has not been changed.

EnergyCalExternallySetFlag Value 0x02

The calibration was sent to the device through a RAPTER message, for example if a technician manually determined and entered the calibration.

EnergyCalSeedSourceFlag Value 0x04

A seed source embedded in or near the detection element was used for calibration. The detection element is always exposed to the source. This could be something like a Cs137 or Eu152 embedded into the detection media, or Thorium welding rods or K40 source near the detection element.

EnergyCalNormFlag Value 0x10

Calibration is performed using naturally occurring radiation.

EnergyCalActiveGammaSourceFlag Value 0x20

Energy calibration performed using an active radiological calibration source that is housed on-board the system and automatically exposed to the detector during dedicated active calibration periods).

EnergyCalActiveLedSourceFlag Value 0x40

A mechanism using a light source, such as a LED or laser embedded in the scintillator, is used.

EnergyCalKnownSourceFlag Value 0x80

A known radiological source was used for calibration. The source is not housed on-board the system.

EnergyCalDefaultValueFlag Value 0x200

Indicates device is using a "default" energy cal, but may update itself in the future. For example when detector is first turned on and hasn't had a chance to calibrate, so is using some pre-programmed defaults.

EnergyCalOtherMethodFlag Value 0x8000

Some other method of calibration was performed, and should be described in the `EnergyCalibration::method_description`.

4.2.18 RadEnergyCalibrationUpdatePush Message

Message sent by a radiation detection module to convey the energy calibration of data provided by the `RadChannelDataPush` or `RadListModeDataPush` messages. If there are multiple subdetectors, all calibrations may be sent as a list of messages within a single WebSocket message. Energy is in units of keV.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x08
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x64
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`calibration_timestamp`	UtcTimePoint
Field Description: When the calibration was completed. Will be in the past, but not the future. If this energy calibration is empty or not valid, this value will be zero.	
`energy_cal_coefficient_type`	`uint8_t` enumerated by `EnergyCalCoefficientType`
Field Description: The types of coefficients used by the calibration.	
`calibration_status`	`uint8_t` enumerated by `EnergyCalibrationStatus`
Field Description: The status of the energy calibration, such as good, out of date, etc.	

Continued on next page

Table continued from previous page

Field Name	Data Type
subdetector_number	uint8_t
Field Description: The subdetector number this calibration is for. If a module only has one detection crystal/He3-tube, this will always be number one. If it has N detection sensors, this value will range from 1 to N (inclusive) according to the subdetector this calibration is for. See also: • DeviceInfoReply::num_subdetectors • RadListModeDataPush::subdetector_number • RadChannelDataPush::subdetector_number	
energy_cal_method_flags	uint16_t bits defined by EnergyCalMethodFlags
Field Description: A bitwise OR of flags representing energy calibration source and method.	
coefficients	Medium Array of float
Field Description: The coefficients necessary to specify the calibration.	
coefficient_uncertainties	Medium Array of float
Field Description: The uncertainty on the coefficients; may have zero entries, the same number of entries as the number of coefficients, twice the number of coefficients, or the number of coefficients squared. The interpretation of the coefficients is dependent on the number of entries and the options are: • No entries: uncertainty is not specified. • Same number of entries as coefficients: symmetrized 1-sigma uncertainties. • Twice the number of entries as coefficients: the positive 1-sigma uncertainties for each coefficient, then the negative 1-sigma uncertainties (e.g., the first N coefficients are positive uncertainties, followed the N negative uncertainties). • The square of number of entries as coefficients: the covariance matrix with entries of the first row listed first, followed by entries of second row, etc. To avoid an ambiguous situation, when there are two coefficients, the covariance matrix can not be specified, but as a work around, a 3rd zero valued coefficient could be added and then the matrix specified.	
deviation_pairs	Short Array of float

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Deviation pairs, as in N42.42, are each two `float`s, first the true energy, then the offset, then next entry in array is the next true energy, and its offset, etc. True energies and offsets are in keV. Unless the `RadDetDeviceFeaturesFlags::SupportsDeviationPairCalFlag` bit is set in `RadSubDetectorInfo::subdetector_features`, no values for this field may be provided. An example procedure for using deviation pairs: • Determine first (offset) and second (gain) polynomial coefficients using the 239 keV and 2614 keV peaks of Th232 • If the k-40 1460 keV peak is now at 1450 keV, a deviation of 10 keV should be set at 1460 keV • Deviation pairs set to zero would be defined for 239 keV and 2614 keV • The value provided in this field would then be [239,0,1460,10,2614,0]. Cubic spline interpolation is used to interpolate between deviation pairs. When solving for the spline coefficients, the second derivative is set to zero at the lowest deviation pair energy, and the first derivative is set to zero at the highest deviation pair energy. Corrections for energies below the lowest value deviation pair are always equal to the offset of the lowest energy deviation pair, and corrections for energies above the highest energy deviation pair are always equal to the offset of the highest energy deviation pair. Note: uncertainties for `RadEnergyCalibrationUpdatePush::deviation_pairs` can not be specified since these are typically taken as well defined fixed quantities, and the polynomial coefficient uncertainties will account for the uncertainties. Note: although non-linear deviation pairs are used within the N42.42-2012 standard, the specification doesnt appear to provide an adequate definition of the,; the open source SpecUtils library, available at https://github.com/sandialabs/SpecUtils/, provides an implementation of non-linear deviation pairs that may be useful for reference.	
`method_description`	`Short String`
Field Description: A free-form description of method used to derive calibration. Examples might be: • "Fixed Factory Calibration" • "Fit to NORM templates" • "Gain matched for Th232 2614 keV and K40 1460 keV peaks" • "Individual PMTs gain matched to each other, followed by a fit to NORM templates" • "Gain matched to internal Cs137 seed" • "5 minute spectrum using Eu152" • etc.	
`calculation_notes`	`Medium String`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Free form text providing additional implementation specific details related to quality of energy calibration procedure. Examples of information that might be given in this text are: • Chi2 of fit to background template. • Endpoint energy of the detector that was determined • Relative gains of PMTs. • Unusual conditions detected, like high background rates, or contamination • Data channel corresponding to peak maximum • Version of calibration algorithm used The text must be UTF-8 encoded.	

4.2.19 RadEnergyCalibrationUpdatePushAck Message

A message sent by the control module to the device acknowledging the receipt of a `RadEnergyCalibrationUpdatePush` message.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x08 *Field Description*: The RAPTER `MessageGroup` this message corresponds too.
`message_type`	`uint8_t` with value 0xe4 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.2.20 RadEnergyCalibrationRequest Message

Request for a detector to send its current energy calibration information to the control module.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x08 *Field Description*: The RAPTER `MessageGroup` this message corresponds too.
`message_type`	`uint8_t` with value 0x65

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`subdetector`	`uint8_t`
Field Description: The subdetector for which calibration is desired.	

4.2.21 RadEnergyCalibrationReplyStatus Enumeration

The overall status for the reply to a `RadEnergyCalibrationRequest`.

Used in message: `RadEnergyCalibrationReply`

Underlying integral representation: `uint8_t`

Enumerated values for `RadEnergyCalibrationReplyStatus`:

EnergyCalReplySuccessfulValue Value 0x00

The energy calibration that is a part of this message was successfully retrieved. Note that devices that do not support energy calibration (ex, He3), still return a valid energy calibration, they are just marked with `EnergyCalCoefficientType::NotApplicableValue`.

EnergyCalReplyInvalidSubDetectorValue Value 0x01

The calibration for an invalid subdetector was requested. The energy calibration that is part of the message should still be marked as `EnergyCalCoefficientType::NotApplicableValu` and otherwise a valid energy calibration.

EnergyCalReplyOtherErrorValue Value 0x02

There was an error retrieving energy calibration - a `NotificationPush` message may be separately sent to provide additional information.

4.2.22 RadEnergyCalibrationReply Message

Energy calibration information in response to a `RadEnergyCalibrationRequest`. If the request specified an individual invalid subdetector number, an energy calibration with `EnergyCalCoefficientType` marked as `EnergyCalCoefficientType::NotApplicableVal` should be returned, along with a notification attached.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x08
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xe5
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
reply_status	uint8_t enumerated by RadEnergyCalibrationReplyStatus
Field Description: The status of retrieving the requested energy calibration.	
calibration_timestamp	UtcTimePoint
Field Description: When the calibration was completed. Will be in the past, but not the future. If this energy calibration is empty or not valid, this value will be zero.	
energy_cal_coefficient_type	uint8_t enumerated by EnergyCalCoefficientType
Field Description: The types of coefficients used by the calibration.	
calibration_status	uint8_t enumerated by EnergyCalibrationStatus
Field Description: The status of the energy calibration, such as good, out of date, etc.	
subdetector_number	uint8_t
Field Description: The subdetector number this calibration is for. If a module only has one detection crystal/He3-tube, this will always be number one. If it has N detection sensors, this value will range from 1 to N (inclusive) according to the subdetector this calibration is for. See also: • DeviceInfoReply::num_subdetectors • RadListModeDataPush::subdetector_number • RadChannelDataPush::subdetector_number	
energy_cal_method_flags	uint16_t bits defined by EnergyCalMethodFlags
Field Description: A bitwise OR of flags representing energy calibration source and method.	
coefficients	Medium Array of float
Field Description: The coefficients necessary to specify the calibration.	
coefficient_uncertainties	Medium Array of float

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The uncertainty on the coefficients; may have zero entries, the same number of entries as the number of coefficients, twice the number of coefficients, or the number of coefficients squared. The interpretation of the coefficients is dependent on the number of entries and the options are: • No entries: uncertainty is not specified. • Same number of entries as coefficients: symmetrized 1-sigma uncertainties. • Twice the number of entries as coefficients: the positive 1-sigma uncertainties for each coefficient, then the negative 1-sigma uncertainties (e.g., the first N coefficients are positive uncertainties, followed the N negative uncertainties). • The square of number of entries as coefficients: the covariance matrix with entries of the first row listed first, followed by entries of second row, etc. To avoid an ambiguous situation, when there are two coefficients, the covariance matrix can not be specified, but as a work around, a 3rd zero valued coefficient could be added and then the matrix specified.	
`deviation_pairs`	`Short Array` of `float`
Field Description: Deviation pairs, as in N42.42, are each two `float`s, first the true energy, then the offset, then next entry in array is the next true energy, and its offset, etc. True energies and offsets are in keV. Unless the `RadDetDeviceFeaturesFlags::SupportsDeviationPairCalFlag` bit is set in `RadSubDetectorInfo::subdetector_features`, no values for this field may be provided. An example procedure for using deviation pairs: • Determine first (offset) and second (gain) polynomial coefficients using the 239 keV and 2614 keV peaks of Th232 • If the k-40 1460 keV peak is now at 1450 keV, a deviation of 10 keV should be set at 1460 keV • Deviation pairs set to zero would be defined for 239 keV and 2614 keV • The value provided in this field would then be [239,0,1460,10,2614,0]. Cubic spline interpolation is used to interpolate between deviation pairs. When solving for the spline coefficients, the second derivative is set to zero at the lowest deviation pair energy, and the first derivative is set to zero at the highest deviation pair energy. Corrections for energies below the lowest value deviation pair are always equal to the offset of the lowest energy deviation pair, and corrections for energies above the highest energy deviation pair are always equal to the offset of the highest energy deviation pair. Note: uncertainties for `RadEnergyCalibrationReply::deviation_pairs` can not be specified since these are typically taken as well defined fixed quantities, and the polynomial coefficient uncertainties will account for the uncertainties. Note: although non-linear deviation pairs are used within the N42.42-2012 standard, the specification doesnt appear to provide an adequate definition of the,; the open source SpecUtils library, available at https://github.com/sandialabs/SpecUtils/, provides an implementation of non-linear deviation pairs that may be useful for reference.	
`method_description`	`Short String`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: A free-form description of method used to derive calibration. Examples might be: • "Fixed Factory Calibration" • "Fit to NORM templates" • "Gain matched for Th232 2614 keV and K40 1460 keV peaks" • "Individual PMTs gain matched to each other, followed by a fit to NORM templates" • "Gain matched to internal Cs137 seed" • "5 minute spectrum using Eu152" • etc.	
`calculation_notes`	`Medium String`
Field Description: Free form text providing additional implementation specific details related to quality of energy calibration procedure. Examples of information that might be given in this text are: • Chi2 of fit to background template. • Endpoint energy of the detector that was determined • Relative gains of PMTs. • Unusual conditions detected, like high background rates, or contamination • Data channel corresponding to peak maximum • Version of calibration algorithm used The text must be UTF-8 encoded.	

4.2.23 RadUseExternalEnergyCalInstructions Enumeration

Enumeration to convey instructions on the use of an energy calibration.

Used in message: `RadUseExternalEnergyCalRequest`

Underlying integral representation: `uint8_t`

Enumerated values for `RadUseExternalEnergyCalInstructions`:

UseThisCalibrationAndSelfUpdateAsNormalValue Value 0x00

Use this calibration, but proceed with normal self calibration procedures in the future if the device has those capabilities.

DoNotPerformSelfCalibrationValue Value 0x01

Accept this energy calibration as fixed, and do not attempt to change it using any built in calibration procedures. Useful if default calibration routine is failing, or an externally provided routine (e.x., a command device is constructed for this purpose) is to be used.

ResetToDefaultCalibrationValue Value 0x02

Do not use the energy calibration sent, if any, but instead reset back to the factory default calibration. Useful if a calibration has previously been set with the `RadUseExternalEnergyCalInstructions::DoNotPerformSelfCalibrationValue` option, or the auto-calibration has wandered into an incorrect area and is no longer producing sensical results.

4.2.24 RadUseExternalEnergyCalRequest Message

Request sent to a device to use the energy calibration contained in this message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x08
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x66
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
calibration_timestamp	UtcTimePoint
Field Description: When the calibration was completed. Will be in the past, but not the future. If this energy calibration is empty or not valid, this value will be zero.	
energy_cal_coefficient_type	uint8_t enumerated by EnergyCalCoefficientType
Field Description: The types of coefficients used by the calibration.	
calibration_status	uint8_t enumerated by EnergyCalibrationStatus
Field Description: The status of the energy calibration, such as good, out of date, etc.	
subdetector_number	uint8_t
Field Description: The subdetector number this calibration is for. If a module only has one detection crystal/He3-tube, this will always be number one. If it has N detection sensors, this value will range from 1 to N (inclusive) according to the subdetector this calibration is for. See also: • DeviceInfoReply::num_subdetectors • RadListModeDataPush::subdetector_number • RadChannelDataPush::subdetector_number	
energy_cal_method_flags	uint16_t bits defined by EnergyCalMethodFlags
Field Description: A bitwise OR of flags representing energy calibration source and method.	
coefficients	Medium Array of float
Field Description: The coefficients necessary to specify the calibration.	
coefficient_uncertainties	Medium Array of float

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The uncertainty on the coefficients; may have zero entries, the same number of entries as the number of coefficients, twice the number of coefficients, or the number of coefficients squared. The interpretation of the coefficients is dependent on the number of entries and the options are: • No entries: uncertainty is not specified. • Same number of entries as coefficients: symmetrized 1-sigma uncertainties. • Twice the number of entries as coefficients: the positive 1-sigma uncertainties for each coefficient, then the negative 1-sigma uncertainties (e.g., the first N coefficients are positive uncertainties, followed the N negative uncertainties). • The square of number of entries as coefficients: the covariance matrix with entries of the first row listed first, followed by entries of second row, etc. To avoid an ambiguous situation, when there are two coefficients, the covariance matrix can not be specified, but as a work around, a 3rd zero valued coefficient could be added and then the matrix specified.	
`deviation_pairs`	`Short Array` of `float`
Field Description: Deviation pairs, as in N42.42, are each two `float`s, first the true energy, then the offset, then next entry in array is the next true energy, and its offset, etc. True energies and offsets are in keV. Unless the `RadDetDeviceFeaturesFlags::SupportsDeviationPairCalFlag` bit is set in `RadSubDetectorInfo::subdetector_features`, no values for this field may be provided. An example procedure for using deviation pairs: • Determine first (offset) and second (gain) polynomial coefficients using the 239 keV and 2614 keV peaks of Th232 • If the k-40 1460 keV peak is now at 1450 keV, a deviation of 10 keV should be set at 1460 keV • Deviation pairs set to zero would be defined for 239 keV and 2614 keV • The value provided in this field would then be [239,0,1460,10,2614,0]. Cubic spline interpolation is used to interpolate between deviation pairs. When solving for the spline coefficients, the second derivative is set to zero at the lowest deviation pair energy, and the first derivative is set to zero at the highest deviation pair energy. Corrections for energies below the lowest value deviation pair are always equal to the offset of the lowest energy deviation pair, and corrections for energies above the highest energy deviation pair are always equal to the offset of the highest energy deviation pair. Note: uncertainties for `RadUseExternalEnergyCalRequest::deviation_pairs` can not be specified since these are typically taken as well defined fixed quantities, and the polynomial coefficient uncertainties will account for the uncertainties. Note: although non-linear deviation pairs are used within the N42.42-2012 standard, the specification doesnt appear to provide an adequate definition of the,; the open source SpecUtils library, available at https://github.com/sandialabs/SpecUtils/, provides an implementation of non-linear deviation pairs that may be useful for reference.	
`method_description`	`Short String`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: A free-form description of method used to derive calibration. Examples might be: • "Fixed Factory Calibration" • "Fit to NORM templates" • "Gain matched for Th232 2614 keV and K40 1460 keV peaks" • "Individual PMTs gain matched to each other, followed by a fit to NORM templates" • "Gain matched to internal Cs137 seed" • "5 minute spectrum using Eu152" • etc.	
`calculation_notes`	`Medium String`
Field Description: Free form text providing additional implementation specific details related to quality of energy calibration procedure. Examples of information that might be given in this text are: • Chi2 of fit to background template. • Endpoint energy of the detector that was determined • Relative gains of PMTs. • Unusual conditions detected, like high background rates, or contamination • Data channel corresponding to peak maximum • Version of calibration algorithm used The text must be UTF-8 encoded.	
`use_instructions`	`uint8_t` enumerated by `RadUseExternalEnergyCalInstructions`
Field Description: Options for implementing the provided calibration: use, do or do not proceed with auto-calibration, and reset to factory default calibration.	

4.2.25 EnergyCalUseStatus Enumeration

Enumeration of values to describe the status of an energy calibration for a radiation detector. Used by a radiation detector module when replying to a request by the control module for a change to a calibration supplied by the control module.

Used in message: `RadUseExternalEnergyCalReply`

Underlying integral representation: `uint8_t`

Enumerated values for `EnergyCalUseStatus`:

EnergyCalUsingCalibrationValue Value 0x00

The calibration received from the control module is being used.

EnergyCalUnsupportedCalibrationTypeValue Value 0x01

The calibration received from the control module is a type that is not supported by the detector.

EnergyCalDeviceDoesNotAcceptCalibrationValue Value 0x02

The calibration received from the control module has been rejected by the detector.

EnergyCalInvalidCalibrationValue Value 0x03

The calibration received from the control module is invalid.

EnergyCalInvalidSubDetectorValue Value 0x04

The calibration received from the control module was directed to an invalid subdetector.

4.2.26 RadUseExternalEnergyCalReply Message

Message replying to the control module's request that the provided energy calibration be applied.

Message Contents:

Field Name	Data Type
`message_group` *Field Description*: The RAPTER `MessageGroup` this message corresponds too.	`uint8_t` with value 0x08
`message_type` *Field Description*: The type of message within the specified message_group, that this message corresponds to.	`uint8_t` with value 0xe6
`message_group_version` *Field Description*: The version of the message_group being used.	`uint8_t`
`message_flags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	`uint8_t` bits defined by `MsgFlags`
`message_id` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	`uint32_t`
`change_effective_timestamp` *Field Description*: Timestamp for when the calibration will become effective. If the calibration won't be used, this must be 0.	UtcTimePoint
`use_status` *Field Description*: Status for the use of the calibration that was sent to the detector. For any value other than `EnergyCalUseStatus::EnergyCalUsingCalibrationValue`, consider sending a separate `NotificationPush` message further explaining the issue.	`uint8_t` enumerated by `EnergyCalUseStatus`

4.3 Vehicle Presence Interface

4.3.1 VehiclePresenceMsgType Enumeration

Values to indicate the type of message being sent as part of the `MessageGroup::VehiclePresenceVal` message group. In this message group, push messages originate at the vehicle presence modules, and request messages originate at the control module. The first byte of messages shall have a value of `MessageGroup::VehiclePresenceValue`, and the second byte of the message will have a value as indicated by this enum, which

will then tell you how to decode the message. For example a second byte value of VehiclePresenceSubDetectorInformationRequest will tell you the message contents are specified by VehiclePresenceSubDetectorInformationRequest.

Underlying integral representation: uint8_t

Enumerated values for VehiclePresenceMsgType:

VehiclePresenceSubDetectorInformationRequestValue Value 0x71

Message sent from the control module to a vehicle presence module requesting information about one of its subdetectors (break beam, distance, camera, etc).

See also: VehiclePresenceSubDetectorInformationRequest

VehiclePresenceSubDetectorInformationReplyValue Value 0xf1

Reply sent in response to a VehiclePresenceSubDetectorInformationRequest message containing information about a subdetector.

See also: DataOutDevicesInfoReply

VehiclePresenceBinaryDataPushValue Value 0x72

Message sent from a vehicle presence module to the control module when one of the binary presence subdetectors (e.g., IR break-beam sensor) changes state.

See also: VehiclePresenceBinaryDataPush

VehiclePresenceBinaryDataPushAckValue Value 0xf2

Acknowledgement of receiving a VehiclePresenceBinaryDataPush message.

See also: VehiclePresenceBinaryDataPushAck

VehiclePresenceCurrentBinaryDataRequestValue Value 0x73

Message sent from the control module to a vehicle presence module requesting the current status of its binary style sensors.

See also: VehiclePresenceCurrentBinaryDataRequest

VehiclePresenceCurrentBinaryDataReplyValue Value 0xf3

Reply to a VehiclePresenceCurrentBinaryDataRequest containing the current status of the binary sensors (e.g., if the IR beams are occluded or not).

See also: VehiclePresenceCurrentBinaryDataReply

VehiclePresenceReadingPushValue Value 0x74

Message from the vehicle presence module to the control module when a new distance or speed measurement is available.

See also: VehiclePresenceReadingPush

VehiclePresenceReadingPushAckValue Value 0xf4

Acknowledgement of receiving a VehiclePresenceReadingPush message.

See also: VehiclePresenceReadingPushAck

VehiclePresenceCurrentReadingRequestValue Value 0x75

Message sent from the control module to a vehicle presence module requesting the current status of a distance or speed sensor.

See also: VehiclePresenceCurrentReadingRequest

VehiclePresenceCurrentReadingReplyValue Value 0xf5

Message sent in response to a VehiclePresenceCurrentReadingRequest message, containing if a distance or speed measurement is available (there may be nothing to measure), and if so, it's value.

See also: VehiclePresenceCurrentReadingReply

VehiclePresenceImagePushValue Value 0x76

Message sent from vehicle presence module to the control module containing an image; the module decides when to send these.

See also: VehiclePresenceImagePush

VehiclePresenceImagePushAckValue Value 0xf6

Acknowledgement of receiving a VehiclePresenceReadingPush message.

See also: VehiclePresenceImagePushAck

VehiclePresenceCurrentImageRequestValue Value 0x77

Message sent from the control module to a vehicle presence device requesting a current image.

See also: VehiclePresenceCurrentImageRequest

VehiclePresenceCurrentImageReplyValue Value 0xf7

Reply to a VehiclePresenceCurrentImageRequest message containing the a current image, or otherwise error.

See also: VehiclePresenceCurrentImageReply

4.3.2 VehiclePresenceSubDetectorInformationRequest Message

A request for the vehicle presence module to send information about all of it's subdetectors.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x71
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.3.3 VehiclePresenceSubDetectorType Enumeration

Enumeration to describe the type of vehicle presence subdetector.

See also: MessageGroup::VehiclePresenceValue

Used in message: VehiclePresenceReadingPush

Underlying integral representation: uint16_t

Enumerated values for VehiclePresenceSubDetectorType:

BreakBeamPresenceValue Value 0x01

Device contains a binary yes/no style presence sensor. For example, a breakbeam style sensor.

VehicleDistanceValue Value 0x02

Device provides a distance of the item, relative to itself.

See also: CmdDeviceSetReferenceInfoRequest

VehicleSpeedValue Value 0x04

Device provides a speed of the item.

ImageValue Value 0x08

Device provides an image. There is currently no message available to query or change resolution.

4.3.4 VehiclePresenceSubDetectorInformation Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold information about a vehicle presence subdetector.

Message Contents:

Field Name	Data Type
subdetector_number	uint8_t
Field Description: The subdetector number of the vehicle presence subdetector that is being described.	

Continued on next page

Table continued from previous page

Field Name	Data Type
subdetector_type	uint16_t enumerated by VehiclePresenceSubDetectorType
Field Description: A UTF-8 encoded, human readable description of this subdetector.	
subdetector_description	Short String
Field Description: A UTF-8 encoded, human readable description of this subdetector.	

4.3.5 VehiclePresenceSubDetectorInformationReply Message

Message from a vehicle presence module containing information on all of the subdetectors in this vehicle presence module.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xf1
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
informations	Short Array of VehiclePresenceSubDetectorInformation
Field Description: Information about all sub-detectors in this module; must contain exactly one entry for each of the subdetectors. Length should be equal to DeviceInfo::num_subdetectors. Sub-detectors start with 1 and go to DeviceInfo::num_subdetectors.	

4.3.6 RecommendedPresenceStatus Enumeration

Enumeration to convey the recommendation from the vehicle presence module as to whether the portal is occupied, as determined from the vehicle presence sensors (e.g., binary break-beam style sensors). Note, if you are not going to include any binary style sensors but you would still like to provide a binary occupied/not-occupied status you could define a binary subdetector that doesn't exist physically, but is determined by a camera or distance sensor.

Used in message: VehiclePresenceBinaryDataPush

Underlying integral representation: uint8_t

Enumerated values for `RecommendedPresenceStatus`:

NotOccupiedValue Value 0x00

The recommendation is that the portal is not occupied.

OccupiedValue Value 0x01

The recommendation is that the portal is occupied.

OccupancyUndeterminedValue Value 0x02

Unable to make a suggestion; either because of an error, or inherent limitation.

4.3.7 VehiclePresenceReadOutFlags Enumeration

Flags that may be bitwise OR'd to describe error conditions relating to vehicle presence data.

Underlying integral representation: `uint32_t`

Enumerated values for `VehiclePresenceReadOutFlags`:

SubDetectorIsSuspectFlag Value 0x01

Indicates that a one or a few subdetectors (e.x., individual beams, or distance sensors) are suspect.

ReadoutExternallyInterruptedFlag Value 0x02

Indicates that an external interrupt (ex panel door being opened) has prevented taking data

ReadoutSystemSuspectFlag Value 0x04

Indicates that occupancy systems health is suspect, and the data may not be reliable.

ReadoutDataCouldNotBeCollectedFlag Value 0x08

Indicates that this data could not be read out.

ReadoutNotAcquiringDataFlag Value 0x10

Indicates that the detector is not taking data, and that this message does not contain valid data (the message must still be in a valid format to allow parsing, but but should be considered to not contain useful vehicle presence data). This bit is useful for when current data of a occupancy detector is requested but the detector isn't in acquisition mode.

ReadOutTargetNotAcquiredFlag Value 0x20

Indicates the sensor was unable to readout a distance or speed as there was no vehicle within its range.

4.3.8 VehiclePresenceBinaryStatus Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold measured vehicle presence data from a binary vehicle presence detector.

Message Contents:

Field Name	Data Type
detection_timestamp	UtcTimePoint
Field Description: The time when the change of occupancy was detected. Microseconds since the UNIX epoch.	
subdetector_number	uint8_t
Field Description: Subdetector number that is reporting the change of occupancy.	
readout_status_flags	uint32_t bits defined by VehiclePresenceReadOutFlags
Field Description: Bitwise OR of flags to indicate state of the vehicle presence data included in this struct. If zero, all is good.	
item_present	uint8_t
Field Description: If nonzero, the subdetector is reporting that an item is present; for a break-beam occupancy sensor this would imply that the beam is broken.	

4.3.9 VehiclePresenceBinaryDataPush Message

This message represents data reported by a occupancy sensor, which may have multiple beams, each of which yield a yes-or-no answer as to whether or not there is an occupancy. Only sub-detectors that have changed need to be included, while when part of VehiclePresenceCurrentBinaryDataReply message, the most recent data from all sub-detectors must be included. Note, no specification is made as to the internal sampling frequency the machine will operate at (ex. some devices may record up to 500 changes per second, some only 10, and others in real time). Current sensor values may be requested using a VehiclePresenceCurrentBinaryDataRequest command.

See Also: VehiclePresenceCurrentBinaryDataRequestValue

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x72
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
presence_recommendation	uint8_t enumerated by RecommendedPresenceStatus

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Recommendation by a vehicle presence module for if the portal is occupied. Timestamp of the recommendation is implied to be the latest one in VehiclePresenceBinaryDataPush::binary_presence_data.	
binary_presence_data	Short Array of VehiclePresenceBinaryStatus
Field Description: Each binary (break-beam type) vehicle presence sensor may have at most one entry in this array and time stamps may not span (come both before and after) a heartbeat. Must have an entry from at least one sensor.	

4.3.10 VehiclePresenceBinaryDataPushAck Message

A message acknowledging the receipt by the control module of a VehiclePresenceBinaryDataPush message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xf2
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.3.11 VehiclePresenceCurrentBinaryDataRequest Message

Requests the current data from the occupancy sensor, when the message is received by the device.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x73
Field Description: The type of message within the specified message_group, that this message corresponds to.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.3.12 VehiclePresenceReplyStatus Enumeration

Enumeration to hold an indicator of whether or not a query for current data was successful.

Used in message: VehiclePresenceCurrentBinaryDataReply

Underlying integral representation: uint8_t

Enumerated values for VehiclePresenceReplyStatus:

InformationAvailableValue Value 0x00

The device is able to provide the requested information.

InvalidSubdetectorNumberValue Value 0x01

The subdetector number provided does not exist.

UnsupportedDataTypeValue Value 0x02

The subdetector number requested does not produce the type of data requested.

UnableToMeasureValue Value 0x03

Measurement was unable to be made. Can happen for example if there is no vehicle present to measure its distance, or if the sensor is busy reading out another measurement.

NotOperatingValue Value 0x04

The device is not in the OperatingMode::OperatingValue mode.

ReadoutErrorValue Value 0x05

There was an error reading out the requested sensor. A NotificationPush may separately be sent to provide more information.

4.3.13 VehiclePresenceCurrentBinaryDataReply Message

A reply from a vehicle presence module to a VehiclePresenceCurrentBinaryDataRequest. If the device is not operating (OperatingMode::OperatingValue) then the VehiclePresenceCurrentBinaryDataReply::reply_status field should have the value of VehiclePresenceReplyStatus::NotOperatingValue. If there is a readout error then VehiclePresenceCurrentBinaryDataReply::reply_status

will have a value of VehiclePresenceReplyStatus::ReadoutErrorValue and a NotificationPush message may be separately sent to provide further information.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xf3
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
reply_status	uint8_t enumerated by VehiclePresenceReplyStatus
Field Description: Status of the reply, as enumerated by VehiclePresenceReplyStatus.	
presence_recommendation	uint8_t enumerated by RecommendedPresenceStatus
Field Description: Recommendation by a vehicle presence module for if the portal is occupied. Timestamp of the recommendation is implied to be the latest one in VehiclePresenceCurrentBinaryDataReply::binary_presence_data.	
binary_presence_data	Short Array of VehiclePresenceBinaryStatus
Field Description: Each binary (break-beam type) vehicle presence sensor may have at most one entry in this array and time stamps may not span (come both before and after) a heartbeat. Must have an entry from at least one sensor.	

4.3.14 VehiclePresenceReadingPush Message

Sent from a vehicle presence module to the control module when something is being measured (e.g., it thinks its measuring something that might be a vehicle) and a new relevant distance measurement is available. The device itself determines when to send this which might be whenever the sensor can take another measurement, or so many times per second (which is a settable parameter, see ParameterInfoRequest, could control).

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_type	uint8_t with value 0x74 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
detection_timestamp	UtcTimePoint *Field Description*: The time the distance measurement was made. Microseconds since the UNIX epoch.
subdetector_number	uint8_t *Field Description*: The subdetector number of the subdetector that is reporting this distance.
subdetector_type	uint16_t enumerated by VehiclePresenceSubDetectorType *Field Description*: The subdetector type this reading corresponds too. Will be either VehiclePresenceSubDetectorType::VehicleDistanceValue or VehiclePresenceSubDetectorType::VehicleSpeedValue. This type must that given by VehiclePresenceSubDetectorInformationReply for the specified subdetector.
readout_status_flags	uint32_t bits defined by VehiclePresenceReadOutFlags *Field Description*: Bitwise OR of flags to indicate state of the distance data reported with this struct. If zero, all is good.
reading_is_useable	uint8_t *Field Description*: If nonzero, the measured distance or speed is useable. If zero, the measured distance or speed should not be used.
reading_value	float *Field Description*: Either the measured speed (in meters per second), or the measured distance (in centimeters), as determined by VehiclePresenceReadingInfo::subdetector_type. The distance from the subdetector to the item in centimeters. When used with the subdetector's position and orientation, an unambiguous location of one point of the item is specified. If no item is being measured, then a value of zero should be reported.

4.3.15 VehiclePresenceReadingPushAck Message

A message acknowledging the receipt by the control module of a VehiclePresenceReadingPush

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0xf4 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.3.16 VehiclePresenceCurrentReadingRequest Message

A request from the control module for the distance measured by a given sensor when this message is received by the vehicle presence sensor.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x75 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
subdetector_number	uint8_t *Field Description*: The subdetector number from which a measurement of distance is wanted.

4.3.17 VehiclePresenceCurrentReadingReply Message

Message from a vehicle presence module containing the currently measured distance using the specified sensor, or if there was an issue reading it out, that issue.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0xf5 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
reply_status	uint8_t enumerated by VehiclePresenceReplyStatus *Field Description*: The success status of reading out the current distance.
detection_timestamp	UtcTimePoint *Field Description*: The time the distance measurement was made. Microseconds since the UNIX epoch.
subdetector_number	uint8_t *Field Description*: The subdetector number of the subdetector that is reporting this distance.
subdetector_type	uint16_t enumerated by VehiclePresenceSubDetectorType *Field Description*: The subdetector type this reading corresponds too. Will be either VehiclePresenceSubDetectorType::VehicleDistanceValue or VehiclePresenceSubDetectorType::VehicleSpeedValue. This type must that given by VehiclePresenceSubDetectorInformationReply for the specified subdetector.
readout_status_flags	uint32_t bits defined by VehiclePresenceReadOutFlags *Field Description*: Bitwise OR of flags to indicate state of the distance data reported with this struct. If zero, all is good.
reading_is_useable	uint8_t *Field Description*: If nonzero, the measured distance or speed is useable. If zero, the measured distance or speed should not be used.
reading_value	float *Field Description*: Either the measured speed (in meters per second), or the measured distance (in centimeters), as determined by VehiclePresenceReadingInfo::subdetector_type. The distance from the subdetector to the item in centimeters. When used with the subdetector's position and orientation, an unambiguous location of one point of the item is specified. If no item is being measured, then a value of zero should be reported.

4.3.18 VehiclePresenceImageType Enumeration

Enumeration to hold the file type of an image contained in the message.

Used in message: VehiclePresenceImagePush

Underlying integral representation: uint8_t

Enumerated values for VehiclePresenceImageType:

NoImageValue Value 0x00

No image is present.

JpegValue Value 0x01

Joint Photographic Experts Group (JPEG)

PngValue Value 0x02

Portable Network Graphics (PNG)

GifValue Value 0x03

Graphics Interchange Format (Gif)

BmpValue Value 0x04

Bitmap image.

PdfValue Value 0x05

Portable Document Format (PDF)

SvgValue Value 0x06

Scalable Vector Graphics (SVG)

4.3.19 VehiclePresenceImagePush Message

Image sent from a vehicle presence module to the control module. This message is sent when the vehicle presence module decides to send it, and may for example be when the vehicle enters or exits the portal. To make it adjustable when, or if, the images are taken, use the parameters mechanism (see ParameterInfoReply).

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x76
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
timestamp	UtcTimePoint
Field Description: Time image was taken. Microseconds since the UNIX epoch.	
subdetector_number	uint8_t
Field Description: The number of the imaging sub-detector (camera) that took the image.	
readout_status_flags	uint32_t bits defined by VehiclePresenceReadOutFlags
Field Description: Bitwise OR of flags to indicate state of the camera that took the picture, and status of the data. If zero, all is good.	
image_type	uint8_t enumerated by VehiclePresenceImageType
Field Description: The file type of the image being transferred.	
image_data	Large Array of uint8_t
Field Description: The image data corresponding to a standard image file.	

4.3.20 VehiclePresenceImagePushAck Message

A message acknowledging the receipt by the control module of a VehiclePresenceImagePush message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x10
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xf6
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.3.21 VehiclePresenceCurrentImageRequest Message

A request from the control module for a picture to be taken as soon as the message is received.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x10
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x77
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`subdetector_number`	`uint8_t`
Field Description: The subdetector number of the camera the pictured is wanted from.	

4.3.22 VehiclePresenceCurrentImageReply Message

Message from a vehicle presence module containing the requested current image, or if not, the reason why not.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x10
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xf7
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`reply_status`	`uint8_t` enumerated by `VehiclePresenceReplyStatus`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The success status of being able to fulfill the request.	
`timestamp`	UtcTimePoint
Field Description: Time image was taken. Microseconds since the UNIX epoch.	
`subdetector_number`	`uint8_t`
Field Description: The number of the imaging sub-detector (camera) that took the image.	
`readout_status_flags`	`uint32_t` bits defined by `VehiclePresenceReadOutFlags`
Field Description: Bitwise OR of flags to indicate state of the camera that took the picture, and status of the data. If zero, all is good.	
`image_type`	`uint8_t` enumerated by `VehiclePresenceImageType`
Field Description: The file type of the image being transferred.	
`image_data`	`Large Array` of `uint8_t`
Field Description: The image data corresponding to a standard image file.	

4.4 Power Management

4.4.1 PowerManagementMsgType Enumeration

Values to indicate the type of message being sent as part of the Power Management Interface. In this message group, push messages originate at the power management module (PMU), and request messages originate at the control module. The first byte of messages shall have a value of `MessageGroup::PowerManagementValue` (0x20), and the second byte of the message will have a value as indicated by this enum, which will then tell you how to decode the message. For example a second byte value of `PowerManagementMsgType::PwrMngmtInformationRequestValue` will tell you the message contents are specified by `PwrMngmtInformationRequest` message.

Underlying integral representation: `uint8_t`

Enumerated values for `PowerManagementMsgType`:

PwrMngmtInformationRequestValue Value 0x56

A request from the control module to the power management module for static information (ie., information that doesn't change, so not things like realtime voltage or current) about the output lines and supply portion of the power management module.

See also: `PwrMngmtInformationRequest`

PwrMngmtInformationReplyValue Value 0xd6

A reply from the power management module to the control module in response to a `PwrMngmtInformationRequest` message. Provides information on the capabilities of the line-outs and supply portion of PMM. Does not provide dynamic status.

See also: `PwrMngmtInformationReply`

PwrMngmtLineOutStatusRequestValue Value 0x58

A request from the control module to the power management module for dynamic information (i.e., power source, voltage, current, etc) about a line out.

See also: `PwrMngmtLineOutStatusRequest`

PwrMngmtLineOutStatusReplyValue Value 0xd8

A reply from the power management module to the control module in response to a `PwrMngmtLineOutStatusRequest` message. Provides dynamic information about the line out such as power source, voltage, current, etc.

See also: `PwrMngmtLineOutStatusReply`

PwrMngmtSupplyStatusRequestValue Value 0x59

A request from the control module to the power management module for dynamic information about the mains power, batteries, charging or inverting circuitry and other components that could cause issues.

See also: `PwrMngmtSupplyStatusRequest`

PwrMngmtSupplyStatusReplyValue Value 0xd9

A reply from the power management module to the control module in response to a `PwrMngmtSupplyStatusRequest` message. Provides dynamic information about mains (supply) power source, such as voltage, current being used, and any issues.

See also: `PwrMngmtSupplyStatusReply`

PwrMngmtLineOutEventPushValue Value 0x5a

A push message from the power management module to the control module notifying it of a condition enumerated by `PwrMngmtLineOutStatusFlags` has changed; ex, switch to on battery power, overloaded, switch back to mains power, etc. See `PwrMngmtLineOutEventPush` for message contents and format.

PwrMngmtLineOutEventPushAckValue Value 0xda

Acknowledgement of receiving a `PwrMngmtLineOutEventPush` message.

See also: `PwrMngmtLineOutEventPushAck`

PwrMngmtSupplyEventPushValue Value 0x5b

A push message from the power management module to the control module notifying it that one of the conditions enumerated by `PwrMngmtSupplyStatusFlags` has changed (ex. over voltage, battery fault, etc). See `PwrMngmtSupplyEventPush` for message contents and format.

PwrMngmtSupplyEventPushAckValue Value 0xdb

Acknowledgement of receiving a `PwrMngmtSupplyEventPush` message.

See also: `PwrMngmtSupplyEventPushAck`

PwrMngmtSelfTestRequestValue Value 0x5c

A request from the control module to the power management module asking the device to perform a self-test. The types of self-tests that can be requested are enumerated in PwrMngmtTestType.

See also: PwrMngmtTestType , PwrMngmtSelfTestRequest

PwrMngmtSelfTestReplyValue Value 0xdc

A reply from the power management module to the control module in response to a PwrMngmtSelfTestRequest message. There may be multiple replies to a single request as some tests can take significant time.

See also: PwrMngmtSelfTestReply

PwrMngmtAutomaticSelfTestResultPushValue Value 0x5d

A push message from the power management module to the control module notifying it of a automated self test has started, in progress, aborted, or completed. Some power management units will perform periodic tests to ensure readiness for mains power failure. This message informs the control module of the results of the test.

See also: PwrMngmtAutomaticSelfTestResultPush

PwrMngmtAutomaticSelfTestResultPushAckValue Value 0xdd

Acknowledgement of receiving a PwrMngmtAutomaticSelfTestResultPush message.

See also: PwrMngmtAutomaticSelfTestResultPushAck

PwrMngmtLineOutPowerCycleRequestValue Value 0x5e

A request from the control module to the power management module asking the device to turn a line out off and then back on, with potential delays for both the off and on actions. PwrMngmtLineOutEventPush message will be sent with the PwrMngmtLineOutStatusFlags::PwrMngmtLineOutImminentShutdownFlag flag set in the PwrMngmtLineOutEventPush::lineout_status field.

See also: PwrMngmtLineOutPowerCycleRequest

PwrMngmtLineOutPowerCycleReplyValue Value 0xde

A reply from the power management module to the control module in response to a PwrMngmtLineOutPowerCycleReply message.

4.4.2 PwrMngmtInformationRequest Message

A request from the control module to the power management module for static information (ie, information that doesn't change, so not things like realtime voltage or current) about a output line.

See Also: PwrMngmtInformationReply

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x20
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x56
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.4.3 PwrMngmtLineOutProtectionType Enumeration

Enumeration to specify the the type of backup protection the line-out corresponds to, using the standard terminology of uninterruptible power supplies.

Underlying integral representation: `uint8_t`

Enumerated values for `PwrMngmtLineOutProtectionType`:

PwrMngmtStandbyTypeValue Value 0x01

The line-out power is directly tied to the input (mains) power, and the inverting circuitry (DC to AC circuitry) is idle until a power disruption occurs. Switching to battery power may lead to a few milliseconds disruption in power output. If the line-out supplies DC voltage and there may be a glitch when switching from mains power to battery power, specify this type.

PwrMngmtDoubleConversionTypeValue Value 0x02

The entire line-out power is always provided by the inverting circuitry from the batteries; the mains power is converted to DC to supply the batteries. Mains power loss or disruptions typically do not cause any interruption to the output power. If the line-out directly provides DC power that is supplied from the battery through a regulated mechanism (ex, DC to DC voltage conversion), specify that it is this type.

PwrMngmtDeltaConversionTypeValue Value 0x04

Some amount of the output power skips the AC to DC back to AC processing stage, to create a more power efficient process but with potentially some loss of protections of the double conversion type.

PwrMngmtLineInteractiveTypeValue Value 0x08

Line interactive line-outs have an ability to automatically regulate voltage to correct low and high mains voltages, without needing to convert to DC.

PwrMngmtNoBatteryPowerValue Value 0x10

The power management module does not provide and buffer from mains losing power.

4.4.4 PwrMngmtLineOutPropertiesFlags Enumeration

Properties of a given lineout. Which properties are marked will trigger `PwrMngmtLineOutEventPush` messages to be sent, and which fields of `PwrMngmtLineOutEventPush` and `PwrMngmtLineOutStatusReply` are expected to be valid.

Underlying integral representation: `uint32_t`

Enumerated values for `PwrMngmtLineOutPropertiesFlags`:

PwrMngmtLineOutCanBePoweredByBatteryFlag Value 0x01

If mains power is lost, this lineout will be powered from the battery.

See also: `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOnBatteryPowerFlag`

PwrMngmtLineOutPowerCanBeCycledFlag Value 0x02

If marked, line can be shut off, and after a delay, turned back on using a `PwrMngmtLineOutPowerCycleRequest` message. Useful for forcing a hard reboot of non-responding devices.

PwrMngmtLineOutVoltageIsMeasuredFlag Value 0x04

The output voltage is actively monitored and reported using `PwrMngmtLineOutStatus::voltage`

PwrMngmtLineOutCurrentIsMeasuredFlag Value 0x08

The output voltage is actively monitored and reported using `PwrMngmtLineOutStatus::current`

PwrMngmtLineOutFrequencyIsMeasuredFlag Value 0x10

The output frequency is actively monitored and reported using `PwrMngmtLineOutStatus::freque`

PwrMngmtLineOutNoiseIsMeasuredFlag Value 0x20

The output is actively monitored for noise conditions.

See also: `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOutOfToleranceFlag`

PwrMngmtLineOutCanReportFaultsFlag Value 0x40

Output faults can be detected and reported.

See also: `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOutputLineFaultFlag`

PwrMngmtLineOutSupportsBypassFlag Value 0x80

The line out may go into bypass mode. This may be due to a manual intervention, power draw overload (while mains power is available), or malfunction.

PwrMngmtLineOutSupportsOverloadWarningFlag Value 0x100

The device monitors for conditions approaching output overload, and will send a `PwrMngmtLineOutEventPush` message when detected.

See also: `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOutputOverloadWarnin`

PwrMngmtLineOutSupportsOverloadProtectionFlag Value 0x200

The device monitors for overload conditions and will take protective actions when necessary.

See also: PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOutputOverloadDetect

PwrMngmtLineOutSupportsBuckFlag Value 0x400

The mains voltage is being actively lowered. Usually applicable to line-interactive units.

See also: PwrMngmtLineOutStatusFlags::PwrMngmtLineOutBuckOnFlag

PwrMngmtLineOutSupportsBoostFlag Value 0x800

The mains voltage is being actively increased. Usually applicable to line-interactive units.

See also: PwrMngmtLineOutStatusFlags::PwrMngmtLineOutBoostOnFlag

PwrMngmtLineOutMonitorsDiagnosticsFlag Value 0x1000

The device will report issues with diagnostics if they happen. For example if voltage or current cant be read, or internal diagnostics fail.

See also: PwrMngmtLineOutStatusFlags::PwrMngmtLineOutDiagnosticsFailFlag

4.4.5 PwrMngmtSupplyPropertiesFlags Enumeration

Features of the "supply" portion of the power management unit. Which features are marked as supported will determine what type of events may trigger a PwrMngmtSupplyEventPush message to be sent, and what fields of PwrMngmtSupplyEventPush and PwrMngmtSupplyStatusRe and PwrMngmtBatteryStatus are expected to be valid.

Underlying integral representation: uint32_t

Enumerated values for PwrMngmtSupplyPropertiesFlags:

PwrMngmtSupplyMainsVoltageIsMeasuredFlag Value 0x01

The power management module actively measures and monitors the mains voltage.

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyMainsOverVoltageFlag , PwrMngmtSupplyStatusFlags::PwrMngmtSupplyMainsUnderVoltageFlag

PwrMngmtSupplyMainsCurrentIsMeasuredFlag Value 0x02

The power management module actively measures and monitors the mains current.

See also: PwrMngmtSupplyStatus::mains_current_amps

PwrMngmtSupplyMainsFrequencyIsMeasuredFlag Value 0x04

The power management module actively measures and monitors the mains frequency.

See also: PwrMngmtSupplyStatusReply::mains_frequency_hz

PwrMngmtSupplyMainsNoiseIsMeasuredFlag Value 0x08

The power management module actively measures and monitors the mains noise (e.g., looks for spike, or sags, etc).

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyMainsNoisyFlag

PwrMngmtSupplyBatteryVoltageIsMeasuredFlag Value 0x10

The power management module actively measures and monitors the battery voltage.

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyBatteryLowCapacityFlag, PwrMngmtSupplyStatusFlags::PwrMngmtSupplyBatteryDepletedFlag

PwrMngmtSupplyBatteryCurrentIsMeasuredFlag Value 0x20

The power management module actively measures and monitors the battery current.

See also: PwrMngmtLineOutStatus::current_amps

PwrMngmtSupplyBatteryFaultsCanBeDetectedFlag Value 0x40

The power management module actively measures and monitors for battery faults.

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyBatteryFaultFlag

PwrMngmtSupplyAmbientTemperatureIsMeasuredFlag Value 0x80

The power management module actively measures and monitors the ambient temperature (i.e., the module enclosure).

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyAmbientTemperatureBad

PwrMngmtSupplyBatteryTemperatureIsMeasuredFlag Value 0x100

The power management module actively measures and monitors the battery temperature(s).

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyBatteryTemperatureBad

PwrMngmtSupplyOtherTemperatureIsMeasuredFlag Value 0x200

The power management module actively measures and temperature(s) other than battery and ambient.

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyOtherTemperatureBadFla

PwrMngmtSupplyFanIsMonitoredFlag Value 0x400

The power management module includes a fan that is actively monitored for operation.

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyFanFailureFlag,

PwrMngmtSupplyHumidityIsMonitoredFlag Value 0x800

The power management module actively monitors relative humidity.

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyHumidityBadFlag

PwrMngmtSupplyWaterPresenceIsMonitoredFlag Value 0x1000

The power management module actively monitors for the presence of water.

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplyWaterDetectedFlag

PwrMngmtSupplyDiagnosticsIsMonitoredFlag Value 0x2000

The power management module is capable of detecting when a diagnostic measurements fail.

See also: PwrMngmtSupplyStatusFlags::PwrMngmtSupplySupplyDiagnosticsFail

4.4.6 PwrMngmtLineOutInformation Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Struct to hold the information relevant to line-outs of a `PwrMngmtInformationReply` message.

Message Contents:

Field Name	Data Type
`subdetector_number`	`uint8_t`
Field Description: The line out number to which this message corresponds. Must be between one and `DeviceInfoReply::num_subdetectors`, inclusive.	
`subdetector_type`	`uint8_t` enumerated by `PwrMngmtLineOutProtectionType`
Field Description: The type of power protection provided by this line out.	
`subdetector_features`	`uint32_t` bits defined by `PwrMngmtLineOutPropertiesFlags`
Field Description: Features of this line-out	
`nominal_output_frequency_hz`	`float`
Field Description: The nominal output frequency, in hertz, of the power supplied for this line-out. DC power output will have a value of zero, and standard US AC will have a value 60.	
`nominal_output_voltage_volts`	`float`
Field Description: The nominal output voltage for the line-out. If AC, then the RMS value is used.	
`nominal_output_max_current_amps`	`float`
Field Description:	

4.4.7 PwrMngmtInformationReply Message

A reply from the power management module to the control module in response to a `PwrMngmtInformationRequest` message. Provides information on the capabilities of the module. Does not provide dynamic status.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x20
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xd6
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
lineout_informations	Short Array of PwrMngmtLineOutInformation
Field Description: Information about all of the lineouts of the unit.	
supply_features	uint32_t bits defined by PwrMngmtSupplyPropertiesFlags
Field Description: Bitfield listing features supported by the supply. Which features are marked will determine which PwrMngmtSupplyStatusFlags properties will trigger a PwrMngmtSupplyEventPush message or measured quantities of PwrMngmtSupplyStatusReply or PwrMngmtBatteryStatus are valid.	
description	Short String
Field Description: Free-form description of power management supply status.	

4.4.8 PwrMngmtLineOutStatusRequest Message

A request from the control module to the power management module for dynamic information (i.e., power source, voltage, current, etc) about the line outs.

See Also: PwrMngmtLineOutStatusReply

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x58
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.4.9 PwrMngmtLineOutStatusFlags Enumeration

Flags to indicate current status of a line out. A nominally operating line-out will not have any flags set. Any condition that will cause a change to any of the statuses (either cause flag to be set, or unset) will also trigger sending a `PwrMngmtLineOutEventPush` to let the control module know of the change. If no connection to the control module is established, then the `PwrMngmtLineOutEventPush` message does not need to be generated, unless data buffering is enabled.

Underlying integral representation: `uint32_t`

Enumerated values for `PwrMngmtLineOutStatusFlags`:

PwrMngmtLineOutOnBatteryPowerFlag Value 0x01

The batteries are currently the source of power for this output line (ie, mains power is not being used).

See also: `PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutCanBePoweredByBa`

PwrMngmtLineOutOffByRequestFlag Value 0x02

This output line is not currently not powered because of a requested power cycle.

See also: `PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutPowerCanBeCycle`

PwrMngmtLineOutOutOfToleranceFlag Value 0x04

The lineout is out of tolerance, for some reason other than overload; usually either too low of voltage, or too high, but may be other factors like noise or frequency.

See also: `PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutNoiseIsMeasuredF`
, `PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutVoltageIsMeasuredFlag`

PwrMngmtLineOutOutputLineFaultFlag Value 0x40

There is a fault on the lineout; too much power was being drawn, or other failure. If the line is shutoff, the `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOffByFailureFlag` flag will also be marked.

See also: `PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutCanReportFaultsF`

PwrMngmtLineOutOnBypassFlag Value 0x80

The lineout is on bypass; usually due to failure of overload.

See also: `PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutSupportsBypassFl`

PwrMngmtLineOutOutputOverloadWarningFlag Value 0x100

The power being drawn from a lineout is approaching maximum levels.

See also: `PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutSupportsOverloa`

PwrMngmtLineOutOutputOverloadDetectedFlag Value 0x200

The power draw has reached overload levels. Line may enter bypass mode, shut the line off, or cope for a limited amount of time before taking further action. If line is turned off the `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOutputLineFaultFlag`

and PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOffByFailureFlag flags will also be set.

See also: PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutSupportsOverloa

PwrMngmtLineOutBuckOnFlag Value 0x400

The mains voltage is being actively lowered. Usually applicable to line-interactive units.

See also: PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutSupportsBuckFla

PwrMngmtLineOutBoostOnFlag Value 0x800

The mains voltage is being actively lowered. Usually applicable to line-interactive units.

See also: PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutSupportsBoostFl

PwrMngmtLineOutDiagnosticsFailFlag Value 0x1000

A diagnostic for the lineout has failed. This might mean the voltage or current cant be measured, or an internal test has failed.

See also: PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutMonitorsDiagnost

PwrMngmtLineOutOffByFailureFlag Value 0x2000

This output line is not currently not powered because of a fault or battery depletion,

See also: PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutCanReportFaultsF

PwrMngmtLineOutImminentShutdownFlag Value 0x8000

This output line will imminently shutdown; you should immediately place all devices connected to it in a safe state.

4.4.10 PwrMngmtLineOutStatus Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Struct to hold the information of a PwrMngmtLineOutStatusReply message.

Message Contents:

Field Name	Data Type
subdetector_number	uint8_t
Field Description: The line out number this message corresponds to. Must be between one and DeviceInfoReply::num_subdetectors, inclusive.	
status_time	UtcTimePoint
Field Description: Time at which the status measurements are effective.	
lineout_status	uint32_t bits defined by PwrMngmtLineOutStatusFlags
Field Description: Bitfield indicating any current issues for the line-out.	
current_source_time_seconds	uint32_t
Field Description: The number of seconds this output line has been on the current power source	

Continued on next page

Table continued from previous page

Field Name	Data Type
voltage_volts	float
Field Description: The measured output voltage, in volts. AC voltages are RMS. If output voltage is not measured, a value of zero will be provided. See also: • PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutVoltageIsMeasuredF	
current_amps	float
Field Description: The measured output current, in amps. AC current given in RMS. If output current is not measured, a value of zero will be provided. See also: • PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutCurrentIsMeasuredF	
frequency_hz	float
Field Description: The frequency of the output. If output frequency is not measured, then the nominal designed frequency will be reported. See also: • PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutFrequencyIsMeasure	
load_percent	float
Field Description: The current load, in percent, of maximum sustainable current. If output load is not measured, a value of zero will be provided. See also: • PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutCurrentIsMeasuredF	
battery_capacity_percent	float
Field Description: The current capacity of the battery(s) (potentially) supplying this lineout. If batteries are not present, or their status is not monitored, a value of zero will be provided. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyBatteryVoltageIsMeas	
estimated_battery_capacity_seconds	int32_t
Field Description: The estimated time in seconds the batteries can sustain this line out if no mains power is supplied. If batteries are not present, or their status is not monitored, a value of zero will be provided. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyBatteryVoltageIsMeas	
status_description	Short String
Field Description: A free form English description of the status; if any flags are set in PwrMngmtLineOutStatus::lineout_status, then this field may provide more information	

4.4.11 PwrMngmtLineOutStatusReply Message

A reply from the power management module to the control module in response to a PwrMngmtLineOutStatusRequest message. Provides dynamic information about the line out such as power source, voltage, current, etc.

See Also: PwrMngmtLineOutStatusRequest

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xd8
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
informations	Short Array of PwrMngmtLineOutStatus
Field Description:	

4.4.12 PwrMngmtSupplyStatusRequest Message

A request from the control module to the power management module for dynamic information about the mains power, batteries, charging or inverting circuitry and other components that could cause issues. The dynamic status of mains power, batteries, and circuits are queried and reported together since their effect on portal operations are largely the same, and for components like circuitry or fans failing it may not always be possible, or particularly helpful, to distinguish the various components of the power management module.

See Also: PwrMngmtSupplyStatusReply

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x59
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.4.13 PwrMngmtSupplyStatusFlags Enumeration

The status of the mains input to the power management system. When the change of a property defined in this enum changes, a `PwrMngmtLineOutEventPush` will be sent out.

Underlying integral representation: `uint32_t`

Enumerated values for `PwrMngmtSupplyStatusFlags`:

PwrMngmtSupplyMainsOverVoltageFlag Value 0x01

Mains (supply) voltage is above tolerances. Some units may switch to battery power, which would be seperately indicated by the `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOn` bit of the `PwrMngmtLineOutEventPush::lineout_status` field.

PwrMngmtSupplyMainsUnderVoltageFlag Value 0x02

Mains (supply) voltage is below tolerances. Some units may switch to battery power, which would be separately indicated by the `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOn` bit of the `PwrMngmtLineOutEventPush::lineout_status` field.

PwrMngmtSupplyMainsNoisyFlag Value 0x04

Mains (supply) voltage is noisy. Some units may switch to battery power, which would be separately indicated by the `PwrMngmtLineOutStatusFlags::PwrMngmtLineOutOnBattery` bit of the `PwrMngmtLineOutEventPush::lineout_status` field.

PwrMngmtSupplyBatteryLowCapacityFlag Value 0x08

The batteries have been discharged to a low capacity. The battery capacity that is considered low may either be fixed at the factory, or settable through the Parameter mechanism, but a typical value might be 10 minutes of runtime left.

PwrMngmtSupplyBatteryDepletedFlag Value 0x10

The battery reserves are at a level such that the current load can not be sustained without mains power; if mains power is not available, any attached devices should be powered off to insure a safe shutdown. The battery level that is considered depleted may either be fixed, or settable through the Parameter mechanism, but a typical value might be one minute.

PwrMngmtSupplyBatteryIsChargingFlag Value 0x20

Battery has started charging to recover from a lowered charge level (devices were powered from batteries, or testing that drained the battery was performed). Not for indicating typical maintenance charging.

PwrMngmtSupplyBatteryFaultFlag Value 0x40

A fault has been detected with a battery. May indicate a battery needs replacing or servicing.

PwrMngmtSupplyFanFailureFlag Value 0x80

A fan has been detected as failed.

PwrMngmtSupplyAmbientTemperatureBadFlag Value 0x100

The ambient temperature is out of tolerance.

PwrMngmtSupplyBatteryTemperatureBadFlag Value 0x200

The battery temperature is out of tolerance.

PwrMngmtSupplyOtherTemperatureBadFlag Value 0x400

A temperature, other than ambient and battery, is out of tolerance. For example, power transistor are running too hot.

PwrMngmtSupplyHumidityBadFlag Value 0x800

Relative humidity is out of tolerance.

PwrMngmtSupplyWaterDetectedFlag Value 0x1000

Water has been detected.

PwrMngmtSupplySupplyDiagnosticsFailFlag Value 0x2000

Diagnostics has failed; this could be something like voltage or current measurements aren't working for some reason.

4.4.14 PwrMngmtBatteryStatusFlags Enumeration

Flags to indicate the gross level of battery health.

Underlying integral representation: `uint32_t`

Enumerated values for `PwrMngmtBatteryStatusFlags`:

PwrMngmtBatteryDiagnosticsFailedFlag Value 0x01

Battery state unable to be determined.

PwrMngmtBatteryLoweredCapacityFlag Value 0x02

Potential battery capacity is below a predefined threshold, but unit will still operate at specified levels.

PwrMngmtBatteryCriticalDegradedFlag Value 0x04

Potential battery capacity has degraded to a level that will effect performance.

PwrMngmtBatteryNotFunctioningFlag Value 0x08

Battery is functionally no longer operable. For example potential capacity is degraded to a point where it will only be able to provide power, in absence of mains power, for a very limited or no time.

4.4.15 PwrMngmtBatteryStatus Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold information about a batteries current status.

Message Contents:

Field Name	Data Type
battery_number	uint8_t
Field Description: If multiple batteries, or banks of batteries within the system are reported separately, this field distinguishes them. The value starts at zero and increments by one for each subsequent battery or bank.	
battery_status_flags	uint32_t bits defined by PwrMngmtBatteryStatusFlags
Field Description: Flags to indicate any issues with the battery. No flags set means all is fine.	
voltage_volts	float
Field Description: The voltage of the battery. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyBatteryVoltageIsMeas	
capacity_percent	float
Field Description: The present capacity of the battery, in percent. Between 0 and 100. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyBatteryVoltageIsMeas	
temperature_centigrade	float
Field Description: The present temperature of the battery. If this isn't the battery itself, it may be the battery enclosure or otherwise representative temperature. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyBatteryTemperatureIs	
battery_description	Short String
Field Description:	

4.4.16 PwrMngmtSupplyStatusReply Message

A reply from the power management module to the control module in response to a PwrMngmtSupplyStatusRequest message. Provides dynamic information about mains (supply) power source, battery(s) and other components of the module. The status of the output lines are reported separately in the PwrMngmtLineOutStatusReply message.

See Also: PwrMngmtSupplyStatusRequest

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xd9
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
status_time	UtcTimePoint
Field Description: Time at which the status measurements are effective.	
supply_status_flags	uint32_t bits defined by PwrMngmtSupplyStatusFlags
Field Description: Bitfield to represent any abnormal conditions Under normal conditions no flags will be set. Check this field to see	
battery_statuses	Short Array of PwrMngmtBatteryStatus
Field Description: Array of battery statuses. Statuses of batteries can be reported either individually, or grouped together. Reporting individual statuses is preffered for the additional information, but may not always be possible due to the design of the hardware.	
mains_voltage_volts	float
Field Description: The current voltage of the supply power. If AC, then the RMS value is used. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsVoltageIsMeasur	
mains_current_amps	float
Field Description: The current amperage being used of the supply power. If AC, then the RMS value is used. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsCurrentIsMeasur	
mains_frequency_hz	float
Field Description: The frequency of the supply power either measured or assumed. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsFrequencyIsMeas	
ambient_temperature_celsius	float
Field Description: The ambient temperature of the power management module enclosure. Will have a value of zero if temperature is not measured. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyAmbientTemperatureIs	
status_description	Short String
Field Description: Free form text description of the status.	

4.4.17 PwrMngmtLineOutEventPush Message

A push message from the power management module to the control module notifying it of a condition enumerated by PwrMngmtLineOutStatusFlags has changed; ex, switch to on battery power, overloaded, switch back to mains power, etc. If a power management unit has multiple line-outs, and more than one changes status (e.g., goes on battery power), then multiple PwrMngmtLineOutEventPush messages will be sent.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x5a
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
subdetector_number	uint8_t
Field Description: The line out number this message corresponds to. Must be between one and DeviceInfoReply::num_subdetectors, inclusive.	
status_time	UtcTimePoint
Field Description: Time at which the status measurements are effective.	
lineout_status	uint32_t bits defined by PwrMngmtLineOutStatusFlags
Field Description: Bitfield indicating any current issues for the line-out.	
current_source_time_seconds	uint32_t
Field Description: The number of seconds this output line has been on the current power source	
voltage_volts	float
Field Description: The measured output voltage, in volts. AC voltages are RMS. If output voltage is not measured, a value of zero will be provided. See also: • PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutVoltageIsMeasuredF	
current_amps	float
Field Description: The measured output current, in amps. AC current given in RMS. If output current is not measured, a value of zero will be provided. See also: • PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutCurrentIsMeasuredF	
frequency_hz	float
Field Description: The frequency of the output. If output frequency is not measured, then the nominal designed frequency will be reported. See also: • PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutFrequencyIsMeasure	
load_percent	float
Field Description: The current load, in percent, of maximum sustainable current. If output load is not measured, a value of zero will be provided. See also: • PwrMngmtLineOutPropertiesFlags::PwrMngmtLineOutCurrentIsMeasuredF	
battery_capacity_percent	float

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The current capacity of the battery(s) (potentially) supplying this lineout. If batteries are not present, or their status is not monitored, a value of zero will be provided. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyBatteryVoltageIsMeas	
estimated_battery_capacity_seconds	int32_t
Field Description: The estimated time in seconds the batteries can sustain this line out if no mains power is supplied. If batteries are not present, or their status is not monitored, a value of zero will be provided. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyBatteryVoltageIsMeas	
status_description	Short String
Field Description: A free form English description of the status; if any flags are set in PwrMngmtLineOutStatus::lineout_status, then this field may provide more information	

4.4.18 PwrMngmtLineOutEventPushAck Message

Acknowledgement of receiving a PwrMngmtLineOutEventPush message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xda
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.4.19 PwrMngmtSupplyEventPush Message

A push message from the power management module to the control module notifying it that one of the conditions enumerated by PwrMngmtSupplyStatusFlags has changed (ex. over voltage, battery fault, etc).

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x5b
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
status_time	UtcTimePoint
Field Description: Time at which the status measurements are effective.	
supply_status_flags	uint32_t bits defined by PwrMngmtSupplyStatusFlags
Field Description: Bitfield to represent any abnormal conditions Under normal conditions no flags will be set. Check this field to see	
battery_statuses	Short Array of PwrMngmtBatteryStatus
Field Description: Array of battery statuses. Statuses of batteries can be reported either individually, or grouped together. Reporting individual statuses is preffered for the additional information, but may not always be possible due to the design of the hardware.	
mains_voltage_volts	float
Field Description: The current voltage of the supply power. If AC, then the RMS value is used. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsVoltageIsMeasur	
mains_current_amps	float
Field Description: The current amperage being used of the supply power. If AC, then the RMS value is used. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsCurrentIsMeasur	
mains_frequency_hz	float
Field Description: The frequency of the supply power either measured or assumed. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsFrequencyIsMeas	
ambient_temperature_celsius	float
Field Description: The ambient temperature of the power management module enclosure. Will have a value of zero if temperature is not measured. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyAmbientTemperatureIs	
status_description	Short String

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Free form text description of the status.	

4.4.20 PwrMngmtSupplyEventPushAck Message

Acknowledgement of receiving a `PwrMngmtSupplyEventPush` message.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x20
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xdb
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.4.21 PwrMngmtTestType Enumeration

Type of test performed, or to be performed.

Used in message: `PwrMngmtSelfTestRequest`

Underlying integral representation: `uint8_t`

Enumerated values for `PwrMngmtTestType`:

PwrMngmtTestGeneralValue Value 0x01

A general test of the systems functionality.

PwrMngmtTestBatteryValue Value 0x02

A test sufficient to check if battery needs replacing.

PwrMngmtTestBatteryDeepCalibrationValue Value 0x04

A deep test of batteries, usually involving discharging and recharging them.

4.4.22 PwrMngmtSelfTestRequest Message

A request from the control module to the power management module asking the device to perform a self-test. The types of self-tests that can be requested are enumerated in

PwrMngmtTestType. There may be multiple replies to this message as some tests may take some time to perform so may send one or more updates during testing.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x5c *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
test_type	uint8_t enumerated by PwrMngmtTestType *Field Description*: The type of test requested to be performed.

4.4.23 PwrMngmtTestStatus Enumeration

The status of the test request.

Used in message: PwrMngmtSelfTestReply

Underlying integral representation: uint8_t

Enumerated values for PwrMngmtTestStatus:

PwrMngmtTestNotSupportedValue Value 0x01

The requested test type is not supported.

PwrMngmtTestCantBePerformedNowValue Value 0x02

The requested test cant be performed right now. This may happen, for example, because mains power is not available, or batteries are charging.

PwrMngmtTestInProgressValue Value 0x03

The requested test has been started. There may be additional progress updates sent, but there must be a final result sent with status PwrMngmtTestStatus::PwrMngmtTestCompleted or PwrMngmtTestStatus::PwrMngmtTestCantBePerformedNowValue.

PwrMngmtTestCompletedValue Value 0x04

Test is complete.

4.4.24 PwrMngmtSelfTestReply Message

A reply from the power management module to the control module in response to a PwrMngmtSelfTestRequest message. There may be multiple replies to a single request as some tests can take significant time.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0xdc *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
timestamp	UtcTimePoint *Field Description*: Time at which event happened.
test_type	uint8_t enumerated by PwrMngmtTestType *Field Description*: The type of test requested.
test_status	uint8_t enumerated by PwrMngmtTestStatus *Field Description*: The status of performing the test.
remaining_time_seconds	uint32_t *Field Description*: The estimated time remaining for the test to complete when PwrMngmtSelfTestReply::test_status is equal to PwrMngmtTestStatus::PwrMngmtTestInProgressValue.
description	Short String *Field Description*: Free form description of test results.

4.4.25 PwrMngmtAutomaticSelfTestResultPush Message

A push message from the power management module to the control module notifying it of a automated self test has started, in progress, aborted, or completed. Some power management units will perform periodic tests to ensure readiness for mains power failure. This message informs the control module of the results of the test.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x5d
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
timestamp	UtcTimePoint
Field Description: Time at which event happened.	
test_type	uint8_t enumerated by PwrMngmtTestType
Field Description: The type of test requested.	
test_status	uint8_t enumerated by PwrMngmtTestStatus
Field Description: The status of performing the test.	
remaining_time_seconds	uint32_t
Field Description: The estimated time remaining for the test to complete when PwrMngmtSelfTestReply::test_status is equal to PwrMngmtTestStatus::PwrMngmtTestInProgressValue.	
description	Short String
Field Description: Free form description of test results.	

4.4.26 PwrMngmtAutomaticSelfTestResultPushAck Message

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xdd
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.4.27 PwrMngmtLineOutPowerCycleRequest Message

A request from the control module to the power management module asking the device to turn a line out off and then back on, with potential delays for both the off and on actions. There may be multiple replies to this request.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x20
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x5e
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`subdetector_number`	`uint8_t`
Field Description: The lineout to power cycle.	
`delay_until_power_off_seconds`	`uint32_t`
Field Description: The delay to wait, in seconds, before powering off the line. This is intended to allow safely powering down devices attached to the line (which may include the control module making the request). A value of zero means no delay.	
`delay_while_powered_off_seconds`	`uint32_t`
Field Description: Number of seconds to wait with the power off, before powering the line back on. This is only a request; the power management module may power the line back on before this timeout, or the total time off may take longer. Reasons for this include: • There may be a set pre-defined maximum off time limit intended to minimize chance of accidentally remotely powering the system down for extensive periods of time. • After the elapsed amount of time, the batteries may not be charged enough to turn the power back on. A value of zero means no delay.	

4.4.28 PwrMngmtLineOutPowerCycleRequestStatus Enumeration

The status of the `PwrMngmtLineOutPowerCycleRequest`

Used in message: PwrMngmtLineOutPowerCycleReply

Underlying integral representation: uint8_t

Enumerated values for PwrMngmtLineOutPowerCycleRequestStatus:

PwrMngmtLineOutPowerCycleInvalidLineOutValue Value 0x01

An invalid line-out was requested. No further replies to the request will be sent.

PwrMngmtLineOutPowerCycleNotSupportedValue Value 0x02

Cycling the power is not supported for the requested line-out. No further replies to the request will be sent.

PwrMngmtLineOutPowerCycleWontToggleValue Value 0x03

The line-out can not currently be power cycled. This may be because of a self-test being run, or another issue. A NotificationPush message may be separately sent explaining the issue. No further replies to the request will be sent.

PwrMngmtLineOutPowerCycleScheduledValue Value 0x04

Request was received, and now waiting the specified delay before powercycling. If the scheduled time is near or greater than RapterConstants::FOLLOWUP_REPLY_TIMEOUT_MS, than more than one reply with this status may need to be sent. Additional replies to the request will be sent when the power cycling happens.

PwrMngmtLineOutPowerCycleTurnedOffValue Value 0x05

Power has been turned off. May not be able to be sent if the line-out powered the control module. Additional replies will be sent when power cycling is completed, or as needed while the power is off.

PwrMngmtLineOutPowerCycleInProgressValue Value 0x06

If the requested delay while power is off is approaching or great than RapterConstants::FOLLOWUP then replies with this status may need to be sent.

PwrMngmtLineOutPowerCycleCompletedValue Value 0x07

The power cycle has been completed, and line-out is turned back on. May not be able to be sent if the line-out powered the control module. No further replies to the request will be sent.

PwrMngmtLineOutPowerCycleCompletedWithIssueValue Value 0x08

The power cycle has been completed, and line-out is turned back on. May not be able to be sent if the line-out powered the control module. No further replies to the request will be sent.

4.4.29 PwrMngmtLineOutPowerCycleReply Message

A reply from the power management module to the control module in response to a PwrMngmtLineOutPowerCycleReply message. In addition to sending this reply, a PwrMngmtLineOutEventPush message with PwrMngmtLineOutStatusFlags::PwrMngmtLine

bit of PwrMngmtLineOutStatus::lineout_status appropriately set when the line is turned off and on.

See Also: PwrMngmtLineOutStatusFlags::PwrMngmtLineOutImminentShutdownFlag

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x20
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xde
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
timestamp	UtcTimePoint
Field Description: Time at which event happened.	
request_status	uint8_t enumerated by PwrMngmtLineOutPowerCycleRequestStatus
Field Description: Status of the request.	
subdetector_number	uint8_t
Field Description: The lineout the power cycle was requested for.	

4.5 Analysis Interface

4.5.1 AnalysisMsgType Enumeration

Values that specify the type of message being conveyed for within the analysis message group. That is, when the first byte of the message has a value of MessageGroup::AnalysisValue, then this enumeration defines the value of the second byte of the message so that the receiver can interpret the contents of the message body.

Underlying integral representation: uint8_t

Enumerated values for AnalysisMsgType:

AnalysisInterimResultRequestValue Value 0x51

Message sent from the control module to the analysis module requesting a preliminary analysis result up through the most recent DataOutDataPacketPush message.

See also: AnalysisInterimResultRequest

AnalysisInterimResultReplyValue Value 0xd1

Reply to a AnalysisInterimResultRequest message, containing the preliminary results for the occupancy so far.

See also: AnalysisInterimResultReply

AnalysisItemFinalResultsRequestValue Value 0x52

Message sent from the control module to the analysis module requesting the final analysis results for the most recent item.

See also: AnalysisItemFinalResultsRequestMsg

AnalysisItemFinalResultsReplyValue Value 0xd2

Reply to a AnalysisItemFinalResultsRequest message, containing the final alarming status for the most recent item.

See also: AnalysisItemFinalResultsReply

4.5.2 AnalysisInterimResultRequest Message

Message sent from the control module to the analysis module requesting a preliminary analysis result up through the most recent DataOutDataPacketPush message. The primary purpose of this message is to allow alarming while the vehicle is still traversing the portal, if the analysis algorithm is able to identify a potential issue before it exits. The control module can make this request at any time during an occupancy, during a background period, test source, etc. How the information in the AnalysisInterimResultReply message is determined is up to the implementation of the algorithm; for example, it could consider only the most recent DataOutDataPacketPush, or most recent 3 seconds, or time since the occupancy began, etc.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x04
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x51
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.5.3 InterimAnalysisDataUsageStatusFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more conditions regarding the usage of data in an interim analysis. Used by the analysis module to indicate the current status of the preliminary analysis.

Underlying integral representation: `uint16_t`

Enumerated values for `InterimAnalysisDataUsageStatusFlags`:

NotUsingDataFlag Value 0x01

This data is being ignored, and not used, and will not affect future analysis results. Possible because device is not in `OperatingMode::OperatingValue`, or the current `MeasurementType` is not `MeasurementType::ItemValue` or `MeasurementType::BackgroundValue`.

DataAnomalousForBackgroundFlag Value 0x02

The data currently is claimed to be background, but the analysis module finds it suspicious, so is not using it to aggregate background.

AggregatingBackgroundFlag Value 0x04

This data is being used to build up the background. Note that this doesn't specify that this data will actually affect the next analysis, since circular buffer, or other such mechanisms may be used internally to the analysis module.

UsingAsPartOfItemOfInterestFlag Value 0x10

Data is being used to evaluate an item of interest.

StateErrorFlag Value 0x8000

There is currently an error; a notification may be attached to the message.

4.5.4 AlarmTypeFlags Enumeration

Bitwise OR of flags indicating that one or more alarms have been triggered based on radiation detector data or vehicle presence data. If this is zero, then no alarm has been found yet. (See N42.42 2012 standard for

<RadAlarmCategoryCode >)

Underlying integral representation: `uint16_t`

Enumerated values for `AlarmTypeFlags`:

NoneFlag Value 0x00

Analysis of radiation detector data has resulted in no alarm.

NeutronFlag Value 0x01

Analysis of radiation detector data has resulted in an alarm due to neutrons.

GammaFlag Value 0x02

Analysis of radiation detector data has resulted in an alarm due to gamma radiation.

IsotopeFlag Value 0x10

Analysis of radiation detector data has resulted in the identification of one or more radionuclides the identity of which is reported in `AnalysisItemFinalResultsReply::n42_xml`

VehicleSpeedFlag Value 0x20

Analysis of vehicle presence data has resulted in an alarm based on vehicle speed

OccupancyDurationTooShortFlag Value 0x40

The duration of vehicle occupancy was too short for analysis to be completed.

OccupancyDurationTooLongFlag Value 0x80

The duration of vehicle occupancy was longer than allowed by the analysis module.

OtherFlag Value 0x100

Analysis of radiation detector data has produced results not properly described by other values of `AlarmTypeFlags`

4.5.5 NuclideResult Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Representation of a specific radioactive source identified by the analysis algorithm. May be a nuclide (e.x., U235, Co60, Th232, etc.) or a more general source (e.x., NORM, Bremsstrahlung, U-xray, SNM, etc.), or even a finding (e.x., Gross Count, Window Ratio, Background Suppression).

Message Contents:

Field Name	Data Type
`nuclide_id_confidence_value`	`float`
Field Description: Indication of confidence ranging from 0.0% to 100.0% (specified by a numerical value ranging from 0.0 to 1.0), in the identification status of a nuclide, where increasing values indicate more certainty that the nuclide is present. The interpretation of this value is dependent on the characteristics of the nuclide identification algorithm. If no confidence value is available, then this field must have a negative value other than negative infinity or negative zero; the magnitude of negative values may not carry any further meaning (e.g., a change in magnitude of a negative value must not affect the fields interpretation). (Roughly equivalent to N42 2012 standard for <NuclideIDConfidenceValue >)	
`start_sample_number`	`uint32_t`
Field Description: The starting sample number that this `NuclideResult` applies to; the value must correspond to a sample number in the relevant occupancy, unless it is zero, in which case the start of the occupancy is then implied.	
`end_sample_number`	`uint32_t`
Field Description: The ending sample number that this `NuclideResult` applies to; the value must correspond to a sample number in the relevant occupancy, unless it is zero, in which case the latest sample number of the occupancy is then implied.	

Continued on next page

Table continued from previous page

Field Name	Data Type
nuclide	Short String
Field Description: Radioactive source identified; does not strictly have to name a nuclide, but should name a radioactive source. For example valid entries include: Plutonium, Co-60, Bremsstrahlung, U-xray, HEU, U-235, I-131, Ag-110m, Annihilation, Gross Count, Window Ratio, (last two not in N42 standard, but it allows additions) etc. (Roughly equivalent to N42 2012 <NuclideName > elements. See section 5.1.3 of the 2012 N42 standard, and its subsections, for formatting of source names).	
nuclide_category	Short String
Field Description: Free form description of the identified source category. Some example entries are: SNM, Industrial, Medical, etc. May be blank. (Roughly equivalent of N42 2012 standard for <NuclideCategoryDescription >)	

4.5.6 AnalysisInterimResultReply Message

Reply to an AnalysisInterimResultRequest message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x04
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xd1
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
analysis_module_uuid	Uuid128
Field Description: The UUID of the analysis module that produced this analysis result. Must match the UUID given by DeviceInfoReply of the analysis module.	
result_timestamp	UtcTimePoint
Field Description: TODO: decide if we need a UUID for each analysis result The time the analysis was started.	
result_uuid	Uuid128
Field Description: UUID generated by the analysis module for this result. This must be generated in such a way that it uniquely identifies this result.	
interim_analysis_status	uint16_t bits defined by InterimAnalysisDataUsageStatusFlags

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Indication of whether the data are being used for background aggregation or for the analysis of an item of interest.	
alarm_type	uint16_t bits defined by AlarmTypeFlags
Field Description: Bitwise OR of flags indicating a gamma, neutron, etc, alarm has been triggered. If this is zero, then no alarm has been found.	
sample_number	uint32_t
Field Description: The item of interest number these results are for; must match DataOutPacket::sample_number of the data this result is for.	
analysis_confidence_value	float
Field Description: Indication of confidence, as a percent ranging from 0.0 to 100.0 (specified with a filed value ranging from 0.0 to 1.0), in the overall accuracy of the analysis, where increasing values indicate higher confidence. If a confidence is not ascribed, this should be set to a negative value (other than negative infinity or negative 0), with the magnitude having not carrying any meaning. (Taken roughly from N42 2012 standard for <AnalysisConfidenceValue >)	
nuclides	Short Array of NuclideResult
Field Description: The analysis algorithms identified results; does not strictly have to be nuclides (e.x., U235, Co60, etc.), but may be other sources as well (e.x., Bremsstrahlung, Annihilation, Gross Count, etc.). The same nuclide may be specified multiple times covering different ranges of sample numbers.	
analysis_result_description	Short String
Field Description: Free-form text describing the overall conclusion of the analysis regarding the source of concern. (See N42.42 2012 standard for <AnalysisResultDescription >)	

4.5.7 AnalysisItemFinalResultsRequest Message

Request sent from the control module to the analysis module to return the final analysis of most recent occupancy. The control module should not make this request except for at the end of an occupancy because the additional computational and bandwidth requirements to send the results may overwhelm the system. If the most recent DataOutDataPacketPush was marked as MeasurementType::ItemValue, the analysis module should assume the occupancy lasted up through the last DataOutDataPacketPush and is now over. If the control module would like updates in real time as the vehicle is traversing the portal, use AnalysisInterimResultRequest requests.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x04
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x52
Field Description: The type of message within the specified message_group, that this message corresponds to.	

Continued on next page

Table continued from previous page

Field Name	Data Type
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.5.8 AnalysisFinalResultStatusFlags Enumeration

Flags that may be bitwise OR'd to indicate the alarm results from the final analysis of radiation data from an occupancy. If no flags are set, the final analysis of occupancy radiation data concluded that no threat was present.

Underlying integral representation: `uint16_t`

Enumerated values for `AnalysisFinalResultStatusFlags`:

InconclusiveFlag Value 0x01

The final analysis of occupancy radiation data was inconclusive as to whether a threat was present.

ExcessNeutronFlag Value 0x02

Analysis of radiation detection data has found a source of neutrons.

SNMFlag Value 0x04

Analysis of radiation detection data has found a source that is categorized as SNM.

IndustrialFlag Value 0x08

Analysis of radiation detection data has found a source that is categorized as Industrial.

MedicalFlag Value 0x10

Analysis of radiation detection data has found a source that is categorized as Medical.

UnknownFlag Value 0x20

Analysis of radiation detection data has found a source that is categorized as Unknown.

NORMFlag Value 0x40

The final analysis of occupancy radiation data concluded that an innocent NORM source was present

ReferralFlag Value 0x80

The algorithm recommends additional inspection.

NotOperatingFlag Value 0x100

The final analysis of occupancy radiation data could not be produced because a necessary part of the system was not operating.

AnalysisFinalResultErrorFlag Value 0x8000

The final analysis of occupancy radiation data was accompanied by an error. Analysis results should not be trusted. The `AnalysisItemFinalResultData::analysis_result_des` field must include a description of the issue.

4.5.9 AnalysisItemFinalResultsReply Message

Message to convey the final results from analysis of the radiation data of an occupancy. Reply to a `AnalysisItemFinalResultsRequest` message.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x04
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xd2
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`analysis_module_uuid`	`Uuid128`
Field Description: The UUID of the analysis module that produced this analysis result. Must match the UUID given by `DeviceInfoReply` of the analysis module.	
`result_timestamp`	UtcTimePoint
Field Description: Timestamp of when the analysis results were generated	
`result_uuid`	`Uuid128`
Field Description: UUID generated by the analysis module for this result. This must be generated in such a way that it uniquely identifies this result.	
`item_number`	`uint32_t`
Field Description: The item of interest number these results are for; must match `DataOutPacket::item_number` of the data this result is for.	
`analysis_results_status`	`uint16_t` bits defined by `AnalysisFinalResultStatusFlags`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Bitwise OR of flags including the large picture final result of the radiation data analysis, i.e., whether to refer the vehicle for further inspection. (See DataOutFinalAnalysisPush::alarm_type field for additional alarm details, plus results from occupancy sensors).	
alarm_type	uint16_t bits defined by AlarmTypeFlags
Field Description: Bitwise OR of AlarmTypeFlags indicating an alarm has been triggered based on radiation data or vehicle presence data. If this is zero, then no alarm has been found. (See AnalysisItemFinalResultsReply::analysis_results_status for referral recommendation and further alarm detail.)	
analysis_confidence_value	float
Field Description: Indication of confidence, as a percent ranging from 0.0 to 100.0 (specified with a filed value ranging from 0.0 to 1.0), in the overall accuracy of the analysis, where increasing values indicate higher confidence. If a confidence is not ascribed, this should be set to a negative value (other than negative infinity or negative 0), with the magnitude having not carrying any meaning. (Taken roughly from N42 2012 standard for <AnalysisConfidenceValue >)	
nuclides	Short Array of NuclideResult
Field Description: The analysis algorithms identified results; does not strictly have to be nuclides (e.x., U235, Co60, etc.), but may be other sources as well (e.x., Bremsstrahlung, Annihilation, Gross Count, etc.). The same nuclide may be specified multiple times covering different ranges of sample numbers.	
analysis_result_description	Short String
Field Description: Free-form text describing the overall conclusion of the analysis regarding the source of concern. (See N42.42 2012 standard for <AnalysisResultDescription >)	
n42_xml	Large String
Field Description: This string provides the UTF-8 encoded contents of a ANSI N42.42-2012 file, including analysis results. The N42.42 contents must include the data used to make the determination of results.	

4.6 Data Out Interface

4.6.1 DataOutMsgType Enumeration

Values to indicate the type of message being sent as part of the DataOut message group. Request messages of the DataOut message group originate at a device and are directed to the control module. The control module issues push messages that are directed to all DataOut devices. The first byte of DataOut messages shall have a value of MessageGroup::DataOutValue. The second byte of the message will have a value as indicated by this enum, which will then tell you how to decode the message. For example a second byte value of DataOutDataPacketPush will tell you the message contents are specified by DataOutDataPacketPush message.

Underlying integral representation: `uint8_t`

Enumerated values for `DataOutMsgType`:

DataOutDataPacketPushValue Value 0x21

Message sent from the control module to DataOut, analysis, and control devices containing packaged data from the portal devices for a given time interval.

See also: `DataOutDataPacketPush`

DataOutDataPacketPushAckValue Value 0xa1

Acknowledgement of receiving a `DataOutDataPacketPush` message.

See also: `DataOutDataPacketPushAck`

DataOutDevicesInfoRequestValue Value 0x22

Message sent from a device to the control module requesting information about all the devices in the system.

See also: `DeviceInfoReply`, `SystemDeviceInfoRequest`, `DataOutSubDetectorInfo`

DataOutDevicesInfoReplyValue Value 0xa2

Reply sent in response to a `DataOutDevicesInfoRequest` containing information about all the devices currently connected to the control module.

See also: `DataOutDevicesInfoReply`

DataOutDeviceParametersRequestValue Value 0x3d

Request to get the parameters of a specific device.

See also: `DataOutDeviceParametersRequest`

DataOutDeviceParametersReplyValue Value 0xbd

Reply sent in response to a `DataOutMsgType::DataOutDeviceParametersRequestValue` containing the parameter information of the requested device.

See also: `DataOutDeviceParametersReply`

DataOutEventAcknowledgementPushValue Value 0x3e

Message notifying DataOut devices that an event has been acknowledged.

See also: `DataOutEventAcknowledgementPush`

DataOutEventAcknowledgementPushAckValue Value 0xbe

Acknowledgement of receiving a `DataOutEventAcknowledgementPush` message.

See also: `DataOutEventAcknowledgementPushAck`

DataOutSubDetectorInformationRequestValue Value 0x23

A request from a DataOut device to the control module for sub-device information for a given device. Note only radiation detectors or vehicle presence modules have sub-devices.

See also: `DataOutDevicesInfoRequest`, `RadSubDetectorInformationRequest`, `VehiclePresenceSubDetectorInformationRequest`

DataOutSubDetectorInformationReplyValue Value 0xa3

The reply from the control module to the DataOut device containing information about the sub-devices of a given module.

See also: DataOutDevicesInfoReply , RadSubDetectorInformationReply , VehiclePresenceSubDetectorInformationReply

DataOutGetStatusOfDeviceRequestValue Value 0x24

Request from a device to the control module requesting the status of a device in the system.

See also: DeviceStatusReply

DataOutGetStatusOfDeviceReplyValue Value 0xa4

Reply to a DataOutGetStatusOfDeviceRequest with either the device's status, or information about why it isn't available at this time.

See also: DataOutGetStatusReply

DataOutSystemOperabilityCheckRequestValue Value 0x25

Request from a device to the control module asking if the system is ready to start operations.

See also: DataOutSystemOperabilityCheckRequest , DeviceOperabilityCheckRequ

DataOutSystemOperabilityCheckReplyValue Value 0xa5

Reply sent in response to a DataOutSystemOperabilityCheckRequest with status of if the system is ready to operate.

See also: DataOutDevicesInfoReply

DataOutDeviceConnectedPushValue Value 0x26

Notice sent from the control module to all DataOut devices when a new device has connected to the control module via a websocket connection; this message is sent regardless of the newly connected device's state (e.g. this information not communicated as part of a DataOutDataPacketPush).

See also: DataOutDeviceConnectedPush

DataOutDeviceConnectedPushAckValue Value 0xa6

Acknowledgement of receiving a DataOutDeviceConnectedPush message.

See also: DataOutDeviceConnectedPushAck

DataOutDeviceDisconnectedPushValue Value 0x27

Notice sent from the control module to all DataOut devices when a device's WebSocket connection has disconnected.

See also: DataOutDeviceDisconnectedPush

DataOutDeviceDisconnectedPushAckValue Value 0xa7

Acknowledgement of receiving a DataOutDeviceDisconnectedPush message.

See also: DataOutDeviceDisconnectedPushAck

DataOutHandshakeFinishedPushValue Value 0x28

Notice sent from the control module to devices using the data message group, letting them know that another device has completed the handshake process. For devices which do not complete the handshake process, see DataOutDeviceDisconnectedPush.

See also: DataOutHandshakeFinishedPush

DataOutHandshakeFinishedPushAckValue Value 0xa8

Acknowledgement of receiving a DataOutHandshakeFinishedPush message.

See also: DataOutHandshakeFinishedPushAck

DataOutDeviceResponseIssuePushValue Value 0x29

Notice sent from the control module to all DataOut devices to let them know that a connection to a device hasn't been terminated (as far as the control module knows), but the device has timed out, or not responded to a message, or that the issue that caused a previous one of these messages to be sent out, has been cleared up.

See also: DataOutDeviceResponseIssuePush

DataOutDeviceResponseIssuePushAckValue Value 0xa9

Acknowledgement of receiving a DataOutDeviceResponseIssuePush message.

See also: DataOutDeviceResponseIssuePushAck

DataOutSystemStateChangePushValue Value 0x2a

Message sent from the control module to a device when a system wide operating state has changed.

See also: DataOutSystemStateChangePush , DataOutSystemStateChangePushAck

DataOutSystemStateChangePushAckValue Value 0xaa

Acknowledgement of receiving a DataOutSystemStateChangePush message.

See also: DataOutSystemStateChangePushAck

DataOutMiscNotificationPushValue Value 0x2b

Message sent from the control module to all DataOut devices as a catch-all message allowing information not communicated elsewhere; useful for error messages, or unusual conditions.

See also: DataOutMiscNotificationPush

DataOutMiscNotificationPushAckValue Value 0xab

Acknowledgement of receiving a DataOutMiscNotificationPush message.

See also: DataOutMiscNotificationPushAck

DataOutDeviceReferenceInfoRequestValue Value 0x2c

Message sent from a device to the control module asking for the physical position information of a specific subdevice; note that this information is stored in the control module and manually entered at portal setup.

See also: DataOutDeviceReferenceInfoRequest

DataOutDeviceReferenceInfoReplyValue Value 0xac

Reply sent in response to a DataOutDeviceReferenceInfoRequest message giving the physical position of the subdetector or device, if it has been set.

See also: DataOutDevicesInfoReply

DataOutTimeStatisticsRequestValue Value 0x2d

Message sent from a device to the control module requesting the time statics (up time, total on time, etc) of the control module and all devices connected to the portal. (The core message group contains a message for obtaining the statistics of a single device.)

See also: DataOutTimeStatisticsRequest

DataOutTimeStatisticsReplyValue Value 0xad

Reply sent in response to a DataOutTimeStatisticsRequest message giving time statistics of the control module and all the devices.

See also: DataOutDevicesInfoReply

DataOutBufferingEnableRequestValue Value 0x2e

Message sent from device asking the control module to buffer eligible messages for the device, in case the device temporarily disconnects. This option is only available to DataOut, Command, and Analysis devices, and only if allowed by the control module.

See also: DataOutBufferingEnableRequest

DataOutBufferingEnableReplyValue Value 0xae

Reply message sent by the control module in response to a DataOutBufferingEnableRequest message giving the status of the buffering.

See also: DataOutBufferingEnableReply

DataOutBufferedMessagesRequestValue Value 0x2f

Message sent by a device to the control module upon re-establishing a connection in order to start retrieval of buffered messages.

See also: DataOutBufferedMessagesRequest

DataOutBufferedMessagesReplyValue Value 0xaf

Reply sent in response to a DataOutBufferedMessagesRequest message, supplying status of the request. There may be multiple replies for a request.

See also: DataOutBufferedMessagesReply

DataOutInterimAnalysisPushValue Value 0x3a

An interim, not final, alarm status, most typically from when a measurement is not yet complete (example: vehicle still in portal); useful to alarm early if the analysis module already knows the item has alarmed before the occupancy is finished. Lack of an alarm from this message does not mean that the vehicle won't alarm during the occupancy.

See also: DataOutInterimAnalysisPush

DataOutInterimAnalysisPushAckValue Value 0xba

Acknowledgement of receiving a `DataOutInterimAnalysisPush` message.

See also: `DataOutInterimAnalysisPushAck`

DataOutFinalAnalysisPushValue Value 0x3b

The final adjudication by the analysis module for the most recent item.

See also: `DataOutFinalAnalysisPush` , `AnalysisItemFinalResultData` , `AnalysisItemFinalResultsReply`

DataOutFinalAnalysisPushAckValue Value 0xbb

Acknowledgement of receiving a `DataOutFinalAnalysisPush` message.

See also: `DataOutFinalAnalysisPushAck`

DataOutCurrentSystemStateRequestValue Value 0x3c

Message sent from a device to the control module requesting the current system state.

See also: `DataOutCurrentSystemStateRequest`

DataOutCurrentSystemStateReplyValue Value 0xbc

Reply sent in response to a `DataOutCurrentSystemStateRequest` message giving the current system state.

See also: `DataOutCurrentSystemStateReply`

4.6.2 DataOutDataPropertiesFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more conditions regarding data in a data out message.

Underlying integral representation: `uint32_t`

Enumerated values for `DataOutDataPropertiesFlags`:

InformationalUpdateNotFinalPacketFlag Value 0x01

Indicates that this `DataOutDataPacketPush` is not complete for the specified interval, and that it does not contain any radiation data or vehicle presence data (break-beam, photos, or distance), but rather includes other information such as state changes, parameter updates, or notifications. A future `DataOutDataPacketPush` that does not have this flag set will be sent out that covers this time period (and possibly more) which contains all information in this packet, plus the radiation and occupancy data. This feature may be used for example, when doing a long dwell measurement but you would still like to send parameter information, such as temperature, or high voltage to connected devices to keep them informed. All information in this message will be re-sent at a later time.

ReTransmitOnTimeoutRecoverFlag Value 0x02

If expected data isn't received from a device by a the control module's timeout interval, then a `DataOutDataPacketPush` message will be sent anyway. However if the data is eventually received from that device, the control module will resend that entire `DataOutDataPacketPush` message, but with this flag marked.

Note that the data frame(s) that were not previously sent must be marked with a DataOutDataFramePropertiesFlags::RecoveredDataFlag, while the previously transmitted frames will not be marked.

DeviceMissingFlag Value 0x04

Indicates that at least one device's data is missing from the packet; if the data is eventually received then the data will be re-sent, but with the DataOutDataPropertiesFlags::ReTransmi bit set, and the devices frame will be marked with the DataOutDataFramePropertiesFlags::Re

NotAllDevicesTakingDataYetFlag Value 0x08

After data acquisition is started, some detectors may start slightly before others so you might get a few time-slices where some are taking data, and others are not. This bit indicates this condition; is not an error condition.

4.6.3 DataOutDataFramePropertiesFlags Enumeration

Flags that may be bitwise OR'd to indicate one or more conditions regarding the properties of a data frame.

Underlying integral representation: `uint32_t`

Enumerated values for `DataOutDataFramePropertiesFlags`:

DataMissingFlag Value 0x01

Indicates that this frame was expected to exist, but the control module never received the data within the timings expected. Data for this device may be sent later, and if it is, it will be marked with the DataOutDataFramePropertiesFlags::RecoveredDataFlag. Note: if nothing was expected from the device in the time frame, then the control module must not include a data frame for that device.

See also: DataOutDataPropertiesFlags::DeviceMissingFlag

RecoveredDataFlag Value 0x02

Indicates that the data in this frame was previously reported as missing data (i.e. marked with DataOutDataFramePropertiesFlags::DataMissingFlag), but the missing data have since been received; the entire frame is being re-transmitted with both the recovered data and the original data.

PreparingToTakeDataFlag Value 0x04

The detector this frame corresponds to is still in the process of getting ready to take data (ex. turning HV on, etc).

DeviceErrorFlag Value 0x08

The device this frame corresponds to is having some sort of issue.

4.6.4 DeviceStatus Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold information on device operating state and status.

Message Contents:

Field Name	Data Type
`status_timestamp`	UtcTimePoint
Field Description: The timestamp of when this status indicated took effect. That is, the timestamp of the last event which caused the reported status to change. For example, if the last reportable thing to happen was the high voltage turned on, then this field would indicate that time.	
`device_status_flags`	`uint32_t` bits defined by `DeviceStatusFlags`
Field Description: Bitwise OR of flags representing target device status.	
`operating_mode`	`uint8_t` enumerated by `OperatingMode`
Field Description: The operating mode that the device is currently in.	
`data_collection_mode`	`uint8_t` enumerated by `DataCollectionModes`
Field Description: The data collection mode that the device is currently in. Set to `DataCollectionModes::NoOriginateValue` if the module is in any state other than of `DeviceStatus::operating_mode` `OperatingMode::OperatingValue`	
`measurement_type`	`uint8_t` enumerated by `MeasurementType`
Field Description: The type of measurement being taken (item of interest, background, not specified, possible interfering source, or active maintenance).	
`collection_interval_ms`	`uint32_t`
Field Description: The data collection interval is specific to the data collection mode, and zero if not applicable. For information on allowed intervals, see the `DataCollectionModes` enumeration. For `DataCollectionModes::RealTimeDwellValue` and `DataCollectionModes::LiveTimeDwellValue` modes, this interval specifies the entire duration of the measurement, and for the other modes it specifies the interval between sending data or `HeartbeatPush` (if non-zero) messages.	

4.6.5 ParameterUpdate Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Represents `ParameterUpdatePush` message that is sent when a parameter changes (either measured value, or one of another field was forced to change with out being asked to), and/or its health status changes. This message should not be sent if a specific request to change a parameter via a `SetParameterRequest` is made (and succeeds), unless there is a side effect like changing a different parameter (which the `SetParameterRequest` should be for the that parameter) or a change in health status or similar.

See Also: ParameterUpdatePushAck

Message Contents:

Field Name	Data Type
change_timestamp	UtcTimePoint
Field Description: Timestamp of when the change became effective, or at least when it was noticed to have become effective. Microseconds since the Unix epoch.	
value_type	uint8_t enumerated by ParameterValueDataType
Field Description: Data type of the value field. The data type cannot change during a connection, once it has been provided.	
parameter_field	uint8_t enumerated by ParameterField
Field Description: Enumeration identifying which field of the parameter changed. Only one field change can be reported per message. Note that if a measured value changes, which also causes a health status change, this field should report that it was the measured value that changed, while the health status change is reported in the ParameterUpdate::parameter_health_impact field, and the new value of the measurement goes in the ParameterUpdate::value field. If the ParameterField::SetValueValue, ParameterField::LowerHealthValue or ParameterField::UpperHealthValue are changed, causing a change to the health status, then it should be reported what was changed, and the new health status should be noted in ParameterUpdate::parameter_health_impact.	
parameter_health_impact	uint8_t enumerated by HealthSeverityLevel
Field Description: The impact of the current value of the parameter on health.	
subdetector_number	uint8_t
Field Description: The subdetector this Parameter applies to, or zero if not associated with a specific sub-detector.	
parameter_name	Short String
Field Description: Parameter name; only ascii letters, numbers, underscore, period, dash, and space characters are allowed.	
value	Short String
Field Description: Data type must match what is specified by ParameterUpdate::value_type, encoded as a string as in https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html	

4.6.6 EnergyCalibration Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold information on the energy calibration for a radiation detector. Used for both internally and externally obtained calibrations.

Message Contents:

Field Name	Data Type
calibration_timestamp	UtcTimePoint
Field Description: When the calibration was completed. Will be in the past, but not the future. If this energy calibration is empty or not valid, this value will be zero.	
energy_cal_coefficient_type	uint8_t enumerated by EnergyCalCoefficientType
Field Description: The types of coefficients used by the calibration.	
calibration_status	uint8_t enumerated by EnergyCalibrationStatus
Field Description: The status of the energy calibration, such as good, out of date, etc.	
subdetector_number	uint8_t
Field Description: The subdetector number this calibration is for. If a module only has one detection crystal/He3-tube, this will always be number one. If it has N detection sensors, this value will range from 1 to N (inclusive) according to the subdetector this calibration is for. See also: • DeviceInfoReply::num_subdetectors • RadListModeDataPush::subdetector_number • RadChannelDataPush::subdetector_number	
energy_cal_method_flags	uint16_t bits defined by EnergyCalMethodFlags
Field Description: A bitwise OR of flags representing energy calibration source and method.	
coefficients	Medium Array of float
Field Description: The coefficients necessary to specify the calibration.	
coefficient_uncertainties	Medium Array of float
Field Description: The uncertainty on the coefficients; may have zero entries, the same number of entries as the number of coefficients, twice the number of coefficients, or the number of coefficients squared. The interpretation of the coefficients is dependent on the number of entries and the options are: • No entries: uncertainty is not specified. • Same number of entries as coefficients: symmetrized 1-sigma uncertainties. • Twice the number of entries as coefficients: the positive 1-sigma uncertainties for each coefficient, then the negative 1-sigma uncertainties (e.g., the first N coefficients are positive uncertainties, followed the N negative uncertainties). • The square of number of entries as coefficients: the covariance matrix with entries of the first row listed first, followed by entries of second row, etc. To avoid an ambiguous situation, when there are two coefficients, the covariance matrix can not be specified, but as a work around, a 3rd zero valued coefficient could be added and then the matrix specified.	
deviation_pairs	Short Array of float

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Deviation pairs, as in N42.42, are each two `float`s, first the true energy, then the offset, then next entry in array is the next true energy, and its offset, etc. True energies and offsets are in keV. Unless the `RadDetDeviceFeaturesFlags::SupportsDeviationPairCalFlag` bit is set in `RadSubDetectorInfo::subdetector_features`, no values for this field may be provided. An example procedure for using deviation pairs: • Determine first (offset) and second (gain) polynomial coefficients using the 239 keV and 2614 keV peaks of Th232 • If the k-40 1460 keV peak is now at 1450 keV, a deviation of 10 keV should be set at 1460 keV • Deviation pairs set to zero would be defined for 239 keV and 2614 keV • The value provided in this field would then be [239,0,1460,10,2614,0]. Cubic spline interpolation is used to interpolate between deviation pairs. When solving for the spline coefficients, the second derivative is set to zero at the lowest deviation pair energy, and the first derivative is set to zero at the highest deviation pair energy. Corrections for energies below the lowest value deviation pair are always equal to the offset of the lowest energy deviation pair, and corrections for energies above the highest energy deviation pair are always equal to the offset of the highest energy deviation pair. Note: uncertainties for `EnergyCalibration::deviation_pairs` can not be specified since these are typically taken as well defined fixed quantities, and the polynomial coefficient uncertainties will account for the uncertainties. Note: although non-linear deviation pairs are used within the N42.42-2012 standard, the specification doesnt appear to provide an adequate definition of the,; the open source SpecUtils library, available at https://github.com/sandialabs/SpecUtils/, provides an implementation of non-linear deviation pairs that may be useful for reference.	
`method_description`	`Short String`
Field Description: A free-form description of method used to derive calibration. Examples might be: • "Fixed Factory Calibration" • "Fit to NORM templates" • "Gain matched for Th232 2614 keV and K40 1460 keV peaks" • "Individual PMTs gain matched to each other, followed by a fit to NORM templates" • "Gain matched to internal Cs137 seed" • "5 minute spectrum using Eu152" • etc.	
`calculation_notes`	`Medium String`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Free form text providing additional implementation specific details related to quality of energy calibration procedure. Examples of information that might be given in this text are: • Chi2 of fit to background template. • Endpoint energy of the detector that was determined • Relative gains of PMTs. • Unusual conditions detected, like high background rates, or contamination • Data channel corresponding to peak maximum • Version of calibration algorithm used The text must be UTF-8 encoded.	

4.6.7 ListModeDataPacket Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct that holds data of one or more detection events recorded by a single radiation subdetector. If the device is operating with a non-zero interval time (e.g., it is sending `HeartbeatPush` messages at regular intervals), then the contained detection events will all be from the same time interval.

Message Contents:

Field Name	Data Type
`reference_timestamp`	UtcTimePoint
Field Description: The timestamp all contained listmode detection events time offsets (i.e., the `ListModeEvent::relative_time` field) will be added to to get the detection events time. Note, this reference timestamp will come before must be less than or equal to the first detection event.	
`rad_data_flags`	`uint32_t` bits defined by `RadDataReadoutFlags`
Field Description: Bitwise OR of the relevant `RadDataReadoutFlags`. The same flags must be applicable to all contained detection events.	
`subdetector_number`	`uint8_t`
Field Description: If a detector contains multiple detection devices (ex. multiple He3 tubes, or multiple NaI blocks) that are individually read out as subdetectors, then the subdetector that detected the event should be specified here. If only a single detection device is present in the module, this value will always be one. See also: • `DeviceInfoReply::num_subdetectors` • `RadChannelDataPush::subdetector_number` • `RadEnergyCalibrationUpdatePush::subdetector_number`	
`listmode_events`	`Medium Array` of `ListModeEvent`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The detected listmode events. Events must be in increasing time order (e.g., the earliest detected events of this message come first in the array). Note that if there are more than 65,536 detection events, multiple `RadListModeDataPush` messages must be sent.	

4.6.8 ChannelDataPacket Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold the histogram radiation data for a `RadChannelDataPush` message

Message Contents:

Field Name	Data Type
`start_timestamp`	UtcTimePoint
Field Description: The clock time of the module at the start of the interval to which this channel data corresponds. Microseconds since the UNIX epoch.	
`end_timestamp`	UtcTimePoint
Field Description: The clock time of the module at the end of the interval to which the channel data corresponds. This value will nominally be the last microsecond in an interval, and not the beginning of the next interval (e.g., it will be one less than the next interval start time). Microseconds since the UNIX epoch. Note that because of time adjustments (via the PTP synchronization mechanisms), or interruptions to data taking, this time might not be `ChannelDataPacket::start_timestamp` + 1.0E6*clock_time_seconds.	
`live_time_seconds`	`float`
Field Description: The time in seconds that the detection system was available to take data during this data taking interval. The difference between `ChannelDataPacket::clock_time_seconds` and `ChannelDataPacket::live_time_seconds` is often referred to as "dead time" and can be caused, for example, by limitations in the MCA that prevent it from recording more detection events for a brief time after an event (maybe due to inherent decay time of scintillator, or electronics reset time), or from internal queues filling up, or an external interrupt (e.g., from a nearby radiography system) indicating data should not be taken.	
`clock_time_seconds`	`float`
Field Description: The acquisition clock time elapsed, in seconds, to which this data corresponds. A common use for this value, in combination with `ChannelDataPacket::live_time_seconds`, is to determine the "dead time" of the acquisition electronics to account for pulse-pileup effects. This time is not simply be the difference of the absolute times of the beginning and end of the data taking interval. For example, PTP time synchronizations should not affect this value. Time periods where data acquisition is blocked due to an external interrupt (i.e., from non-intrusive inspection system), or other reason (e.x., data acquisition did not start until part way through the interval) do not contribute to this value.	
`rad_data_flags`	`uint32_t` bits defined by `RadDataReadoutFlags`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Bitwise OR of flags to indicate unusual, but important conditions during data taking.	
`subdetector_number`	`uint8_t`
Field Description: The subdetector this data is from.	
`channel_data`	`Large Array` of `float`
Field Description: The channel data as recorded by the detector.	

4.6.9 BinarySensorsData Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold measurements from one or more binary vehicle presence detectors (e.g., break beam sensors), with overall recommendation from the vehicle presence module as to the presence (or not) of a vehicle.

Message Contents:

Field Name	Data Type
`presence_recommendation`	`uint8_t` enumerated by `RecommendedPresenceStatus`
Field Description: Recommendation by a vehicle presence module for if the portal is occupied. Timestamp of the recommendation is implied to be the latest one in `BinarySensorsData::binary_presence_data`.	
`binary_presence_data`	`Short Array` of `VehiclePresenceBinaryStatus`
Field Description: Each binary (break-beam type) vehicle presence sensor may have at most one entry in this array and time stamps may not span (come both before and after) a heartbeat. Must have an entry from at least one sensor.	

4.6.10 ImageData Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold image data from a subdetector.

Message Contents:

Field Name	Data Type
`timestamp`	UtcTimePoint
Field Description: Time image was taken. Microseconds since the UNIX epoch.	
`subdetector_number`	`uint8_t`
Field Description: The number of the imaging sub-detector (camera) that took the image.	

Continued on next page

Table continued from previous page

Field Name	Data Type
readout_status_flags	uint32_t bits defined by VehiclePresenceReadOutFlags
Field Description: Bitwise OR of flags to indicate state of the camera that took the picture, and status of the data. If zero, all is good.	
image_type	uint8_t enumerated by VehiclePresenceImageType
Field Description: The file type of the image being transferred.	
image_data	Large Array of uint8_t
Field Description: The image data corresponding to a standard image file.	

4.6.11 VehiclePresenceReadingInfo Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold a completed measurement of distance or speed from a vehicle presence subdetector to a vehicle.

Message Contents:

Field Name	Data Type
detection_timestamp	UtcTimePoint
Field Description: The time the distance measurement was made. Microseconds since the UNIX epoch.	
subdetector_number	uint8_t
Field Description: The subdetector number of the subdetector that is reporting this distance.	
subdetector_type	uint16_t enumerated by VehiclePresenceSubDetectorType
Field Description: The subdetector type this reading corresponds too. Will be either VehiclePresenceSubDetectorType::VehicleDistanceValue or VehiclePresenceSubDetectorType::VehicleSpeedValue. This type must that given by VehiclePresenceSubDetectorInformationReply for the specified subdetector.	
readout_status_flags	uint32_t bits defined by VehiclePresenceReadOutFlags
Field Description: Bitwise OR of flags to indicate state of the distance data reported with this struct. If zero, all is good.	
reading_is_useable	uint8_t
Field Description: If nonzero, the measured distance or speed is useable. If zero, the measured distance or speed should not be used.	
reading_value	float

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Either the measured speed (in meters per second), or the measured distance (in centimeters), as determined by VehiclePresenceReadingInfo::subdetector_type. The distance from the subdetector to the item in centimeters. When used with the subdetector's position and orientation, an unambiguous location of one point of the item is specified. If no item is being measured, then a value of zero should be reported.	

4.6.12 PwrMngmtSupplyStatus Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Not a RAPTER message, but a struct to hold information about the "supply" portion of the Power Management module.

See Also: PwrMngmtSupplyStatusReply

Message Contents:

Field Name	Data Type
`status_time`	UtcTimePoint
Field Description: Time at which the status measurements are effective.	
`supply_status_flags`	`uint32_t` bits defined by PwrMngmtSupplyStatusFlags
Field Description: Bitfield to represent any abnormal conditions Under normal conditions no flags will be set. Check this field to see	
`battery_statuses`	Short Array of PwrMngmtBatteryStatus
Field Description: Array of battery statuses. Statuses of batteries can be reported either individually, or grouped together. Reporting individual statuses is preffered for the additional information, but may not always be possible due to the design of the hardware.	
`mains_voltage_volts`	`float`
Field Description: The current voltage of the supply power. If AC, then the RMS value is used. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsVoltageIsMeasur	
`mains_current_amps`	`float`
Field Description: The current amperage being used of the supply power. If AC, then the RMS value is used. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsCurrentIsMeasur	
`mains_frequency_hz`	`float`
Field Description: The frequency of the supply power either measured or assumed. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyMainsFrequencyIsMeas	

Continued on next page

Table continued from previous page

Field Name	Data Type
ambient_temperature_celsius	float
Field Description: The ambient temperature of the power management module enclosure. Will have a value of zero if temperature is not measured. See also: • PwrMngmtSupplyPropertiesFlags::PwrMngmtSupplyAmbientTemperatureIs	
status_description	Short String
Field Description: Free form text description of the status.	

4.6.13 PwrMngmtSelfTestResult Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Not a RAPTER message. Holds information related to a power management unit test.

Message Contents:

Field Name	Data Type
timestamp	UtcTimePoint
Field Description: Time at which event happened.	
test_type	uint8_t enumerated by PwrMngmtTestType
Field Description: The type of test requested.	
test_status	uint8_t enumerated by PwrMngmtTestStatus
Field Description: The status of performing the test.	
remaining_time_seconds	uint32_t
Field Description: The estimated time remaining for the test to complete when PwrMngmtSelfTestReply::test_status is equal to PwrMngmtTestStatus::PwrMngmtTestInProgressValue.	
description	Short String
Field Description: Free form description of test results.	

4.6.14 RequestReceivedSummary Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct used by the control module to inform DataOut devices of a single request message it has received from a command device. Only the first 255 bytes of the command request are included. Even though the whole message is not distributed to the DataOut devices, key parts of the message are included and may be useful for auditing what requests have been made to the control module.

Message Contents:

Field Name	Data Type
timestamp_received	UtcTimePoint
Field Description: The time the control module received the request from a command device. Microseconds since the UNIX epoch.	
request_message_length	uint32_t
Field Description: The total length of the received request.	
message_content	Short Array of uint8_t
Field Description: Up to the first 255 bytes of the request message received from the command device.	

4.6.15 DataOutDataFrame Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold measurement data and related information to be sent to DataOut devices. See DataOutDataPacketPush message for use within a RAPTER message. When sending information a DataOutDataPacketPush message, the control module will group information into relevant time segments, and further group information by unique device. Information for one time segment for a unique device is represented by a one DataOutDataFrame.

Message Contents:

Field Name	Data Type
device_uuid	Uuid128
Field Description: The UUID of the device from which the data in this frame originate.	
device_type	uint8_t enumerated by MessageGroup
Field Description: The device type this frame represents as the device reports in DeviceInfo::device_type. See also: • DeviceInfo::device_type	
frame_properties_flags	uint32_t bits defined by DataOutDataFramePropertiesFlags
Field Description: Bitwise OR of DataOutDataFramePropertiesFlags	
changed_statuses	Short Array of DeviceStatus
Field Description: An array of DeviceStatus, in which the elements record each operating state experienced by the device during this time interval.	
changed_params	Medium Array of ParameterUpdate
Field Description: Parameter changes that occurred during the time interval of this frame are included here. Note that these might be sensor-derived parameters like a temperature change, a changed setting from via a SetParameterRequest, or any other parameter change.	
changed_energy_calibrations	Short Array of EnergyCalibration
Field Description: Energy calibrations that went into effect on this device during the interval of this frame.	
listmode_data	Large Array of ListModeDataPacket
Field Description: List mode data produced by this device during the interval of this frame.	

Continued on next page

Table continued from previous page

Field Name	Data Type
heartbeats	Short Array of HeartbeatPacket
Field Description: Heartbeat information received for the sub-devices; note that only radiation detectors and vehicle presence sensors will have entries as DataOut, Analysis, and Command devices do not have any sub-devices.	
channel_data	Short Array of ChannelDataPacket
Field Description: Channel data produced by this device during the interval of this frame.	
presence_binary_data	Short Array of BinarySensorsData
Field Description: Binary occupancy data produced (e.g., break-beam style IR sensors) during the interval of this frame.	
presence_image_data	Short Array of ImageData
Field Description: Any images produced during the time interval of this frame.	
presence_speed_distance_data	Short Array of VehiclePresenceReadingInfo
Field Description: Any speed or location measurements produced during the time interval of this frame.	
power_management_line_out_statuses	Short Array of PwrMngmtLineOutStatus
Field Description: Information from PwrMngmtLineOutStatusReply and PwrMngmtLineOutEventPush messages.	
power_management_supply_statuses	Short Array of PwrMngmtSupplyStatus
Field Description: Information from PwrMngmtSupplyStatusReply and PwrMngmtSupplyEventPush messages.	
power_management_test_results	Short Array of PwrMngmtSelfTestResult
Field Description: Information from PwrMngmtSelfTestReply and PwrMngmtAutomaticSelfTestResultPush messages.	
received_requests	Short Array of RequestReceivedSummary
Field Description: Any MessageGroup::CommandValue interface request messages that the control module received from this device during the time interval of this data frame (will have zero entries for all non-command devices).	
notifications	Short Array of Notification
Field Description: Notifications sent to the control module by this device during the interval of this frame.	

4.6.16 DataOutDataPacketPush Message

Message pushed from the control module to all DataOut devices. The message contains information produced during normal operations by all devices connected to the control module, and including the control module, for a discrete amount of time. For example, if a radiation portal is operating with gamma detectors in DataCollectionModes::ClockTimeIntervalValue mode with interval 100ms, then this packet will be sent from the control module after each 100ms period. If performing a dwell measurement this packet will be sent at the end of

the measurement interval; however intermediate packets may also be sent, but with the DataOutDataPropertiesFlags::InformationalUpdateNotFinalPacketFlag bit of DataOutPacket::properties set to indicate that it doesnt contain radiation data, but may include other information that may be useful to monitor during the dwell, like parameters. Each DataOutDataPacketPush message that does not have the DataOutDataPropertiesFlags::InformationalUpdateNotFinalPacketFlag bit marked covers a unique period of time, so there is no time-overlap between DataOutDataPacketPush message, with the exception of re-sending DataOutDataPacketPush messages due to data being recovered from a device for this time period that was not originally included (see DataOutDataPropertiesFlags::ReTransmitOnTimeoutRecoverFlag). Data that is from different system states, or from different time period will not be combined into the same DataOutDataPacketPush. DataOutDataPacketPush messages may be sent out while not collecting radiation data in order to convey parameter updates or other information; again time periods will not overlap, but when and how often to send the the messages is up to the control module.

Message Contents:

Field Name	Data Type
message_group *Field Description*: The RAPTER MessageGroup this message corresponds too.	uint8_t with value 0x02
message_type *Field Description*: The type of message within the specified message_group, that this message corresponds to.	uint8_t with value 0x21
message_group_version *Field Description*: The version of the message_group being used.	uint8_t
message_flags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	uint8_t bits defined by MsgFlags
message_id *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	uint32_t
interval_start_time *Field Description*: The start of the time interval covered by the data in this struct.	UtcTimePoint
interval_end_time *Field Description*: The end of the time interval covered by the data in this struct.	UtcTimePoint
sample_number *Field Description*: The data frame index of the current occupancy; the first data frame of an occupancy will have a value of one, and each subsequent data frame of the occupancy will have a value that increases by one from the previous value. All non-occupancy frames will have a value of zero.	uint32_t
item_number *Field Description*: Item number increments monotonically for a particular control module for each occupancy. If the data is not associated with an occupancy, this should be marked as a zero.	uint32_t
properties	uint32_t bits defined by DataOutDataPropertiesFlags

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: A bitwise OR of DataOutDataProperties.	
system_operating_mode *Field Description*: The operating mode of the system. This can either be set using the CmdSystemStateChangeRequest message or may be automatically determined and set by the control module. Individual devices may be in differing operating modes, or transitioning. See also: • DataOutSystemStateChangePush::requested_operating_mode	uint8_t enumerated by OperatingMode
system_measurement_type *Field Description*: What is being measured (item of interest, background, not specified, possible interfering source, or active maintenance).	uint8_t enumerated by MeasurementType
system_data_collection_interval_ms *Field Description*: The data collection interval of the system.	uint32_t
device_frame_uuids *Field Description*: The UUIDs of the entries in DataOutDataPacketPush::device_frames. This array must be in the same order, same size, and have the same UUIDs as the entries in DataOutDataPacketPush::device_frames.	Short Array of Uuid128
device_frame_offsets *Field Description*: The relative offset of each frame in DataOutDataPacketPush::device_frames. The entries in this array must correspond one-to-one, with the entries in DataOutDataPacketPush::device_frames and DataOutDataPacketPush::device_frame_uuids. The offsets are relative to where the first DataOutDataFrame in DataOutDataPacketPush::device_frames begins, therefore the first entry in this array will always have a value of zero. The second offset will be the number of bytes the first frame took up, plus any padding bytes to get to the multiple of 8 needed for serializing the struct in an array, and so on.	Short Array of uint32_t
device_frames *Field Description*: Information from each device connected to the control module, who reported information for this time interval. This information includes sensor data, parameter updates, notifications, state changes (either pushed, or that were requested), or a summary of requests messages received by the control module from the device.	Short Array of DataOutDataFrame

4.6.17 DataOutDataPacketPushAck Message

A message acknowledging the receipt of a DataOutDataPacketPush message sent by the control module. Only required when buffering for the connection is enabled.

Message Contents:

Field Name	Data Type
message_group *Field Description*: The RAPTER MessageGroup this message corresponds too.	uint8_t with value 0x02
message_type	uint8_t with value 0xa1

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.6.18 DataOutDevicesInfoRequest Message

Request from a DataOut device to the control module requesting information about all the devices in the system.

See Also: `DeviceInfoReply` , `SystemDeviceInfoRequest`

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02 *Field Description*: The RAPTER `MessageGroup` this message corresponds too.
`message_type`	`uint8_t` with value 0x22 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.6.19 DataOutDeviceInfo Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Holds the information about a connected device. For use within `DataOutDevicesInfoReply` messages. Note, the control modules will not include devices that they havent yet gotten `DataOutDeviceInfo` for yet.

Message Contents:

Field Name	Data Type
handshake_finish_timestamp	UtcTimePoint
Field Description: Approximate timestamp of finishing the handshake.	
ipaddress	Short String
Field Description: IP address of the device. This must match to the IP address specified in DataOutDeviceConnectedPush::ipaddress and DataOutDeviceDisconnectedPush::ipaddress.	
device_type	uint8_t enumerated by MessageGroup
Field Description: The device type, as determined from the following options (designated by the associated MessageGroup value): command device (Command), analysis device (Analysis), radiation detection device (RadDetector), vehicle presence device (VehiclePresence), or control module (Core). The device type determines the message groups that the device must support. See Chapter 2 for details.	
data_collection_modes	uint8_t bits defined by DataCollectionModes
Field Description: Bitwise OR of flags representing data collection modes supported by the device.	
device_features	uint64_t bits defined by DeviceFeaturesFlags
Field Description: Bitwise OR of flags representing device features.	
buffer_size	uint64_t
Field Description: The size of the buffer for buffering. This size can be an estimate. Should be set to 0 if the device does not support data buffering when the connection is down. When the network connection is down, Push messages should be buffered for retrieval once the connection is back up; see BufferedMessagesRequest and BufferedMessagesReply for details.	
num_subdetectors	uint8_t
Field Description: The number of sub detectors the device contains; applicable only to radiation detector, vehicle presence, and power management modules. For other devices this must be set to 0. As examples, for gamma-ray detectors this might indicate independent crystals (e.g. 4 crystals in one panel or module), for neutron detectors this might be the number of independently read-out He-3 tubes, and for vehicle presence sensors, the number of IR beams plus number of cameras. For power management modules it represents the number of independent output lines. Note that all sub-detectors must operate in the same data collection mode (DataCollectionModes::ClockTimeIntervalValue, DataCollectionModes::OnEventValue, etc) and at the same data collection intervals (10ms, 100ms, etc.). See also: • RadListModeDataPush::subdetector_number • RadChannelDataPush::subdetector_number • RadEnergyCalibrationUpdatePush::subdetector_number • RequestEnergyCalibration::subdetector • VehiclePresenceSubDetectorInformationRequest • PwrMngmtInformationRequest	
device_uuid	Uuid128

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The universally unique ID for this device. This number must be globally unique, and can be assigned according to the deploying agency conventions; for example assigned as the thumbprint of the devices cryptographic certificate. This value will not change for a given physical device.	
`serial_number`	`Short String`
Field Description: The manufacturer specified serial number that is expected to be unique at least with respect to the vendor and model. This value must remain fixed across firmware upgrades, power cycles, setting changes, and if at all possible hardware repairs. No restrictions on format other than length, and valid UTF-8 text.	
`manufacturer`	`Short String`
Field Description: Specifies the manufacturer of this device. No restrictions other than length, and valid UTF-8 text.	
`model`	`Short String`
Field Description: Specifies the model of this device. No restrictions other than length, and valid UTF-8 text.	
`hardware_version`	`Short String`
Field Description: Specifies the version of the hardware for the device. No restrictions other than length, and valid UTF-8 text. This should be descriptive and change with respect to hardware changes.	
`firmware_version`	`Short String`
Field Description: Specifies the version of the current firmware for the device. No restrictions other than length, and valid UTF-8 text. This should be descriptive and change with respect to firmware changes or updates.	
`other_component_versions`	`Short Array` of `ComponentVersionInformation`
Field Description: Versions of additional components in the device, that may be informative for compatibility, troubleshooting, or maintenance. Examples include: underlying operating system version, high voltage daughter card version, nuclide library version, analysis library version, etc.	
`specialized_capabilities_description`	`Short String`
Field Description: A free form English language description of the capabilities and hardware specific to the specialized purpose of this device. No restrictions other than length, and valid UTF-8 text. Examples might include: • "3x3 NaI gamma detector, 7% FWHM @ 661 keV, temperature compensated energy calibration, pulse pileup filter, 250MHz waveform sampling" • "Data archiving device to save all health information to a database" • "Graphical user interface device for portal operations. Touchscreen." • "Analysis software based on DHSIsotopeID v13, customized for PVT."	
`computing_platform_description`	`Short String`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: A free form English language description of the computing resources of the device, such as descriptions of its CPUs, ram memory, hard drive space, Ethernet adapter capabilities, operating system, etc. No restrictions other than length, and valid UTF-8 text. Examples include: • "PC4F1453 MCU, 120 MHz, 256 KB SRAM, 4GB SD-card storage, IEEE-1588 Ethernet transceiver, FreeRTOS 9.0.0" • "Dual core 2 GHz Arm CPU, 1 GB ram, Linux kernel 3.4."	

4.6.20 DataOutDevicesInfoReply Message

A reply by the control module to a `DataOutDevicesInfoRequest` message sent by a data-out device requesting information on all connected devices.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xa2
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`device_infos`	`Short Array` of `DataOutDeviceInfo`
Field Description: The information describing each and every device, one device per array element.	

4.6.21 DataOutDeviceParametersRequest Message

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x3d
Field Description: The type of message within the specified message_group, that this message corresponds to.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
device_uuid	Uuid128
Field Description: UUID of device you would like the status of.	

4.6.22 DataOutDeviceParametersRequestStatus Enumeration

Status enumeration of a DataOutDeviceParametersRequest.

Used in message: DataOutDeviceParametersReply

Underlying integral representation: uint8_t

Enumerated values for DataOutDeviceParametersRequestStatus:

RequestSuccessfulValue Value 0x00

Information in this DataOutSubDetectorInformationReply is valid.

RequestToDeviceInProgressValue Value 0x01

A request has been sent to the device for this information an a reply from it is being waited on.

RequestToDeviceTimedOutValue Value 0x02

The request for this information to the device timed out.

InvalidDeviceUuidValue Value 0x03

The device UUID specified was not the UUID of a device in the system.

RequestFailedOtherErrorValue Value 0x04

A DataOutMiscNotificationPush may be sent to provide additional information.

4.6.23 ParameterInfo Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Not a RAPTER message but a struct that holds all information about a parameter.

Message Contents:

Field Name	Data Type
value_type	uint8_t enumerated by ParameterValueDataType
Field Description: The data type of this parameter. Each applicable field in ParameterInfo::set_value, ParameterInfo::measured_value, ParameterInfo::lower_healthy_measured_value, ParameterInfo::upper_healthy_measured_value, ParameterInfo::lower_settable_value, ParameterInfo::upper_settable_valu and ParameterInfo::reportable_change_delta (as can be determined from the ParameterInfo::value_properties field) must be lexically encoded as this value type (non applicable fields must be empty strings).	
parameter_health_impact	uint8_t enumerated by HealthSeverityLevel
Field Description: The devices assessment as to the impact of the value of this parameter. For both ParameterPropertiesFlags::HealthyValueRangedFlag and ParameterPropertiesFlags::AffectsHealthWithoutLimitsFlag parameters, it is the responsibility of the sender to mark the health status, regardless of healthy limits or current measured value.	
subdetector_number	uint8_t
Field Description: The subdetector this Parameter applies to, or zero if not associated with a specific sub-detector.	
value_properties	uint32_t bits defined by ParameterPropertiesFlags
Field Description: A bitwise OR of ParameterPropertiesFlags flags.	
set_timestamp	UtcTimePoint
Field Description: The timestamp at which this parameter was last set. May be zero for factory values, or other situations where it wouldn't be able to be tracked, or if the parameter is not settable.	
effective_timestamp	UtcTimePoint
Field Description: The timestamp at which this parameter was last measured, or health status determined. If this parameter does not change, or effect health status this field must be zero. (i.e. zero unless ParameterInfo::value_properties has ParameterPropertiesFlags::AffectsHealthWithoutLimitsFlag or ParameterPropertiesFlags::MeasuredFlag bits set)	
parameter_name	Short String
Field Description: Name of the parameter; must not be empty, and use only only ascii letters, numbers, underscore, dash, period, and space characters.	
set_value	Short String
Continued on next page	

Table continued from previous page

Field Name	Data Type
Field Description: Value of the parameter, if ParameterInfo::value_properties contains ParameterPropertiesFlags::ValueFixedFlag or ParameterPropertiesFlags::SettableFlag; otherwise the string shall be empty. Value is always represented as a UTF8 string which is a lexical encoding of the type specified by ParameterInfo::value_type. Lexical encoding of non-string values is according to the same rules as for XML data types, with the exception that Infs and NaNs are not allowed for floating points; see https://www.w3.org/TR/xmlschema11-2/. If ParameterInfo::value_properties is ParameterPropertiesFlags::ValueFixedFlag or ParameterPropertiesFlags::SettableFlag, then this field may only be empty if ParameterInfo::value_type has a value of ParameterValueDataType::Utf8StringValue, ParameterValueDataType::FloatingPointListValue, or ParameterValueDataType::IntegerListValue	
measured_value	Short String
Field Description: The measured value of the parameter if, and only if, ParameterInfo::value_properties has the ParameterPropertiesFlags::MeasuredFlag bit set, otherwise string must be empty. Value is always represented as a UTF8 string which is a lexical encoding of the type specified by ParameterInfo::value_type. Lexical encoding of non-string values is according to the same rules as for XML data types; see https://www.w3.org/TR/xmlschema11-2/, with the exception floats can not be Infs or NaNs.	
lower_healthy_measured_value	Short String
Field Description: The lower limiting value in the range of values over which this parameter is still considered 'healthy'. Used if and only if ParameterInfo::value_properties has the ParameterPropertiesFlags::HealthyValueRangedFlag bit set, and otherwise must be an empty string. Value must be lexically encoded according to same rules as XML data types.	
upper_healthy_measured_value	Short String
Field Description: The highest value at which this parameter is still considered 'healthy'. Used only if ParameterInfo::value_properties has the ParameterPropertiesFlags::HealthyValueRangedFlag bit set, otherwise must be an empty string. Value must be lexically encoded according to same rules as XML data types.	
lower_settable_value	Short String

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The lowest value at which this parameter can be set. If this value is dependent upon a value of another parameter (e.x. both parameters must multiply together and be higher than a given number), then this value lists the absolute lowest value the parameter can take, but the parameter description should give this limitation, and if the parameter is requested to be changed to a value that it cant be set to (partially due to the other dependent parameter), a reply with a status of `CommandReplyStatus::FailedValue` should be given. Used if and only if `ParameterInfo::value_properties` field has the `ParameterPropertiesFlags::SetValueRangedFlag` bit set, otherwise this string must be empty. Value must be lexically encoded according to same rules as XML data types.	
`upper_settable_value`	`Short String`
Field Description: The absolutely highest value at which this parameter can be set. If this upper threshold value is dependent upon a value of another parameter (e.x. both parameters must add together and be lower than a given number), then this value should list the absolute highest value the parameter can take, but the parameter description should give this limitation, and if the parameter is requested to be changed to a value that it cant be set to (partially due to the other dependent parameter), a reply with a status of `CommandReplyStatus::FailedValue` should be given. Used if and only if `ParameterInfo::value_properties` has the `ParameterPropertiesFlags::SetValueRangedFlag` bit is set, otherwise this string must be empty. Value must be lexically encoded according to same rules as XML data types.	
`reportable_change_delta`	`Short String`
Field Description: The amount a measured value needs to change before an update will be sent. Used only if `ParameterInfo::value_properties` has the `ParameterPropertiesFlags::ReportOnChangeFlag` bit set, otherwise this string will be empty. For boolean parameters, setting delta to false indicates report on change (such that a change in value results in a Boolean change from zero to one), while true indicates do not report (for which a change of one or more does not result in a change of Boolean value). Value must be lexically encoded according to same rules as XML data types, with the restriction that for `floating` point types Infs and NaNs are not allowed.	
`parameter_description`	`Medium String`
Field Description: Human readable, UTF-8 description of the parameter.	

4.6.24 DataOutDeviceParametersReply Message

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xbd
Field Description: The type of message within the specified message_group, that this message corresponds to.	

Continued on next page

Table continued from previous page

Field Name	Data Type
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`requested_device_uuid`	`Uuid128`
Field Description: UUID of requested	
`request_status`	`uint8_t` enumerated by `DataOutDeviceParametersRequestStatus`
Field Description: Status of the request. Check this to see if there are issues, information is not yet available, or the information is available in the `DataOutDeviceParametersReply::parameters` field.	
`parameters`	`Medium Array` of `ParameterInfo`
Field Description: The parameters from the device. If the `DataOutDeviceParametersReply::request_status` value is anything besides `DataOutDeviceParametersRequestStatus::RequestSuccessfulValue` this array may be empty.	

4.6.25 AcknowledgeableEventType Enumeration

The type of event an acknowledgment is for. Any events added in the future that can be acknowledged must have a UUID generated by the originating device.

Used in message: `DataOutEventAcknowledgementPush`

Underlying integral representation: `int32_t`

Enumerated values for `AcknowledgeableEventType`:

FinalAnalysisAlarmAcknowledgement Value 0x00

Acknowledgment of an alarming vehicle.

4.6.26 EventAcknowledgmentType Enumeration

The type of acknowledgment a device provided. Additional values may be added in the future when operational needs are better defined.

Used in message: `DataOutEventAcknowledgementPush`

Underlying integral representation: `int32_t`

Enumerated values for `EventAcknowledgmentType`:

EventDealtWith Value 0x00

Event has been dealt with.

4.6.27 DataOutEventAcknowledgementPush Message

Message telling data out devices that an event (possible events defined by AcknowledgeableEventType) has been acknowledged by a command device. This message can be used to indicate, for example, that a gate arm can lift up after a vehicle alarms. This message can also serve as a record that a condition was acknowledged and dealt with. Currently only final analysis results can be acknowledged, but other events such as errors will be added in the future.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x3e
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
acknowledging_device_uuid	Uuid128
Field Description: The UUID of the device that acknowledged the event.	
acknowledging_device_timestamp	UtcTimePoint
Field Description: The timestamp of the device that acknowledged the event.	
acknowledgment_recieved_timestamp	UtcTimePoint
Field Description: The time at which the control module received the Ack.	
event_type	uint32_t enumerated by AcknowledgeableEventType
Field Description: The type of event that was acknowledged.	
event_uuid	Uuid128
Field Description: The UUID of the event that See also: • AnalysisItemFinalResultData::result_uuid • InterimAnalysisData::result_uuid	
acknowledgment_type	uint32_t enumerated by EventAcknowledgmentType
Field Description: The type of acknowledgement that was done.	
message_from_acknowledger	Short String
Field Description: Optional message included by the device that gave this acknowledgement. See also: • CmdDeviceEventAcknowledgmentRequest::message	

4.6.28 DataOutEventAcknowledgementPushAck Message

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xbe
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.29 DataOutSubDetectorInformationRequest Message

Request from a MessageGroup::DataOutValue device to the control module requesting information about all the sub-devices for a given device in the system. When the control module receives this request, it is free to then query the specified device, or the control module can cache this information and return that. This means the control module may, or may not, be able to return information on devices that were previously connected, but now are disconnected.

See Also: DataOutDevicesInfoRequest , RadSubDetectorInformationRequest , VehiclePresenceSubDetectorInformationRequest

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x23
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

Continued on next page

Table continued from previous page

Field Name	Data Type
device_uuid	Uuid128
Field Description: UUID of device you would like the status of.	

4.6.30 DataOutSubDetectorInformationReplyStatus Enumeration

The status of a DataOutSubDetectorInformationRequest

Used in message: DataOutSubDetectorInformationReply

Underlying integral representation: uint8_t

Enumerated values for DataOutSubDetectorInformationReplyStatus:

RequestSuccessfulValue Value 0x00

Information in this DataOutSubDetectorInformationReply is valid.

DeviceDoesNotHaveSubDevicesValue Value 0x01

The request was for a valid UUID, but the device did not have any sub-devices (e.g., was a data out, command device, analysis module or the control module).

InvalidDeviceUuidValue Value 0x02

The device UUID specified was not the UUID of a device in the system.

RequestToDeviceTimedOutValue Value 0x03

The RadSubDetectorInformationRequest or VehiclePresenceSubDetectorInform
from the control module to the device timed out or otherwise failed.

HandshakeInProgressValue Value 0x04

The handshake between the specified device and the control module is currently in progress. Try again later.

4.6.31 PwrMngmtSupplyInformation Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold information about the supply portion of the power management unit.

Message Contents:

Field Name	Data Type
supply_features	uint32_t bits defined by PwrMngmtSupplyPropertiesFlags
Field Description: Bitfield listing features supported by the supply. Which features are marked will determine which PwrMngmtSupplyStatusFlags properties will trigger a PwrMngmtSupplyEventPush message or measured quantities of PwrMngmtSupplyStatusReply or PwrMngmtBatteryStatus are valid.	

Continued on next page

Table continued from previous page

Field Name	Data Type
description	Short String
Field Description: Free-form description of power management supply status.	

4.6.32 DataOutSubDetectorInformationReply Message

The reply from the control module to a data out device containing information about the sub-devices of a given module.

See Also: DataOutDevicesInfoReply , RadSubDetectorInformationReply , VehiclePresenceSubDetectorInformationReply

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xa3
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
device_uuid	Uuid128
Field Description: UUID of device requested.	
status	uint8_t enumerated by DataOutSubDetectorInformationReplySta
Field Description: The status of this request.	
rad_detectors_info	Short Array of RadSubDetectorInfo
Field Description: The sub-device information if the requested device was a radiation detector. Each RadSubDetectorInfo must have a unique RadSubDetectorInfo::subdetector_number starting at one and increasing by one for each subsequent RadSubDetectorInfo.	
vehicle_presence_info	Short Array of VehiclePresenceSubDetectorInformation
Field Description: The sub-device information if the requested device was a vehicle presence module. Each VehiclePresenceSubDetectorInformation must have a unique VehiclePresenceSubDetectorInformation::subdetector_number starting at one and increasing by one for each subsequent RadSubDetectorInfo.	

Continued on next p

Table continued from previous page

Field Name	Data Type
power_management_lineout_info	Short Array of PwrMngmtLineOutInformation
Field Description: The lineout information of the power management unit if that is what was requested.	
power_management_supply_info	Short Array of PwrMngmtSupplyInformation
Field Description: The supply information of the power management unit if that is what was requested.	

4.6.33 DataOutGetStatusOfDeviceRequest Message

Request from a DataOut device to the control module requesting the status of a device in the system.

See Also: DeviceStatusReply

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x24
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
device_uuid	Uuid128
Field Description: UUID of device you would like the status of.	
force_refresh	uint8_t
Field Description: If 0, the control module can decide to either return its cached or tracked version of the status, or query the device; if non-zero, the control module must query the individual device for status. In the case of querying the device, there may be delays or potential errors due to the overhead of handling the request in the control module and possibility of the request to the device timing out.	

4.6.34 DataOutDeviceStatusFlags Enumeration

Underlying integral representation: uint8_t

Enumerated values for DataOutDeviceStatusFlags:

InvalidDevice Value 0x01

Device with requested Uuid is not part of the system. The remainder of the message must be valid to decode, but should not be used.

DeviceNoLongerConnected Value 0x02

Device is no longer connected, so status is last known status, but is valid.

CouldNotReachDevice Value 0x04

The device is not responding, so couldn't update status from device, however the control module does not recognized the device as being disconnected.

StatusNotAvailable Value 0x08

The status for the device is not available for some reason; a DataOutMiscNotificationPush may be separately sent explaining the reason. The remainder of the message must be valid to decode, but should not be used.

QueriedDevice Value 0x10

Device status was retrieved by querying the device itself.

4.6.35 DataOutGetStatusOfDeviceReply Message

A reply by the control module to a DataOutGetStatusOfDeviceRequest message sent by a data-out device requesting status of a specific device.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xa4
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
device_uuid	Uuid128
Field Description: The UUID of the device being reported.	

Continued on next page

Table continued from previous page

Field Name	Data Type
reply_status_flags	uint8_t bits defined by DataOutDeviceStatusFlags
Field Description: Bitwise OR of the DataOutDeviceStatusFlags indicating the status of this reply to the request.	
status_timestamp	UtcTimePoint
Field Description: The timestamp of when this status indicated took effect. That is, the timestamp of the last event which caused the reported status to change. For example, if the last reportable thing to happen was the high voltage turned on, then this field would indicate that time.	
device_status_flags	uint32_t bits defined by DeviceStatusFlags
Field Description: Bitwise OR of flags representing target device status.	
operating_mode	uint8_t enumerated by OperatingMode
Field Description: The operating mode that the device is currently in.	
data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: The data collection mode that the device is currently in. Set to DataCollectionModes::NoOriginateValue if the module is in any state other than of DataOutGetStatusOfDeviceReply::operating_mode OperatingMode::OperatingValue	
measurement_type	uint8_t enumerated by MeasurementType
Field Description: The type of measurement being taken (item of interest, background, not specified, possible interfering source, or active maintenance).	
collection_interval_ms	uint32_t
Field Description: The data collection interval is specific to the data collection mode, and zero if not applicable. For information on allowed intervals, see the DataCollectionModes enumeration. For DataCollectionModes::RealTimeDwellValue and DataCollectionModes::LiveTimeDwellValue modes, this interval specifies the entire duration of the measurement, and for the other modes it specifies the interval between sending data or HeartbeatPush (if non-zero) messages.	

4.6.36 DataOutSystemOperabilityCheckRequest Message

A message sent by a DataOut device to the control module requesting to check if the system is operable.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x25

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t` *Field Description*: The version of the message_group being used.
`message_flags`	`uint8_t` bits defined by `MsgFlags` *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
`message_id`	`uint32_t` *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.6.37 SystemOperabilityStatus Enumeration

Enumeration of descriptions of system operability.

Used in message: `DataOutSystemOperabilityCheckReply`

Underlying integral representation: `uint8_t`

Enumerated values for `SystemOperabilityStatus`:

SystemIsDeterminingOperabilityStatusValue Value 0x00

Operability is still being determined; this message is being sent to avoid timeouts.

SystemIsOperableNoIssuesValue Value 0x01

There is nothing preventing operation, or the system is already in the `OperatingMode::ReadyValue` or `OperatingMode::OperatingValue` operating mode; however, there may still be a delay before things can start, so check the time estimate.

SystemIsOperableWithIssuesValue Value 0x02

The system can operate, but one or more devices have issues that may prevent optimal operation; examples might be that one gamma detector can't be found, or that a neutron detectors state of health is bad. The `DataOutSystemOperabilityCheckReply::description` field will provide a description of the issue(s).

SystemIsNotOperableValue Value 0x03

The system can not operate - for example if too many detectors are missing, or have bad states of health, or a critical interrupt is tripped somewhere. The `DataOutSystemOperabilityCheckReply::description` field will provide a description of the issue(s).

4.6.38 DataOutSystemOperabilityCheckReply Message

A reply by the control module to a `DataOutSystemOperabilityCheckRequest` message sent by a data-out device requesting system operability status.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xa5
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`status`	`uint8_t` enumerated by `SystemOperabilityStatus`
Field Description: The status of the system's operability.	
`estimate_time_to_operating_ms`	`uint32_t`
Field Description: Estimated time to resolve any issues that prevent operability, if known; zero if the system is already operable.	
`description`	`Medium String`
Field Description: Provides a description of any issues or errors present. Intended to help users diagnose issues.	

4.6.39 DeviceConnectionLevel Enumeration

The different level of connection a `DataOutDeviceConnectedPush` might convey. Not all implementations of control modules will support all connection levels.

Used in message: `DataOutDeviceConnectedPush`

Underlying integral representation: `uint8_t`

Enumerated values for `DeviceConnectionLevel`:

PhysicalConnection Value 0x00

A device was physically connected to the network.

DhcpEstablished Value 0x01

A device has acquired a IP address.

WebSocketConnected Value 0x02

A new WebSocket connection was created to the control module.

4.6.40 DataOutDeviceConnectedPush Message

Message pushed from the control module to all DataOut devices when a new device has connected to the control module via a WebSocket connection.

See Also: DataOutDeviceConnectedPush

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x26 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
timestamp	UtcTimePoint *Field Description*: Approximate timestamp that connection was established.
ipaddress	Short String *Field Description*: IP address of the device .
device_uuid	Uuid128 *Field Description*: UUID of the connecting device, if known, for example through the SSDP mechanism or the MAC address. All zeros if not known.
connection_level	uint8_t enumerated by DeviceConnectionLevel *Field Description*: The type of connection being reported on.

4.6.41 DataOutDeviceConnectedPushAck Message

A message acknowledging the receipt of a DataOutDeviceConnectedPush message sent by the control module.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0xa6 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.42 DeviceDisconnectReason Enumeration

Enumeration of some reasons a disconnect may have happened. See also Section 7.4.1 or RFC 6455.

Used in message: `DataOutDeviceDisconnectedPush`

Underlying integral representation: `uint8_t`

Enumerated values for `DeviceDisconnectReason`:

CleanCloseValue Value 0x00

The WebSocket close code was 1000 or 1001.

DisappearedValue Value 0x01

The device disappeared, and no close frame was received for the WebSocket connection.

DisappearedPhysicallyValue Value 0x02

The device disappeared, and the control module is able to determine it is effectively no longer physically connected to the network (e.g., link integrity test pulses of the Ethernet have stopped); not all control module configurations may be able to detect this.

DueToPowerDownValue Value 0x03

The control module requested the device power down via a `PowerDownRequest`, and the connection was subsequently closed.

DueToFirmwareUpgradeValue Value 0x04

A `FirmwareUpgradeRequest` was completed successfully and device restarted.

DueToTimeoutValue Value 0x05

Communications with the device have timed out, so the control module has closed the connection.

UnallowedNetworkCommunicationValue Value 0x06

The control module detected a network communication attempt outside of the allowed channels.

InvalidWebSocketCommunicationValue Value 0x07

An invalid RAPTER message was received from the device by the control module within the WebSocket connection.

MessageReceivedInInvalidContext Value 0x08

A message was received from the device at a time when that message was not expected. Ex., A reply to a request that was never sent. Or a message was received before that `MessageGroup` version was negotiated.

InvalidMessageTypeReceivedValue Value 0x09

The device sent the control module an invalid message type. E.g.,a radiation detector module sent a message type in the vehicle presence message group.

InvalidMessageFormatReceivedValue Value 0x0a

The device sent a message that was not able to be decoded by the control module. E.g., message was not long enough to contain a string of the claimed length; or the message received was shorter than that type of message would be allowed.

InvalidMessageFieldEntryReceivedValue Value 0x0b

A value for one of the fields of a message was invalid. E.g., the value for an enumeration field did not match any defined value for the enumeration.

RequiredResponseAbsentValue Value 0x0c

The device is not responding to a message requiring a response.

CoreMessageGroupVersionInvalidValue Value 0x0d

A mutually supported version of the core message group could not be agreed upon.

NonCoreMessageGroupVersionInvalidValue Value 0x0e

A mutually supported version of a non-core message group could not be agreed upon.

DuplicateResourceValue Value 0x0f

The device was a duplicate in some way. Either it was the second analysis module, or it contained a already existing UUID, or something similar.

DesiredFeatureNotSupportedValue Value 0x10

The control module is unable to use the device because it does not support a feature the control module requires. E.g., the control module requires gamma detectors to support listmode data collection, but the connected gamma detector does not support listmode data collection.

ExcessiveNetworkTrafficValue Value 0x11

The device was causing excessive network traffic, so was disconnected.

UnsatisfactoryMessagingPerformanceValue Value 0x12

The device messaging performance was not acceptable. For example if responses are consistently taking to long, to many TCP re-transmits are necessary, or connections keep getting dropped.

UnsatisfactoryTimeSynchronizationValue Value 0x13

The time synchronization between the control module and the device could not be established well enough to allow operations. Examples might include: the times reported by the device jump around relative to the control module, or the absolute difference between the devices time and the control modules time is larger than acceptable and has not corrected itself via the PTP mechanism in a reasonable amount of time.

OtherControlModuleInitiatedDisconnectValue Value 0x14

The control module disconnected the device for another reason not otherwise enumerated.

OtherDisconnectTypeValue Value 0x15

The connection was disconnected for another reason not enumerated.

4.6.43 DataOutDeviceDisconnectedPush Message

Message pushed from the control module to all DataOut devices that a device has disconnected from the control module. If the control module itself is re-booting, then the IP address should be "::1" for IPv6. A `DataOutMiscNotificationPush` may be separately sent to explain possible odd conditions.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x27
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`timestamp`	UtcTimePoint
Field Description: Approximate timestamp of disconnect.	
`ipaddress`	`Short String`
Field Description: IP address of the device.	
`device_uuid`	`Uuid128`
Field Description: UUID of the disconnecting device, if known; all zeros if not.	
`reason`	`uint8_t` enumerated by `DeviceDisconnectReason`
Field Description: The reason for the disconnect.	

Continued on next page

Table continued from previous page

Field Name	Data Type
remark	Short String
Field Description: Optional free form text description of disconnect. Examples include: "An analysis module is already present.", "ParameterUpdatePush message indicated unrecognized parameter name."	

4.6.44 DataOutDeviceDisconnectedPushAck Message

A message acknowledging the receipt of a DataOutDeviceDisconnectedPush message sent by the control module.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xa7
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.45 DataOutHandshakeFinishedPush Message

Message sent by the control module to DataOut devices whenever a new device has connected to the control module, and the handshake has completed (i.e, the control module has gotten the device information).

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x28
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`handshake_finish_timestamp`	UtcTimePoint
Field Description: Approximate timestamp of finishing the handshake.	
`ipaddress`	`Short String`
Field Description: IP address of the device. This must match to the IP address specified in `DataOutDeviceConnectedPush::ipaddress` and `DataOutDeviceDisconnectedPush::ipaddress`.	
`device_type`	`uint8_t` enumerated by `MessageGroup`
Field Description: The device type, as determined from the following options (designated by the associated MessageGroup value): command device (Command), analysis device (Analysis), radiation detection device (RadDetector), vehicle presence device (VehiclePresence), or control module (Core). The device type determines the message groups that the device must support. See Chapter 2 for details.	
`data_collection_modes`	`uint8_t` bits defined by `DataCollectionModes`
Field Description: Bitwise OR of flags representing data collection modes supported by the device.	
`device_features`	`uint64_t` bits defined by `DeviceFeaturesFlags`
Field Description: Bitwise OR of flags representing device features.	
`buffer_size`	`uint64_t`
Field Description: The size of the buffer for buffering. This size can be an estimate. Should be set to 0 if the device does not support data buffering when the connection is down. When the network connection is down, Push messages should be buffered for retrieval once the connection is back up; see `BufferedMessagesRequest` and `BufferedMessagesReply` for details.	
`num_subdetectors`	`uint8_t`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The number of sub detectors the device contains; applicable only to radiation detector, vehicle presence, and power management modules. For other devices this must be set to 0. As examples, for gamma-ray detectors this might indicate independent crystals (e.g. 4 crystals in one panel or module), for neutron detectors this might be the number of independently read-out He-3 tubes, and for vehicle presence sensors, the number of IR beams plus number of cameras. For power management modules it represents the number of independent output lines. Note that all sub-detectors must operate in the same data collection mode (DataCollectionModes::ClockTimeIntervalValue, DataCollectionModes::OnEventValue, etc) and at the same data collection intervals (10ms, 100ms, etc.). See also: • RadListModeDataPush::subdetector_number • RadChannelDataPush::subdetector_number • RadEnergyCalibrationUpdatePush::subdetector_number • RequestEnergyCalibration::subdetector • VehiclePresenceSubDetectorInformationRequest • PwrMngmtInformationRequest	
device_uuid	Uuid128
Field Description: The universally unique ID for this device. This number must be globally unique, and can be assigned according to the deploying agency conventions; for example assigned as the thumbprint of the devices cryptographic certificate. This value will not change for a given physical device.	
serial_number	Short String
Field Description: The manufacturer specified serial number that is expected to be unique at least with respect to the vendor and model. This value must remain fixed across firmware upgrades, power cycles, setting changes, and if at all possible hardware repairs. No restrictions on format other than length, and valid UTF-8 text.	
manufacturer	Short String
Field Description: Specifies the manufacturer of this device. No restrictions other than length, and valid UTF-8 text.	
model	Short String
Field Description: Specifies the model of this device. No restrictions other than length, and valid UTF-8 text.	
hardware_version	Short String
Field Description: Specifies the version of the hardware for the device. No restrictions other than length, and valid UTF-8 text. This should be descriptive and change with respect to hardware changes.	
firmware_version	Short String
Field Description: Specifies the version of the current firmware for the device. No restrictions other than length, and valid UTF-8 text. This should be descriptive and change with respect to firmware changes or updates.	
other_component_versions	Short Array of ComponentVersionInformation

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Versions of additional components in the device, that may be informative for compatibility, troubleshooting, or maintenance. Examples include: underlying operating system version, high voltage daughter card version, nuclide library version, analysis library version, etc.	
specialized_capabilities_description	Short String
Field Description: A free form English language description of the capabilities and hardware specific to the specialized purpose of this device. No restrictions other than length, and valid UTF-8 text. Examples might include: • "3x3 NaI gamma detector, 7% FWHM @ 661 keV, temperature compensated energy calibration, pulse pileup filter, 250MHz waveform sampling" • "Data archiving device to save all health information to a database" • "Graphical user interface device for portal operations. Touchscreen." • "Analysis software based on DHSIsotopeID v13, customized for PVT."	
computing_platform_description	Short String
Field Description: A free form English language description of the computing resources of the device, such as descriptions of its CPUs, ram memory, hard drive space, Ethernet adapter capabilities, operating system, etc. No restrictions other than length, and valid UTF-8 text. Examples include: • "PC4F1453 MCU, 120 MHz, 256 KB SRAM, 4GB SD-card storage, IEEE-1588 Ethernet transceiver, FreeRTOS 9.0.0" • "Dual core 2 GHz Arm CPU, 1 GB ram, Linux kernel 3.4."	

4.6.46 DataOutHandshakeFinishedPushAck Message

A message acknowledging the receipt of a DataOutHandshakeFinishedPush message sent by the control module.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xa8
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.47 DataOutResponseErrorType Enumeration

Categories of general communication issues that may arise

Used in message: `DataOutDeviceResponseIssuePush`

Underlying integral representation: `uint8_t`

Enumerated values for `DataOutResponseErrorType`:

SingleMessageResponseTimeoutValue Value 0x01

Device timed out for a single message, but still responds to echos/pings and other RAPTER messages and lower level network traffic, such as pings.

DeviceNotRespondingToRequestsValue Value 0x02

Device isn't responding to RAPTER messages, but device still responds to lower level network traffic, such as pings.

DeviceNotProducingExpectedPushValue Value 0x03

Device is not producing expected "push" messages

DeviceCommunicationsErrorOtherValue Value 0x04

A communications error not captured by other enumerated conditions; message should have an English description as well.

DeviceCommunicationsErrorResolvedValue Value 0x05

Previous communications error has been resolved.

4.6.48 DataOutDeviceResponseIssuePush Message

Message pushed from the control module to all DataOut devices to let them know that a connection to a device hasn't been terminated (as far as the control module knows), but the device has timed out or not responded to a message, or that the issue, that caused a previous one of these messages to be sent out, has been cleared up. The control module shouldn't necessarily send out one of these messages for each timed out message. Sending just the first message that triggers a timeout for a given context avoids flooding the network with messages).

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_type	uint8_t with value 0x29 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
timestamp	UtcTimePoint *Field Description*: Time when issue was first diagnosed
device_uuid	Uuid128 *Field Description*: UUID of device having comms issues
issue_message_id	uint32_t *Field Description*: The id of the message sent to the device that timed out, for debugging purposes; may be zero, especially if an issue with not receiving expected push messages.
time_waited_ms	uint32_t *Field Description*: The approximate time, in milliseconds, that this issue took to manifest, e.g., how long you waited without a message response.
issue_message_group	uint8_t enumerated by MessageGroup *Field Description*: The message group of the message having an issue.
issue_message_type	uint8_t *Field Description*: The message type within the MessageGroup (ex. CoreMsgType::DeviceInfoRequestValue, CoreMsgType::UseMessageGroupVersionRequestValue, VehiclePresenceMsgType::VehiclePresenceSubDetectorInformationRequestV… etc) of the message having trouble. Will have a value of 255 if this is a general connection issue.
issue_error_type	uint8_t enumerated by DataOutResponseErrorType *Field Description*: The type of comm error detected
issue_description	Short String *Field Description*: Description of the issue; should especially be filled out in the case of DataOutResponseErrorType::DeviceCommunicationsErrorOtherValue

4.6.49 DataOutDeviceResponseIssuePushAck Message

A message acknowledging the receipt of a DataOutDeviceResponseIssuePush message sent by the control module.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0xa9 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.6.50 BufferingForDataOutOption Enumeration

Enumeration to represent the buffering option a device requests of the control module to do on its behalf.

Used in message: DataOutBufferingEnableRequest

Underlying integral representation: uint8_t

Enumerated values for BufferingForDataOutOption:

DisableBufferMessagesValue Value 0x00

Useful to turn off buffering by the control module for the device. For example if the device knows it will be shutting down (or otherwise going away) and the buffering will be pointless. When the control module receives a DataOutBufferingEnableRequest with this status, it should clear all contents of its buffer for this device, even if it is currently sending buffered messages to the device. By default the control module does not buffer, so no need to send a DataOutBufferingEnableRequest message with this value unless buffering was explicitly already enabled.

EnableBufferingMessageValue Value 0x01

The device is requesting that the control module buffer data for the device.

4.6.51 DataOutBufferingEnableRequest Message

A message sent by a DataOut device to the control module requesting that the control module buffer its push DataOut messages, in the event of a temporary disconnection.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x2e
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
buffering_option	uint8_t enumerated by BufferingForDataOutOption
Field Description: Whether the device requests that the control module buffer push messages to the device.	

4.6.52 DataOutBufferingStatus Enumeration

Enumeration to represent response possibilities to a DataOutBufferingEnableRequest.

Used in message: DataOutBufferingEnableReply

Underlying integral representation: uint8_t

Enumerated values for DataOutBufferingStatus:

BufferingEnabledValue Value 0x00

The control module was able, and did, enable buffering for the device. The device must send acknowledges to push messages to keep the control module's buffer from overflowing.

BufferingDisabledValue Value 0x01

Buffering is not enabled by the control module for this device. The device does not need to send acknowledges to push messages.

BufferingNotSupportedValue Value 0x02

Buffering is not supported by the control module. The device does not need to send acknowledges to push messages.

4.6.53 DataOutBufferingEnableReply Message

A reply by the control module to a DataOutBufferingEnableRequest message sent by a data-out device.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xae
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
buffering_status	uint8_t enumerated by DataOutBufferingStatus
Field Description: The status of the control module buffering for the requesting DataOut device.	
buffer_capacity	uint32_t
Field Description: Approximate buffer capacity for this device, in bytes; zero if buffering is not supported.	
current_buffer_contents	uint32_t
Field Description: Size of the current contents in buffer for this device, in bytes; zero if buffering not supported or not being performed.	

4.6.54 DataOutBufferedMessagesRequest Message

Message sent by a data-out device requesting the control module send any buffered messages after a RAPTER connection is established. This request must be sent after the device has sent the control module a DeviceInfoReply message, as well as negotiated the version of MessageGroup::DataOutValue to use, but before the device has sent acknowledges to any push messages the control module may have sent the device, and before the device sends a DataOutBufferingEnableRequest to the control module.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x2f
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
recovery_mode	uint8_t enumerated by BufferRecoveryMode
Field Description: How the recovered data should be sent.	

4.6.55 DataOutBufferedMessagesReply Message

Message sent in response to an DataOutBufferedMessagesRequest request; multiple instances of these messages will be sent in response to the original request message. One of these messages will be sent before re-sending the buffered data, and one of these messages will be sent once sending of the buffered data is complete. However there may be other of these messages sent, if, for instance, there is a failure in re-sending the data, the original message will be followed up to indicate the failure. If the control module is able to provide the buffered data, it should return this message followed by the buffered data.

If recovery mode is BufferRecoveryMode::SerialValue, then data currently being taken should not be sent until the last currently buffered message is sent. (This applies to essentially any message falling into the buffer category, having an Ack in the name of the reply.) Processing of commands which do not fall into the buffering scheme (so do not have an "Ack" in the name of the replies) can proceed even while the recovery (i.e. the communications 'catch up') is still happening. Note that if any buffer overflow (causing discarded messages) happens during this catch-up, a DeviceStatusPush update message should be sent to indicate this. Upon successful completion of the buffer transfer a DataOutBufferedMessagesReply message should be sent with the DataOutBufferedMessagesReply::buffered_status field set to BufferedDataRequestStatus::SendCompletedSuccessfullyValue.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xaf
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
buffered_status	uint8_t enumerated by BufferedDataRequestStatus
Field Description: Status of the buffering operation.	
oldest_recovered_msg_id	uint32_t
Field Description: The message ID of the oldest message that will be, or was sent. If DataOutBufferedMessagesReply::buffered_status is BufferedDataRequestStatus::NotSupportedValue, BufferedDataRequestStatus::NoBufferedContentsValue, or BufferedDataRequestStatus::SendAlreadyInProgressValue, then this variable should be ignored.	
newest_recovered_msg_id	uint32_t
Field Description: When DataOutBufferedMessagesReply::buffered_status is BufferedDataRequestStatus::WillSendValue, then this is the most recent message in the buffer. When DataOutBufferedMessagesReply::buffered_status is BufferedDataRequestStatus::SendFailedValue, this is the most recent message to enter the buffer, or zero if none. If status is BufferedDataRequestStatus::SendCompletedSuccessfullyValue then this was the last message sent from the buffer. For all other DataOutBufferedMessagesReply::buffered_status values this variable should be ignored. Note that if BufferRecoveryMode::SerialValue option was selected, there may be subsequent messages after this ID that are marked with the MsgFlags::MessageHasBeenBufferedFlag bit, and the final BufferedMessagesReply sent after buffer has been emptied will have a DataOutBufferedMessagesReply::newest_recovered_msg_id indicating the last message transfered before the buffer was emptied (so might be data taken after the buffer recovery was begun).	
number_failed_messages	uint32_t
Field Description: Indicates the number of buffered messages that either will not be, or were not, recovered. When BufferedMessagesReply::buffered_status is BufferedDataRequestStatus::WillSendValue, this field indicates the number of messages that are known to not be able to be recovered, for example because the buffer overflowed. When BufferedMessagesReply::buffered_status has a value BufferedDataRequestStatus::SendFailedValue or BufferedDataRequestStatus::SendCompletedSuccessfullyValue this variable indicates the number of message that were not able to be sent. For other values of BufferedDataRequestStatus, this value must be zero.	

4.6.56 SystemStateChangeStatus Enumeration

Enumeration to hold possible statuses of the reply to a CmdSystemStateChangeRequest request.

Used in message: DataOutSystemStateChangePush

Underlying integral representation: uint8_t

Enumerated values for SystemStateChangeStatus:

CompletedNoIssuesValue Value 0x00

The system state change has successfully completed, and no devices gave any issues or transition failures.

InProgressValue Value 0x01

The state change is in progress; at least one more reply will be sent to indicate the final transition status.

NoStateChangeNeededValue Value 0x02

System was already in the requested state so no action was taken.

InvalidRequestValue Value 0x03

The CmdSystemStateChangeRequest request was invalid, for example if not all the neutron detectors supported the requested DataCollectionModes. A DataOutMiscNotificationPush may be separately sent to provide insight into the reason.

StateChangeCanceledValue Value 0x04

Either before the transition was scheduled to occur, or while the transition was underway, another CmdSystemStateChangeRequest was received that canceled this transition.

CompletedWithIssueValue Value 0x05

The system state change was completed, however there was at least one issue; a DataOutMiscNotificationPush may be separately sent to provide additional information.

CompletedWithDeviceFailureValue Value 0x06

The system state change was completed, however there was at least one device that failed the transition.

CouldNotCompleteValue Value 0x07

The system state change could not be completed. A DataOutMiscNotificationPush may be separately sent to provide insight into the reason.

4.6.57 DeviceStateChangeInfo Struct

This struct is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold information on a change in operating state of a device.

Message Contents:

Field Name	Data Type
device_uuid *Field Description*: The universally unique ID for this device.	Uuid128
status_of_transition *Field Description*: Status of the transition. Note that the device must send a message with status of CommandReplyStatus::CompletedValue or CommandReplyStatus::CompletedWithIssueValue after all data of the previous state have been sent and before any data of this new state are sent.	uint8_t enumerated by CommandReplyStatus
previous_device_operating_mode *Field Description*: The operating mode of the device at the time that the request to change state was received.	uint8_t enumerated by OperatingMode
previous_device_data_collection_mode *Field Description*: The data collection mode of the device at the time that the request to change state was received.	uint8_t enumerated by DataCollectionModes
previous_device_collection_interval_ms *Field Description*: The data collection interval of the device at the time that the request to change state was received.	uint32_t
previous_measurement_type *Field Description*: The measurement type in effect on the device at the time that the request to change state was received.	uint8_t enumerated by MeasurementType
requested_device_operating_mode *Field Description*: The requested new operating mode for the device.	uint8_t enumerated by OperatingMode
requested_device_data_collection_mode *Field Description*: The requested new data collection mode for the device.	uint8_t enumerated by DataCollectionModes
requested_device_collection_interval_ms *Field Description*: The requested new data collection interval for the device.	uint32_t
requested_measurement_type *Field Description*: The requested new measurement type to be applied by the device.	uint8_t enumerated by MeasurementType
new_state_effective_timestamp *Field Description*: Time that the state change was completed (CommandReplyStatus::CompletedValue or CommandReplyStatus::CompletedWithIssueValue), or is predicted to be completed (CommandReplyStatus::InProgressValue), or failed (CommandReplyStatus::FailedValue), or was cancelled (CommandReplyStatus::CanceledValue), or a redundant request (CommandReplyStatus::RedundantValue) was received. If the status is CommandReplyStatus::InProgressValue, then this value may be zero if there is no estimate available for when the transition will finish.	UtcTimePoint

4.6.58 DeviceNotification Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

A struct to hold information relating to an error or issue. This struct is used by the control module in DataOut messages, and includes the UUID of the device that has reported an issue along with the information included from the `NotificationPush` message from the originating device.

Message Contents:

Field Name	Data Type
`device_uuid`	`Uuid128`
Field Description: The universally unique ID for this device.	
`timestamp`	UtcTimePoint
Field Description: The timestamp when the error actually happened, not when it was sent on the network. If the time is not applicable to this particular message, this integer should have the value of zero.	
`reference_message_id`	`uint32_t`
Field Description: If this notification is being sent with regards to another message, this field provides the 'message_id' of that message. If this notification is not in regards to another message, then this field must be zero. An example of when a notification may reference another message would be if `CmdSystemStateChangeRequest` can not be successfully completed, in addition to the device sending a `CmdSystemStateChangeReply` indicating the request could not be completed, the device could also send a notification with an English description with further device specific information that may help to diagnose the issue. If non-zero, this message_id field must correspond to a message sent within the time specified by `RapterConstants::DATA_PUSH_TIMEOUT_MS`.	
`severity`	`uint16_t` enumerated by `NotificationSeverity`
Field Description: Indicates the severity of the issue reported by this notification.	
`cause`	`uint8_t` enumerated by `NotificationCause`
Field Description: The cause of the notification; should be considered to be indicative or descriptive, and not an absolute or definitive form of information, since there will likely be errors or issues that are not enumerated in the `NotificationCause` notification types.	
`description`	`Large String`
Field Description: Human readable, UTF-8 encoded text which describes the error or issue in a manner appropriate to display to a technician to provide information or to help diagnose the problem.	

4.6.59 DataOutSystemStateChangePush Message

Message sent by the control module to all devices that receive DataOut messages, in response to a valid `CmdSystemStateChangeRequest`. This message forwards the status, as of the time of the message, of those devices that were asked to change their status by the request. Devices that were not affected by the `CmdSystemStateChangeRequest` (e.g. a DataOut device already in the requested state, that will continue being in the same state) do

not have to be reported in this message. For each CmdSystemStateChangeRequest message that the control module decides to act on, it will send out at least two DataOutSystemStateChangePush messages, but may send out more. The first DataOutSystemStateChangePush message should be sent out as early as possible from the control module to let the DataOut devices know about the request. The final DataOutSystemStateChangePush gives the result of the transition, and also the final ChangeDeviceStateReply of any of the transitioning devices that hadn't been communicated yet. Additional intermediate DataOutSystemStateChangePush messages may be utilized to to convey status updates, or estimates. For each device whose state changed as a result of the CmdSystemStateChangeRequest, at least their final transition information should be included in DataOutSystemStateChangePush::device_statuses.

See Also: DataOutSystemStateChangePush , DataOutSystemStateChangePushAck , CmdSystemStateChangeRequest

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x2a
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
requestor_uuid	Uuid128
Field Description: The UUID of the device that requested this transition.	
requested_transition_time	UtcTimePoint
Field Description: The time the requestor targeted the transition to finish; if no time was specified (e.g. an immediate transition), this time represents the time the control module began processing the request. All DataOutSystemStateChangePush messages that are in response to a CmdSystemStateChangeRequest must have the same values of DataOutSystemStateChangePush::requested_transition_time.	
requested_system_operating_mode	uint8_t enumerated by OperatingMode
Field Description: At the completion of the requested system state change, all devices and the control module should ideally be in this operating mode.	
requested_gamma_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: The data collection mode to which all gamma radiation detection modules are to transition.	
requested_neutron_data_collection_mode	uint8_t enumerated by DataCollectionModes

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The data collection mode to which all neutron radiation detection modules are to transition.	
requested_vehicle_presence_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: The data collection mode to which all vehicle presence detection modules are to transition.	
requested_collection_interval_ms	uint32_t
Field Description: The data collection interval to which all data collecting devices are to transition.	
requested_measurement_type	uint8_t enumerated by MeasurementType
Field Description: The new measurement type for the system.	
effective_measurement_type	uint8_t enumerated by MeasurementType
Field Description: The currently measurement type of the system.	
status_of_system_transition	uint8_t enumerated by SystemStateChangeStatus
Field Description: The overall status of the system transition. (The initial and requested status of the individual devices is given in the array of structs contained in the DataOutSystemStateChangePush::device_statuses field of this message, in the struct field StateChangeInfo::status_of_transition)	
device_statuses	Short Array of DeviceStateChangeInfo
Field Description: Statuses sent from each of the devices that will be changing status due to a particular CmdSystemStateChangeRequest message. The control module may collect state change information from multiple devices before sending out the DataOutSystemStateChangePush message.	
device_notifications	Short Array of DeviceNotification
Field Description: Array of structs, one per error or notification condition being reported. Each of the structs contains the identity of the device, timestamp of the issue, the cause and severity of the issue, and a string field to contain a description.	

4.6.60 DataOutSystemStateChangePushAck Message

A message acknowledging the receipt of a DataOutSystemStateChangePush message sent by the control module.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xaa
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.61 DataOutCurrentSystemStateRequest Message

Message sent by a data-out device to the control module requesting information on the current system state.

See Also: DataOutSystemStateChangePush

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x3c
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.62 DataOutCurrentSystemStateReply Message

Reply to a DataOutCurrentSystemStateRequest message, conveying information on the system state of the portal; this information should match the information in the most recent DataOutSystemStateChangePush message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xbc

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`state_requestor_uuid`	`Uuid128`
Field Description: The UUID of the command device, or the control module, that requested the portal transition to the current state. If the system state transition was driven by default behaviour of the control module, or operating procedures (e.x., devices failed necessitating a state change), then the UUID of the control module will be given. If the state was explicitly requested by a command device, then its UUID will be specified.	
`system_state_start_time`	UtcTimePoint
Field Description: The starting time of the current system state.	
`system_operating_mode`	`uint8_t` enumerated by `OperatingMode`
Field Description: The current system operating mode.	
`gamma_data_collection_mode`	`uint8_t` enumerated by `DataCollectionModes`
Field Description: Data collection mode of gamma modules.	
`neutron_data_collection_mode`	`uint8_t` enumerated by `DataCollectionModes`
Field Description: Data collection mode of neutron modules.	
`vehicle_presence_data_collecti↲ on_mode`	`uint8_t` enumerated by `DataCollectionModes`
Field Description: Data collection mode of vehicle presence modules.	
`collection_interval_ms`	`uint32_t`
Field Description: Current collection interval in milliseconds.	
`requested_measurement_type`	`uint8_t` enumerated by `MeasurementType`
Field Description: The system measurement type specified by the device that requested the current state. A value of `MeasurementType::NotSpecifiedValue` indicates the `MeasurementType` is being determined based on sensor (e.g., vehicle presence sensor) values.	
`effective_system_measurement_type`	`uint8_t` enumerated by `MeasurementType`
Field Description: The effective measurement type of the systems (i.e. if auto determined, its actual auto determined value)	

4.6.63 DataOutMiscNotificationPush Message

A message sent from the control module to all DataOut devices to convey information for which there is no dedicated mechanism to convey. The intent of this message is that its content will go somewhere like a log file or operational GUI. This message may be sent to an individual DataOut or Command device to notify that device of issues related to one of its requests, or it may be the case that effectively a copy of the same message may be sent out to all DataOut and Command devices.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x2b
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`timestamp`	UtcTimePoint
Field Description: Time when issue happened.	
`reference_message_id`	`uint32_t`
Field Description: If this notification is being sent with regards to another message, this field provides the 'message_id' of that message. If this notification is not in regards to another message, then this field must be zero. An example of when a notification may reference another message would be if `CmdSystemStateChangeRequest` can not be successfully completed, in addition to the device sending a `CmdSystemStateChangeReply` indicating the request could not be completed, the device could also send a notification with an English description with further device specific information that may help to diagnose the issue. If non-zero, this message_id field must correspond to a message sent within the time specified by `RapterConstants::DATA_PUSH_TIMEOUT_MS`.	
`severity`	`uint16_t` enumerated by `NotificationSeverity`
Field Description: The type of message within the message group.	
`device_uuid`	`Uuid128`
Field Description: The UUID of the device that has experienced the issue that is the subject of the notification. If the issue does not concern a specific device, or a device whose UUID is not yet known, this should be left blank (all zeros). The control module can also put its own UUID to communicate issues related to itself.	
`description`	`Medium String`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: A human readable UTF-8 encoded string providing a description of the issue being reported.	

4.6.64 DataOutMiscNotificationPushAck Message

A message acknowledging the receipt of a DataOutMiscNotificationPush message sent by the control module.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xab
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.65 DataOutDeviceReferenceInfoRequest Message

A message sent by a DataOut device to the control module requesting the reference info (position, name, comments) of a device, or a sub-detector, that was manually entered.

See Also: `CmdDeviceSetReferenceInfoRequest`

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x2c
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`device_uuid`	`Uuid128`
Field Description: The UUID for the device you would like the reference information for.	
`subdetector_number`	`uint8_t`
Field Description: The subdetector you would like the reference information for. A value of zero indicates information is wanted for the device. A non-zero value indicates information is wanted for the specified sub-detector.	

4.6.66 DeviceReferenceInfoRequestStatus Enumeration

An enum to describe the status of a `DataOutDeviceReferenceInfoRequest`.

Used in message: `DataOutDeviceReferenceInfoReply`

Underlying integral representation: `uint8_t`

Enumerated values for `DeviceReferenceInfoRequestStatus`:

ReferenceInfoSuccessfulValue Value 0x00

Request was successful.

ReferenceInfoNotSetValue Value 0x01

Information has not been set for the requested device and/or sub-detector.

ReferenceInfoInvalidUuidValue Value 0x02

A UUID for a target device not connected to the control module was specified in the request.

ReferenceInfoInvalidSubDeviceNumberValue Value 0x03

An invalid subdetector number for the particular target device was specified in the request.

4.6.67 DeviceReferenceInfoSetFlags Enumeration

Flags to specify what reference information is set, for variables that could be ambiguous.

Underlying integral representation: `uint8_t`

Enumerated values for `DeviceReferenceInfoSetFlags`:

ReferenceInfoPositionIsSetFlag Value 0x01

The position is set. If this flag is not set, the position is not usable. Position follows the coordinate system and portal configuration conventions, as defined in Chapter 2, including the naming of radiation detection panels which sets the positive z direction.

4.6.68 DataOutDeviceReferenceInfoReply Message

A reply by the control module to a DataOutDeviceReferenceInfoRequest message sent by a data-out device requesting the reference information of a device.

See Also: CmdDeviceSetReferenceInfoRequest

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0xac *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
device_uuid	Uuid128 *Field Description*: The UUID for the device for which the reference information was requested. Must match the UUID specified in the request.
subdetector_number	uint8_t *Field Description*: The subdetector the reference information is for. Must match the DataOutDeviceReferenceInfoRequest::subdetector_number specified in the request. A value of zero indicates the information is for the device as a whole.
status_of_request	uint8_t enumerated by DeviceReferenceInfoRequestStatus *Field Description*: The status of the reference information query.
time_set	UtcTimePoint *Field Description*: Time when the coordinates reference information was set. Microseconds after the UNIX epoch
name	Short String *Field Description*: Name for this device, or sub-device that should be used when storing information in N42 files, databases, or log files. For gamma and neutron sub-detectors, examples are: 'Aa1', 'Ca1N', 'Da2'. For RSPs (e.g., subdetector zero), examples are: 'Panel1', 'RSP2'. For vehicle presence sensors, examples are: 'BreakBeam1', 'EntranceCamera'. Note that if no name is entered, it is unspecified what name should be used in N42 files or databases, but some possibilities are the UUID of the device, a guessed name based on location, or a randomly assigned name.
set_fields	uint8_t bits defined by DeviceReferenceInfoSetFlags

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags to indicate if ambiguous quantities have been set.	
x_cm	float
Field Description: The x-location; see CmdDeviceSetReferenceInfoRequest for definition.	
y_cm	float
Field Description: The y-location; see CmdDeviceSetReferenceInfoRequest for definition.	
z_cm	float
Field Description: The z-location; see CmdDeviceSetReferenceInfoRequest for definition.	
position_description	Short String
Field Description: A free-form description of the location. Examples might be: 'Lower passenger side RSP', or 'Lower detection element'.	
comment	Short String
Field Description: A free-form text comment about this module or subdetector.	

4.6.69 DataOutTimeStatisticsRequest Message

Message sent from a data out device to the control module in order to request time statistics of all devices currently connected to the control module, as well as the control module itself.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x02
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x2d
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.70 DataOutTimeStatisticsReplyStatus Enumeration

The status of a reply to a DataOutTimeStatisticsRequest message.

Used in message: DataOutTimeStatisticsReply

Underlying integral representation: uint8_t

Enumerated values for `DataOutTimeStatisticsReplyStatus`:

FinalAllConnectedDevicesIncludedValue Value 0x00

The time statistics of all devices connected to the control module, as well as for the control module itself, are included in this `DataOutTimeStatisticsReply` message.

FinalErrorGettingSomeDevicesResponsesValue Value 0x01

Querying at least one of the connected devices for time statistics failed. Statistics for the devices for which it was successful are included in this `DataOutTimeStatisticsReply` message.

WaitingOnResponseFromDevicesValue Value 0x02

The request has been received, and devices are being queried. Another `DataOutTimeStatisticsR` will be sent with a status of either `DataOutTimeStatisticsReplyStatus::FinalAllConnec` or `DataOutTimeStatisticsReplyStatus::FinalErrorGettingSomeDevicesRespon`

4.6.71 IdentifiedDeviceTimeStatistics Struct

This `struct` is not a RAPTER message, but is used as an element in an array of a RAPTER message.

Message Contents:

Field Name	Data Type
`device_uuid` *Field Description*:	`Uuid128`
`timestamp` *Field Description*: The time at which these statistics were calculated. Microseconds since the UNIX epoch.	UtcTimePoint
`current_uptime_seconds` *Field Description*: The time, in seconds, since the device has been started or restarted. Similar to 'uptime' command in Linux and BSD.	`uint32_t`
`current_session_time_seconds` *Field Description*: For non-control modules, the number of seconds the current RAPTER WebSocket session has been established for the device. For the control module, the amount of time, in seconds, continuously available, up to the present, to accept WebSocket connections.	`uint32_t`
`current_session_time_operating_seconds`	`uint32_t`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: For non-control modules, the number of seconds the device has been in OperatingMode::OperatingValue during the current RAPTER WebSocket session. Must be no greater than IdentifiedDeviceTimeStatistics::current_session_time_seconds, i.e., even if the device was operating when session began, because buffering was enabled, this value should start accumulating at the beginning of the current session, not when data acquisition actually started. For the control module, then this should be the longest uninterrupted time that any connected device has been in OperatingMode::OperatingValue.	
current_operating_mode_time_seconds	uint32_t
Field Description: For devices, the number of seconds since the most recent change of the device's OperatingMode. May be larger than IdentifiedDeviceTimeStatistics::current_session_time_seconds if not a control module and the device was previously in the current state before the RAPTER WebSocket connection was started. For the control modules, the time since DataOutSystemStateChangePush::requested_system_operating_mode has been changed, or if it hasn't been changed, the time since system start up.	
cumulative_time_on_seconds	uint32_t
Field Description: The number of seconds this device has been powered on. This value should monotonically accumulate across power cycles, firmware upgrades, and if at all possible, hardware upgrades.	
cumulative_time_operating_seconds	uint32_t
Field Description: The number of seconds this device has been powered on and in the OperatingMode::OperatingValue. This value should monotonically accumulate across power cycles, firmware upgrades, mode changes, and if at all possible, hardware upgrades.	
number_restarts	uint32_t
Field Description: Number of times, at the application level, the device has restarted. That is, if the device is implemented as a software program application running within an operating system, this number would be incremented each time the application is started by the operating system, wether due to it crashing, or the device power cycling. If the device is implemented as a real time operating system, Unikernel, or dedicated hardware, this number would increment each power cycle. This value should monotonically accumulate across power cycles, firmware upgrades, and if at all possible, hardware component upgrades.	
number_websocket_connections_initiated	uint32_t
Field Description: If a device: cumulative number of WebSocket connection it has attempted to initiate over its lifetime, whether the connection was successful or not. If a control module: cumulative number of WebSocket connections devices have attempted to make to it, whether the connection was successful or not. This value should monotonically accumulate across power cycles, firmware upgrades, and if at all possible, hardware component upgrades.	
number_completed_handshakes	uint32_t

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The number of times a WebSocket connection resulted in at least negotiating the message group. This value should monotonically accumulate across power cycles, firmware upgrades, and if at all possible, hardware component upgrades.	

4.6.72 DataOutTimeStatisticsReply Message

Message from the control module to the requesting device containing an array of elements, each of which contains time statistics for an individual, identified device.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xad
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`reply_status`	`uint8_t` enumerated by `DataOutTimeStatisticsReplyStatus`
Field Description: The status of the reply, whether final, waiting, or error.	
`device_statistics`	`Short Array` of `IdentifiedDeviceTimeStatistics`
Field Description: Array of IdentifiedDeviceTimeStatistics. Each element of the array holds the time statistics from one device. An element is included for each device in the system.	

4.6.73 DataOutInterimAnalysisPush Message

Message pushed from the control module to each of the DataOut devices informing them of an interim analysis received from the analysis module, with the occupancy still in progress.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x3a

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
analysis_module_uuid	Uuid128
Field Description: The UUID of the analysis module that produced this analysis result. Must match the UUID given by DeviceInfoReply of the analysis module.	
result_timestamp	UtcTimePoint
Field Description: TODO: decide if we need a UUID for each analysis result The time the analysis was started.	
result_uuid	Uuid128
Field Description: UUID generated by the analysis module for this result. This must be generated in such a way that it uniquely identifies this result.	
interim_analysis_status	uint16_t bits defined by InterimAnalysisDataUsageStatusFlags
Field Description: Indication of whether the data are being used for background aggregation or for the analysis of an item of interest.	
alarm_type	uint16_t bits defined by AlarmTypeFlags
Field Description: Bitwise OR of flags indicating a gamma, neutron, etc, alarm has been triggered. If this is zero, then no alarm has been found.	
sample_number	uint32_t
Field Description: The item of interest number these results are for; must match DataOutPacket::sample_number of the data this result is for.	
analysis_confidence_value	float
Field Description: Indication of confidence, as a percent ranging from 0.0 to 100.0 (specified with a filed value ranging from 0.0 to 1.0), in the overall accuracy of the analysis, where increasing values indicate higher confidence. If a confidence is not ascribed, this should be set to a negative value (other than negative infinity or negative 0), with the magnitude having not carrying any meaning. (Taken roughly from N42 2012 standard for <AnalysisConfidenceValue >)	
nuclides	Short Array of NuclideResult
Field Description: The analysis algorithms identified results; does not strictly have to be nuclides (e.x., U235, Co60, etc.), but may be other sources as well (e.x., Bremsstrahlung, Annihilation, Gross Count, etc.). The same nuclide may be specified multiple times covering different ranges of sample numbers.	
analysis_result_description	Short String

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Free-form text describing the overall conclusion of the analysis regarding the source of concern. (See N42.42 2012 standard for <AnalysisResultDescription >)	

4.6.74 DataOutInterimAnalysisPushAck Message

A message acknowledging the receipt of a `DataOutInterimAnalysisPush` message sent by the control module.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xba
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.6.75 DataOutFinalAnalysisPush Message

Message pushed from the control module to all DataOut devices informing and conveying to them a final analysis received from the analysis module.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0x3b
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`analysis_module_uuid`	`Uuid128`
Field Description: The UUID of the analysis module that produced this analysis result. Must match the UUID given by `DeviceInfoReply` of the analysis module.	
`result_timestamp`	UtcTimePoint
Field Description: Timestamp of when the analysis results were generated	
`result_uuid`	`Uuid128`
Field Description: UUID generated by the analysis module for this result. This must be generated in such a way that it uniquely identifies this result.	
`item_number`	`uint32_t`
Field Description: The item of interest number these results are for; must match `DataOutPacket::item_number` of the data this result is for.	
`analysis_results_status`	`uint16_t` bits defined by `AnalysisFinalResultStatusFlags`
Field Description: Bitwise OR of flags including the large picture final result of the radiation data analysis, i.e., whether to refer the vehicle for further inspection. (See `DataOutFinalAnalysisPush::alarm_type` field for additional alarm details, plus results from occupancy sensors).	
`alarm_type`	`uint16_t` bits defined by `AlarmTypeFlags`
Field Description: Bitwise OR of `AlarmTypeFlags` indicating an alarm has been triggered based on radiation data or vehicle presence data. If this is zero, then no alarm has been found. (See `DataOutFinalAnalysisPush::analysis_results_status` for referral recommendation and further alarm detail.)	
`analysis_confidence_value`	`float`
Field Description: Indication of confidence, as a percent ranging from 0.0 to 100.0 (specified with a filed value ranging from 0.0 to 1.0), in the overall accuracy of the analysis, where increasing values indicate higher confidence. If a confidence is not ascribed, this should be set to a negative value (other than negative infinity or negative 0), with the magnitude having not carrying any meaning. (Taken roughly from N42 2012 standard for <AnalysisConfidenceValue >)	
`nuclides`	`Short Array` of `NuclideResult`
Field Description: The analysis algorithms identified results; does not strictly have to be nuclides (e.x., U235, Co60, etc.), but may be other sources as well (e.x., Bremsstrahlung, Annihilation, Gross Count, etc.). The same nuclide may be specified multiple times covering different ranges of sample numbers.	
`analysis_result_description`	`Short String`

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: Free-form text describing the overall conclusion of the analysis regarding the source of concern. (See N42.42 2012 standard for <AnalysisResultDescription >)	
`n42_xml`	`Large String`
Field Description: This string provides the UTF-8 encoded contents of a ANSI N42.42-2012 file, including analysis results. The N42.42 contents must include the data used to make the determination of results.	

4.6.76 DataOutFinalAnalysisPushAck Message

A message acknowledging the receipt of a DataOutFinalAnalysisPush message sent by the control module.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x02
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xbb
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

4.7 Command Interface

4.7.1 CommandMsgType Enumeration

Values to indicate the type of message being sent as part of the command message group. Command requests originate at a device and are directed to the control module. The control module replies directly to the requesting device. The first byte of command messages shall have a value of `MessageGroup::CommandValue`, and the second byte of the message will have a value as indicated by this enum, which will then tell you how to decode the message. For example a second byte value of 0X41 will tell you the message contents are specified by `CmdSystemStateChangeRequest`.

Underlying integral representation: `uint8_t`

Enumerated values for CommandMsgType:

CmdSystemStateChangeRequestValue Value 0x41

Message from a command device to the control module requesting a change to the system's operational state.

See also: CmdSystemStateChangeRequest

CmdSystemStateChangeReplyValue Value 0xc1

Reply sent to the command device that issued a CmdSystemStateChangeRequest message advising the device of the system state to be implemented by the control module and the status of the state change, as of the time of this reply.

See also: CmdSystemStateChangeReply

CmdMeasurementTypeChangeRequestValue Value 0x42

Message from a MessageGroup::CommandValue device to the control module informing it, from external information, about what is currently being measured (item, background, etc.).

See also: CmdMeasurementTypeChangeRequest , MeasurementTypeChangeRequest

CmdMeasurementTypeChangeReplyValue Value 0xc2

Reply sent in response to a CmdMeasurementTypeChangeRequest message.

See also: CmdMeasurementTypeChangeReply

CmdDeviceRelayRequestValue Value 0x43

Message from a command device to the control module requesting to forward an enclosed request message to a specific target device (of any type). Note that this can also be used to send the control module commands like changing a parameter value. Only request and responding reply messages may be relayed.

See also: CmdDeviceRelayRequest

CmdDeviceRelayReplyValue Value 0xc3

The control module's reply to a CmdDeviceRelayRequest message, giving the status of the request, or a reply from the device that was the target of the request, or both. Multiple replies may be sent. The requesting device determines whether the message transaction is complete by examining the status field within the target device's relayed reply.

See also: CmdDeviceRelayReplyValue

CmdDeviceSetReferenceInfoRequestValue Value 0x44

Message from a command device to the control module, informing it of the physical position of a device or subdetector, relative to center of portal, road level. It also specifies the device or subdetectors name, and free-form comments.

See also: CmdDeviceSetReferenceInfoRequestValue

CmdDeviceSetReferenceInfoReplyValue Value 0xc4

Message sent in response to a CmdDeviceSetReferenceInfoRequest message, giving the success status of setting the requested reference information.

CmdDeviceEventAcknowledgmentRequestValue Value 0x45

Message sent to acknowledge any event specified by AcknowledgeableEventType.

CmdDeviceEventAcknowledgmentReplyValue Value 0xc5

Message sent in response to a CommandMsgType::CmdDeviceEventAcknowledgmentReque message.

4.7.2 CmdSystemStateChangeRequest Message

A request from a command device to the control module to change the operating state of the system. The measurement type, operating mode, and data collection interval are applied uniformly to the entire system by this request, whereas data collection modes may be separately applied to gamma, neutron, and vehicle presence modules. The control module determines the OperatingMode and DataCollectionModes for DataOut, Command, Analysis, and Power Management devices; except for the analysis module, these devices will typically be operating whenever possible, even if radiation data is not being collected. Requests to individual modules are governed by the same limitations and requirements as the ChangeDeviceStateRequest message. This command is useful for getting a system with multiple detectors into a consistent operating state. If you thereafter want heterogenous operation (for example two gamma detectors operating in different modes) use CmdDeviceRelayRequest to make the changes (but there is no guarantee that the control module will support heterogeneous operation). The most recently received state change request overrules any previously received state change request. Although there is a CmdSystemStateChangeReply to this message, monitoring for DataOutSystemStateChangePush messages may be a more convenient way of detecting the actual state change.

See Also: ChangeDeviceStateRequest , CmdSystemStateChangeReply , CommandMsgType::CmdSystemStateChangeRequestValue

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x01
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x41
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

Continued on next page

Table continued from previous page

Field Name	Data Type
operating_mode	uint8_t enumerated by OperatingMode
Field Description: The operating mode (operating, ready, or standby) to bring the system to. If the operating mode is OperatingMode::StandByValue or OperatingMode::ReadyValue (i.e., not OperatingMode::OperatingValue), then the collection modes must be DataCollectionModes::NoOriginateValue, the measurement type must be MeasurementType::NotSpecifiedValue, and the collection interval must be zero.	
gamma_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: The requested data collection mode for all modules that detect gamma radiation, even if the same module also detects neutrons	
neutron_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: The requested data collection mode for all modules that only detect neutrons.	
vehicle_presence_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description:	
measurement_type	uint8_t enumerated by MeasurementType
Field Description: The requested type of measurement to perform, or being performed, by the system (item of interest, background, not specified, possible interfering source, or active maintenance). A value of MeasurementType::NotSpecifiedValue indicates no change is being requested (e.g., keep using value determined via the vehicle presence sensors or otherwise).	
collection_interval_ms	uint32_t
Field Description: Requested collection interval of the system when being put into the operating state. For OperatingMode::ReadyValue or OperatingMode::StandByValue operating mode, this value must be zero.	
requested_transition_time	UtcTimePoint
Field Description: See ChangeDeviceStateRequest::requested_transition_time for full semantics of this variable, but to summarize: if zero, or some time in the past, start transition now. If some time in the future, aim to have the transition complete as close as possible to that time. This transition time is managed by the individual devices in the system, not the control module (to allow individual devices to account for their inherent delays in state transitions, and thus get more accurate results). If you request a system state change some time X in the future but then send an individual device a state change request before X, then that device will no longer participate in the original state change request.	

4.7.3 CmdSystemStateChangeReply Message

Reply from the control module to the requesting command device, advising the command device of the changes to be made in response to the device's CmdSystemStateChangeRequest for a change in the system's state. Multiple of these messages may need to be sent for

one transition. The CmdSystemStateChangeReply::status_of_transition field of this reply message provides the current status of the transition, including if it is finished or if there was an error. If there was an error, in addition to the CmdSystemStateChangeReply::status_of_transition field being set to the appropriate SystemStateChangeStatus value, a NotificationPush message referencing the same message id may also be sent to provide additional information. To monitor the status of individual devices or the status of the transitions, watch for DataOutSystemStateChangePush messages.

See Also: DataOutSystemStateChangePush

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x01
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xc1
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
requested_operating_mode	uint8_t enumerated by OperatingMode
Field Description: Reply from the control module to the requesting command device, advising the command device of the operating mode (operating, ready, or standby) to which the system will transition in response to the request.	
requested_gamma_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: Reply from the control module to the requesting command device, indicating the changes to be made in response to the request for gamma data collection mode. All gamma detectors in the system must support the specified mode. If there are no gamma detectors in the system, this value will be ignored. For OperatingMode::ReadyValue or OperatingMode::StandByValue states, this value must be set to DataCollectionModes::NoOriginateValue. For OperatingMode::OperatingValue, this field gives the data collection mode of the radiation detection modules that detect gamma radiation. However, if this value is set to DataCollectionModes::NoOriginateValue, then detectors that detect gamma radiation will be transitioned from OperatingMode::OperatingValue into state of OperatingMode::ReadyValue. If a dwell mode is specified for gamma or neutron, but not both, then the system state should last for the dwell measurement's duration, and the control module should take care of transitioning the other class of devices back to a OperatingMode::ReadyValue state when the dwell finishes.	

Continued on next page

Table continued from previous page

Field Name	Data Type
requested_neutron_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: Reply from the control module to the requesting command device, advising the command device of the changes to be made in response to the request for neutron data collection mode. All neutron detectors in the system must support the specified mode. If there are no neutron detectors in the system, this value will be ignored. For OperatingMode::ReadyValue or OperatingMode::StandByValue states, this value must be set to DataCollectionModes::NoOriginateValue. For OperatingMode::OperatingValue, this field gives the data collection mode of the radiation detection modules that detect neutron radiation. However, if this value is set to DataCollectionModes::NoOriginateValue, then detectors that detect neutron radiation will be transitioned from OperatingMode::OperatingValue into state of OperatingMode::ReadyValue. If a dwell mode is specified for gamma or neutron, but not both, then the system state should last for the dwell measurement's duration, and the control module should take care of transitioning the other class of devices back to a OperatingMode::ReadyValue state when the dwell finishes.	
requested_vehicle_presence_data_collection_mode	uint8_t enumerated by DataCollectionModes
Field Description: Reply from the control module to the requesting command device, advising the command device of the change in vehicle presence data collection mode to be made in response to the device's CmdSystemStateChangeRequest. If there are no vehicle presence detectors in the system, this value will be ignored.	
requested_data_collection_interval_ms	uint32_t
Field Description: The control module's reply for the collection interval when the system is being put into the operating state. For OperatingMode::ReadyValue or OperatingMode::StandByValue operating mode, this value must be zero.	
requested_measurement_type	uint8_t enumerated by MeasurementType
Field Description: Reply from the control module to the requesting command device, advising the command device of the changes to be made in response to the request for a change in the system's type of measurement to perform or being performed (item of interest, background, not specified, possible interfering source, or active maintenance).	
status_of_transition	uint8_t enumerated by SystemStateChangeStatus
Field Description: Reply from the control module to the requesting command device, advising the command device of the status of the transition requested by the device's CmdSystemStateChangeRequest.	

4.7.4 CmdMeasurementTypeChangeRequest Message

A request from a command device to the control module to change the measurement type (item of interest, background, not specified, possible interfering source, or active maintenance) of the system, in approximately real time.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x01 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x42 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
effective_timestamp	UtcTimePoint *Field Description*: Timestamp for when this state became effective; must be current time or before, and may not be equal to or before any previous time specified by a CmdMeasurementTypeChangeRequest. If you are unable to specify an effective timestamp (ex you dont know when the background period began) then a value of 0 may be indicated to mean taking effect 'now'. The primary purpose of this timestamp is to allow devices to compensate for latencies in receiving this message, but as this message is intended to be closer to a 'in real time' type message, there is no requirements modules have to compensate for these latencies. See also: • MeasurementTypeChangeRequest::effective_timestamp
measurement_type	uint8_t enumerated by MeasurementType *Field Description*: Request by a command device to change the type of measurement to perform or being performed (item of interest, background, not specified, possible interfering source, or active maintenance). For values of MeasurementType::ItemValue, MeasurementType::BackgroundValue, MeasurementType::PossibleInterferingSourceValue, and MeasurementType::ActiveMaintenanceValue, that measurement type should be assumed until the MeasurementType is changed by a CmdMeasurementTypeChangeRequest or CmdSystemStateChangeRequest message, or it becomes infeasible to continue. A value of MeasurementType::NotSpecifiedValue indicates that the control module should resume determining the MeasurementType to use.

4.7.5 CmdMeasurementTypeChangeReply Message

A reply by the control module to a CmdMeasurementTypeChangeRequest message sent by a command device.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x01 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0xc2 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.

4.7.6 CmdDeviceRelayRequest Message

Message sent from a command device to the control module, to relay a specific request message to a specific target device, or the control module itself. This message specifies the target device's UUID as well as the request message to send to it. Note that the control module will check that the message is of the right type to send to the target device, and it is an appropriate time to send it to the device. The control module will also possibly re-encode the message to a version the target device is capable of understanding. This message is also how you can set parameters on the control module, request log files, or similar.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x01 *Field Description*: The RAPTER MessageGroup this message corresponds too.
message_type	uint8_t with value 0x43 *Field Description*: The type of message within the specified message_group, that this message corresponds to.
message_group_version	uint8_t *Field Description*: The version of the message_group being used.
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
device_uuid	Uuid128 *Field Description*: The UUID of the target device (i.e., the device that will receive the request and issue a reply).
request_message	Large Array of char

Continued on next page

Table continued from previous page

Field Name	Data Type
Field Description: The encoded request message to send to the device. The message should contain the standard 8-byte RAPTER header, but bytes 4 through 8, the message ID, will be ignored, so does not have to be unique. The version of the request message should be the version used by the requesting device. Note that the control module will determine if the message should be sent to the target device, assign a message ID, and translate the message into the version used by the target device (if necessary). Note: only request messages can be relayed from the control module to the targeted device. Also, the control module can choose to not allow some request messages through, like `SupportedMessageGroupVersionsRequest` message. (See `CmdDeviceRelayReplyStatus::CmdDeviceRelayReplyRequestRejectedValue`)	

4.7.7 CmdDeviceRelayReplyStatus Enumeration

Enumeration to indicate the status of a `CmdDeviceRelayReply` message. For request messages that are capable of more than one reply, to determine if the message exchange is completed, you have to inspect the contents (i.e., a status field) of the relayed message.

Used in message: `CmdDeviceRelayReply`

Underlying integral representation: `uint8_t`

Enumerated values for `CmdDeviceRelayReplyStatus`:

CmdDeviceRelayReplyFromDeviceValue Value 0x00

Indicates that `CmdDeviceRelayReply::reply_message` is the requested reply message from the target device, that the control module is relaying back to the requesting command device.

CmdDeviceRelayReplyMessageTypeInvalidValue Value 0x01

Indicates the `CmdDeviceRelayRequest` contained a message type that is not allowed for passing through to the target device. For example, if the message group does not match the device, or the contained message is a reply, push, or ack (only requests can go through).

CmdDeviceRelayReplyInvalidUUIDValue Value 0x02

Indicates the `CmdDeviceRelayRequest` specified an invalid UUID for the target device.

CmdDeviceRelayReplyUnableToDecodeRequestValue Value 0x03

Indicates `CmdDeviceRelayRequest::request_message` was unable to be decoded by the control module.

CmdDeviceRelayReplyUnableToDecodeReplyValue Value 0x04

Indicates that the reply returned from the target device could not be decoded; message content still attached.

CmdDeviceRelayReplyDeviceNotRespondingValue Value 0x05

Indicates communications with the requested device is not responding but still has an established WebSocket connection; the control module could not send the command to the device.

CmdDeviceRelayReplyDeviceNotConnectedValue Value 0x06

The device specified is not currently connected to the control module.

CmdDeviceRelayReplyUnableToTranslateRequestValue Value 0x07

Indicates the message could not be translated from the provided `CmdDeviceRelayRequest::requ` to a format the target device could understand. This could happen if the Command message group is a newer version that contains a feature not included in the message group version used by the target device.

CmdDeviceRelayReplyRequestRejectedValue Value 0x08

Indicates the control module opted to not pass the `CmdDeviceRelayRequest` through to the target device for some reason. This might happen, for example, if there is currently an occupancy, and an active calibration is requested (whether it happens or not is implementation defined), or your request tries to a control module decision, like version of the message group to use.

4.7.8 CmdDeviceRelayReply Message

The reply by the control module to a `CmdDeviceRelayRequest` message sent by a command device. There may be multiple replies to the original message, since the device itself may reply multiple times. Also, if the command does something the data out devices should be updated about (ex. parameter update, or energy calibration update), then those respective updates will be sent to all data out devices, including the one that issued the `CmdDeviceRelayRequest`.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x01
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xc3
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

Continued on next page

Table continued from previous page

Field Name	Data Type
device_uuid	Uuid128
Field Description: The UUID of device the reply is coming from (i.e., the device that was the target of the relay request).	
reply_status	uint8_t enumerated by CmdDeviceRelayReplyStatus
Field Description: The status of this relayed reply. Use this to determine if there is an error or issue with the request or reply. Note that this indicates the status of receiving the reply from the target device, not the status the message returned by the target device may indicate (e.g., this status may indicate successful, but the actual response from the target device may indicate failure).	
reply_message	Large Array of char
Field Description: The reply message from the target device, if CmdDeviceRelayReply::reply_status is equal to CmdDeviceRelayReplyStatus::CmdDeviceRelayReplyFromDeviceValue, otherwise must have length zero. Note that the first 8 bytes are as normal, but the message ID should not be assumed to be meaningful (e.g. might be zero, might be equal to the CmdDeviceRelayReply::message_id of this message, or completely random). Also, the message may have been re-encoded by the control module to deal with any message group versioning issues.	

4.7.9 CmdDeviceSetReferenceInfoRequest Message

A request from a command device to the control module to set manually entered reference information (position, name, comment) of a device or subdetector within the device. To set the information for the device, specify a subdetector number of zero. To set the information for a subdetector of the device, specify the appropriate (non-zero) subdetector number. For a given device, a combination of, all, or none of the subdetectors and/or device reference information may be set. The name specified is useful for referencing to detectors in N42 files, databases, and log files. Some analysis algorithms may need subdetector position to perform calculations. Note that all reference information is set, so if you want to only change one field you will first need to retrieve the current reference information and copy the other fields over to the information you send the control module. Portal physical configuration, coordinate system, and position settings of devices are discussed in RAPTER Interface Specification, Chapter 2.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x01
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x44
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	

Continued on next page

Table continued from previous page

Field Name	Data Type
message_flags	uint8_t bits defined by MsgFlags *Field Description*: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.
message_id	uint32_t *Field Description*: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.
device_uuid	Uuid128 *Field Description*: The UUID of device (or module) for which the position is being requested to be set.
subdetector_number	uint8_t *Field Description*: The subdetector this position is for. For a device that does not have subdetectors, then this value must be zero.
name	Short String *Field Description*: Name for this device, or sub-device that should be used when storing information in N42 files, databases, or log files. For gamma and neutron sub-detectors, examples are: 'Aa1', 'Ca1N', 'Da2'. For RSPs (e.g., subdetector zero), examples are: 'Panel1', 'RSP2'. For vehicle presence sensors, examples are: 'BreakBeam1', 'EntranceCamera'. Note that if no name is entered, it is unspecified what name should be used in N42 files or databases, but some possibilities are the UUID of the device, a guessed name based on location, or a randomly assigned name.
set_fields	uint8_t bits defined by DeviceReferenceInfoSetFlags *Field Description*: Flags to indicate if ambiguous quantities have been set.
x_cm	float *Field Description*: The x-location; see CmdDeviceSetReferenceInfoRequest for definition.
y_cm	float *Field Description*: The y-location; see CmdDeviceSetReferenceInfoRequest for definition.
z_cm	float *Field Description*: The z-location; see CmdDeviceSetReferenceInfoRequest for definition.
position_description	Short String *Field Description*: A free-form description of the location. Examples might be: 'Lower passenger side RSP', or 'Lower detection element'.
comment	Short String *Field Description*: A free-form text comment about this module or subdetector.

4.7.10 SetReferenceInfoStatus Enumeration

Enumeration to give the success status of a request to set the position of a device.

Used in message: CmdDeviceSetReferenceInfoReply

Underlying integral representation: uint8_t

Enumerated values for `SetReferenceInfoStatus`:

ReferenceInfoSetSuccessfullyValue Value 0x00

Everything is good, position was set in the control module.

InvalidUuidValue Value 0x01

The UUID does not match any connected device. The request to set position failed.

InvalidSubDeviceNumberValue Value 0x02

An invalid sub device was specified. The request to set position failed.

OtherErrorSettingReferenceInfoValue Value 0x03

Some other issue. You should also attach a notification to the reply.

4.7.11 CmdDeviceSetReferenceInfoReply Message

Reply by the control module to a `CmdDeviceSetReferenceInfoRequest` sent by a command device. This reply gives the status of setting the device's position.

Message Contents:

Field Name	Data Type
`message_group`	`uint8_t` with value 0x01
Field Description: The RAPTER `MessageGroup` this message corresponds too.	
`message_type`	`uint8_t` with value 0xc4
Field Description: The type of message within the specified message_group, that this message corresponds to.	
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
`status`	`uint8_t` enumerated by `SetReferenceInfoStatus`
Field Description: The completion status of implementing the requested position setting.	

4.7.12 CmdDeviceEventAcknowledgmentRequest Message

Message used to request an acknowledgement of a condition that can be acknowledged. Currently, the only condition to be acknowledged is alarming final analysis results. The acknowledgement is to be made by a `CmdDeviceEventAcknowledgmentReply` message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x01
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0x45
Field Description: The type of message within the specified message_group, that this message corresponds to.	
message_group_version	uint8_t
Field Description: The version of the message_group being used.	
message_flags	uint8_t bits defined by MsgFlags
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
message_id	uint32_t
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	
acknowledgment_timestamp	UtcTimePoint
Field Description: The time at which this acknowledgement was generated.	
event_type	uint32_t enumerated by AcknowledgeableEventType
Field Description: The type of event to be acknowledged.	
event_uuid	Uuid128
Field Description: The UUID of the event being acknowledged. See also: • AnalysisItemFinalResultData::result_uuid • InterimAnalysisData::result_uuid	
acknowledgment_type	uint32_t enumerated by EventAcknowledgmentType
Field Description: The type of acknowledgement this is.	
message	Short String
Field Description: Optional free form UTF-8 message associated with this acknowledgement. This can be used to, for example, indicate the user logged into the system that did the acknowledgement, or to include other information into the event record. See also: • DataOutEventAcknowledgementPush::message_from_acknowledger	

4.7.13 CmdDeviceEventAcknowledgmentReply Message

A reply to a CmdDeviceEventAcknowledgmentRequest message.

Message Contents:

Field Name	Data Type
message_group	uint8_t with value 0x01
Field Description: The RAPTER MessageGroup this message corresponds too.	
message_type	uint8_t with value 0xc5
Field Description: The type of message within the specified message_group, that this message corresponds to.	

Continued on next page

Table continued from previous page

Field Name	Data Type
`message_group_version`	`uint8_t`
Field Description: The version of the message_group being used.	
`message_flags`	`uint8_t` bits defined by `MsgFlags`
Field Description: Flags indicating if this message has a notification attached, is a list of messages, or has previously been buffered.	
`message_id`	`uint32_t`
Field Description: The message ID that uniquely identifies the request/reply or push/awknowledge message exchange between the control module and a specific device.	

Appendices

Appendix A

General Message Encoding/Decoding Example

An example of encoding and decoding a RAPTER message to the binary format suitable for sending over the network, in the C language is given bellow. To encode the message, a `WRITE_FIELD` macro is defined and used to encode the message field-by-field; this method is is independent of how the `struct` is laid out in memory. A check is then performed to see if the `struct`'s memory layout is compatible with the RAPTER message structure, and if it is, a briefer method of decoding the message is used (a similar brief method of encoding could have been used as well). This code assumes a little-endian based computer architecture, with IEEE-754 compatible floating point representation, and only a limited amount of crude error checking is performed.

```
1
2  #include <stddef.h>    //offsetof
3  #include <stdint.h>    //sized ints (uin8_t, int64_t, etc)
4  #include <string.h>    //memcpy
5  #include <stdio.h>     //printf
6  #include <sys/time.h>  //gettimeofday (UNIX)
7  #include <assert.h>    //assert
8  #include <stdlib.h>    //malloc / free
9
10 // Compiler specific (GCC, Clang, others) test for a little-↩
     endian machine
11 #if __BYTE_ORDER__ != __ORDER_LITTLE_ENDIAN__
12 error( "Requires little endian; alter code to swap byte order↩
     for integral types" );
13 #endif
14
15 #ifndef __STDC_IEC_559__
16 #pragma message( "May not be using IEEE-754 floats (but could↩
     be, or close enough)" )
17 #endif
```

```
18
19 // Example of getting timestamp in (UNIX specific)
20 int64_t current_timestamp()
21 {
22    struct timeval abs_time;
23    gettimeofday( &abs_time, NULL );
24    return ((int64_t)abs_time.tv_sec) * 1000000 + abs_time.tv_usec;
25 }
26
27 //Represent the SendLogsReply message contents as a C-struct. Note order of
28 //  fields, and integral widths are consistent with RAPTER specification
29 struct SendLogsReplyMsg
30 {
31   uint8_t message_group;
32   uint8_t message_type;
33   uint8_t message_group_version;
34   uint8_t message_flags;
35   uint32_t message_id;
36
37   uint8_t reply_status;
38   int64_t start_time;
39   int64_t end_time;
40   uint32_t log_data_length;
41   const char *log_data;
42 };
43
44 //Define a macro to write integer fields into the buffer
45 #define WRITE_FIELD(buf,buflen,field) do{\
46   /* Get number of bytes to be written to buffer */ \
47   const size_t width = sizeof(field); \
48   /* Make sure to start writing the field at a multiple of its size */ \
49   const size_t num_pad_bytes = (buflen % width) ? width - (buflen % width) : 0; \
50   /* (optional) write zeros to any padding bytes we need to add in */ \
51   memset( buf + buflen, 0, num_pad_bytes ); \
52   buflen += num_pad_bytes; \
53   /* Make sure we dont write past end of buffer. */ \
54   assert( buflen+width < sizeof(buf)); \
55   /* Copy value into the buffer */ \
56   memcpy( buf + buflen, &field, width ); \
57   /* Increment the length of the buffer by how much we wrote */ \
58   buflen += width; \
```

```
59 }while(0)
60
61
62 int main()
63 {
64   uint8_t buffer[1024];  // what we will be sending out over ↩
       the network
65   uint32_t buffer_len = 0;
66   struct SendLogsReplyMsg reply;
67
68   reply.message_group = 0x00;       //Message belongs to Core ↩
       message group
69   reply.message_type = 0x8C;        //Message is a ↩
       SendLogsReply type
70   reply.message_group_version = 0; //Version of Core message ↩
       group being used
71   reply.message_id = 32521;         //randomly chosen message ↩
       id
72   reply.reply_status = 0;           //Indicate ↩
       LogRetrievedSuccessValue
73
74   reply.start_time = current_timestamp(); //example value ↩
       only
75   reply.end_time = current_timestamp();   //example value ↩
       only
76
77   reply.log_data = "An example log message";
78   reply.log_data_length = strlen( reply.log_data );
79
80   buffer_len = 0;
81   //Write message header to buffer[]
82   WRITE_FIELD( buffer, buffer_len, reply.message_group );
83   WRITE_FIELD( buffer, buffer_len, reply.message_type );
84   WRITE_FIELD( buffer, buffer_len, reply.↩
       message_group_version );
85   WRITE_FIELD( buffer, buffer_len, reply.message_flags );
86   WRITE_FIELD( buffer, buffer_len, reply.message_id );
87
88   //Write message contents to buffer[]
89   WRITE_FIELD( buffer, buffer_len, reply.reply_status );
90   WRITE_FIELD( buffer, buffer_len, reply.start_time );
91   WRITE_FIELD( buffer, buffer_len, reply.end_time );
92   WRITE_FIELD( buffer, buffer_len, reply.log_data_length );
93
94   //Perform a memcpy of string contents into buffer, starting↩
       immediatly
95   //  after the strings length.
96   memcpy( buffer + buffer_len, reply.log_data, reply.↩
```

```
    log_data_length );
97  buffer_len += reply.log_data_length;
98
99  //buffer[], with length of buffer_len bytes is ready to be sent
100
101 //Lets also check if devices memory layout happens to match RAPTER message
102 //  format, and if so decod the message in a simpler way.
103 if( offsetof(struct SendLogsReplyMsg, message_group) == 0
104     && offsetof(struct SendLogsReplyMsg, message_type) == 1
105     && offsetof(struct SendLogsReplyMsg, message_group_version) == 2
106     && offsetof(struct SendLogsReplyMsg, message_flags) == 3
107     && offsetof(struct SendLogsReplyMsg, message_id) == 4
108     && offsetof(struct SendLogsReplyMsg, reply_status) == 8
109     //reply_status was one byte, so add in 7 bytes of padding
110     && offsetof(struct SendLogsReplyMsg, start_time) == 16
111     && offsetof(struct SendLogsReplyMsg, end_time) == 24
112     && offsetof(struct SendLogsReplyMsg, log_data_length) == 32 )
113 {
114   printf( "Memory layout allows simpler message encoding/decoding.\n" );
115
116   //Get length up through, and including log_data_length
117   size_t decode_len = ((void*)&reply.log_data_length - (void*)&reply)
118                       + sizeof(reply.log_data_length);
119
120   struct SendLogsReplyMsg decoded;
121   memcpy( &decoded, &buffer, decode_len );
122
123   //A quick sanity check
124   assert( (decode_len + decoded.log_data_length) == buffer_len );
125
126   //Copy string data. Note undefined behavior if log_data_length == 0
127   if( decoded.log_data_length )
128     decoded.log_data = (const char *)malloc( decoded.log_data_length );
129   memcpy( (void *)decoded.log_data, buffer + decode_len, decoded.log_data_length );
130
131   //Lets check that everything made the round trip okay
```

```
132     assert( decoded.message_group == reply.message_group );
133     assert( decoded.message_type == reply.message_type );
134     assert( decoded.message_group_version == reply.↩
          message_group_version );
135     assert( decoded.message_flags == reply.message_flags );
136     assert( decoded.message_id == reply.message_id );
137     assert( decoded.reply_status == reply.reply_status );
138     assert( decoded.start_time == reply.start_time );
139     assert( decoded.end_time == reply.end_time );
140     assert( decoded.log_data_length == reply.log_data_length ↩
          );
141     assert( !memcmp(reply.log_data, decoded.log_data, reply.↩
          log_data_length) );
142
143     //If message was from unknown source would also check ↩
          that enumerated fields
144     //  have valid values, times are valid, and strings are ↩
          UTF-8
145
146     free( (void *)decoded.log_data );  //prevent memory leak
147   }
148
149   printf( "Message contents: [" );
150   for( size_t i = 0; i < buffer_len; ++i )
151     printf( "%s%#x", (i?", ":""), (unsigned int)buffer[i] );
152   printf( "]\n" );
153
154   return 1;
155 }
```

Appendix B

Parameter Encoding and Decoding

A full description of “parameters” can be found in Section 2.1.1. Encoding and decoding of parameter messages follow the rules specified in Section 3.3.3, however the fact that the parameters can be boolean, integer, floating point, or strings introduces the complication that the messages are interpreted differently based on the type of information. Each of the relevant messages specify the data type being relayed, so the messages can be interpreted without external reference (e.g., knowing that a parameter of a given name is a given type). Booleans and integers are represented as signed 32-bit integers; for booleans any value besides zero is interpreted as a true value. Floats are represented as a little-endian, four-byte, IEEE 754-2008 floats [18] (Infs, NaNs, and denormal values prohibited), and strings are represented as a four byte unsigned integer that specifies the string byte length, followed by that many bytes of character data. The values are encoded/decoded using the same rules as in Section 3.3.3. An example C language encoding and decoding of the ParameterInfoReply message (the largest of all the parameter related messages) is given bellow.

```

#include <math.h>      //fabs
#include <stdint.h>    //sized ints (uin8_t, int64_t, etc)
#include <string.h>    //memcpy
#include <stdio.h>     //printf
#include <errno.h>     //for errno
#undef NDEBUG          //Will use assert for error checking
#include <assert.h>    //assert
#include <stdlib.h>    //malloc / free
#include <inttypes.h>  //PRId64

// Compiler specific (GCC, Clang, others) test for a little-endian machine
#if __BYTE_ORDER__ != __ORDER_LITTLE_ENDIAN__
#pragma error( "Requires little endian; alter code to swap" \
               " byte order for integral types" )
#endif

#if !defined(__STDC_IEC_559__) && !defined(__GCC_IEC_559)
#pragma message( "May not be using IEEE-754 floats" \
                 " (but could be, or close enough)" )
#endif

//Extract some convenient enums from RAPTER Interface
//  Specification
enum ParameterReplyValidity
{
  ValidValue       = 0,
  InvalidNameValue = 1
};

enum ParameterValueDataType
{
  BooleanValue           = 0x00,
  IntegerValue           = 0x01,
  FloatValue             = 0x02,
  Utf8StringValue        = 0x03,
  FloatingPointListValue = 0x04,
```

```
  IntegerListValue       = 0x05
};

enum HealthSeverityLevel
{
  NotApplicableValue                          = 0x0,
  NoProblemValue                              = 0x1,
  WarningNotAffectingDeviceOperationValue     = 0x2,
  WarningAffectingDeviceOperationValue        = 0x3,
  FatalEffectOnDeviceOperationValue           = 0x4,
  IntentionalPreventionOfDeviceOperationValue = 0x5,
  NotAbleToMeasureValue                       = 0x6
};

enum ParameterPropertiesFlags
{
  ValueFixedFlag                      = 0x1,
  SettableFlag                        = 0x2,
  MeasuredFlag                        = 0x4,
  HealthyValueRangedFlag              = 0x8,
  SetValueRangedFlag                  = 0x10,
  ReportOnChangeFlag                  = 0x20,
  ReportOnChangeDeltaSettableFlag     = 0x40,
  HealthyLimitsSettableFlag           = 0x80,
  AffectsHealthWithoutLimitsFlag      = 0x100,
  NotSettableWhileOperatingFlag       = 0x200,
  SettableWithPredefinedValuesOnlyFlag = 0x400
};

/** Represent a general ParameterInfoReply as a C-struct,
 * similar to how you might do on the control module.
 *
 * For illustration only, and not a safe or efficient
 * representation.
*/
struct ParameterInfoReply
{
  uint8_t message_group;
  uint8_t message_type;
  uint8_t message_group_version;
  uint8_t message_flags;
  uint32_t message_id;

  uint8_t reply_status;
  uint8_t value_type;
  uint8_t parameter_health_impact;
  uint8_t subdetector_number;

  uint32_t value_properties;
  int64_t set_timestamp;
  int64_t effective_timestamp;

  char *parameter_name;               // ShortString

  char *set_value;                    // ShortString
  char *measured_value;               // ShortString
  char *lower_healthy_measured_value; // ShortString
  char *upper_healthy_measured_value; // ShortString
  char *lower_settable_value;         // ShortString
  char *upper_settable_value;         // ShortString
  char *reportable_change_delta;      // ShortString

  char *parameter_description;        // MediumString
};


/** Example of representing a high voltage parameter
 * like you might do on a gamma detector.  E.g., dont
 * store float values as strings, or information that
 * wont change, but convert floats to string, and add
 * non-changing information during serialization.
*/
struct HighVoltagParameter
{
  uint8_t subdetector_number;
  int64_t set_timestamp;
  int64_t effective_timestamp;

  double set_value;
  double measured_value;
};


/** Helper function to compare strings that both may be null. */
void check_str_equal( const char * const lhs,
                      const char * const rhs )
{
  assert( (!lhs && !rhs) || strcmp(lhs,rhs)==0 );
}


/** Helper function to safe-free strings. */
void checked_free( char **str )
{
  if( *str )
    free( *str );
  *str = NULL;
}

```

```
/** Helper function to free strings in a ParameterInfoReply */
void free_string_fields( struct ParameterInfoReply *msg )
{
  checked_free( &msg->set_value );
  checked_free( &msg->measured_value );
  checked_free( &msg->lower_healthy_measured_value );
  checked_free( &msg->upper_healthy_measured_value );
  checked_free( &msg->lower_settable_value );
  checked_free( &msg->upper_settable_value );
  checked_free( &msg->reportable_change_delta );
  checked_free( &msg->parameter_name );
  checked_free( &msg->parameter_description );
};

/** Define a macro to write int and float fields into the buffer */
#define WRITE_NUMERIC_FIELD(buf,bufpos,max_buff_len,field) do {  \
  /* Get number of bytes to be written to buffer */             \
  const size_t w = sizeof(field);                               \
  /* Start writing the field at a multiple of its size */       \
  const size_t npad_bytes = (bufpos % w) ? w - (bufpos % w) : 0; \
  /* Write zeros to any padding bytes we need to add in */      \
  memset( buf + bufpos, 0, npad_bytes );                        \
  bufpos += npad_bytes;                                         \
  /* Make sure we dont write past end of buffer. */             \
  assert( bufpos+w < max_buff_len );                            \
  /* Copy value into the buffer */                              \
  memcpy( buf + bufpos, &field, w );                            \
  /* Increment the length of the buffer by how much we wrote */ \
  bufpos += w;                                                  \
} while(0)

/** Define a macro to write string fields into the buffer */
#define WRITE_STRING_FIELD(buf,pos,max_buff_len,field) do {      \
  /* Strings start at a multiple of 4 from message begining */   \
  const size_t npad_start = (pos % 4) ? 4 - (pos % 4) : 0;      \
  const uint32_t str_len = (field ? strlen(field) : (size_t)0);  \
  /* Strings end with >=1 zero byte, and length multiple of 4*/  \
  const size_t end_pos = pos + npad_start + 4  + str_len;       \
  const size_t npad_end = (end_pos % 4) ? 4 - (end_pos % 4) : 4; \
  assert( end_pos + npad_end <= max_buff_len );                 \
  pos += npad_start; /* padding byte values not specified */    \
  WRITE_NUMERIC_FIELD(buf,pos,max_buff_len,str_len);            \
  if(str_len) memcpy( buf + pos, field, str_len );              \
  pos += str_len;                                               \
  assert( pos == end_pos );                                     \
  /* Terminated with 1 to 4 zero-valued bytes. */               \
  memset( buf + pos, 0, npad_end );                             \
  pos += npad_end;                                              \
} while(0)


/** Encodes a ParameterInfoReply into a buffer.
 * @param msg Message to encode.
 * @param buffer Buffer to create the RAPTER message in
 * @param buff_size The maximum bytes that can be written to buffer
 * @returns The number of bytes written to the buffer.
*/
size_t encode_ParameterInfoReply( const struct ParameterInfoReply * const msg,
                                  uint8_t *buffer, const size_t buff_size )
{
  assert( buff_size >= 104 );  //minimum size of this struct

  size_t len = 0;
  //Belongs to Core Interface
  assert( msg->message_group == 0x00 );

  //Make sure a ParameterInfoReply
  assert( msg->message_type == 0x93 );

  //Only supporting version 0 of Core Interface
  assert( msg->message_group_version == 0x00 );

  // Attached Notifications, list, and buffering not
  //  supported in example code
  assert( msg->message_flags == 0x00 );

  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->message_group );
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->message_type );
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->message_group_version );
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->message_flags );
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->message_id );
  assert( msg->reply_status == ValidValue
          || msg->reply_status == InvalidNameValue );
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->reply_status );
  assert( msg->value_type == BooleanValue
          || msg->value_type == IntegerValue
          || msg->value_type == FloatValue
          || msg->value_type == Utf8StringValue
          || msg->value_type == FloatingPointListValue
          || msg->value_type == IntegerListValue );

  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->value_type );
  //assert( msg->parameter_health_impact is valid value );
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->parameter_health_impact );
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->subdetector_number );

  //assert( msg->value_properties only has valid combination of bits set )
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->value_properties );
  WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->set_timestamp );
```

```
237   WRITE_NUMERIC_FIELD( buffer, len, buff_size, msg->effective_timestamp );
238
239   WRITE_STRING_FIELD( buffer, len, buff_size, msg->parameter_name );
240   WRITE_STRING_FIELD( buffer, len, buff_size, msg->set_value );
241   WRITE_STRING_FIELD( buffer, len, buff_size, msg->measured_value );
242   WRITE_STRING_FIELD( buffer, len, buff_size, msg->lower_healthy_measured_value );
243   WRITE_STRING_FIELD( buffer, len, buff_size, msg->upper_healthy_measured_value );
244   WRITE_STRING_FIELD( buffer, len, buff_size, msg->lower_settable_value );
245   WRITE_STRING_FIELD( buffer, len, buff_size, msg->upper_settable_value );
246   WRITE_STRING_FIELD( buffer, len, buff_size, msg->reportable_change_delta );
247   WRITE_STRING_FIELD( buffer, len, buff_size, msg->parameter_description );
248
249   return len;
250 }
251
252
253 /** Encode a HighVoltagParameter as a RAPTER ParameterInfoReply message. */
254 size_t encode_HighVoltagParameter( const struct HighVoltagParameter * const par,
255                                    uint8_t *buffer, const size_t buff_size )
256 {
257   assert( buff_size >= 104 );  //minimum size of this struct
258
259   size_t len = 0;
260   buffer[len++] = 0x00;  //Belongs to Core Interface
261   buffer[len++] = 0x93;  //ParameterInfoReply msg type
262   buffer[len++] = 0x00;  //version 0 of Core Interface
263   buffer[len++] = 0x00;  //No notification, list, buffering
264   uint32_t message_id = 3333;
265   WRITE_NUMERIC_FIELD( buffer, len, buff_size, message_id );
266   buffer[len++] = ValidValue;
267   buffer[len++] = FloatValue;
268   buffer[len++] = NoProblemValue;
269   buffer[len++] = 1;  //subdector number
270
271   uint32_t value_properties = (SettableFlag
272                                | MeasuredFlag
273                                | HealthyValueRangedFlag
274                                | ReportOnChangeFlag
275                                | NotSettableWhileOperatingFlag);
276
277   WRITE_NUMERIC_FIELD( buffer, len, buff_size, value_properties );
278   WRITE_NUMERIC_FIELD( buffer, len, buff_size, par->set_timestamp );
279   WRITE_NUMERIC_FIELD( buffer, len, buff_size, par->effective_timestamp );
280
281   const char *par_name = "high_voltage";
282   WRITE_STRING_FIELD( buffer, len, buff_size, par_name );
283
284   char tmpbuf[64];
285   snprintf( tmpbuf, sizeof(tmpbuf), "%f", par->set_value );
286   WRITE_STRING_FIELD( buffer, len, buff_size, tmpbuf );
287
288   snprintf( tmpbuf, sizeof(tmpbuf), "%f", par->measured_value );
289   WRITE_STRING_FIELD( buffer, len, buff_size, tmpbuf );
290
291   WRITE_STRING_FIELD( buffer, len, buff_size, "100.0" );
292   WRITE_STRING_FIELD( buffer, len, buff_size, "2000.0" );
293   WRITE_STRING_FIELD( buffer, len, buff_size, NULL );
294   WRITE_STRING_FIELD( buffer, len, buff_size, NULL );
295   WRITE_STRING_FIELD( buffer, len, buff_size, "1.5" );
296
297   const char *par_desc = "A parameter that is settable, measured, has a"
298                          " define range of healthy values, changes are"
299                          " reported every 1.5 volts.";
300   WRITE_STRING_FIELD( buffer, len, buff_size, par_desc );
301
302   return len;
303 }
304
305
306 /** Define macro to read int and float fields from the buffer */
307 #define READ_NUMERIC_FIELD(buf,pos,max_buff_pos,field) do{       \
308   const size_t w = sizeof(field);                               \
309   pos += (pos % w) ? w - (pos % w) : 0;                         \
310   assert( pos+w <= max_buff_pos );                              \
311   memcpy( &field, buf + pos, w );                               \
312   pos += w;                                                     \
313 } while(0)
314
315
316 /** Define a macro to read a string from the buffer.  Note memory
317  *  is allocated for non-empty strings; make sure to free.
318 */
319 #define READ_STRING_FIELD(buff,pos,max_buff_pos,field) do{       \
320   uint32_t str_len; /* string length specified by uint32_t */    \
321   READ_NUMERIC_FIELD( buff, pos, max_buff_pos, str_len );       \
322   const size_t end_pos = pos + str_len;                         \
323   /* Strings are padded w/ between 1 and 4 '\0' bytes at end */ \
324   /* so overall length for the field used is a multiple of 4 */ \
325   const size_t npad_end = (end_pos % 4) ? 4 - (end_pos % 4) : 4; \
326   assert( (pos + str_len + npad_end) <= max_buff_pos );         \
327   for( size_t i = 0; i < npad_end; ++i ) {                      \
328     assert( buff[pos+str_len+i] == '\0' );                      \
329   }                                                             \
330   /* Allocate memory to hold string.  Depending on use   */     \
331   /* context, you could avoid the allocation and instead */     \
332   /* set pointer to buff+pos.                            */     \
333   if( str_len ) {                                               \
334     field = (char *)malloc(str_len + 1);                        \
335     memcpy( field, buff + pos, str_len + 1 );                   \
```

```
  } else {                                                    \
    field = NULL;                                             \
  }                                                           \
  pos += str_len + npad_end;                                  \
  /* assert( string should be UTF8 encoded ) */               \
} while(0)


/** Helper function to make sure the data represented by a string
 * is the type of data it should be.  For illustration only, not
 * rigorous.
*/
void check_valid_data( const char * const val, const uint8_t val_type )
{
  assert( val );
  assert( strlen(val) <= 255 );  //Filed values are ShortString

  switch( val_type )
  {
    case BooleanValue:
      assert( strcmp(val,"true")==0 || strcmp(val,"false")==0
              || strcmp(val,"0")==0 || strcmp(val,"1")==0 );
      break;

    case IntegerValue:
    {
      int64_t dummy;
      const int nargs = sscanf( val, "%"PRId64, &dummy );
      assert( nargs == 1 );
      break;
    }

    case FloatValue:
    {
      double dummy;
      const int nargs = sscanf( val, "%lf", &dummy );
      assert( nargs == 1 );
      break;
    }

    case Utf8StringValue:
      //assert( is UTF-8 encoded );
      break;

    case FloatingPointListValue:
    {
      errno = 0;
      size_t nval = 0;
      const char *pos = val;
      do
      {
        char *nextpos;
        const double thisval = strtod( pos, &nextpos );
        assert( errno == 0 );
        ++nval;
        pos = (nextpos && (*nextpos)) ? nextpos+1 : nextpos;
      }while( pos && (*pos) );

      assert( nval > 0 );

      break;
    }

    case IntegerListValue:
    {
      errno = 0;
      size_t nval = 0;
      const char *pos = val;
      do
      {
        char *nextpos;
        const long long thisval = strtoll( pos, &nextpos, 10 );
        assert( errno == 0 );
        ++nval;
        pos = (nextpos && (*nextpos)) ? nextpos+1 : nextpos;
      }while( pos && (*pos) );

      assert( nval > 0 );

      break;
    }

    default:
      assert(0);
  }
}


/** Decodes a RAPTER message from the input buffer to the supplied
 * ParameterInfoReply.
 * Asserts on error.
 * @returns Number of bytes read from buffer if successful.
*/
size_t decode_ParameterInfoReply( struct ParameterInfoReply * const msg,
                                  const uint8_t * const buffer,
                                  const size_t buff_size )
{
  assert( buff_size >= 76 );  //minimum size of this struct

```

```
  size_t pos = 0;
  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->message_group );
  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->message_type );
  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->message_group_version );
  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->message_flags );
  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->message_id );

  //Check is a core interface message
  assert( msg->message_group == 0x00 );
  //Check message is a ParameterInfoReply
  assert( msg->message_type == 0x93 );
  //Version 0 of core interface
  assert( msg->message_group_version == 0x00 );
  //Attached notifications, list, or buffering not supported in ex. code
  assert( msg->message_flags == 0x00 );

  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->reply_status );
  assert( msg->reply_status == ValidValue
          || msg->reply_status == InvalidNameValue );

  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->value_type );
  assert( msg->value_type == BooleanValue
          || msg->value_type == IntegerValue
          || msg->value_type == FloatValue
          || msg->value_type == Utf8StringValue
          || msg->value_type == FloatingPointListValue
          || msg->value_type == IntegerListValue );

  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->parameter_health_impact );
  //assert( msg->parameter_health_impact is valid value );

  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->subdetector_number );
  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->value_properties );
  //assert( msg->value_properties only has valid combination of bits set )

  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->set_timestamp );
  READ_NUMERIC_FIELD( buffer, pos, buff_size, msg->effective_timestamp );

  READ_STRING_FIELD( buffer, pos, buff_size, msg->parameter_name );
  READ_STRING_FIELD( buffer, pos, buff_size, msg->set_value );
  READ_STRING_FIELD( buffer, pos, buff_size, msg->measured_value );
  READ_STRING_FIELD( buffer, pos, buff_size, msg->lower_healthy_measured_value );
  READ_STRING_FIELD( buffer, pos, buff_size, msg->upper_healthy_measured_value );
  READ_STRING_FIELD( buffer, pos, buff_size, msg->lower_settable_value );
  READ_STRING_FIELD( buffer, pos, buff_size, msg->upper_settable_value );
  READ_STRING_FIELD( buffer, pos, buff_size, msg->reportable_change_delta );
  READ_STRING_FIELD( buffer, pos, buff_size, msg->parameter_description );


  //Check the string values we expect exist, and are properly formatted.
  if( msg->value_properties & SettableFlag )
    check_valid_data( msg->set_value, msg->value_type );
  else { assert( !msg->set_value ); }

  if( msg->value_properties & MeasuredFlag )
    check_valid_data( msg->measured_value, msg->value_type );
  else { assert( !msg->measured_value ); }

  if( msg->value_properties & HealthyValueRangedFlag )
    check_valid_data( msg->lower_healthy_measured_value, msg->value_type );
  else { assert( !msg->lower_healthy_measured_value ); }

  if( msg->value_properties & HealthyValueRangedFlag )
    check_valid_data( msg->upper_healthy_measured_value, msg->value_type );
  else { assert( !msg->upper_healthy_measured_value ); }

  if( msg->value_properties & SetValueRangedFlag )
    check_valid_data( msg->lower_settable_value, msg->value_type );
  else { assert( !msg->lower_settable_value ); }

  if( msg->value_properties & SetValueRangedFlag )
    check_valid_data( msg->upper_settable_value, msg->value_type );
  else { assert( !msg->upper_settable_value ); }

  if( msg->value_properties & ReportOnChangeFlag )
    check_valid_data( msg->reportable_change_delta, msg->value_type );
  else { assert( !msg->reportable_change_delta ); }

  assert( msg->parameter_name );
  // Check name only has ascii characters
  for( const char *p = msg->parameter_name; *p; ++p )
  { assert( ((unsigned char)*p) <= 127 ); }

  const size_t namelen = strlen(msg->parameter_name);
  assert( msg->parameter_name && namelen>0 && namelen <= 255 );

  assert( !msg->parameter_description
          || strlen(msg->parameter_description) <= 65535 );

  return pos;
}


int main()
{
  /*  We will create a ParameterInfoReply struct and fill its information
      out, serialize it to a buffer that could be sent as a RAPTER message
      then deserialize the buffer back into a ParameterInfoReply struct.
```

```
534     Then we will serialize a HighVoltagParameter struct to a RAPTER
535     ParameterInfoReply message in a buffer, and then deserialize it
536     to a ParameterInfoReply struct.
537 */
538
539 struct ParameterInfoReply msg;
540
541 //Initialize msg to all zeros.
542 memset( &msg, 0, sizeof(struct ParameterInfoReply) );
543
544 msg.message_group = 0;          //Message belongs to Core Interface
545   msg.message_type = 0x93;        //Message is a ParameterInfoReply
546   msg.message_group_version = 0;  //Version of Core interface
547   msg.message_flags = 0;          //no notif., not list, not buffered
548 msg.message_id = 3333;          //Message id
549
550 msg.reply_status = ValidValue;
551 msg.value_type = FloatValue;
552 msg.parameter_health_impact = NoProblemValue;
553 msg.value_properties = (SettableFlag
554                         | MeasuredFlag
555                         | HealthyValueRangedFlag
556                         | ReportOnChangeFlag
557                         | NotSettableWhileOperatingFlag);
558 msg.set_timestamp = 1508830130000000;
559 msg.effective_timestamp = 1508830130000000;
560
561 msg.subdetector_number = 1;
562
563 const char *par_name = "high_voltage";
564 msg.parameter_name = (char *)malloc( strlen(par_name) + 1 );
565 strcpy( msg.parameter_name, par_name );
566
567
568 const char *par_desc = "A parameter that is settable, measured, has a"
569                        " define range of healthy values, changes are"
570                        " reported every 1.5 volts.";
571 msg.parameter_description = (char *)malloc( strlen(par_desc) + 1 );
572 strcpy( msg.parameter_description, par_desc );
573
574
575 const char *set_value = "1330";
576 msg.set_value = (char *)malloc( strlen(set_value) + 1 );
577 strcpy( msg.set_value, set_value );
578
579 const char *measured_value = "1331.32";
580 msg.measured_value = (char *)malloc( strlen(measured_value) + 1 );
581 strcpy( msg.measured_value, measured_value );
582
583 const char *low_healthy = "100.0";
584 msg.lower_healthy_measured_value
585                  = (char *)malloc( strlen(low_healthy) + 1 );
586 strcpy( msg.lower_healthy_measured_value, low_healthy );
587
588 const char *up_healthy = "2000.0";
589 msg.upper_healthy_measured_value
590                  = (char *)malloc( strlen(up_healthy) + 1 );
591 strcpy( msg.upper_healthy_measured_value, up_healthy );
592
593 msg.lower_settable_value = NULL;
594 msg.upper_settable_value = NULL;
595
596 const char *change_delta = "1.5";
597 msg.reportable_change_delta
598                  = (char *)malloc( strlen(change_delta) + 1 );
599 strcpy( msg.reportable_change_delta, change_delta );
600
601
602
603 //Lets write msg to a buffer as a RAPTER message
604 const size_t buffer_size = 2048;
605 uint8_t buffer[buffer_size] = { 0 };
606
607 size_t encoded_len = encode_ParameterInfoReply(
608                                 &msg, buffer, buffer_size );
609
610 //Now test that we can decode a ParameterInfoReply message from buffer
611 struct ParameterInfoReply decoded_msg;
612 size_t decoded_len = decode_ParameterInfoReply(
613                                 &decoded_msg, buffer, encoded_len );
614
615 //Lets make sure we made the round trip okay
616 assert( encoded_len == decoded_len );
617
618 assert( msg.message_group == decoded_msg.message_group );
619 assert( msg.message_type == decoded_msg.message_type );
620 assert( msg.message_group_version
621         == decoded_msg.message_group_version );
622 assert( msg.message_flags == decoded_msg.message_flags );
623 assert( msg.message_id == decoded_msg.message_id );
624
625 assert( msg.reply_status == decoded_msg.reply_status );
626 assert( msg.value_type == decoded_msg.value_type );
627 assert( msg.parameter_health_impact
628         == decoded_msg.parameter_health_impact );
629 assert( msg.subdetector_number
630         == decoded_msg.subdetector_number );
631
632 assert( msg.value_properties == decoded_msg.value_properties );
```

```
633   assert( msg.set_timestamp == decoded_msg.set_timestamp );
634   assert( msg.effective_timestamp == decoded_msg.effective_timestamp );
635
636   check_str_equal( msg.set_value,
637                    decoded_msg.set_value );
638
639   check_str_equal( msg.measured_value,
640                    decoded_msg.measured_value );
641
642   check_str_equal( msg.lower_healthy_measured_value,
643                    decoded_msg.lower_healthy_measured_value );
644
645   check_str_equal( msg.upper_healthy_measured_value,
646                    decoded_msg.upper_healthy_measured_value );
647
648   check_str_equal( msg.lower_settable_value,
649                    decoded_msg.lower_settable_value );
650
651   check_str_equal( msg.upper_settable_value,
652                    decoded_msg.upper_settable_value );
653
654   check_str_equal( msg.reportable_change_delta,
655                    decoded_msg.reportable_change_delta );
656
657   check_str_equal( msg.parameter_name,
658                    decoded_msg.parameter_name );
659
660   check_str_equal( msg.parameter_description,
661                    decoded_msg.parameter_description );
662
663
664   //
665   struct HighVoltagParameter simplepar;
666   simplepar.set_value = 1330;
667   simplepar.measured_value = 1331.32;
668   simplepar.set_timestamp = msg.set_timestamp;
669   simplepar.effective_timestamp = msg.effective_timestamp;
670
671   memset( buffer, 0, buffer_size );
672   encoded_len = encode_HighVoltagParameter(
673                              &simplepar, buffer, buffer_size );
674
675   //Lets reuse decoded_msg, so free its strings, and zero it out
676   free_string_fields( &decoded_msg );
677   memset( &decoded_msg, 0, sizeof(decoded_msg) );
678
679   decoded_len = decode_ParameterInfoReply(
680                              &decoded_msg, buffer, encoded_len );
681
682   //Check to make sure the HighVoltagParameter decoded to a
683   //  ParameterInfoReply gave values expected.
684   assert( encoded_len == decoded_len );
685   assert( decoded_msg.value_type == FloatValue );
686   assert( fabs(1330 - atof(decoded_msg.set_value)) < 0.0001 );
687   assert( fabs(1331.32 - atof(decoded_msg.measured_value)) < 0.0001 );
688
689   free_string_fields( &msg );
690   free_string_fields( &decoded_msg );
691
692   printf( "Passed encoding/decoding checks.\n" );
693
694   return 1;
695 }
```

Bibliography

[1] S. Bradner, “Key words for use in RFCs to Indicate Requirement Levels,” RFC 2119, The Internet Society, March 1997. 7, 8, 10

[2] A. M. I. Fette, “The WebSocket Protocol,” RFC 6455, The Internet Society, December 2011. 11, 24, 29

[3] “Ieee standard for a precision clock synchronization protocol for networked measurement and control systems - redline,” *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002) - Redline*, pp. 1–300, July 2008. 17, 29

[4] “American national standard data format for radiation detectors used for homeland security,” *ANSI N42.42-2012 (Revision of ANSI N42.42-2006)*, pp. 1–248, April 2013. 18, 20

[5] C. Hornig, “A Standard for the Transmission of IP Datagrams over Ethernet Networks,” RFC 894, The Internet Society, April 1984. 24

[6] e. a. J. Mogu, “INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION,” RFC 791, The Internet Society, September 1981. 24

[7] R. H. S. Deering, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 2460, The Internet Society, December 1998. 24

[8] B. H. S. Deering, W. Fenner, “Multicast Listener Discovery (MLD) for IPv6,” RFC 2710, The Internet Society, October 1999. 24

[9] L. S. Committee, “Ieee standard for ethernet,” *IEEE Std 802.3 - 2015 (Revision of IEEE Std 802.3-2012)*, pp. 1–698, September 2015. 24

[10] L. S. Committee, “Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications,” *IEEE Std 802.11 - 2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–300, March 2012. 25

[11] “Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements,” *IEEE Std 802.3af-2003 (Amendment to IEEE Std 802.3-2002, including IEEE Std 802.3ae-2002)*, pp. 1–121, 2003. 25

[12] e. a. R. Droms, J. Bound, “Dynamic Host Configuration Protocol for IPv6 (DHCPv6),” RFC 3315, The Internet Society, July 2003. 25

[13] e. a. K. Stouffer, “Guide to Industrial Control Systems (ICS) Security,” NIST Pubs, NIST, June 2015. 25

[14] E. R. T. Dierks, “The Transport Layer Security (TLS) Protocol Version 1.2,” RFC 5246, The Internet Society, August 2008. 26

[15] S. Bellovin, “Guidelines for Specifying the Use of IPsec Version 2,” RFC 5406, The Internet Society, February 2009. 26

[16] e. a. A. Donoho, “Upnp device architecture 2.0,” *UPnP Forum*, pp. 1–196, Febraury 2015. 26, 28

[17] T. Yoshino, “Compression Extensions for WebSocket,” RFC 7692, The Internet Society, December 2015. 29

[18] “Ieee standard for floating-point arithmetic,” *IEEE Std 754-2008*, pp. 1–70, Aug 2008. 45, 47, 305

[19] I. Sun Microsystems, “XDR: External Data Representation Standard,” RFC 1014, The Internet Society, June 1987. 47

[20] M. Wildgrube, “Structured Data Exchange Format,” RFC 3072, The Internet Society, March 2001. 47

[21] e. a. R. Housley, W. Ford, “Internet X.509 Public Key Infrastructure Certificate and CRL Profile,” RFC 2459, The Internet Society, January 1999.

[22] e. a. J. Mogul, D. Mills, “Pulse-Per-Second API for UNIX-like Operating Systems, Version 1.0,” RFC 2783, The Internet Society, March 2000.

[23] e. a. A. Presser, “Upnp device architecture 1.1,” *UPnP Forum*, pp. 1–129, October 2008.

[24] e. a. D. Mills, “Network Time Protocol Version 4: Protocol and Algorithms Specification,” RFC 5905, The Internet Society, June 2010.