Date: February 2026

Ground Equipment Monitoring Service (GEMS)

Version 1.7

OMG Document Number: dtc/26-03-22

Normative reference: https://www.omg.org/spec/GEMS/

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 9C Medway Road, PMB 274, Milford, MA 01757, U.S.A.

TRADEMARKS

IMM®, MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Specifications, Report a Bug/Issue (https://www.omg.org/report_issue.)

Table of Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

http://www.omg.org/spec

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
9C Medway Road
PMB 274
Milford, MA 01757
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to http://www.omg.org/report_issue.htm.

Introduction to Specification

Communication with space vehicles often requires a sophisticated suite of ground equipment ranging from network devices to antennas. These devices work together to create an end-to-end signal-processing suite. For example, an antenna, frequency converter, demodulator, bit synchronizer, decryption device, and frame synchronizer process a telemetry down-link signal before the end-user software receives the data. These devices must be configured to properly process the signal and provide status to user applications.

The Ground Equipment Monitoring Service (GEMS) specification defines a common messaging interface to do just this. The intent of GEMS is to define a lightweight, easy-to-use interface model suitable for control and status of nearly all types of devices within space related ground systems. In truth, these devices are not unique to space communications. Modern ground systems include many commercial devices such as networking devices and digital archive systems. While this specification focuses specifically on space applications, it is designed to apply to a broad range of devices.

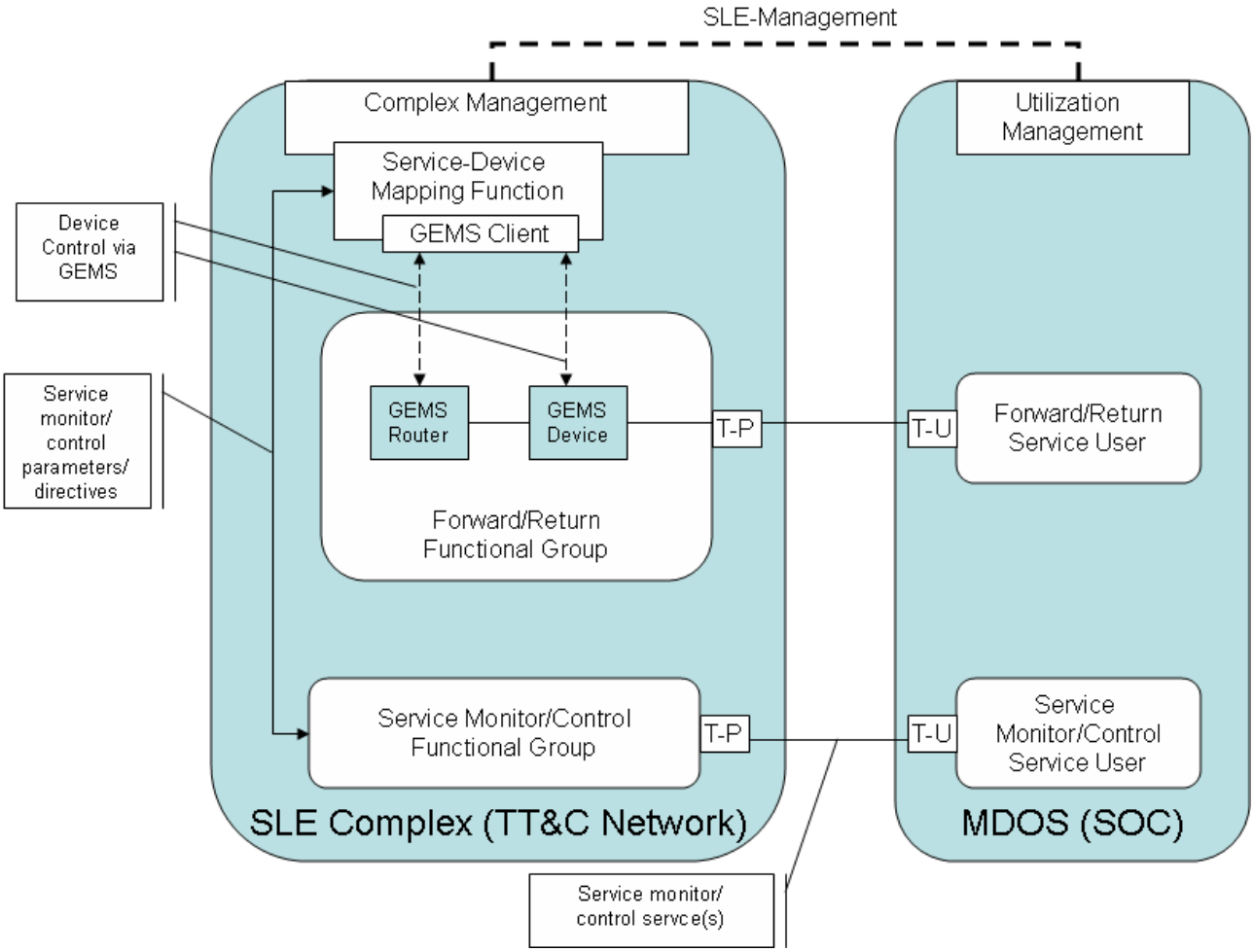
User's Perspective

To understand the GEMS design, it is useful to first look at it from a user's perspective. Typically, high-level software, such as a TT&C (Telemetry, Tracking & Commanding) application, controls the ground system equipment. The main purpose of these types of applications is controlling the space vehicle – a complicated task in its own right. The ground system enables communication between the TT&C applications and the space vehicle by performing low-level signal and data processing functions. While the ground system is a significant part of the signal processing capabilities, if operating properly it should be transparent to the user. Only initial configuration and basic status is needed. This should be simple and ideally use a standard model for all types of devices. That is where the GEMS model comes in. The GEMS model defines a simple, message-based interface suitable for controlling all types of devices using a variety of protocols and transport mechanisms. It allows system integrators to develop control and status applications that can easily interface with a wide range of device types.

Vendor's Perspective

From a vendor's perspective, a standard device control protocol should be relatively easy to implement and require limited resources. In addition, to ensure the widest possible range of integrations, the protocol should not require any specific third-party middleware implementation. Using GEMS, vendors achieve exactly this. The GEMS protocol is a lightweight protocol requiring only a TCP/IP connection. The Platform Independent Model (PIM) simply describes the messages and interactions necessary to configure and obtain status on a device. The Platform Specific Models (PSM) map the PIM to basic ASCII and XML messages. The ACII PSM is intended for the most basic devices with limited processing capabilities. It is well suited for Serial (RS-232) and terminal style devices. The XML PSM incorporates the more sophisticated capabilities of XML such as XML Schemas and validation. While these capabilities require additional libraries, formatting and processing the messages is still quite simple.

The GEMS interface compliments other existing standards such as CCSDS SLE. The CCSDS SLE specification provides facility level interfaces but does not address the lower level device interfaces. It accomplishes this through a Service-Device mapping function. The GEMS specification enables a standard Service-Device mapping function to be created for a wide range of device types. The following figure depicts this interaction.



In this figure, the SLE Complex Management Service layers on top of the GEMS client to control the physical devices in the system.

1 Scope

The GEMS specification defines a lightweight standard for the control and status of typical ground equipment found in the space domain using a model driven approach. A conceptual model for the GEMS specification is shown below.

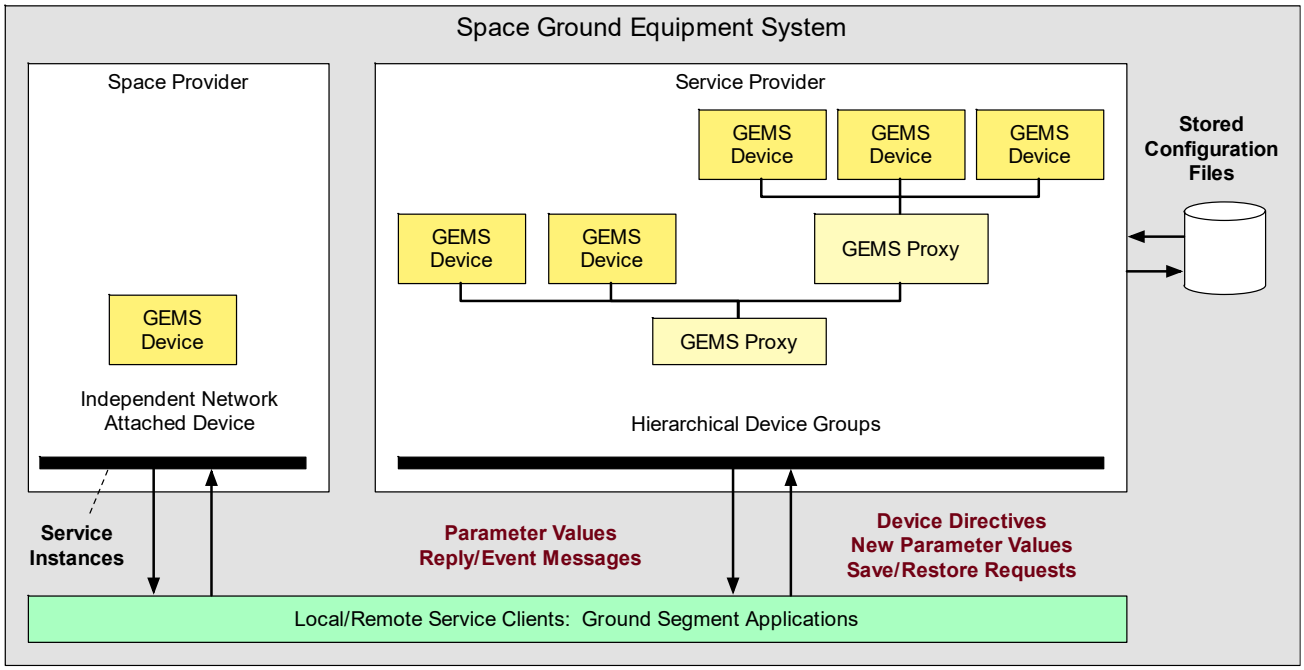


Figure 1.1 GEMS Overview

The GEMS specification is most commonly used for the control of a single device. However, in more complicated systems, multiple devices may be controlled using a single container message (MessageSequence) combined with a GEMS Proxy.

The GEMS PIM defines the message structure and behavior standard to all GEMS Devices. In the case of GEMS, the platform is defined as the middleware, network protocol, or transport mechanism used to communicate with the device. By defining the platform at this level, the GEMS PIM focus is primarily on the necessary message content and behavior and leaves the specifics of defining the message format and transporting that message to the PSMs.

Two GEMS PSMs are defined. The raw GEMS-ASCII PSM is a terse ASCII-based protocol well suited for serial or terminal based devices. The focus of this PSM is short, human-readable messages that are easily formatted and processed. These messages are transported directly across either a network or serial bus. The GEMS-XML PSM utilizes the features of XML to provide message definition and validation. These messages use an HTML like header for transport across a network. Otherwise, GEMS-XML messages leverage the structure inherent in an XML document to define the content.

Many other platform specific mappings of the GEMS PIM are possible. These potentially include other standard middleware or protocol definitions such as CORBA, SNMP, or the GPIB-SCPI protocol. In fact, the large number of potential protocols used for device control further illustrates the need for the GEMS specification. By defining a specific mapping of each of these protocols to the GEMS PIM, automatic translation between these protocols is possible. This enables a device vendor to focus on the features and capabilities of the device without the need to support multiple protocols. Similarly, the device user can utilize a single protocol, with appropriate translators, to control numerous devices.

2 Conformance

The primary point of conformance is support of the PIM. Conformance to any defined PSM is optional, but if a defined platform is used, such as XML and raw ASCII, the implementation must conform to the appropriate PSM. In the event that a PSM does not exist for a specific protocol, implementers are encouraged to define a PSM and submit it for standardization to the OMG.

3 References

3.1 Normative References

XML Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler, editors. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* World Wide Web Consortium, 2008. (See: https://www.w3.org/TR/xml).

ASCII American National Standard for Information Systems – Coded Character Sets – 7 Bit American National Standard Code for Information Interchange (7-Bit ASCII), ANSI X3.4-1 986, American National Standards Institute, Inc., March 26, 1986

4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

TT&C: Telemetry Tracking & Commanding

CCSDS: Consultative Committee for Space Data Systems SLE: CCSDS Space Link Extensions

SNMP: Simple Network Management Protocol

GPIB: General Purpose Interface Bus

SCPI: Standard Commands for Programmable Instruments

N/A: Not Applicable

5 Additional Information

5.1 Acknowledgments

The following companies submitted this specification:

- Kratos S1, Inc.
- AMERGINT Technologies Inc.

The following companies supported this specification:

- Harris Corp
- Boeing Corp

6 PIM

6.1 Overview

The GEMS specification defines a standard, platform independent model (PIM) for controlling a wide range of devices. The GEMS model does not presume or try to define a specific system level architecture. Instead, it defines generic concepts such as devices, parameters, and directives that are relatively simple to implement and provide system integrators common ways to control heterogeneous suites of space related ground equipment.

The central concept of GEMS is the GEMS device. GEMS devices have typed parameters, accept directives with typed arguments, and can optionally save and restore their configuration using persistent storage. Users utilize the GEMS interface within the device to configure and obtain status.

For more sophisticated systems with multiple devices, a GEMS proxy is deployed and routes message traffic to a system of devices. This allows for several devices to be configured in a single transaction. In addition, the proxy often supports the Save/Restore functionality, thus keeping the GEMS devices simple.

The GEMS PIM consists of message related classes that allow the user to send directives, configure devices, obtain configuration information and device status, save the configuration, and restore the configuration.

6.2 GEMS Use Cases

The following diagram depicts the GEMS use cases. These use cases define common interactions and activities associated with creating and using suites of devices.

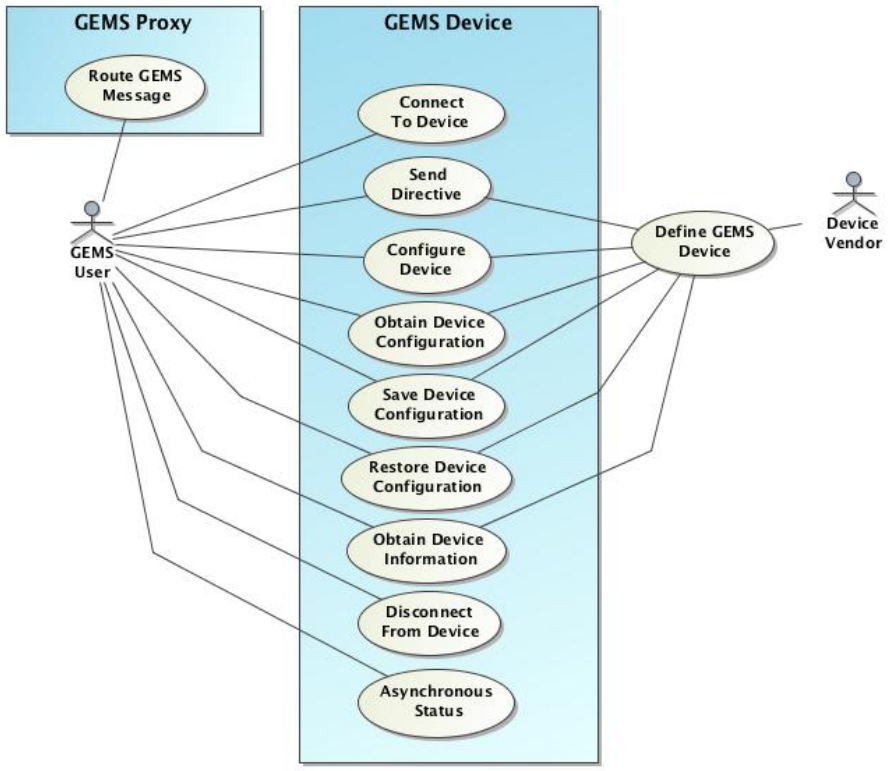


Figure 6.1 GEMS Use Cases

6.2.1 GEMS User

This actor represents a user of a GEMS device or system. The GEMS user commonly takes the form of a controlling software application that uses the GEMS interface to manipulate a device or system of devices.

6.2.2 Device Vendor

As the name indicates, the Device Vendor is the manufacturer of the device supporting the GEMS interface. The Device Vendor's role is to define the parameters and directives supported by the device and provide those along with the device to the System Integrator and/or GEMS User.

6.2.3 Define GEMS Device

This use case represents the activities necessary to define a GEMS device. These include defining the names, types, and ranges for all device parameters as well as the directives and associated arguments.

6.2.4 Connect To Device

The first action that a GEMS user must take to establish an interaction with a GEMS device is to connect to the device. In doing this, the GEMS user identifies whether control of the device or status only is requested. This is similar to requesting read/write access to a file. The GEMS device responds to a connection request by indicating whether or not the connection was successful and if successful, provides a token that the GEMS user will use in all future interactions with the device.

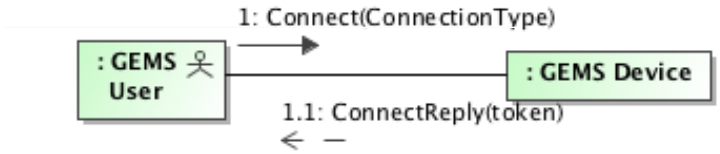


Figure 6.2 Connecting

Authentication may optionally be performed by passing GEMS User credentials in the token field of the Connect Request. This is described in section 6.6.1.

6.2.5 Send Directive

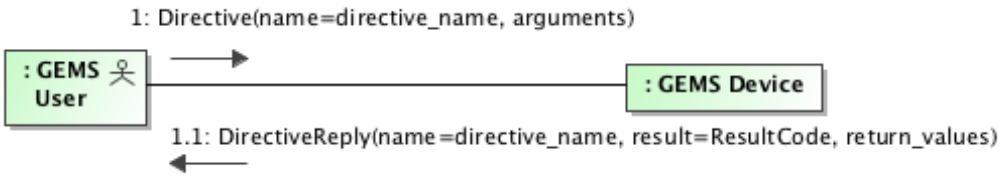


Figure 6.3 Sending a directive

GEMS directives allow the user to manipulate the device or system in a scoped action. Common examples relating to ground system equipment are enabling modulation of a signal or sending a vehicle command. In this use case, the user formats a GEMS directive message and sends it to the GEMS device. The GEMS device performs the actions necessary to fulfill the directive and then sends a reply message back to the user indicating the result of the directive. The response includes any return values appropriate for the directive.

6.2.6 Configure Device

A common use case is configuring a device. In this use case, the user sends a set of configuration parameters to the device. The device performs the appropriate validation of the parameters and then applies the values to its configuration. The GEMS device sends a response back to the user indicating success or failure. If successful, the response also includes the number of parameters affected.

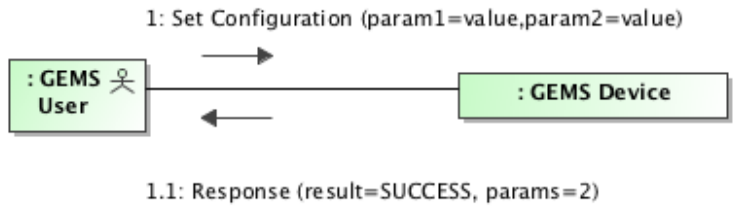


Figure 6.4 Configuring a device

6.2.7 Obtain Configuration

In this use case, the user wishes to obtain information about the device configuration. The user sends a message to the GEMS device requesting configuration information. The request can specify specific parameters that the user wishes information on. The GEMS device receives the request and populates a reply with the appropriate parameters.

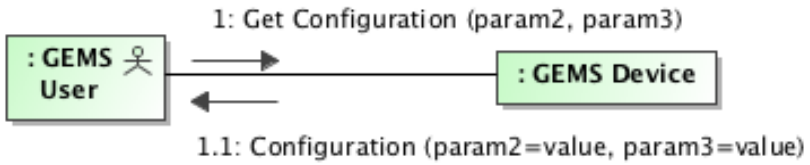


Figure 6.5 Obtaining device configuration

6.2.8 Save Configuration

A convenient use case in satellite operations is configuring the device for a specific space vehicle and then saving that configuration for later use. Often, configurations are saved for different missions. To accomplish this, the user sends a message requesting the GEMS device to save its configuration to local storage. The user specifies the name of the configuration for later recall.

This use case is optional since saving the configuration to persistent storage, such as a hard drive or Flash RAM, is not necessarily available on all types of devices. For these types of devices, it is expected that higher-level applications such as a GEMS router will provide this capability by first obtaining the device configuration and then writing it to persistent storage.

6.2.9 Restore Configuration

Once the user has saved a configuration to persistent storage, it can be recalled later. The user sends a restore configuration message to the GEMS device. The message includes the name of the configuration to restore. The GEMS device then loads that configuration and applies the values specified to the device. The response message indicates success or failure, and the number of parameters modified.

It is not required for the named configuration to contain all device parameters. Only the parameters specified are modified. This enables a useful approach to controlling a device. GEMS users can load full configuration followed by selected subsets.

Like saving the configuration, this use case is optional for devices that do not have a persistent storage capability. In these cases, it is expected that higher-level software applications such as a GEMS router store the configurations and then apply those configurations to the device.

6.2.10 Obtain Device Information

In this use case, the user obtains information about what directives and parameters the GEMS device accepts. The user sends a message to the GEMS device requesting service information. The reply contains descriptions of the directives (including argument name and type information) and the parameters (including name, type, and range information).

Additional device definitions may be available from the device vendor as described in section 6.9.

6.2.11 Asynchronous Status

In this use case, the user obtains asynchronous status messages containing the current values of various GEMS parameters. The user configures the frequency and type of update using parameters and directives on the GEMS device. Standardization of these parameters and directives are left for later versions of GEMS. The messages are "pushed" asynchronously and can be provided on either the current GEMS connection or an alternate out-of-band connection.

6.2.12 Proxy & Route GEMS Message

Typically, GEMS messages target a specific device. However, in more complex systems, it is convenient and sometimes required to manipulate multiple devices in a single transaction. In this case, multiple messages are combined into a message set and sent to a GEMS proxy for processing. The GEMS Proxy separates the messages and passes them to the designated targets sequentially.

This capability is not required of GEMS devices, though it can be used to offer sequential execution of messages. A GEMS proxy can also provide functions such as Save/Restore and message logging. These functions are not required in a GEMS proxy.

A GEMS proxy is considered to be a device and can be the target of messages such as the SetConfigMessage and GetConfigMessage. The specifics of any proxy parameters, directives, or other behavior is left to the implementor to define. At a minimum, in the event a GetConfigMessage is sent to a proxy, the resulting GetConfigResponse will contain at least one parameter named 'targets.' That parameter is an array of strings listing all of the targets handled by that specific proxy instance.

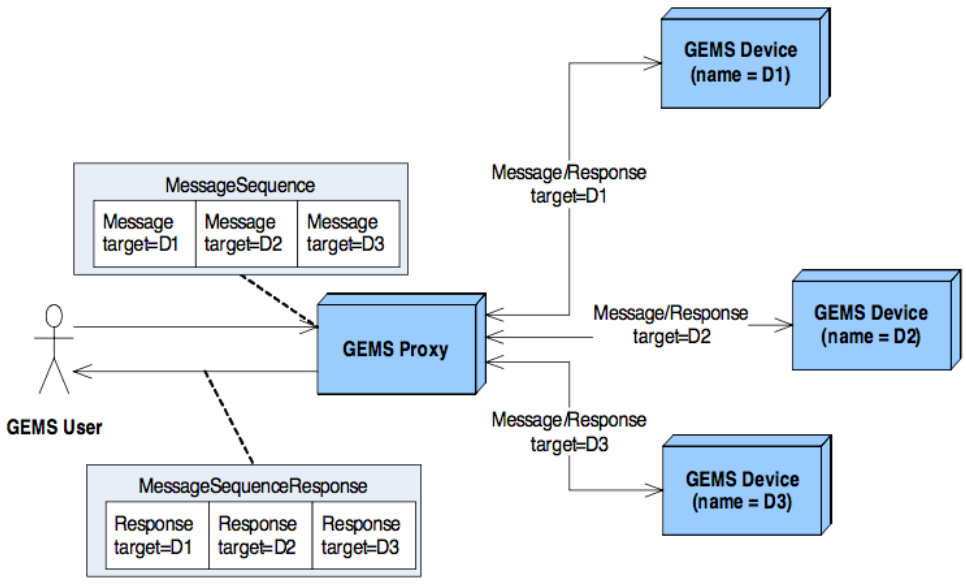


Figure 6.6 Message routing

6.2.13 GEMS High Level Design

The GEMS protocol defines the basic message structure and interaction between a user and a GEMS device. It does not specify the exact parameters, types, or ranges for any specific device or device type. That is beyond the scope of this specification. Instead, GEMS defines the approach to use when defining device specific parameters. The device vendor provides custom device information in a format compliant with the PIM and PSM used. To represent this interaction, GEMS defines two notional packages. These packages are not part of this specification. The Custom Devices package contains definitions of concrete devices. These devices meet the GEMS specification but are custom to a given vendor. The Standard Devices package contains definitions of standard devices, their parameters (names, types, and ranges) and directives. It is envisioned that this package eventually becomes an addendum to this specification.

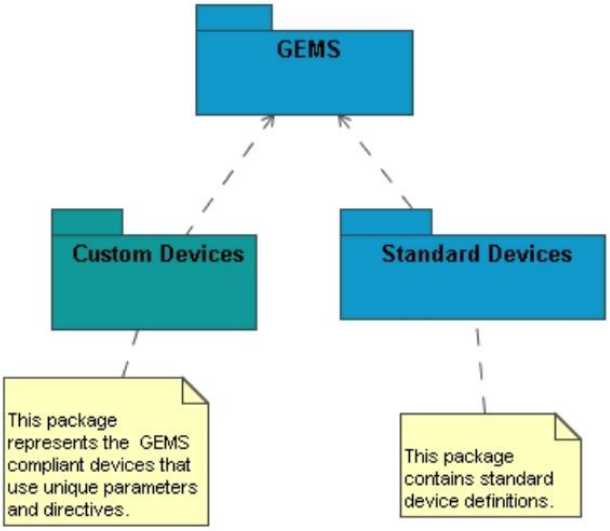


Figure 6.7 GEMS Packages

Typically, the device vendor defines and maintains the associated GEMS device definitions. The device definitions are either distributed manually by the vendor (e.g. incorporated into product documentation or as a GEMS Device Definition file), or directly from the GEMS device as an electronic document. See section 6.8 for more information on Device Definitions. By supporting a GEMS interface, the device vendor enables customers to easily control the device in a standard manner following the GEMS model. However, there is no guarantee that the parameter names, types, and ranges used will be interoperable with other similar devices produced by other vendors. For example, a common status parameter supported by receivers is signal lock. This parameter indicates when the receiver has locked on to the desired signal. One vendor might name that parameter LOCK_STATE while another vendor might name it SIGNAL_LOCK. To help standardize parameter choices a dictionary of device types and parameters will be developed.

6.3 GEMS UML Design

The following class diagram shows the GEMS classes and their relationships. At the top are the message classes. These classes define the various types of messages available through the GEMS protocol. Each message class, with the exception of the Disconnect class, has an associated response message. In several cases, the message contains one or more parameters.

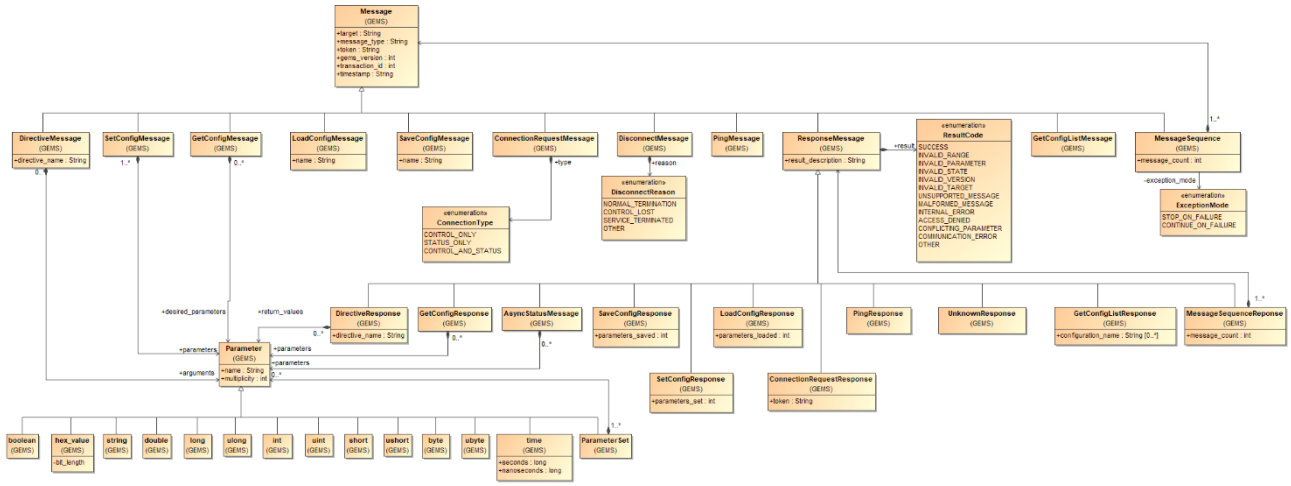


Figure 6.8 GEMS UML

These classes are described in detail in the following sections.

6.3.1 Parameters

Parameters represent the actual values used to configure and provide status on a given device. Each Parameter has a name, type, and a multiplicity. The name is a free text string and cannot contain any spaces. The multiplicity represents arrays of the same type. The specific implementation of the multiplicity is left to the PSM.

For completeness, the PIM defines both signed and unsigned types as well as a variety of integer sizes. If appropriate, these various integer types and precisions may be mapped to a single integer type within a PSM.

6.3.1.1 UML Diagram

This diagram shows the base parameter class and all of the specific types.

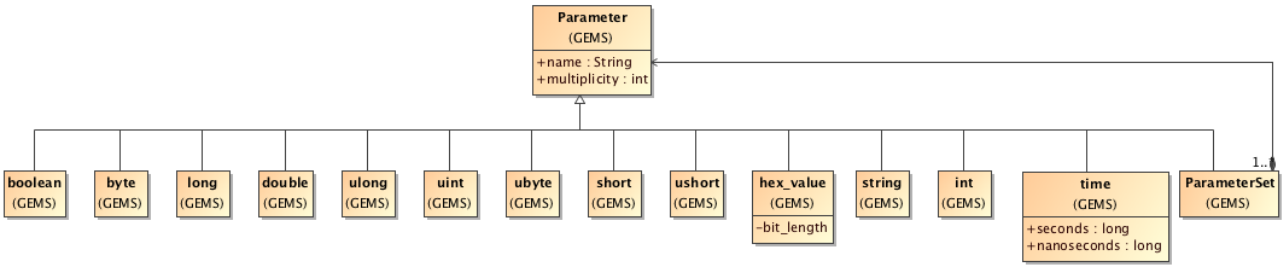


Figure 6.9 GEMS Standard Types

6.3.1.2 boolean

Represents a boolean true/false value.

6.3.1.3 byte

Represents a single signed 8 bit byte or octet.

6.3.1.4 ubyte

Represents a single unsigned 8 bit byte or octet.

6.3.1.5 long

Represents a signed 8 byte value.

6.3.1.6 ulong

Represents an unsigned 8 byte value.

6.3.1.7 int

Represents a signed 4 byte value.

6.3.1.8 uint

Represents an unsigned 4 byte value.

6.3.1.9 short

Represents a signed 2 byte value.

6.3.1.10 ushort

Represents an unsigned 2 byte value.

6.3.1.11 double

Represents a double precision floating point number.

6.3.1.12 string

Represents a free text ASCII string of characters.

6.3.1.13 hex_value

Represents an ASCII representation of a hexadecimal value. The string optionally may be preceded by a '0x.' The bit_length is used to specify the number of bits represented in the hex_value. See the PSM specific mapping for the format of this field.

6.3.1.14 time

Represents the number of seconds and nanoseconds elapsed since midnight UTC of January 1, 1970. The time value is represented by integer values for seconds and nanoseconds.

6.3.1.15 utime

Represents a time value as Coordinated Universal Time (UTC). This format allows for the representation of leap seconds within a time parameter.

6.3.1.16 Parameter Sets

Parameter sets allow for the creation of mixed-type structures of parameters or "complex types" using the composite design pattern. The intent is to offer device vendors an option for creating arbitrarily complex data structures.

6.3.2 Messages

GEMS defines a set of messages that allow parameters and directives to be sent between a GEMS client and GEMS device. These messages all have a common header and structure. Each message sent to the GEMS device has a corresponding response.

The following diagram depicts the GEMS message class structure.

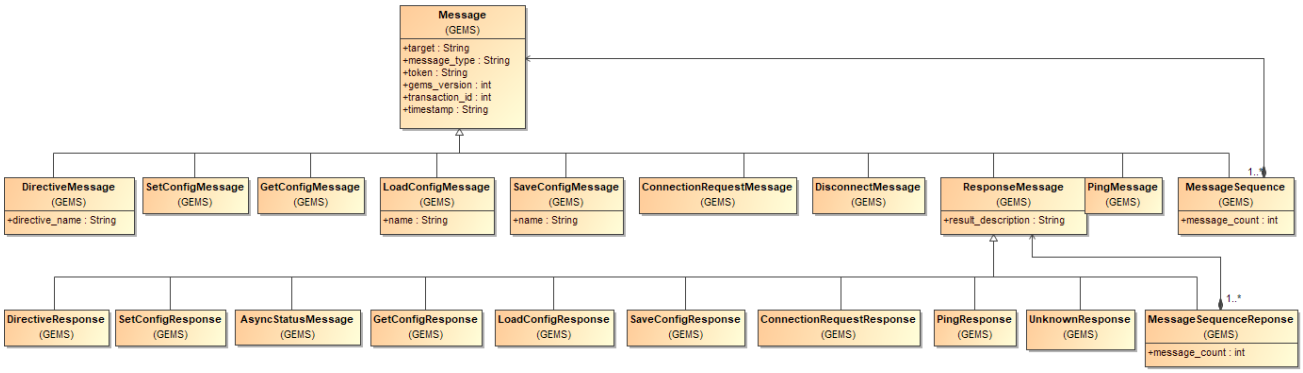


Figure 6.10 GEMS Message Classes

6.3.2.1 Message Base Class

The Message base class defines the header information necessary to determine the target for the messages and other information such as the version number.

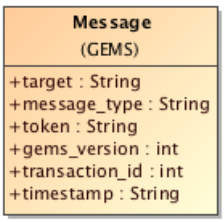


Figure 6.11 GEMS Message Base Class

target

The target is a free text field containing the name of the target device. For a single device, the target is a word naming the device. If the device is part of a system hierarchy, the levels within the hierarchy are concatenated using '/' characters similar to UNIX directory paths. This naming scheme allows GEMS proxies to properly route messages.

Example:

/SiteA/Modem/Modulator1

The target field is optional if the message is sent directly to the targeted device. For distributed systems that utilize a GEMS proxy, the proxy is responsible for mapping the target names to network addresses as necessary.

message_type

The message _type field is an alpha-numeric field containing a message identifier for a specific message type. The specific values are defined in the PSM.

token

The token field is a free-text field containing an ASCII token. The exact format and content of the token is dependent on the GEMS device. The GEMS device gives the token to the GEMS user as part of the initial connection. The token is then passed back to the GEMS device with every message.

While this specification does not define the use of the token field, it does offer recommendations. For example, if strict authentication is desired, a GEMS device can encode values in to the token that clearly identify the client and any privileges that client may have.

A common use of the message token is for limiting access to device control related features. When a client connect message is received, the message contains the type of access being requested (e.g., control or status).

If there are currently no control clients connected, the GEMS device gives the control token to the new client. While this mechanism is not fool-proof, it does work well in controlled environments.

version

The version field contains the version of GEMS message being used. It provides backwards compatibility. Examples of supported versions are: 1.0, 1.1, 1.2, 1.3, 1.4, etc..

timestamp

The timestamp field provides useful debugging and message sorting information. The value in the field is the current time when the message is sent in UTC. The format of the field is defined in the PSM.

6.3.2.2 Response Messages

All GEMS response messages contain a result code and an optional free-text description. The result code indicates the success of failure of the original message. If a failure occurs, the specific result code can be inspected programmatically and the appropriate action taken. In the event an unrecognizable message is received, a generic base response message is returned with the result code set to indicate the error. Most commonly the result code for this type of error is MALFORMED_MESSAGE.

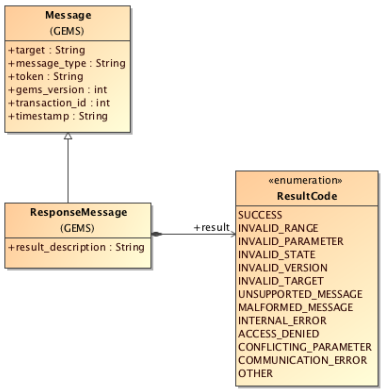


Figure 6.12 GEMS Response Messages

The acceptable result codes are as follows:

SUCCESS

This result code indicates that the associated message or directive was successful. The result_description is optional.

INVALID_RANGE

This result code indicates that a parameter or argument within the associated message or directive was out of the acceptable range. In this error condition, the result_description should contain a free-text description of which parameter was in error.

INVALID_PARAMETER

This result code indicates that an unsupported or unknown parameter was named in the associated message. In this error condition, the result_description should contain a free-text description of which parameter was in error.

INVALID_STATE

This result code indicates that an invalid state was reached within the device or GEMS interface. Common reasons for this are attempts to set or get parameter values before a connection is established. In this error condition, the result_description should contain a free-text description of the error.

INVALID_VERSION

This result code indicates that the GEMS message version was unrecognized or unsupported by the device. In this error condition, the result_description should contain a list of the supported GEMS versions.

INVALID_TARGET

This result code indicates that the target was unrecognized. In this error condition, the result_description should contain the provided (unrecognized) target.

UNSUPPORTED_MESSAGE

This result code indicates that the message type is not supported by the device. This applies to the optional save/restore messages. In this error condition, the result_description should contain a free-text description of the error.

MALFORMED_MESSAGE

This result code indicates that the message was malformed or otherwise unrecognized. In this error condition, the reply message header should reflect as much of the original message as possible. The result description should contain the malformed message with any necessary modification such that the reply is properly formed.

INTERNAL_ERROR

This result code indicates that the device or proxy experienced an internal error while processing the original message. In this error condition, the result_description should contain a free-text description of the error, if possible.

ACCESS_DENIED

This result code indicates that the GEMS user does not have appropriate access to invoke the action defined in the original message. This commonly is the result of a status-only client attempting to configure the device using either a SetMessage or LoadConfigMessage. In this error condition, the result_description should contain a free-text description of the error.

CONFLICTING_PARAM ETERS

This result code indicates that the original message defined a set of parameters that conflicted with one another. For example, a device might have different ranges depending on the mode specified. In this error condition, the result_description should identify the conflicting parameters.

COMMUNICATION_ERROR

This result code indicates that a communication error occurred. This commonly occurs as a result of network socket errors, serial bus errors, or other transport mechanism errors. In this error condition, the result_description should contain a description of the error, if possible.

OTHER

This result code indicates that an error occurred not already defined in one of the other possible result codes. In this error condition, the result _description should contain a description of the error. If necessary, other non-standard error codings may be used within the result_description. However, it should be noted that these types of codings are not interoperable with other implementation and will be treated as free-text.

6.3.3 Controlling and Monitoring Devices

This section describes the process of controlling and monitoring GEMS devices.

6.3.3.1 Setting Configurations

To control a GEMS Device, the user sends a set of parameters. The parameters each have a name, type, and value as described in Section 6.3.1, Parameters. These parameters are applied in a transactional manner to the device. Validation checks are performed prior to changing the configuration of the device itself.

If these checks fail (e.g., values out of range), the transaction is cancelled. A description of the error is sent back to the GEMS user in the response message. If all values validate, the new parameter settings are applied to the device.

The following diagram depicts this sequence.

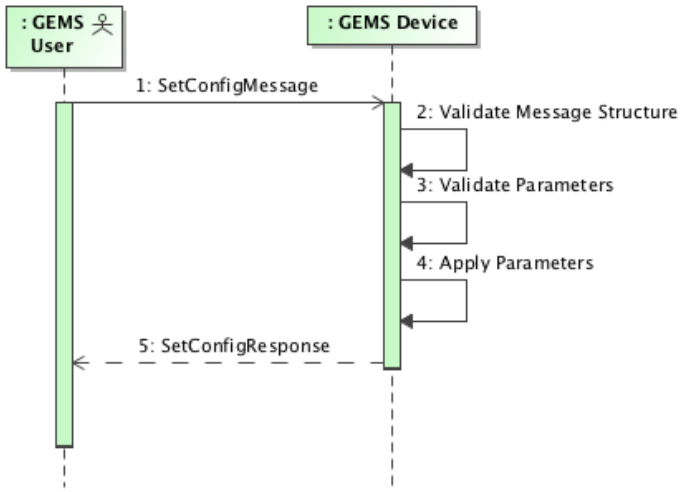


Figure 6.13 Set Configuration Sequence

Errors found in steps 2 or 3 are immediately returned and no changes to the device configuration are made. In the event that an error occurs during step 4, the GEMS device attempts to return to the previous configuration if possible.

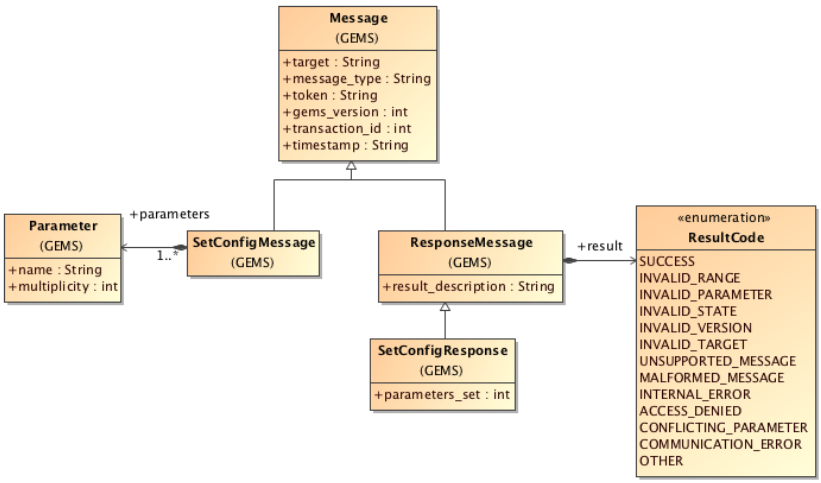


Figure 6.14 Set Configuration Message Classes

SetConfigMessage

The *SetConfigMessage* contains the list of parameters to set. These parameters are applied to the device as a single transaction.

SetConfigResponse

The *SetConfigResponse* message indicates the result of the *SetConfigMessage*. This message has 1 parameter.

parameters_set

The *parameters _set* field indicates the number of parameters affected by the *SetConfigMessage*. This value provides the GEMS User feedback on the number of values actually modified.

For example, a SetConfigMessage containing 20 parameter values might only change five parameters. From the perspective of the GEMS device, this is a valid request. However, the GEMS User might have expected to change 20 parameters.

6.3.3.2 Retrieving Configurations & Status

To obtain the current configuration of a device or monitor the runtime status of a device, the GEMS user sends a *GetConfigMessage*. The message can optionally contain the list of parameters desired. If specific parameters are specified, only those parameters are returned to the GEMS user. If no parameters are specified, then all device parameters are returned.

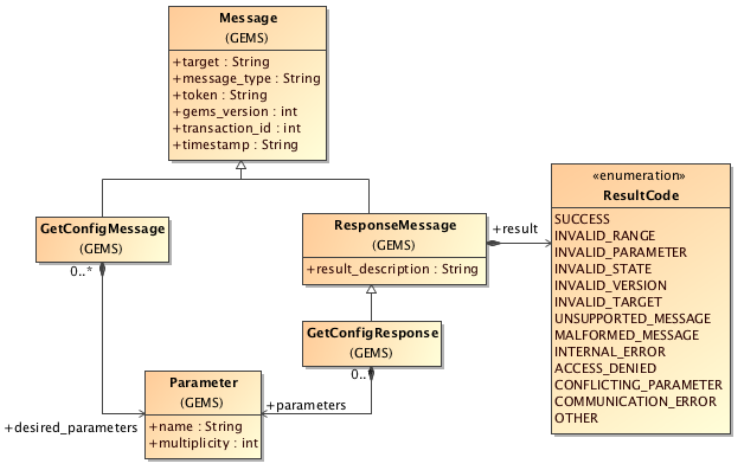


Figure 6.15 GetConfigMessage classes

GetConfigMessage

The GetConfigMessage requests the current configuration from the GEMS device. The message can optionally contain a list of desired parameters.

GetConfigResponse

The GetConfigResponse message contains the device parameters requested.

6.3.3.3 Storing Configurations

To store a configuration, the GEMS User sends a SaveConfigMessage to the GEMS device. This message contains the desired name of the configuration. All device parameters are saved to this configuration. It is expected that multiple device configurations can be contained in a single named configuration. While the format of the persisted configuration is at the vendor's discretion, it is recommended that it use a standard PSM. For example, if ASCII files are used to store configuration information, an appropriate PSM such as XML should be used.

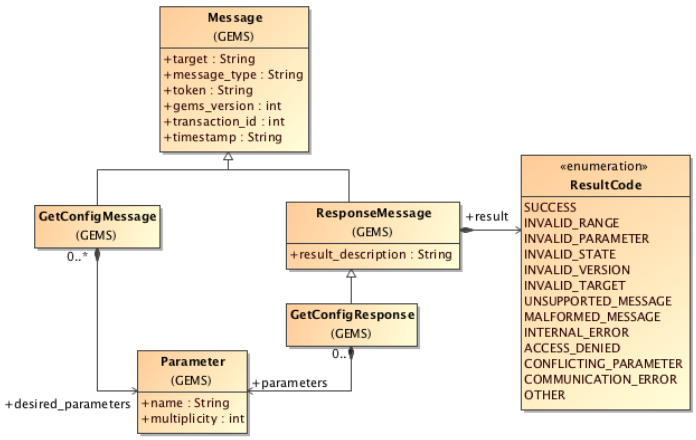


Figure 6.16 Save Config Message Classes

SaveConfigMessage

The SaveConfigMessage contains the name of the configuration to save. The configuration name is a string containing a sequence of letters (A-Z) and/or numbers. Spaces are not allowed as they are not fully supported on all operating systems. The configuration name is case sensitive.

SaveConfig Response

The SaveConfigResponse contains the number of parameters saved if the save was successful. In the case of an error, the ResultCode indicates the type of error.

parameters_saved

The parameters_saved field indicates the number of parameters saved to the named configuration. This value provides the GEMS User feedback on the number of values actually saved. This value can be compared to a later restore configuration request as a means of ensuring expected behavior.

6.3.3.4 Restoring Configurations

To restore a named configuration to a GEMS device, the GEMS user sends a LoadConfigMessage. This message identifies the name of the configuration to load. The following diagram depicts the UML structure for these messages.

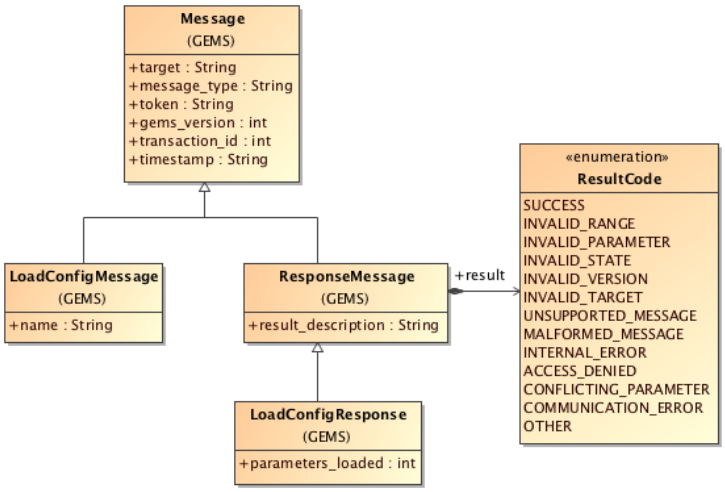


Figure 6.17 Load Config Message Classes

It is not required that the configuration contain a full set of parameter values.

LoadConfigMessage

The LoadConfigMes sage identifies the name of the configuration to load.

LoadConfigResponse

The LoadConfigResponse contains the number of parameters loaded. If an error occurred, it also indicates whether or not the device was left in a valid state.

parameters_loaded

The parameters_loaded field indicates the number of parameters affected by the LoadConfigMessage. This value provides the GEMS User feedback on the number of values actually modified. For example, a configuration containing 20 parameter values might only change five parameters. From the perspective of the GEMS device, this is a valid request. However, the GEMS User might have expected to change 20 parameters.

6.3.3.5 Retrieving Available Configurations

In order to restore a named configuration to a GEMS device, the GEMS user needs to be provided a mechanism for retrieving all the configurations available. This message identifies the list of configuration names available to load. The following diagram depicts the UML structure for these messages.

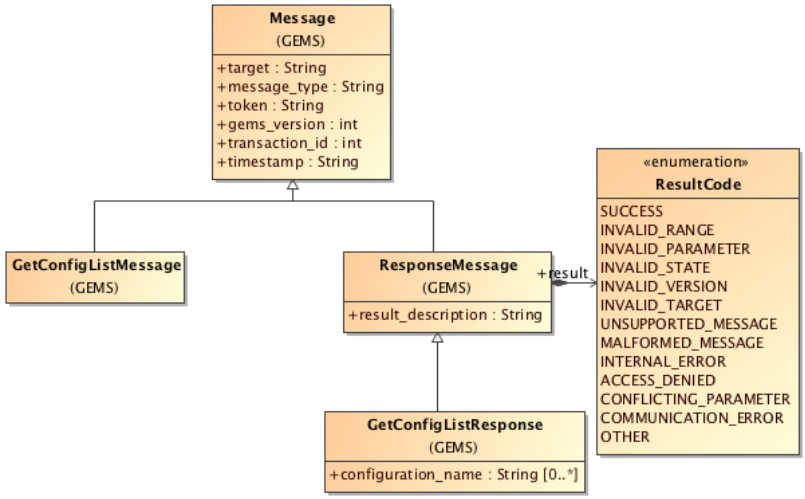


Figure 6.18 Get Config List Message Classes

GetConfigListMessage

The GetConfigListMessage does not contain any additional information above and beyond the base message class.

GetConfigListResponse

configuration_name

The configuration_name field contains all of the available configuration names for the GEMS device. The format of the names is dictated by the device developer.

6.3.3.6 Ping Message and Response

The ping message provides a simple mechanism for determining if the GEMS device is still responding to messages. The use of this message is optional. The response to the message is a Ping Response.

6.4 Directives

Directives allow the GEMS user to invoke a scoped action on the GEMS device. These actions typically involve the purpose of the device rather than the configuration. For example, a device that formats space vehicle commands might have a send _vehicle _command directive. GEMS directives can have a list of parameters (or arguments) and can return values in the response.

The supported directives and associated arguments are defined by the vendor as part of the device definition. The format of this definition is based on the PSM used.

The following diagram shows the sequence of directive messages.

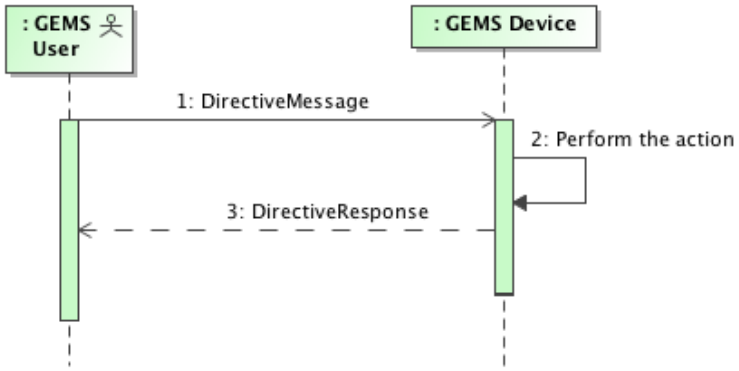


Figure 6.19 Directive Message Sequence

The DirectiveMessage contains the name of the directive and the list of arguments. The response contains the name of the directive and a list of return values. If an error occurs, the ResultCode indicates the reason for the error.

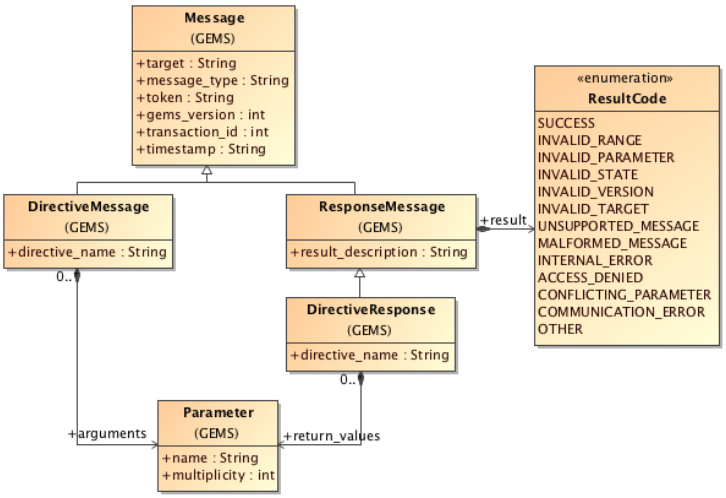


Figure 6.20 Directive Message Classes

6.4.1 DirectiveMessage

The DirectiveMessage contains the name of the directive and a list of arguments.

6.4.2 DirectiveResponse

The DirectiveResponse contains the name of the directive and a list of return values. The name is provided to allow the GEMS user to correlate the response with the original directive.

6.5 Asynchronous Status Messages

Asynchronous status messages allow users to obtain parameter values through a push mechanism. These are normally provided either periodically, or on change. The messages are structured as ResponseMessages with a result_description and a ResultCode. The purpose of this structure is to allow the GEMS Device to communicate any non-nominal conditions to the GEMS user in the event that the periodic status update fails. It also allows GEMS user software to handle the message in a similar fashion to normal response messages.

The following diagram shows the sequence of an AsyncStatusMessage. The configuration of the message occurs using the SetConfigMessage. Then the GEMS device enters a polling loop in which it generates AsyncStatusMessages at the configured rate.

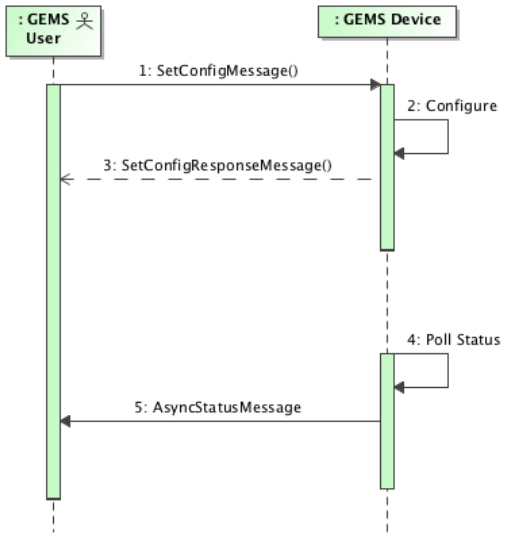


Figure 6.21 Asynchronous Status Message

The AsyncStatusMessage contains a list of parameters. The list of parameters may be all or a subset of the parameters available on the GEMS Device. The structure of the message is similar to the GetConfigResponse. If an error occurred during the asynchronous status polling, it is reflected in the ResultCode and/or the result_description.

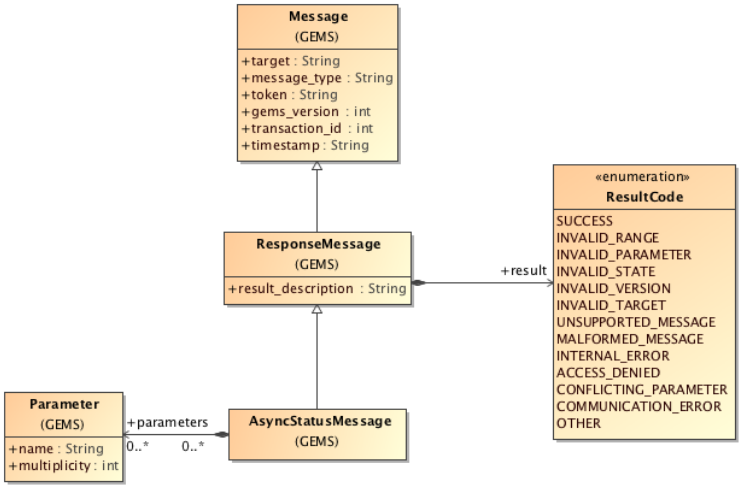


Figure 6.22 AsyncStatusMessage Class

6.6 GEMS Security

The GEMS PIM defines only basic security concepts. GEMS Users connecting to a GEMS device are provided a token to use for all future interactions until a Disconnect message is sent or the connection is in some way broken (e.g., the socket is disconnected). In the basic GEMS model, the content of the token is left undefined. Implementers of GEMS devices are encouraged to use the token to identify the client and the connection type. The encoding of this information varies between PSMs and between vendors.

6.6.1 GEMS Authentication

Optionally, GEMS devices may use a lightweight authentication scheme that allows GEMS-Users to identify themselves to the server using a user name and password. This information is passed to the server in the token field of the original Connect Request. The GEMS PIM defines only the information to be passed. It is left to the PIM and transport layers to appropriately secure the information. This is often done using transports such as SSL/TLS.

6.6.1.1 Authentication

The GEMS-User sends a GEMS Connect Request message to the server. The connect message contains the username and password encoded as a string in the GEMS token field using the following format:

up:<username>:<password>

The prefix up indicates that a username:password pair follows. The intention is to allow other forms of credentials such as a digital certificate encoded in base64 etc. Only the up form is currently defined.

The following provides an example authentication token with the user john and password John'sSecret#1

up:john:John'sSecret#1

6.6.1.2 Successful Authentication Response

If the username and password successfully authenticate, the a GEMS Connect Response is returned indicating success and providing the token to use for all subsequent transactions. The content of returned token is implementation specific.

6.6.1.3 Failed Authentication Response

If the authentication fails, the server responds with ACCESS_DENIED and the description: "Authentication failed". Note that there is no additional information about why the authentication failed.

6.7 GEMS Internationalization

At this time no consideration is provided for internationalization.

6.8 Standard Devices

In future revisions this section will contain industry standard device definitions. The intent of these definitions is to offer a degree of interoperability across device vendors. This allows GEMS users to swap GEMS devices of like types with minimal impact to application software.

6.9 Device Definitions

A GEMS Device may optionally provide a GEMS Device Definition to clients. The intent of the device definition is to describe the parameters and directives supported by a GEMS Device, including textual descriptions and any constraints, in a machine-readable format. The format, and mechanism for obtaining the information, is PSM specific. A GEMS Device Definition may contain more than one device if a logical grouping of devices exists within the system.

When a device definition is provided by the device vendor, it can be retrieved through various methods, e.g. manually from the vendor via some form of electronic media, file transfer, product package, or programmatically via API request to the ground equipment or software application serving the GEMS interface.

The GEMS Device Definition contains the following information:

A list of one or more devices, each containing:

- Device Target Name
- List of Parameter Get Definitions and/or Parameter Set Definitions
- List of Directive Definitions

The Device Target Name is the target name used in all GEMS messages when referencing the device.

The device definition typically comes in an XML format for the purpose of accurately describing device capabilities, and applies to both the ASCII and XML PSMs as described in sections 7.5 and 8.5, respectively. The device definition may contain additional device capabilities such as:

- Async Status Message support
- Save Configuration Message support
- Restore Configuration Message support

6.9.1 Parameter Definitions

Parameter definitions are used to define an individual parameter on a GEMS device. Parameter Definitions contain the following:

- Parameter Name
- Parameter Type
- Description of the parameter (free text)
- A list of constraints, if any, restricting values of the parameter

Parameter constraints may contain one or more of the following:

- maximum/minimum/fixed length (applies to strings)
- pattern as a regular expression (applies to strings)
- enumerated list of values (applies to strings)
- inclusive maximum/minimum values (applies to numeric and time values)
- exclusive maximum/minimum values (applies to numeric and time values)
- maximum/minimum number of nano seconds (applies to time values)

For parameter arrays (i.e. lists), the multiplicity attribute defines the number of items that the array can contain. For example, a multiplicity of 5 indicates that the array may contain between 0 and 5 elements. A multiplicity of -1 indicates that the array is unbounded.

6.9.2 Parameter Set Definitions

Parameter Set definitions contain the following:

- Name

- Description of the parameter set (free text)
- List of Parameter Definitions as defined in 6.8.1

6.9.3 Directive Definitions

Directive definitions are used to define the directives supported by a GEMS Device. Parameter Definitions are used to define the arguments and return values of the directive.

Each Directive Definition includes the following:

- Directive Name
- Description of the directive (free text)
- A list of arguments (represented as a list of Parameter or Parameter Set Definitions)
- A list of return values (represented as a list of Parameter or Parameter Set Definitions)

This page intentionally left blank.

7 PSM (XML)

7.1 General

XML (Extensible Markup Language) is a W3C initiative that allows information and services to be encoded with meaningful structure and semantics that computers and humans can understand. XML is great for information exchange, and can easily be extended to include user-specified and industry-specified tags.

The GEMS PSM mapping for XML leverages these capabilities to define and transfer GEMS messages. XML schemas provide a concise language for defining messages, device parameters, and directives. These schemas also enable strict validation of the message contents all the way down to range values.

7.2 PIM To PSM Mapping

Most of the GEMS PIM TO PSM Mapping is captured in a single, standard XML schema file: GEMS_base_types.xsd. This file defines the message classes and parameters described in the PIM.

7.2.1 GEMS_base_types.xsd

The PIM to PSM mapping for GEMS is best described by working through the GEMS_base_types.xsd file itself. This file starts out with a standard XML header that defines the namespace and pulls in other standard XML schemas. These values are PSM specific and have no direct mapping to the GEMS PSM.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="Error! Hyperlink reference not valid."
    targetNamespace="http://www.omg.org/spec/gems/20110323/basetypes"
    xmlns="http://www.omg.org/spec/gems/20110323/basetypes"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
```

7.2.1.1 Parameter Types

GEMS parameters map to specific complex types.

String Type

The mapping of the GEMS string type is defined in the following XML Schema.

```
<xsd:complexType name="StringParameter">
    <xsd:simpleContent>
        <xsd:extension base= "xsd:string">
            <xsd:attribute name="ParameterType" fixed="string" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Boolean Type

The mapping of the GEMS boolean type is defined in the following XML Schema.

```
<xsd:complexType name="BooleanParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:boolean">
            <xsd:attribute name="ParameterType" fixed="boolean" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
```

```
</xsd:complexType>
```

Byte Type

The mapping of the GEMS byte type is defined in the following XML Schema.

```
<xsd:complexType name="ByteParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:byte">
            <xsd:attribute name="ParameterType" fixed="byte" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Unsigned Byte Type

The mapping of the GEMS unsigned byte type is defined in the following XML Schema.

```
<xsd:complexType name="UnsignedByteParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:unsignedByte ">
            <xsd:attribute name="ParameterType" fixed="ubyte" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Hex Value

The mapping of the GEMS hex_value type is defined in the following XML Schema. The xsd:hexBinary type requires an even number of hex characters (nibbles), therefore the hex parameter must contain full bytes. Odd bit lengths may be specified by including an even number of hex characters and specifying the correct bit length. For example E0/4 represents 0xE.

```
<xsd:complexType name="HexParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:hexBinary">
            <xsd:attribute name="ParameterType" fixed="hex" type="xsd:string"/>
            <xsd:attribute name="bit_length" type="xsd:int" use="required"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Double Type

The mapping of the GEMS double type is defined in the following XML Schema.

```
<xsd:complexType name="DoubleParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:double">
           <xsd:attribute name="ParameterType" fixed="double" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Long Type

The mapping of the GEMS long type is defined in the following XML Schema.

```
<xsd:complexType name="LongParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:long">
            <xsd:attribute name="ParameterType" fixed="long" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
```

```
</xsd:complexType>
```

Unsigned Long Type

The mapping of the GEMS unsigned long type is defined in the following XML Schema.

```
<xsd:complexType name="UnsignedLongParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:unsignedLong">
            <xsd:attribute name="ParameterType" fixed="ulong" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Integer Type

The mapping of the GEMS integer type is defined in the following XML Schema.

```
<xsd:complexType name="IntParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:int">
            <xsd:attribute name="ParameterType" fixed="int" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Unsigned Integer Type

The mapping of the GEMS unsigned integer type is defined in the following XML Schema.

```
<xsd:complexType name="UnsignedIntParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:unsignedInt">
            <xsd:attribute name="ParameterType" fixed="uint" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Short Integer Type

The mapping of the GEMS short integer type is defined in the following XML Schema.

```
<xsd:complexType name=" ShortParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:short ">
            <xsd:attribute name="ParameterType" fixed="short" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Unsigned Short Integer Type

The mapping of the GEMS unsigned short integer type is defined in the following XML Schema.

```
<xsd:complexType name="UnsignedShortParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:unsignedShort">
            <xsd:attribute name="ParameterType" fixed="ushort" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Time Type

The mapping of the GEMS time type is defined in the following XML Schema.

```
<xsd:complexType name="TimeParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:anySimpleType">
            <xsd:attribute name="ParameterType" fixed="time" type="xsd:string"/>
            <xsd:attribute name="seconds" type="xsd:int" use="required"/>
            <xsd:attribute name="nanoseconds" type="xsd:int" use="optional"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Utime Type

The mapping of the GEMS utime type is defined in the following XML Schema.

```
<xsd:complexType name="UtimeParameter">
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute name="ParameterType" fixed="utime" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

The UTC time string in ISO 8601 ordinal date format as follows:

YYYY-DDDThh:mm:ss[.nnnnnnnnn]Z

where: YYYY - four digits of year

DDD – day of the year (001 through 365)

T - the character "T" denotes beginning of the Time field

hh - hours (00 through 23)

mm - minutes (00 through 59)

ss - seconds (00 through 59)

[.nnnnnnnnn] - optional 1-9 digit fractional seconds in nanoseconds resolution

Z - the character "Z" denotes Zulu time which represents time zone UTC+00:00, equivalent to Greenwich Mean Time (GMT).

For example, "2009-273T09:14:50.02Z" represents 9:14:50.02 AM GMT on 30 September 2009. The fractional portion of the seconds in this example represents 20 milliseconds.

7.2.1.2 Parameters

GEMS Parameters are named values with a specific type. GEMS-XML maps this to a Parameter element that contains a single type element. The choice schema element constrains the parameter types to the available GEMS types. The multiplicity attribute is used only if the parameter is an array. The multiplicity then specifies the number of items. If no multiplicity attribute is specified, the parameter is a scalar.

```
<xsd:complexType name="Parameter">
    <xsd:sequence>
        <xsd:choice maxOccurs="unbounded" minOccurs="0"> <xsd:element name="string"
         type="StringParameter"/>
            <xsd:element name="boolean" type="BooleanParameter"/>
            <xsd:element name="byte" type="ByteParameter"/>
            <xsd:element name="ubyte" type="UnsignedByteParameter"/>
```

```
            <xsd:element name="hex_value" type="HexParameter"/>
            <xsd:element name="double" type="DoubleParameter"/>
            <xsd:element name="long" type="LongParameter"/>
            <xsd:element name="ulong" type="UnsignedLongParameter"/>
            <xsd:element name="int" type="IntParameter"/>
            <xsd:element name="uint" type="UnsignedIntParameter" />
            <xsd:element name="short" type="ShortParameter"/>
            <xsd:element name="ushort" type="UnsignedShortParameter"/>
            <xsd:element name="time" type="TimeParameter"/>
            <xsd:element name="utime" type="UtimeParameter"/>
        </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="type" type="xsd:string" use="optional"/>
    <xsd:attribute name="multiplicity" type="xsd:int" use="optional"/>
</xsd:complexType>
```

7.2.1.3 Parameter Sets

Parameter sets contain a heterogeneous grouping of parameters modeled after the Composite Design Pattern. The GEMS-XML mapping represents ParameterSets as a sequence of Parameters and/or ParameterSets.

If the multiplicity is set, indicating an array of ParameterSets, then the contained ParameterSet elements are entries in the array. The name and multiplicity attributes (if specified) are ignored. This is to ensure a consistent mapping with the equivalent GEMS-ASCII message.

```
<xsd:complexType name="ParameterSet">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="Parameter" type="Parameter" minOccurs="0"
         maxOccurs="unbounded"/>
        <xsd:element name="ParameterSet" type="ParameterSet" minOccurs="0"
         maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attribute name="name" type="xsd:string" use="optional"/>
    <xsd:attribute name="type" type="xsd:string" use="optional"/>
    <xsd:attribute name="multiplicity" type="xsd:int" use="optional"/>
</xsd:complexType>
```

7.2.1.4 Message

The GEMS-XML mapping for the GEMS message class is shown in the XML schema below. The Message element contains attributes for the version, token, target, transaction _id, and timestamp specified in UTC. The timestamp string format follows the format specified for the time type in the ASCII PSM which is "seconds.nanoseconds." See Table 8.1.

```
<xsd:complexType name="MessageType">
    <xsd:attribute name="gems_version" type="xsd:decimal" use="required"/>
    <xsd:attribute name="token" type="xsd:string" use="optional"/>
    <xsd:attribute name="target" type="xsd: string" use="optional "/>
    <xsd:attribute name="transaction_id" type="xsd:long" use="optional"/>
    <xsd:attribute name="timestamp" type="xsd:string" use="optional"/>
</xsd:complexType>
```

7.2.1.5 MessageSequence

The XML mapping for the GEMS MessageSequence is as follows:

```
<xsd:complexType name="MessageSequenceType">
    <xsd:complexContent>
        <xsd:restriction base="MessageType">
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0" name="SetConfigMessage"
                 type="SetConf igMessageType"/>
                <xsd:element maxOccurs="unbounded" minOccurs="0" name="GetConfigMessage"
                 type="GetConf igMessageType"/>
                <xsd:element maxOccurs="unbounded" minOccurs="0" name="LoadConfigMessage"
```

```
                type="LoadConfigMessageType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="SaveConfigMessage"
                type="SaveConf igMessageType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="SetConfigResponse"
                type="SetConf igResponseType" />
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="LoadConfigResponse"
                type="LoadConf igResponseType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="SaveConfigResponse"
                type="SaveConf igResponseType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="DirectiveMessage"
                type="DirectiveMessageType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="UnknownResponse"
                type="ResponseMessageType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="PingMessage"
                type="PingMessageType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="PingResponse"
                type="PingResponseType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="GetConfigResponse"
                type="GetConfigResponseType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0”
                name="GetConfigListMessage" type="GetConfigListMessageType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0"
                name="GetConfigListResponse" type="GetConfigListResponseType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0"
                name="GetConfigListResponse" type="GetConfigListResponseType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0"
                name="ConnectionRequestMessage" type="ConnectionRequestMessageType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0"
                name="ConnectionRequestResponse" type="ConnectionRequestResponseType"/>
               <xsd:element maxOccurs="unbounded" minOccurs="0" name="DisconnectMessage"
                type="DisconnectMessageType"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="MessageSequence" type="MessageSequenceType"/>
```

7.2.1.6 SetConfigMessage And GetConfigMessage

The XML mapping for GEMS uses the same schema definition for both SetConfigMessage and GetConfigMessage mappings.

```
<!-- Define a SetGetConfigMessageType -->
<xsd:complexType name="SetGetConfigMessageType">
    <xsd:complexContent>
        <xsd:restriction base="MessageType">
            <xsd:sequence>
                <xsd:element name="Parameter" type="Parameter" minOccurs="0"
                 maxOccurs="unbounded"/>
                <!-- 0 or more Parameters -->
                <xsd:element name="ParameterSet" type="ParameterSet" minOccurs="0"
                 maxOccurs="unbounded"/>
                <!-- 0 or more ParameterSets -->
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<!-- Define a SetConfigMessage -->
<xsd:element name="SetConfigMessage" type="SetGetConfigMessageType"/>
<!-- Define a GetConfigMessage -->
<xsd:element name="GetConfigMessage" type="SetGetConfigMessageType"/>
```

7.2.1.7 LoadConfigMessage And SaveConfigMessage

The XML mapping for GEMS uses the same schema definition for both SetConfigMessage and GetConfigMessage mappings.

```
<!-- Define a LoadSaveConfigMessageType -->
<xsd:complexType name="LoadSaveConfigMessageType">
    <xsd:complexContent>
        <xsd:restriction base="MessageType">
            <xsd:sequence>
               <xsd:element name="name" type="StringParameter"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<!-- Define a LoadConfigMessage -->
<xsd:element name="LoadConfigMessage" type="LoadSaveConfigMessageType"/>
<!-- Define a SaveConfigMessage -->
<xsd:element name="SaveConfigMessage" type="LoadSaveConfigMessageType"/>
```

7.2.1.8 GetConfigListMessage

The XML mapping for a GEMS GetConfigListMessage is as follows:

```
<xsd:complexType name="GetConfigListMessageType">
    <xsd:complexContent>
        <xsd:extension base="MessageType">
           </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GetConfigListMessage" type="GetConfigListMessageType"/>
```

7.2.1.9 ConnectionType

GEMS ConnectionType values are passed with the connection request and indicate the type of connection desired.

```
<xsd:simpleType name="ConnectionType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="CONTROL_ONLY"/>
        <xsd:enumeration value="STATUS_ONLY"/>
        <xsd:enumeration value="CONTROL_AND_STATUS"/>
    </xsd:restriction>
</xsd:simpleType>
```

7.2.1.10 ConnectionRequestMessage

The XML mapping for the GEMS ConnectionRequestMessage is as follows:

```
<xsd:complexType name="ConnectionRequestMessageType">
    <xsd:complexContent>
        <xsd:extension base="MessageType">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" name="type"
                 type="ConnectionType"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="ConnectionRequestMessage" type="ConnectionRequestMessageType"/>
```

7.2.1.11 DisconnectReason

GEMS DisconnectReason values are passed with the disconnect message and indicate the reason for disconnecting.

```
<xsd:simpleType name="DisconnectReason">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="NORMAL_TERMINATION"/>
        <xsd:enumeration value="CONTROL_LOST"/>
        <xsd:enumeration value="SERVICE_TERMINATED"/>
```

```
        <xsd:enumeration value="OTHER"/>
    </xsd:restriction>
</xsd:simpleType>
```

7.2.1.12 DisconnectMessage

The GEMS-XML mapping for the GEMS DisconnectMessage is as follows:

```
<xsd:complexType name="DisconnectMessageType">
    <xsd:complexContent>
        <xsd:extension base="MessageType">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" name="reason"
                 type="DisconnectReason" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="DisconnectMessage" type="DisconnectMessageType"/>
```

7.2.1.13 ResultCode

The GEMS-XML mapping for the ResultCode utilizes an XML enumeration.

```
<xsd:simpleType name="ResultCode">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="SUCCESS"/>
        <xsd:enumeration value="INVALID_RANGE"/>
        <xsd:enumeration value="INVALID_PARAMETER"/>
        <xsd:enumeration value="INVALID_STATE"/>
        <xsd:enumeration value="INVALID_VERSION"/>
        <xsd:enumeration value = "INVALID_TARGET"/>
        <xsd:enumeration value="CONFLICTING_PARAMETER"/>
        <xsd:enumeration value="UNSUPPORTED_MESSAGE"/>
        <xsd:enumeration value="MALFORMED_MESSAGE"/>
        <xsd:enumeration value="COMMUNICATION_ERROR"/>
        <xsd:enumeration value="INTERNAL_ERROR"/>
        <xsd:enumeration value="ACCESS_DENIED"/>
        <xsd:enumeration value="OTHER"/>
    </xsd:restriction>
</xsd:simpleType>
```

7.2.1.14 Response

The GEMS-XML mapping for the base ResponseMessage is as follows:

```
<xsd:complexType name="ResponseMessageType">
    <xsd:complexContent>
        <xsd:restriction base="MessageType">
            <xsd:sequence>
                <xsd:element minOccurs="1" maxOccurs="1" name="Result" type="ResultCode"/>
                <xsd:element minOccurs="0" maxOccurs="1" name="description"
                 type="xsd:string"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

The GEMS-XML mapping for a generic error response message is as follows:

```
<xsd:element name="UnknownResponse" type="ResponseMessageType"/>
```

7.2.1.15 LoadConfigResponse

The GEMS-XML mapping for the LoadConfigResponse is as follows:

```
<xsd:complexType name="LoadConfigResponseType">
    <xsd:complexContent>
        <xsd:extension base="ResponseMessageType">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" name="parameters_loaded"
                 type="xsd:int"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="LoadConfigResponse" type="LoadConfigResponseType"/>
```

7.2.1.16 Save Config Response

The GEMS-XML mapping for the SaveConfigResponse message is as follows:

```
<xsd:complexType name="SaveConfigResponseType">
    <xsd:complexContent>
        <xsd:extension base="ResponseMessageType">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" name="parameters_saved"
                 type="xsd:int"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="SaveConfigResponse" type="SaveConfigResponseType"/>
```

7.2.1.17 Get Config List Response

The GEMS-XML mapping for the GetConfigListResponse message is as follows:

```
<xsd:complexType name="GetConfigListResponseType">
    <xsd:complexContent>
        <xsd:extension base="ResponseMessageType">
            <xsd:sequence>
                <xsd:element minOccurs="0" maxOccurs="unbounded" name="ConfigurationName"
                 type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GetConfigListResponse" type="GetConfigListResponseType"/>
```

7.2.1.18 SetConfigResponse

The GEMS-XML mapping for the SetConfigResponse message is as follows:

```
<xsd:complexType name="SetConfigResponseType">
    <xsd:complexContent>
        <xsd:extension base="ResponseMessageType">
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" name="parameters_set"
                 type="xsd:int"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="SetConfigResponse" type="SetConfigResponseType"/>
```

7.2.1.19 GetConfigResponse

The GEMS-XML mapping for the GetConfigResponse message is as follows:

```
<!-- Define a GetConfigResponseType -->
<xsd:complexType name="GetConfigResponseType">
    <xsd:extension base=”ResponseMessageType”>
        <xsd:choice minOccurs=”0” maxSelector=”unbounded”>
            <xsd:element name="Parameter" type="Parameter" minOccurs="0"
             maxOccurs="unbounded"/> <!-- 0 or more Parameters -->
            <xsd:element name="ParameterSet" type="ParameterSet" minOccurs="0"
             maxOccurs="unbounded"/> <!-- 0 or more ParameterSet -->
        </xsd:choice>
    </xsd:extension>
</xsd:complexType>
<!-- Define a GetConfigResponse -->
<xsd:element name="GetConfigResponse" type="GetConfigResponseType"/>
```

7.2.1.20 ConnectionRequestResponse

The GEMS-XML mapping for the ConnectionRequestResponse message is as follows:

```
<!-- Define a ConnectionRequestResponse -->
<xsd:complexType name="ConnectionRequestResponseType">
    <xsd:complexContent>
        <xsd:extension base="ResponseMessageType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="ConnectionRequestResponse" type="ConnectionRequestResponseType"/>
```

7.2.1.21 Directive Arguments

Directive arguments are represented in GEMS-XML as a sequence of Parameters.

```
<!--Define an ArgumentsType to be used by GEMS Directives -->
<xsd:complexType name="ArgumentsType">
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="Parameter"
        type="Parameter"/>
    </xsd: sequence>
    <xsd:attribute name="directive_name" type="xsd:string" use="optional"/>
</xsd:complexType>
```

7.2.1.22 Directive Return Values

Directive return values are represented in GEMS-XML as a sequence of parameters.

```
<!– Define a ReturnValuesType to be used by GEMS Directive Responses -->
<xsd:complexType name="ReturnValuesType">
    <xsd:choice minOccurs=”0” maxOccurs=”unbounded”>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="Parameter"
         type="Parameter"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="ParameterSet"
         type="ParameterSet"/>
    </xsd:choice>
</xsd:complexType>
```

7.2.1.23 Directive Messages

The DirectiveMessage is the container for a directive. It is standard for all devices.

```
<!-- Define a DirectiveMessageType -->
<xsd:complexType name="DirectiveMessageType">
    <xsd:complexContent>
        <xsd:restriction base= "MessageType">
            <xsd:sequence>
                <xsd:element minOccurs="1" maxOccurs="1" name="directive_name"
```

```
                type="xsd:string"/>
                <xsd:element minOccurs="0" maxOccurs="1" name="arguments" type="ArgumentsType" />
            </xsd: sequence>
        </xsd:restriction>
    </xsd : complexContent>
</xsd : complexType>
<!-- Define a directive message -->
<xsd: element name="DirectiveMessage" type="DirectiveMessageType"/>
```

7.2.1.24 Directive Response

The DirectiveResponse is the contain for a directive response. It is standard for all devices.

```
<!-- Define a DirectiveResponseType -->
<xsd : complexType name=" DirectiveResponseType">
    <xsd: complexContent>
        <xsd: extension base="ResponseMessageType">
        <xsd: sequence>
            <xsd: element minOccurs="1" maxOccurs="1" name="directive _name"
             type="xsd: string"/>
             <xsd: element minOccurs="0" maxOccurs="1" name="return _values"
             type=" ReturnValuesType" />
       </xsd: sequence>
</xsd : extension>
</xsd : complexContent>
</xsd: complexType>
<!-- Define a directive response -->
<xsd:element name="DirectiveResponse" type="DirectiveResponseType"/>
```

7.2.1.25 Ping Message

The GEMS XML-Mapping for the PingMessage is as follows:

```
 <xsd:complexType name="PingMessageType">
    <xsd:complexContent>
        <xsd:extension base="MessageType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="PingMessage" type="PingMessageType"/>
```

7.2.1.26 Ping Response Message

The GEMS XML-Mapping for the PingResponseMessage is as follows:

```
 <!—Define a PingResponse -->
<xsd:complexType name="PingResponseType">
    <xsd:complexContent>
        <xsd:extension base="ResponseMessageType"/>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="PingResponse" type="PingResponseType"/>
```

7.2.1.27 AsyncStatusMessage

The GEMS-XML mapping for the AsyncStatusMessage message is as follows:

```
<!-- Define an AsyncStatusMessageType -->
<xsd:complexType name="AsyncStatusMessageType">
    <xsd:complexContent>
        <xsd:restriction base="MessageType">
            <xsd:sequence>
                <xsd:element name="Parameter" type="Parameter" minOccurs="0" maxOccurs="unbounded"/>
                <!-- 0 or more Parameters -->
```

```
                <xsd:element name="ParameterSet" type="ParameterSet" minOccurs="0"
maxOccurs="unbounded"/>
                <!-- 0 or more ParameterSets -->
              </xsd:sequence>
            </xsd:restriction>
        </xsd:complexContent>
</xsd:complexType>
<!-- Define an AsyncStatusMessage -->
<xsd:element name="AsyncStatusMessage" type="AsyncStatusMessageType"/>
```

7.2.1.28 GEMS Scheme End

```
</xsd: schema>
```

7.3 XML Examples

Examples of the GEMS XML messages are available on the OMG website at http://www.omg.org/space. Navigate to the GEMS specific page and look for GEMS Examples.

7.4 TCP/IP Message Structure

Transfer of GEMS-XML messages across a network transport is done by writing the XML directly to the transport layer. If the transport layer includes the message length, the XML text is all that is required. This is the case with several commercially available message oriented middleware solutions. However, if the message length is not included, as is the case with normal TCP/IP sockets, then a standard HTTP header and the standard XML encoding pseudo-attribute is added to the message. This is done to simplify processing of the message on the other side.

The HTTP header format is as follows. Note that all lines must end in a carriage-control and line-feed. A blank line with a carriage-control and line feed is used to separate the HTTP header from the body.

```
POST /target HTTP/1.1
User-Agent: OMG-GEMS
Content-Type: text/xml
Content-Length: 646
```

Response messages use the standard HTTP response header.

```
HTTP/1.1 200 OK
User-Agent: OMG-GEMS
Content-Type: text/xml
Content-Length: 616

The Content-Length specifies the number of bytes to read after the HTML header and includes any
formatting characters (spaces, end-of-line etc.)
Example:

POST /target HTTP/1.1
User-Agent: OMG-GEMS
Content-Type: text/xml
Content-Length: 346

<?xml version="1.0" ?>
<gems:PingMessage gems_version="1.2"
 target="System/Device1"
 timestamp="1410540242.29"
 token="" transaction_id="1"
 xmlns:gems=http://www.omg.org/spec/gems/20110323/basetypes
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=http://www.omg.org/spec/gems/20110323/basetypes GEMS_base_types.xsd
/>
```

When using client-side HTTP libraries, the GEMS connection requires a persistent socket connection. If the connection to the TCP socket is closed at some point during a connection, the GEMS device should automatically disconnect the GEMS User.

7.5 GEMS Device Definitions

The GEMS-XML mapping of a Device Definition takes the form of an XML file. The schema for this file is defined in the normative GEMS_Device.xsd file, included as part of the GEMS-XML PSM. To avoid redundancy between documents, comments have been added in the schema file, describing the various portions of the schema.

The XML Device Definition file is either distributed manually by the device vendor as part of the system delivery, or automatically using an HTTP GET request. It is expected that the HTTP GET request is handled using standard HTTP protocols on port 80. However, alternate port number may be specified by the vendor.

If provided automatically using an HTTP GET request, the URL for the device definition file shall be as follows:

```
http://<hostname>/omg/gems/DeviceDefinitions.xml
```

where <hostname> is either the hostname or a valid IP address for the web server on the device.

Refer to section 6.9 for details on device definitions.

7.5.1 Example Device Definition File

Examples of the GEMS Device Definition File are available on the OMG website at http://www.omg.org/space. Navigate to the GEMS specific page and look for GEMS Examples.

This page intentionally left blank.

8 PSM (ASCII)

8.1 General

The ASCII PSM defines a simple ASCII message protocol usable across a variety of transport mechanisms, including networks, serial lines and internal data buses such as PCI. The message structure is human-readable and easy to process.

8.2 PIM to PSM Mapping

For the GEMS PIM TO PSM ASCII Mapping each supported message type is captured in a single table describing the message format. The message table defines the order of the fields within the message, the field tags placed within the message, and the field's original range of values (ROVs).

All messages consist of a standard message header, followed by data in a message body consisting of fields uniquely associated with each type of message. Each message is terminated using a standard message trailer and is constructed from ASCII character fields.

8.2.1 Parameters

GEMS-ASCII parameters are represented within messages using a name, type, value triad as follows:

```
parameter_name:type=value
```

Multiplicity is represented using an array-like syntax common to many scripting languages. The values are specified using a comma separated list.

```
parameter_name:type[3]=value1,value2,value3
```

8.2.1.1 Parameter Types

The PSM ASCII Mapping supports all the parameter types defined by the GEMS PIM. The general format in ASCII for parameters is '**parameter_name:parameter_type=parameter_value**.' Following is a table of example ASCII formats for all of the parameter types.

Table 8.1: Parameter Types and Formatting

Parameter Type	Example Format	Comments
string	param:string=abc	
boolean	param:boolean=true	'true' and 'false' case insensitive
byte	param:byte=127	
ubyte	param:ubyte=255	
hex_value	param:hex_value=faf320/22 param:hex_value=0xfaf320/22	The value of the bit length field follows the slash (/). Specify a zero length bit field with a 0/0.
double	param:double=10.12	
long	param:long=-999999999	
ulong	param:ulong=9999999999	

int	param:int=-123	
uint	param:uint=123	
short	param:short=-12	
ushort	param:ushort=12	
time	param:time=1111111111.222000000	Value is seconds.nanoseconds in UTC where nanoseconds are fractional.
utime	param:utime=2009-273T09:14:50.02Z	The UTC time string in ISO 8601 ordinal date format as follows: YYYY-DDDThh:mm:ss[.nnnnnnnnn]Z Where: YYYY - four digits of year DDD - day of the year (001 through 365) T - the character "T" denotes beginning of the Time field hh - hours (00 through 23) mm - minutes (00 through 59) ss - seconds (00-59) [.nnnnnnnnn] - optional 1-9 digit fractional seconds in nanoseconds resolution Z - the character “Z” denotes Zulu time which represents time zone UTC+00:00, equivalent to Greenwich Mean Time (GMT)

8.2.1.2 Parameter Sets

In the PSM ASCII Mapping parameter sets may be defined by the additional use of semi-colons. For example a simple parameter set containing mixed parameters would be as follows:

```
mixed_set:set_type=param_1:int=1;param_2:boolean=true;param_3:string=a short string;
```

To represent an array of parameter sets use commas to separate the parameter sets.

```
setlist:set_type[2]=A:hex_value=FA/7;B:string[2]=text1,text2;,A:hex_value=2F/6;B:string[1]=text2;
```

Note: The final member of the parameter set is followed by a semi-colon. This allows for easy parsing when dealing with parameter set arrays.

8.2.2 Reserved Words and Special Characters

The following words are reserved and should not be used as parameter or target names: “|GEMS” and “|END|”

The following characters are special but can be escaped using the following escape sequences:

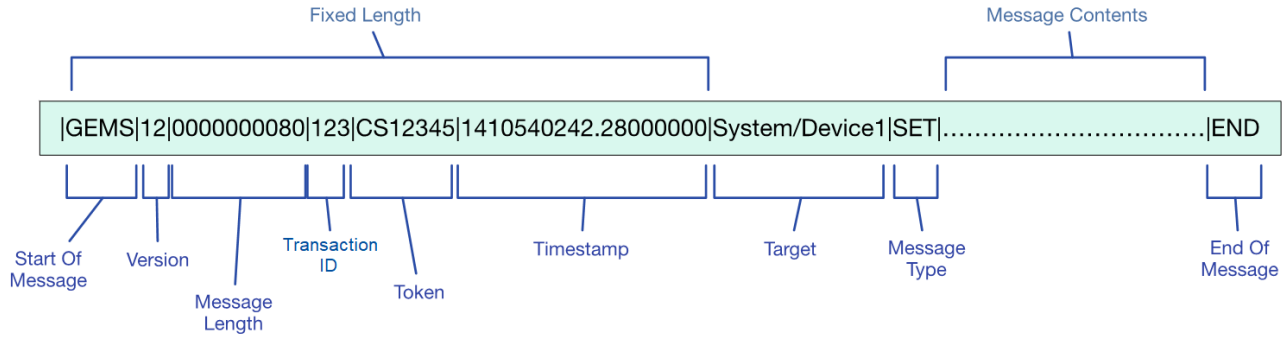
Character	Escape Sequence
&	&a
|	&b
.	&c
:	&d

These escape sequences were chosen for the ASCII PSM to allow for simple parsing. Notionally, implementations first tokenize on a vertical bar '|' character. Then, for messages containing parameters, split the parameter fields on the first equal sign '='. Escaped characters should only show up to the right of the equal sign in string parameters.

8.2.3 Message Header

The message header maps directly to the GEMS base message class. The first three fields are fixed length to simplify processing. After that, all fields are variable length and use the pipe symbol, |, as a delimiter.

The following diagram shows the message header and footer.



The following table defines each field, the expected field length and the range of values.

Table 8.2: Standard Header

Field Names	Length (char)	Range of Values	Comments
Field Delimiter	1	|	Pipe character (ASCII 124)
Start of Message	4	GEMS	Invariant
Field Delimiter	1	|	Pipe character (ASCII 124)
GEMS Version	2	01 – 99	Message Format Version Mapping - For versions 1.2 and greater, ASCII code for version is version number with decimal removed 01 : 1.0 02 : 1.1 12 : 1.2 13 : 1.3 (etc.)

Field Delimiter	1	|	Pipe character (ASCII 124)
Message Length	10	up to 9999999999	Total Length of the message (in bytes), including the start of message, standard header, message body and the end of message.
Field Delimiter	1	|	Pipe character (ASCII 124)
Transaction ID	Variable	0-999,999,999	Client specified transaction ID. The reply will reflect this number back to the client for message correlation.
Field Delimiter	1	|	Pipe character (ASCII 124)
Token	Variable	Alphanumeric	Token provided by the device or proxy in the connect response.
Field Delimiter	1	|	Pipe character (ASCII 124)
Timestamp	19	111111111.222000000	Timestamp of when the message was sent in UTC. Uses the format of the time parameter specified in Table 8.1.
Field Delimiter	1	|	Pipe character (ASCII 124)
Target	Variable	Alphanumeric	The target field identifies the target of the message for the initial request and the source of the message for the reply.
Field Delimiter	1	|	Pipe character (ASCII 124)
Message Type	Variable	Alphanumeric	The Message Type identifies the type of message being sent. These types map directly to the PIM message types and are listed in the following sections.
Field Delimiter	1	|	Pipe character (ASCII 124)
Message Body	Variable	Alphanumeric	Specific to each supported message type. See following sections for definitions and examples.
Field Delimiter	1	|	Pipe character (ASCII 124)
End of Message	3	END	Invariant

The message header for a response contains the same first 7 fields as the message plus an additional 2 fields providing result codes and descriptions.

Table 8.3: Response Message Fields

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	
Field Delimiter	1	|	Pipe character (ASCII 124)

Result Code	Variable	Alphanumeric	One of the following GEMS PIM result codes: SUCCESS INVALID_RANGE INVALID_PARAMETER INVALID_TARGET INVALID_VERSION INVALID_STATE CONFLICTING_PARAMETER UNSUPPORTED_MESSAGE MALFORMED_MESSAGE COMMUNICATION_ERROR INTERNAL_ERROR ACCESS_DENIED OTHER
Field Delimiter	1	|	Pipe character (ASCII 124)
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Field Delimiter	1	|	Pipe character (ASCII 124)
End of Message	3	END	Invariant

The GEMS-ASCII mapping for a generic error response message (i.e. an UnknownResponse message) is simply a standard Response Message per Table 8.3 plus trailer with the Message Type set to UNK-R and Result Code and Result Description set as needed.

8.2.4 Message Trailer

The message trailer ends all request and response messages and is represented as follows. It is always separated from the message body by a '|' character.

Table 8.4: Message Trailer Fields

Field Names	Length (char)	Range of Values	Comments
End of Message	3	END	Invariant

8.2.5 ConnectMessage and Response

The following table describes the ConnectMessage body and response format.

8.2.5.1 ConnectMessage

Table 8.5: Connect Request Message

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = CON
Field Delimiter	1	|	Pipe character (ASCII 124)

Connection Type	Variable	Alphanumeric	Supported Values: CONTROL_ONLY CONTROL_AND_STATUS STATUS_ONLY
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.5.2 ConnectMessage Response

The ConnectMessage Response nothing more than the standard response header and trailer with the message type set to CON-R. See Sections 8.2.3 and 8.2.4.

8.2.6 DisconnectMessage

The following table describes the DisconnectMessage body. There is no corresponding response message.

Table 8.6: Disconnect Message Request

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = DISC
Field Delimiter	1	|	Pipe character (ASCII 124)
Disconnect Reason	Variable	Alphanumeric	Supported Values: NORMAL_TERM INATION
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.7 GetConfig Message and Response

The following tables describe the message body for the GetConfigMessage and its corresponding response.

8.2.7.1 GetConfigMessage

Table 8.7: Get Configuration Request Message

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = GET
Field Delimiter	1	|	Pipe character (ASCII 124)
Number of Parameters	Variable	Numeric	Number of parameters requested. A blank entry indicates the client desires all parameters available.
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter Name 1	Variable	Alphanumeric	Name of first parameter value requested. Only required if Number of Parameters field is greater than 0.

Field Delimiter	1	|	Pipe character (ASCII 124)
...			
Parameter Name n	Variable	Alphanumeric	Name of nth parameter requested. Only required if Number of Parameters field is greater than n -1.
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.7.2 GetConfigMessage Response

Table 8.8: Get Configuration Response Message

Field Names	Length (char)	Range of Values	Comments
Standard Response Header	Variable	<See Table 8.2 >	Message Type = GET-R
Field Delimiter	1	|	Pipe character (ASCII 124)
Number of Parameters Returned	Variable	Numeric	Number of parameters returned
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter Value 1	Variable	<See Table 8.1 >	Value of first parameter requested
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter Value n	Variable	<See Table 8.1 >	Value of nth parameter requested
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.8 SetConfigMessage and Response

The following tables describe message body for the SetConfigMessage and its corresponding response.

8.2.8.1 SetConfigMessage

Table 8.9: Set Configuration Request Message

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = SET
Field Delimiter	1	|	Pipe character (ASCII 124)

Number of Parameters	Variable	Numeric (> 0)	Number of parameters modified
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter Value 1	Variable	<See Table 8.1 >	Value of first parameter
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter Value n	Variable	<See Table 8.1 >	Value of nth parameter
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.8.2 SetConfigMessage Response

Table 8.10: Set Configuration Response Message

Field Names	Length (char)	Range of Values	Comments
Standard Response Header	Variable	<See Table 8.2 >	Message Type = SET-R
Field Delimiter	1	|	Pipe character (ASCII 124)
Number of Parameters Successfully Set	Variable	Numeric (>= 0)	Number of parameters returned
Field Delimiter	1	|	Pipe character (ASCII 124)
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.9 Save/LoadConfigMessage and Response

The following tables describe the message body and response for both the Save and LoadConfigMessages.

8.2.9.1 Save and LoadConfigMessage

Table 8.11: SaveConfig and LoadConfig Messages

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = SAVE or LOAD
Field Delimiter	1	|	Pipe character (ASCII 124)
File Name	Variable	Standard File Naming Conventions	Name of file configuration is saved to or loaded from
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.9.2 Save and LoadConfig Response

Table 8.12: SaveConfig and LoadConfig Response Messages

Field Names	Length (char)	Range of Values	Comments
Standard Response Header	Variable	<See Table 8.2 >	Message Type = SAVE-R or LOAD-R
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameters Successfully Saved or Loaded	Variable	Numeric (>= 0)	Number of parameters successfully saved to or loaded from file.
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.10 GetConfigListMessage and Response

The following tables describe the message body and response for the GetConfigListMessage.

8.2.10.1 GetConfigListMessage

Table 8.13: GetConfigList Request Message

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = GETL
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.10.2 GetConfig ListMessage Response

Table 8.14: GetConfigListMessage Response

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = GETL-R
Field Delimiter	1	|	Pipe character (ASCII 124)
Number of Configuration Names	Variable	Numeric (>= 0)	Number of configuration names available
Configuration Name 1	Variable	Alphanumeric	Name of 1st configuration available
Field Delimiter	1	|	Pipe character (ASCII 124)
Configuration Name n	Variable	Alphanumeric	Name of nth configuration available

Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.11 DirectiveMessage and Response

The following tables describe the message body and response for the DirectiveMessage.

8.2.11.1 DirectiveMessage

Table 8.15: Directive Request Message

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = DIR
Field Delimiter	1	|	Pipe character (ASCII 124)
Directive Name	Variable	Alphanumeric	Name of the directive to execute
Field Delimiter	1	|	Pipe character (ASCII 124)
Number of Parameters	Variable	Numeric (>= 0)	Number of parameters to be passed into directive
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter 1	Variable	<See Table 8.1 >	1st parameter of directive. Only if Number of Parameters field is > 0.
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter n	Variable	<See Table 8.1 >	nth parameter of directive Only if Directive Number of Parameters field is > n -1.
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.11.2 DirectiveMessage Response

Table 8.16: Directive Response Message

Field Names	Length (char)	Range of Values	Comments
Standard Header	Variable	<See Table 8.2 >	Message Type Field = DIR-R
Field Delimiter	1	|	Pipe character (ASCII 124)
Directive Name	Variable	Alphanumeric	Name of the directive

Field Delimiter	1	|	Pipe character (ASCII 124)
Number of Return Values	Variable	Numeric (>= 0)	Number of returned values
Field Delimiter	1	|	Pipe character (ASCII 124)
Return Value 1	Variable	<See Table 8.1 >	1st return parameter. Only if Number of Return Values field is > 0.
Field Delimiter	1	|	Pipe character (ASCII 124)
...			
Return Value n	Variable	<See Table 8.1 >	nth return parameter Only if Number of Return Values field is > n -1
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.2.12 AsyncStatusMessage

Table 8.17: Asynchronous Status Message

Field Names	Length (char)	Range of Values	Comments
Standard Response Header	Variable	<See Table 8.2 >	Message Type = ASYNC
Field Delimiter	1	|	Pipe character (ASCII 124)
Number of Parameters Returned	Variable	Numeric	Number of parameters returned
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter Value 1	Variable	<See Table 8.1 >	Value of first parameter requested
Field Delimiter	1	|	Pipe character (ASCII 124)
Parameter Value n	Variable	<See Table 8.1 >	Value of nth parameter requested
Field Delimiter	1	|	Pipe character (ASCII 124)
Standard Trailer	3	<See Table 8.4 >	

8.3 ASCII Examples

Examples of the GEMS ASCII messages are available on the OMG website at http://www.omg.org/space. Navigate to the GEMS specific page and look for GEMS Examples.

8.4 TCP/IP Message Structure

The GEMS-ASCII messages contain all of the information necessary for transport across both networks and serial data buses. The messages are written directly to a socket or serial port. No additional header information is required.

When using TCP, the GEMS connection requires a persistent socket connection. If the connection to the TCP socket is closed at some point during a connection, the GEMS device should automatically disconnect.

8.5 GEMS Device Definitions

The GEMS-ASCII Device Definitions utilize the GEMS-XML Device Definition file as specified in section 7.4. This file may be distributed manually by the device vendor or by using a locally hosted web server and an HTTP GET message similar to GEMS-XML.

Refer to section 6.9 for details on device definitions.

9 PSM (JSON)

9.1 General

This PSM defines a RESTful message protocol using JSON encoding that can be transferred via HTTP or HTTPS. The message structure is in the body of the HTTP request/response and is encoded in JSON (Content-Type: application/json). JSON Encoding is described here: http://www.ietf.org/rfc/rfc4627.txt, and JSON is supported across programming languages. Libraries for parsing/emitting JSON are listed here: https://www.json.org/json-en.html.

9.2 PIM to PSM Mapping

This section maps both the GEMS Parameters and the GEMS Messages from the GEMS PIM to the JSON/REST PSM.

9.2.1 Parameters

GEMS parameters are represented within messages using JSON encoding. JSON supports numbers, strings, booleans, arrays, and objects (unordered key/value sets that represent mixed-type structures).

9.2.1.1 Parameter Types

The JSON PSM mapping supports all the parameter types defined by the GEMS PIM. The PIM defines explicit types for numbers. Since JSON is not explicit when it comes to numbers, the encoded "type" enables validation. Table 9.1 maps each GEMS parameter type to the respective JSON body format. These same formats are used in all GEMS messages for Parameters and Directives (which can contain GEMS parameters). Note `"param"` in Table 9.1 is the *parameter name*, e.g. `"frameLength"`. Note: An error shall be returned if the `"value"` does not adhere to the JSON Value Type.

Table 9.1 - Parameter Types and Formatting

GEMS Parameter Type	JSON Body (Encoding Format)	JSON Value Type
boolean	`{"param": {"type": "bool", "value": true}}`	**JSON Boolean**
byte	`{"param": {"type": "byte", "value": -127}}`	**JSON Number**
ubyte	`{"param": {"type": "ubyte", "value": 255}}`	**JSON Number**
long	`{ "param": { "type": "long", "value": -9223372036854775807 } }`	**JSON Number** Implementations must treat the "long" value as a 64-bit signed integer and not represent it as a floating-point number which would lose precision.
ulong	`{ "param": { "type": "ulong", "value": 18446744073709551615 } }`	**JSON Number** Implementations must treat the "ulong" value as a 64-bit unsigned integer and not represent it as a floating-point number which would lose precision.
int	`{"param": {"type": "int", "value": -2147483647}}`	**JSON Number**
uint	`{"param": {"type": "uint", "value": 4294967295}}`	**JSON Number**
short	`{"param": {"type": "short", "value": -32767}}`	**JSON Number**
ushort	`{"param": {"type": "ushort", "value": 65535}}`	**JSON Number**
double	`{"param": {"type": "double", "value": 0.12}}`	**JSON Number**
string	`{"param": {"type": "string", "value": "abcdef"}}`	**JSON String**
hex_value	`{ "param": { "type": "hex_value", "value": "FAF320/24" } }`	**JSON String** The value of the bit length field follows the slash (/) and is in decimal. To specify a zero-length bit field, use "0/0".

time	<pre>{
	  "param": {
	  "type": "time",
	  "value": "1111111111111111111.123456789"
	  }
	}</pre>
		JSON String
		Value is seconds.nanoseconds, where ".nanoseconds" are the fractional seconds portion of the time. Represented as a JSON String to support adequate resolution.

utime	<pre>{
	  "param": {
	  "type": "utime",
	  "value": "2009-073T09:14:50.123456789Z"
	  }
	}</pre>
		JSON String
		Value is the UTC time string in ISO 8601 ordinal date format per the "utime" description in Appendix A.6 JSON Schema: GEMS Get/Set Config

Array Parameter	<pre>{
	  "param": {
	  "type": "string[]",
	  "array": [
	  {
	  "type": "string",
	  "value": "abc"
	  },
	  {
	  "type": "string",
	  "value": "def"
	  },
	  {
	  "type": "string",
	  "value": "ghi"
	  }
	  ]
	  }
	}</pre>
		JSON Array, elements are typed according to their individual GEMS type. Multiple parameters are simply multiple Key/Value pairs. This example is an array of strings.

Parameter Set (Structure)	<pre>{
	  "param": {
	  "type": "myStruct",
	  "structure": {
	  "member1": {
	  "type": "string",
	  "value": "abc"
	  },
	  "member2": {
	  "type": "uint",
	  "value": 6
	  },
	  "member3": {
	  "type": "bool",
	  "value": true
	  }
	  }
	  }
	}</pre>
		JSON Structure, bracketed list of structure members. Elements are typed according to their individual GEMS type. The parameter "type" name represents this unique collection of structure members.

Array of Parameter Sets (Array of structures)
```
{
  "param": {
    "type": "myStruct[]",
    "array": [
      {
        "type": "myStruct",
        "structure": {
          "member1": {
            "type": "string",
            "value": "abc"
          },
          "member2": {
            "type": "uint",
            "value": 6
          },
          "member3": {
            "type": "bool",
            "value": true
          }
        }
      },
      {
        "type": "myStruct",
        "structure": {
          "member1": {
            "type": "string",
            "value": "def"
          },
          "member2": {
            "type": "uint",
            "value": 7
          },
          "member3": {
            "type": "bool",
            "value": false
          }
        }
      }
    ]
  }
}
```
 | **JSON array of structures**, is a list of JSON objects, where each object in the array is of a previously defined Parameter Set/structure type. Elements are typed according to their individual GEMS type. |

Array of Parameter Sets with nested array member
```
{
  "myArrayOfStructsParamName": {
    "type": "myHobbit[]",
    "array": [
      {
        "type": "myHobbit",
        "structure": {
          "myHobbMember1": {
            "type": "string",
            "value": "Smeagol"
          },
          "myHobbMember2": {
            "type": "myPrecious[]",
            "array": [
              {
                "type": "myPrecious",
                "structure": {
                  "myPrecMember1": {
                    "type": "string",
                    "value": "Ring, no wait..."
                  },
                  "myPrecMember2": {
                    "type": "hex_value",
                    "value": "FAF320/24"
                  }
                }
              },
              {
                "type": "myPrecious",
                "structure": {
                  "myPrecMember1": {
                    "type": "string",
                    "value": "Rings!!"
                  },
                  "myPrecMember2": {
                    "type": "hex_value",
                    "value": "1ACFFC1D/32"
                  }
                }
              }
            ]
          }
        }
      },
      {
        "type": "myHobbit",
        "structure": {
          "myHobbMember1": {
            "type": "string",
            "value": "Frodo"
          },
          "myHobbMember2": {
            "type": "myPrecious[]",
            "array": [
              {
```
 | **JSON array of structures** with a member of the struct being another JSON array of structures. Both top and nested arrays have the same format as the above “Array of Parameter Sets” containing a list of JSON objects. |

```
                    "type": "myPrecious",
                    "structure": {
                      "myPrecMember1": {
                        "type": "string",
                        "value": "Journey, or..."
                      },
                      "myPrecMember2": {
                        "type": "hex_value",
                        "value": "AABB/11"
                      }
                    }
                  },
                  {
                    "type": "myPrecious",
                    "structure": {
                      "myPrecMember1": {
                        "type": "string",
                        "value": "Friends!"
                      },
                      "myPrecMember2": {
                        "type": "hex_value",
                        "value": "CCDDEE/19"
                      }
                    }
                  }
                ]
              }
            }
          }
        ]
      }
}
```


9.2.1.2 Device Definitions (Parameter Metadata)

This JSON PSM supports the optional PIM requirement of a GEMS Device/service providing Device Definitions to GEMS Clients. These device definitions represent metadata that aids in describing the parameters and directives supported by a GEMS Device. This metadata can include descriptions and any constraints in a machine-readable format. The format used in this PSM is JSON, and the mechanism for obtaining the metadata is via REST over HTTP/HTTPS.

All GEMS metadata fields can be returned using the GetConfig message with a query string per section 9.2.2. The standard GEMS metadata is normative and shall be supported by GEMS implementations.

The vendor-specific GEMS metadata is optional and may be supported by the GEMS device at the vendor's discretion. Any vendor metadata shall be located under the `"vendor"` field in the below JSON body to prevent conflict with the standard metadata.

The normative GEMS metadata definition is found in A.6 JSON Schema: GEMS Get/Set Config.

The JSON body below describes the format of GEMS *metadata* (device definitions) using example values. This metadata can be returned using the `"?meta"` query string described in section 9.2.2 and in section 9.2.10. These same formats apply to Directive arguments which are also GEMS parameters.

All metadata is read-only and can only be *retrieved* from the GEMS device, not set. Note that the `"type"` filed is considered metadata but is also returned with the `"value"` field in all GetConfigResponse messages to be compliant with the PIM; this provides type-checking and is consistent with the other PSMs in this specification that provide both type and value in their messages.

Example JSON Body containing Device Definitions (metadata) for a GEMS Parameter:

```
{
  "param": {
    "description": "Textual description of this GEMS parameter.",
    "readOnly": [false, true],
    "type": "GEMS parameter type",
    "units": "Name of any applicable units for this GEMS parameter, or empty if N/A",
    "ranges": [
      {
        "minValue": -1.0,
        "maxValue": 1.0
      },
      {
        "minValue": 1.0,
        "maxValue": 2.0,
        "minInclusive": false,
        "maxInclusive": false
      },
      {
        "minValue": 3.0
      }
    ],
    "enums": [
      0.0, 1.5, 3.2
    ],
    "vendor": {
      "healthStatus": "good",
      "properties": {
        "_l": "Label",
        "_v": true
      }
    }
  }
}
```

The “enums” field is an array of enumerated values of the same “type” as the parameter. The example is a string enum.

The "ranges" field represents one or more numeric range(s) for integer or floating-point parameters and is optional. Setting minInclusive and/or maxInclusive to false indicates that value is *exclusive* to the range. Omitting the minInclusive and/or maxInclusive field(s) or setting to true indicates that value is *inclusive* to the range.

The minValue and maxValue ranges default to minInclusive = true and maxInclusive = true when omitted. minValue and maxValue must adhere to the parameter’s “type”.

The first range example above represents: $-1.0 \leq x \leq 1.0$, where both the values are *inclusive* to the range.

The second range example above represents: $1.0 < x < 2.0$, where both the values are *exclusive* to the range.

The third range example above represents: $3.0 \leq x$, where the range is 3.0 to +infinity.

Hence, an infinity range is represented by omitting the minValue or maxValue field.

Standard GEMS metadata are: `"description"`, `"readOnly"`, `"type"`, `"units"`, `"enums"`, and **`"ranges"`**.

Any fields contained in the “vendor” section are vendor-specific metadata, i.e. are not standard GEMS metadata. In the above example, the “good” value for “healthStatus” may indicate if the parameter value is within the given range. The “properties” field has a vendor-specific value for “_l”, that may be the parameter’s “label” in the Graphical User Interface (GUI), and the vendor-specific value for “_v” may indicate if the parameter is “visible” in the GUI.

9.2.2 Messages

RESTful GEMS Message semantics map to an HTTP method/verb, a URI with an optional HTTP JSON body, and a JSON Response. Table 9.2 summarizes these mappings and provides links to sections detailing each mapping. The JSON Response format and HTTP Status mapping is in section 9.2.2.1. The format of a GEMS REST URI is as follows, where the various `<PARAMETER_DIRECTIVE_STRING>` options are described in each of the sections linked from Table 9.2:

GEMS REST URI: **`http[s]://<host>:<port>/gems/<TARGET>/<PARAMETER_DIRECTIVE_STRING>`**

This structured GEMS URI supports integration with an API Gateway where multiple GEMS devices can sit behind a single webserver/API gateway. This URI conforms to standard HTTP/HTTPS protocols commonly used for backend services, allowing API Gateways to route, manage, and secure traffic to this endpoint.

RESTful GEMS <TARGET> semantics:

- <TARGET> is a REST endpoint that can contain one or more levels of hierarchy separated by slashes **`'/'`**, e.g.
  - `https://<host>:<port>/gems/tlm/chan1/frameSync`
- The leaf (rightmost) level is the ***concrete* target** containing parameters and directives, e.g. `frameSync`.
- Levels to the left of the leaf are ***context* target(s)** that *contain* concrete and/or additional context targets.
  - Context targets do not themselves contain parameters or directives.
  - A concrete target must exist at some level under every context target.
  - Context targets are not required however they provide a hierarchal way to describe a GEMS device.
- The root target of a GEMS device is an empty <TARGET> with only a single slash **`'/'`** between `gems` and `<PARAMETER_DIRECTIVE_STRING>`, i.e. GEMS device root is: `https://<host>:<port>/gems/`

GEMS <PARAMETER_DIRECTIVE_STRING> semantics:

- The <PARAMETER_DIRECTIVE_STRING> is added after the <TARGET> in the GEMS URI to access GEMS parameters and directives using the specified formats in the subsequent sections and appendices.
- Two reserved words: **`_parameters`** and **`_directives`** may follow the <TARGET> to access these resources. These are required to allow parameters and directives to have the same resource name, and to uniquely identify the resource since directives entail an invocation signature.
- A third reserved word: **`_ping`** is added after the <TARGET> in the GEMS URI to determine liveliness of a GEMS target. Refer to section 9.2.17 for the specified format of this message.

Table 9.2 – GEMS Messages to HTTP/REST Mappings

GEMS Message (from PIM)	**HTTP Method/Verb**	**REST Mapping** (links to sections)
Connect	POST	Connecting
Disconnect	N/A	Disconnecting
Discovering Targets (applicable for all messages)	GET	• Discovering targets • Discovering target resources • Discovering target parameters • Discovering target directives
GetConfig and GetConfigResponse	GET (use query strings for multiple parameters)	Getting Config Parameters
SetConfig and SetConfigResponse	PUT	Setting Config parameters
Directive and DirectiveResponse	GET to retrieve directive signature, POST for all other cases	Discovering a directive signature Invoking a directive
SaveConfig	POST	Save Configuration
LoadConfig	POST	Load Configuration
List Configurations	GET	List Configurations
Ping and PingResponse	GET	Ping target resource
AsyncStatus	Not supported in this version.	Not supported in this version.

9.2.2.1 REST Query Strings

The <PARAMETER_DIRECTIVE_STRING> portion of the GEMS REST URI contains the protocol string and optional query strings needed to access GEMS parameter and directive resources. This REST/JSON GEMS protocol was designed to lever a flat folder-based URI structure for the most common discover, get, set, and invoke operations, however REST query strings are widely used as a conventional method of filtering large JSON responses, so they provide an *option* for more advanced queries when needed. Please refer to the message sections linked in the Table 9.2 **REST Mappings** for examples of these REST query strings.

This section provides the normative definitions of all GEMS REST Query Strings. Refer to section 9.2.10 for example URIs and JSON responses. GEMS query strings shall use the standard REST query string format using '?' and '&' as follows:

`.../<TARGET>/<PARAMETER_DIRECTIVE_STRING>?query1=<ITEMS>&query2=<ITEMS>&query3=<ITEMS>`

Brackets "[]" in the below **Query Strings** definitions represent optional portions of the string as described.

Table 9.3 – GEMS REST Query Strings

Query String	<ITEMS> Definition	Description
?targets=<ITEMS>	**<ITEMS>** can be **_all** for all targets under the specified <TARGET> or a comma-separated list of targets under the specified <TARGET>	This query string is appended to _parameters or _directives to retrieve multiple GEMS parameters or directive signatures from the target(s) specified in <ITEMS>. The preceding <TARGET> can be a concrete *or* context target. This query can be combined with the **meta** query string.
?list=<ITEMS>	**<ITEMS>** can be **_all** for all target parameters or a comma-separated list of specific parameter names	This query string is appended to _parameters to get or set one or more GEMS value(s) of the parameters specified in <ITEMS>. The <TARGET> must be a *concrete* target. This query can be combined with the **meta** query string.
?meta=<ITEMS>	**<ITEMS>** can be **_all** for all standard and vendor metadata, or a comma-separated list of metadata fields per section 9.2.1.2: **description,** **readOnly,** **type,** **units,** **ranges,** **enums,** **vendor**	This query string is appended to _parameters[/<PARAM NAME>] to retrieve metadata from the specified <PARAM NAME> or from multiple parameters if a <PARAM NAME> is not specified and either the **list** or **targets** query is used, as follows: If <TARGET> is concrete, then the meta query can be combined with the **list** query to get metadata from the specified parameters. If <TARGET> is a context target, the meta query can be combined with the **targets** query to get metadata from multiple targets.

9.2.3 Response Messages

The GEMS-REST mapping for GEMS Response Messages is shown in the JSON schema below. When the HTTP status code is SUCCESS (200), the "result" and "description" shall be omitted. When HTTP status is an error (code is not 200), the "result" shall be set to the GEMS Result Code per Table 9.3 below with a corresponding error "description". The normative definition for the GEMS Error Response is in A.1 JSON Schema: GEMS Error Response. The “Server” HTTP header is optional and may contain the web server or other description of the software servicing the request.

```
HTTP/1.1 <HTTP Status Code> <HTTP Status Text>
Server: myWebServer.com
Content-Type: application/json
Content-Length: <length in octets>
{
  "result": "<GEMS Result Code String per Table 9.3>",
  "description": "Blank when SUCCESS, otherwise description of the error",
  <JSON Body for the applicable GEMS Response provided in subsequent sections>
}
```

9.2.3.1 HTTP Response Codes

Since the JSON PSM uses JSON over HTTP, the following table maps the possible standard HTTP responses to the applicable GEMS Result code. In addition to assigning the GEMS result code defined by the PIM in the above response message, the HTTP status code should be set appropriately as documented in this table:

Table 9.3 – GEMS to HTTP Response Code Mapping

GEMS Result Code (string)	HTTP Status Code
"SUCCESS"	200 Success
"INVALID_RANGE"	400 Bad Request
"INVALID_PARAMETER"	400 Bad Request
"INVALID_STATE"	400 Bad Request
"CONFLICTING_PARAMETER"	400 Bad Request
"UNSUPPORTED_MESSAGE"	400 Bad Request
"MALFORMED_MESSAGE"	400 Bad Request
"COMMUNICATION_ERROR"	500 Internal Server Error
"INTERNAL_ERROR"	500 Internal Server Error
"ACCESS_DENIED"	401 Unauthorized / 403 Forbidden Unauthorized for invalid/missing credentials, Forbidden if credentials are present and access is denied
"OTHER"	500 Internal Server Error/400 Bad Request
Invalid URL – Result not returned since not found	404 Not Found

9.2.4 Connecting

HTTP was designed to be stateless/connectionless, however as of HTTP/1.1 a persistent connection may be implemented to reduce latency by reusing the same TCP connection for multiple requests. The concept of GEMS “Connect” message is mapped differently depending on the type of authentication supported by the server. Servers may support one of the following authentication schemes:

HTTP Authentication	Transport	GEMS Connect
No Authentication	HTTP/HTTPS	Implied on each request
Basic Authentication	HTTP/HTTPS	Implied on each request
JWT Authentication	HTTP/HTTPS	Target embedded in the returned JSON Web Token (JWT)

With **No Authentication**, requests are allowed as though a proper GEMS Connect had been issued and can be done with or without HTTP encryption. This is the only method supported over HTTP, the other methods require the transfer of private information (passwords and tokens) and should not use HTTP.

With **Basic Authentication**, the standard method is used for authenticating users when accessing resources over HTTP. It functions by transmitting user credentials (username and password) in the Authorization header of an HTTP request which are authenticated each time before the request is carried out. If authentication is successful, the request is allowed as though a GEMS Connect had been issued.

With **JWT Authentication**, the client provides user, password, desired control and/or status authorization request, and GEMS target information to the server for authentication. If authenticated, the client receives a digitally signed JSON Web Token (JWT) which encodes the user, control/status, and GEMS target information in a secure manner. This JWT serves the role of the GEMS token and contains information on what target was connected as well as whether control is available. Other JWT fields are allowed and the standard fields iat (Issued At), nbf (Not Before), exp (Expiration Time), iss (Issuer), and aud (Audience) are recommended.

9.2.4.1 HTTP Headers

The following are the minimum HTTP Headers required:

- Content-Type: application/json
- Content-Length

Content-Type is required whenever the body contains JSON encoded information.

The following are optional HTTP Headers:

- Authorization

9.2.4.2 Connect Request (JWT)

The POST /con endpoint is used for JWT Authentication as described above. This serves the purpose of the GEMS Connect message containing the GEMS target and Control (ctl) fields in the JSON body. An example is shown below:

```
POST /con HTTP/1.1
Host: http[s]://<host>:<port>
Content-Type: application/json
{
  "username": "myuser",
  "password": "secret",
  "ctl": true,
  "target": " tlmRx/frameSync"
}
```

9.2.4.3 Connect Response (JWT)

The JWT response to the connection request is encoded per IETF RFC 7519. This serves the purpose of the GEMS Connect Response message. An example is shown below:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4g
RG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
  "refreshToken": "anotherLongTokenString",
  "expiresIn": 3600
}
```

For example, the `accessToken` payload would contain the following to uniquely identify the user connection:

```
{
  "aud": "10.0.8.2:8888",
  "iss": "10.0.8.2:8888",
  "exp": 1474367329,
  "iat": 1471367329,
  "nbf": 1471367329,
  "user": "myuser",
  "ctl": true,
  "target": "tlmRx/frameSync"
}
```

The encoded JWT `accessToken` is used for all subsequent requests in the HTTP Authorization header followed by the word “Bearer” and a space, i.e. Authorization header: `Bearer <accessToken>`

9.2.5 Disconnecting

GEMS Disconnect is handled by the JWT expiring per the `expiresIn` field in the Connect Response (JWT), thus lasts as long as the server deems necessary to balance re-authentication costs with nominal operations.

9.2.6 Discovering targets

Discovering targets from the root level of a GEMS device is the first step needed to access its parameters and directives.

The normative definition for these success-case responses is in A.2 JSON Schema: Discover GEMS Targets.

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/
```

Example JSON Response for success case:

```
{
  "resources": [
    "tlm",
    "cmd",
    "configService"
  ]
}
```

In the above example, `"tlm"` and `"cmd"` are top-level context targets and `"configService"` is a concrete target (identified via the "resources" described in section 9.2.7).

Example JSON Response for an error case (see section 0 for this response message format):

```
{
  "result": "INTERNAL_ERROR",
  "description": "Device is busy."
}
```

Discover targets from a top-level context (this is the second step when recursing into layered targets of a device):

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm
```

Example JSON Response:

```
{
  "resources": [
    "chan1",
    "chan2"
  ]
}
```

In the above example, `"tlm"` is a top-level context target where `"chan1"` and `"chan2"` are the next level context targets under the "tlm" target.

Discover targets from the next level context (this step is repeated for each level when recursing into a device hierarchy):

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1
```

Example JSON Response:

```
{
  "resources": [
    "frameSync",
    "rsDecoder",
    "ccsdsProc"
  ]
}
```

In the above example, `"frameSync"`, `"rsDecoder"`, and `"ccsdsProc"` are concrete targets under "tlm/chan1" context target. Concrete targets are identified via the "resources" described in section 9.2.7.

9.2.7 Discovering target resources

Discover the resources that a concrete target provides. The normative definition for this success-case response is in A.3 JSON Schema: Discover GEMS Target Resources.

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/frameSync
```

Example JSON Response:

```
{
  "resources": [
    "_parameters",
    "_directives"
  ]
}
```

When `"_parameters"` and `"_directives"` are returned from a discovery call, the requested target is known to be **concrete**. In the above example, `"frameSync"` is the concrete target where `"_parameters"` and `"_directives"` are the available resources under that target. The reserved "_parameters" and/or "_directives" resource descriptions shall exist for all concrete targets, meaning that the requested target provides GEMS parameters and/or directives.

9.2.8 Discovering target parameters

Discover all parameters supported by a concrete target. The normative definition for this success-case response is in A.4 JSON Schema: Discover GEMS Target Parameters.

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_parameters
```

Example JSON Response:

```
{
  "parameters": [
    "frameLength",
    "lockState",
    "inverted"
  ]
}
```

In the above example, `"frameLength"`, `"lockState"`, and `"inverted"` are the *list* of all available GEMS parameters under the "frameSync" target.

9.2.9 Discovering target directives

Discover all directives supported by a concrete target. The normative definition for this success-case response is in A.5 JSON Schema: Discover GEMS Target Directives.

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_directives
```

Example JSON Response:

```
{
  "directives": [
    "resetStatistics",
    "clearFault"
  ]
}
```

In the above example, `"resetStatistics"`, and `"clearFault"` are the *list* of all available GEMS directives under the "frameSync" target.

9.2.10 Getting Config parameters

The normative definition for all success-case Get Config responses (including the below examples) is found in A.6 JSON Schema: GEMS Get/Set Config.

Get a SINGLE parameter-value from a GEMS concrete target (the "type" is returned by default to comply with the PIM):

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_parameters/frameLength
```

Example JSON Response for success case:

```
{
  "frameLength": {
    "type": "uint",
    "value": 8192
  }
}
```

Get ALL parameter-values from a GEMS concrete target (the "list" query only applies to concrete targets):

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_parameters?list=_all
```

Example JSON Response:

```
{
  "frameLength": {
    "type": "uint",
    "value": 8192
  },
  "lockState": {
    "type": "string",
    "value": "search"
  },
  "inverted": {
    "type": "bool",
    "value": true
  }
}
```

Get values of *specified parameters* from a GEMS concrete target (the "list" query only applies to concrete targets):

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_parameters?list=lockState,
       inverted
```

Example JSON Response:

```
{
  "lockState": {
    "type": "string",
    "value": "search"
  },
  "inverted": {
    "type": "bool",
    "value": true
  }
}
```

Get all parameter values for all **concrete** targets under a GEMS **context** target (the “targets” query only applies to context targets or the root/device target):

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/_parameters?targets=_all
```

Example JSON Response:


```
{
  "resources": [
    {
      "target": "tlm/chan1/frameSync",
      "parameters": {
        "frameLength": {
          "type": "uint",
          "value": 8192
        },
        "lockState": {
          "type": "string",
          "value": "search"
        },
        "inverted": {
          "type": "bool",
          "value": true
        }
      }
    },
    {
      "target": "tlm/chan1/rsDecoder",
      "parameters": {
        "algorithm": {
          "type": "string",
          "value": "RS_255_239"
        },
        "interleave": {
          "type": "uint",
          "value": 4
        }
      }
    },
    {
      "target": "tlm/chan1/ccsdsProc",
      "parameters": {
        "dataLinkVersion": {
          "type": "string",
          "value": "AOS"
        },
        "crcErrors": {
          "type": "uint",
          "value": 7
        }
      }
    }
  ]
}
```

Get parameter values from a SUBSET of GEMS targets at the device (root) level:

HTTP Verb: GET

URI: https://<host>:<port>/**gems/_parameters?targets=tlm/chan1/rsDecoder,tlm/chan2/ccsdsProc**

Example JSON Response:


```
{
  "resources": [
    {
      "target": "tlm/chan1/rsDecoder",
      "parameters": {
        "algorithm": {
          "type": "string",
          "value": "RS_255_239"
        },
        "interleave": {
          "type": "uint",
          "value": 4
        }
      }
    },
    {
      "target": "tlm/chan2/ccsdsProc",
      "parameters": {
        "dataLinkVersion": {
          "type": "string",
          "value": "AOS"
        },
        "crcErrors": {
          "type": "uint",
          "value": 7
        }
      }
    }
  ]
}
```


Get a SINGLE parameter's selected metadata (vs all the metadata) from a GEMS target:

HTTP Verb: GET

URI: https://<host>:<port>/**gems/tlm/chan1/frameSync/_parameters/frameLength?meta=readOnly,units**

Example JSON Response:


```
{
  "frameLength": {
    "readOnly": false,
    "units": "bits"
  }
}
```

Get a SINGLE parameter's metadata from a GEMS target (returns all standard and vendor metadata):

HTTP Verb: GET

URI:

https://<host>:<port>/**gems/tlm/chan1/frameSync/_parameters/frameLength?meta=_all**

Example JSON Response:


```
{
  "frameLength": {
    "description": "Length of the frame.",
    "readOnly": false,
    "type": "uint",
    "units": "bits",
    "ranges": [
      {
        "minValue": 8,
        "maxValue": 524280
      }
    ],
    "vendor": {
      "healthStatus": "good",
      "properties": {
        "_l": "Frame Length",
        "_v": true
      }
    }
  }
}
```


Get SELECTED parameter's vendor metadata from a GEMS target:

HTTP Verb: GET

URI: https://<host>:<port>/**gems/tlm/chan1/frameSync/_parameters?list=frameLength,lockState?meta=vendor**

Example JSON Response:


```
{
  "frameLength": {
    "vendor": {
      "healthStatus": "fault",
      "properties": {
        "_l": "Frame Length",
        "_v": true
      }
    }
  },
  "lockState": {
    "vendor": {
      "healthStatus": "good",
      "properties": {
        "_l": "Lock State",
        "_v": true
      }
    }
  }
}
```

Get all metadata of all parameters from all targets of a GEMS device (i.e. *get the entire Device Definition*):

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/_parameters?targets=_all?meta=_all
```

Example JSON Response:

```
{
  "resources": [
    {
      "target": "tlm/chan1/frameSync",
      "parameters": {
        "frameLength": {
          "description": "Length of the frame.",
          "readOnly": false,
          "type": "uint",
          "units": "bits",
          "ranges": [
            {
              "minValue": 8,
              "maxValue": 524280
            }
          ]
        },
        "lockState": {
          "description": "Frame Sync Lock State of the frame.",
          "readOnly": true,
          "type": "string",
          "units": "states",
          "enums": [
            "NoData","Search","Verify","Lock","Check","Stopped"
          ]
        },
        ... additional frameSync parameters
      },
    },
    {
      "target": "tlm/chan2/rsDecoder",
      "parameters": {
        "algorithm": {
          "description": "Reed-Solomon Decoder algorithm/mode.",
          "readOnly": false,
          "type": "string",
          "units": "modes",
          "enums": [
            "CCSDS 255/223","CCSDS 255/239"
          ]
        },
        ... additional rsDecoder parameters
      }
    },
    ... additional targets
  ]
}
```

Get all parameter values from all targets of a GEMS device (i.e. *get the entire Device Configuration*):

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/_parameters?targets=_all
```

Example JSON Response:

```
{
  "resources": [
    {
      "target": "tlm/chan1/frameSync",
      "parameters": {
        "frameLength": {
          "type": "uint",
          "value": 1024,
        },
        "lockState": {
          "type": "string",
          "value": "Search",
        },
        ... all remaining frameSync parameters
      },
    },
    {
      "target": "tlm/chan2/rsDecoder",
      "parameters": {
        "algorithm": {
          "type": "string",
          "value": "CCSDS 255/223",
        },
        ... all remaining rsDecoder parameters
      }
    },
    ... all remaining targets and their parameters
  ]
}
```

9.2.11 Setting Config parameters

The normative definition for all success-case Set Config JSON bodies (including all the examples below) is found in A.6 JSON Schema: GEMS Get/Set Config. Note Set Config levers the same schema as Get Config for reuse and consistency.

Set a SINGLE parameter-value in a GEMS concrete target (the "type" is used by default to comply with the PIM):

```
HTTP Verb: PUT
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_parameters/frameLength
Content-Type: application/json
{
  "frameLength": {
    "type": "uint",
    "value": 4096
  }
}
```

Example JSON Response for the above Set Config success case:

```
{
  "frameLength": {
    "type": "uint",
    "value": 4096
  }
}
```

Set values of MULTIPLE parameters in a GEMS concrete target:

```
HTTP Verb: PUT
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_parameters
Content-Type: application/json
{
  "frameLength": {
    "type": "uint",
    "value": 1024
  },
  "syncPattern": {
    "type": "hex_value",
    "value": "FAF320/24"
  }
}
```

Example JSON Response:

```
{
  "frameLength": {
    "type": "uint",
    "value": 1024
  },
  "syncPattern": {
    "type": "hex_value",
    "value": "FAF320/24"
  }
}
```

Set parameter values in ALL or a SUBSET of GEMS targets, i.e. the JSON body sent with the PUT call can contain all targets and all parameters, or a subset of targets and a subset of parameters):

```
HTTP Verb: PUT
URI: https://<host>:<port>/gems/_parameters
Content-Type: application/json
{
  "resources": [
    {
      "target": "tlm/chan1/frameSync",
      "parameters": {
        "frameLength": {
          "type": "uint",
          "value": 4096
        }
        ... subset or all frameSync parameters
      }
    },
    {
      "target": "tlm/chan1/rsDecoder",
      "parameters": {
        "algorithm": {
          "type": "string",
          "value": "RS_255_239"
        },
        "interleave": {
          "type": "uint",
          "value": 5
        }
        ... subset or all rsDecoder parameters
      }
    },
    {
      "target": "tlm/chan2/ccsdsProc",
      "parameters": {
        "dataLinkVersion": {
          "type": "string",
          "value": "TM"
        },
        "enableCrc": {
          "type": "bool",
          "value": true
        }
      }
    }
    ... subset or all remaining targets and all or subset of their parameters
  ]
}
```

The JSON Response contains the same JSON body as the above requested content.

9.2.12 Discovering a directive signature

The normative definition for the success-case response (including the below examples) is found in A.6 JSON Schema: GEMS Get/Set Config. Note this definition levers the same schema as Get Config for reuse and consistency because a directive signature may contain any form of GEMS parameter, including complex structures, arrays, etc.

Get a SINGLE directive's signature including invocation and return parameter(s). The `"value"` returned represents the default value of each directive argument:

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_directives/resetChanSeqNum
```

Example JSON Response for success case:

```
{
  "channelName": {
    "type": "string",
    "value": ""
  },
  "seqNum": {
    "type": "uint",
    "value": 0
  }
}
```

9.2.13 Invoking a directive

The normative definition for all success-case JSON bodies (including all the examples below) is found in A.6 JSON Schema: GEMS Get/Set Config. Note this definition levers the same schema as Get Config for reuse and consistency because a directive signature may contain any form of GEMS parameter, including complex structures, arrays, etc.

Invoke a Directive in a GEMS concrete target:

```
HTTP Verb: POST
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_directives/resetChanSeqNum
Content-Type: application/json
{
  "channelName": {
    "type": "string",
    "value": "WB"
  },
  "seqNum": {
    "type": "uint",
    "value": 3
  }
}
```

Example JSON Response containing a return value:

```
{
  "resetSeqNumStatus": {
    "type": "string",
    "value": "RESET_TO_VAL"
  }
}
```

Example JSON Response for a Directive that does not contain a return value:

```
{}
```

9.2.14 Save Configuration

The normative definition for the save configuration message is defined below. Everything is normative other than the example `values`. The TARGET is the configuration service for the given GEMS device which must provide the **`saveConfiguration`** directive that implements the following API to comply with this standard.

Save Configuration in a GEMS device (i.e. saves the device parameters to a configuration file, e.g. XML file):

```
HTTP Verb: POST
URI: https://<host>:<port>/gems/<configService>/_directives/saveConfiguration
Content-Type: application/json
{
  "name": {
    "type": "string",
    "value": "nominal"
  }
}
```

JSON Response for Save Configuration:

```
{
  "parameterCount": {
    "type": "uint",
    "value": 754
  },
  "result": {
    "type": "string",
    "value": "Successfully saved 754 parameters to configuration 'nominal_setup'"
  }
}
```

9.2.15 Load Configuration

The normative definition for the load configuration message is defined below. Everything is normative other than the example `values`. The TARGET is the configuration service for the given GEMS device which must provide the **`loadConfiguration`** directive that implements the following API to comply with this standard.

Load Configuration in a GEMS device (i.e. restores the device parameters to those from specified configuration file):

```
HTTP Verb: POST
URI: https://<host>:<port>/gems/<configService>/_directives/loadConfiguration
Content-Type: application/json
{
  "name": {
    "type": "string",
    "value": "nominal"
  }
}
```

JSON Response for Load Configuration:

```
{
  "parameterCount": {
    "type": "uint",
    "value": 754
  },
  "result": {
    "type": "string",
    "value": "Successfully loaded 754 parameters from configuration 'nominal'"
  }
}
```

9.2.16 List Configurations

The normative definition for the list configurations message is defined below. Everything is normative other than the example `values`. The TARGET is the configuration service for the given GEMS device which must provide the **`listConfigurations`** directive that implements the following API to comply with this standard.

List Configurations in a GEMS device (i.e. provides a list of configuration files that can be used to set parameters):

```
HTTP Verb: POST
URI: https://<host>:<port>/gems/<configService>/_directives/listConfigurations
There is no JSON data provided with this message.
```

JSON Response for List Configurations:

```
{
  "configurations": {
    "type": "string[]",
    "array": [
      {
        "type": "string",
        "value": "nominal"
      },
      {
        "type": "string",
        "value": "test_1"
      },
      {
        "type": "string",
        "value": "test_2"
      }
    ]
  }
}
```

9.2.17 Ping target resource

The normative definition for the Ping target resource message is defined below. Everything is normative other than the example `values`. The TARGET can be any of the GEMS device targets which must implement the following ping API to comply with this standard.

Ping target in a GEMS device:

```
HTTP Verb: GET
URI: https://<host>:<port>/gems/tlm/chan1/frameSync/_ping
There is no JSON data provided with this message.
```

JSON Response for Ping target that responds successfully:

```
{
  "result": {
    "type": "string",
    "value": "SUCCESS"
  }
}
```

If the Ping target fails at the server for any reason, a GEMS REST Response message shall be returned per section 9.2.3 with the respective result and description assigned.

The Ping target may also fail by never reaching the GEMS device, in which case the HTTP request times out and fails with “site can’t be reached” and/or “<host> refused to connect.”, and no JSON Response is returned.

Appendix A GEMS JSON Schema

A.1 JSON Schema: GEMS Error Response

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "JSON Schema for GEMS Error Response",
  "type": "object",
  "required": [
    "result",
    "description"
  ],
  "properties": {
    "result": {
      "type": "string",
      "description": "GEMS Result Code",
      "enum": [
        "SUCCESS",
        "INVALID_RANGE",
        "INVALID_PARAMETER",
        "INVALID_STATE",
        "CONFLICTING_PARAMETER",
        "UNSUPPORTED_MESSAGE",
        "MALFORMED_MESSAGE",
        "COMMUNICATION_ERROR",
        "INTERNAL_ERROR",
        "ACCESS_DENIED",
        "OTHER"
      ]
    },
    "description": {
      "type": "string",
      "description": "Description of the error"
    }
  },
  "additionalProperties": false
}
```

A.2 JSON Schema: Discover GEMS Targets

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "JSON Schema for Discover GEMS Targets",
  "type": "object",
  "required": [
    "resources"
  ],
  "properties": {
    "resources": {
      "type": "array",
      "description": "A list of GEMS targets from the requested device level.",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    }
  },
  "additionalProperties": false
}
```

A.3 JSON Schema: Discover GEMS Target Resources


```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "JSON Schema for Discover GEMS Target Resources",
  "type": "object",
  "required": [
    "resources"
  ],
  "properties": {
    "resources": {
      "type": "array",
      "description": "A list of GEMS resource(s) from the requested concrete target.",
      "uniqueItems": true,
      "items": {
        "type": "string",
        "enum": [
          "_parameters",
          "_directives"
        ]
      },
      "anyOf": [
        {
          "contains": {
            "const": "_parameters"
          }
        },
        {
          "contains": {
            "const": "_directives"
          }
        }
      ]
    }
  },
  "additionalProperties": false
}
```

A.4 JSON Schema: Discover GEMS Target Parameters

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "JSON Schema for Discover GEMS Target Parameters",
  "type": "object",
  "properties": {
    "parameters": {
      "type": "array",
      "description": "A list of GEMS parameters from the requested concrete target.",
      "items": {
        "type": "string"
      },
      "uniqueItems": true
    }
  },
  "required": [
    "parameters"
  ],
  "additionalProperties": false
}
```

A.5 JSON Schema: Discover GEMS Target Directives

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "JSON Schema for Discover GEMS Target Directives",
  "type": "object",
  "properties": {
    "directives": {
      "type": "array",
      "description": "A list of GEMS directives from the requested concrete target.",
      "items": {
        "type": "string"
      },
      "uniqueItems": true
    }
  },
  "required": [
    "directives"
  ],
  "additionalProperties": false
}
```

A.6 JSON Schema: GEMS Get/Set Config

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "JSON Schema for GEMS Get and Set Config",
  "type": "object",
  "properties": {
    "resources": {
      "type": "array",
      "items": {
        "$ref": "#/$defs/resource"
      }
    }
  },
  "additionalProperties": {
    "$ref": "#/$defs/parameter"
  },
  "$defs": {
    "resource": {
      "type": "object",
      "required": [
        "target",
        "parameters"
      ],
      "properties": {
        "target": {
          "type": "string"
        },
        "parameters": {
          "type": "object",
          "additionalProperties": {
            "$ref": "#/$defs/parameter"
          }
        }
      },
      "additionalProperties": false
    },
    "parameter": {
      "type": "object",
      "properties": {
        "type": {
          "type": "string"
        },
        "value": {},
        "structure": {
          "type": "object",
          "additionalProperties": {
            "$ref": "#/$defs/parameter"
          }
        },
        "array": {
          "type": "array",
          "items": {
            "$ref": "#/$defs/parameter"
          }
        },
```

```
"description": {
  "type": "string"
},
"readOnly": {
  "type": "boolean"
},
"units": {
  "type": "string"
},
"enums": {
  "type": "array",
  "items": {
    "anyOf": [
      {
        "type": "string"
      },
      {
        "type": "number"
      },
      {
        "type": "integer"
      },
      {
        "type": "boolean"
      }
    ]
  }
},
"ranges": {
  "type": "array",
  "items": {
    "type": "object",
    "anyOf": [
      {
        "required": [
          "minValue"
        ]
      },
      {
        "required": [
          "maxValue"
        ]
      }
    ],
    "properties": {
      "minValue": {
        "type": "number"
      },
      "maxValue": {
        "type": "number"
      },
      "minInclusive": {
        "type": "boolean"
      },
      "maxInclusive": {
```

```
            "type": "boolean"
          }
        },
        "additionalProperties": false
      }
    },
    "vendor": {
      "type": "object",
      "properties": {
        "healthStatus": {
          "type": "string"
        },
        "properties": {
          "type": "object",
          "additionalProperties": true
        }
      },
      "additionalProperties": true
    }
  },
  "allOf": [
    {
      "if": {
        "required": [
          "array"
        ],
        "not": {
          "properties": {
            "array": {
              "items": {
                "required": [
                  "structure"
                ]
              }
            }
          }
        }
      },
      "then": {
        "properties": {
          "type": {
            "enum": [
              "bool[]",
              "byte[]",
              "ubyte[]",
              "long[]",
              "ulong[]",
              "int[]",
              "uint[]",
              "short[]",
              "ushort[]",
              "double[]",
              "string[]",
              "hex_value[]",
              "time[]",
```

```
            "utime[]"
          ]
        }
      }
    }
  },
  {
    "if": {
      "required": [
        "array"
      ],
      "properties": {
        "array": {
          "items": {
            "required": [
              "structure"
            ]
          }
        }
      }
    },
    "then": {
      "properties": {
        "type": {
          "pattern": ".+\\[\\]$"
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "bool[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "bool"
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
```

```
          "const": "byte[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "byte"
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "ubyte[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "ubyte"
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "long[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "long"
```

```
            }
          }
        }
      }
    }
  }
},
{
  "if": {
    "properties": {
      "type": {
        "const": "ulong[]"
      }
    }
  },
  "then": {
    "properties": {
      "array": {
        "items": {
          "properties": {
            "type": {
              "const": "ulong"
            }
          }
        }
      }
    }
  }
},
{
  "if": {
    "properties": {
      "type": {
        "const": "int[]"
      }
    }
  },
  "then": {
    "properties": {
      "array": {
        "items": {
          "properties": {
            "type": {
              "const": "int"
            }
          }
        }
      }
    }
  }
},
{
  "if": {
    "properties": {
      "type": {
```

```
          "const": "uint[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "uint"
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "short[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "short"
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "ushort[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "ushort"
```

```
            }
          }
        }
      }
    }
  }
},
{
  "if": {
    "properties": {
      "type": {
        "const": "double[]"
      }
    }
  },
  "then": {
    "properties": {
      "array": {
        "items": {
          "properties": {
            "type": {
              "const": "double"
            }
          }
        }
      }
    }
  }
},
{
  "if": {
    "properties": {
      "type": {
        "const": "string[]"
      }
    }
  },
  "then": {
    "properties": {
      "array": {
        "items": {
          "properties": {
            "type": {
              "const": "string"
            }
          }
        }
      }
    }
  }
},
{
  "if": {
    "properties": {
      "type": {
```

```
          "const": "hex_value[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "hex_value"
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "time[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "time"
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "utime[]"
        }
      }
    },
    "then": {
      "properties": {
        "array": {
          "items": {
            "properties": {
              "type": {
                "const": "utime"
```

```
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "not": {
        "anyOf": [
          {
            "required": [
              "structure"
            ]
          },
          {
            "required": [
              "array"
            ]
          }
        ]
      }
    },
    "then": {
      "properties": {
        "type": {
          "enum": [
            "bool",
            "byte",
            "ubyte",
            "long",
            "ulong",
            "int",
            "uint",
            "short",
            "ushort",
            "double",
            "string",
            "hex_value",
            "time",
            "utime"
          ]
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "enum": [
            "uint",
            "ulong",
            "ushort",
```

```
            "ubyte"
          ]
        }
      },
      "required": [
        "type"
      ]
    },
    "then": {
      "properties": {
        "value": {
          "type": "integer",
          "minimum": 0
        },
        "enums": {
          "items": {
            "type": "integer",
            "minimum": 0
          }
        },
        "ranges": {
          "items": {
            "properties": {
              "minValue": {
                "type": "integer",
                "minimum": 0
              },
              "maxValue": {
                "type": "integer",
                "minimum": 0
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "enum": [
            "int",
            "long",
            "short",
            "byte"
          ]
        }
      },
      "required": [
        "type"
      ]
    },
    "then": {
      "properties": {
```

```
        "value": {
          "type": "integer"
        },
        "enums": {
          "items": {
            "type": "integer"
          }
        },
        "ranges": {
          "items": {
            "properties": {
              "minValue": {
                "type": "integer"
              },
              "maxValue": {
                "type": "integer"
              }
            }
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "double"
        }
      },
      "required": [
        "type"
      ]
    },
    "then": {
      "properties": {
        "value": {
          "type": "number"
        },
        "enums": {
          "items": {
            "type": "number"
          }
        },
        "ranges": {
          "items": {
            "properties": {
              "minValue": {
                "type": "number"
              },
              "maxValue": {
                "type": "number"
              }
            }
          }
        }
```

```
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "string"
        }
      },
      "required": [
        "type"
      ]
    },
    "then": {
      "properties": {
        "value": {
          "type": "string"
        },
        "enums": {
          "items": {
            "type": "string"
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
        "type": {
          "const": "bool"
        }
      },
      "required": [
        "type"
      ]
    },
    "then": {
      "properties": {
        "value": {
          "type": "boolean"
        },
        "enums": {
          "items": {
            "type": "boolean"
          }
        }
      }
    }
  },
  {
    "if": {
      "properties": {
```

```
              "type": {
                "const": "time"
              }
            },
            "required": [
              "type"
            ]
          },
          "then": {
            "properties": {
              "value": {
                "type": "string",
                "pattern": "^[0-9]{1,19}\\.[0-9]{1,9}$",
                "description": "Seconds.nanoseconds elapsed since midnight UTC of
January 1, 1970"
              }
            }
          }
        },
        {
          "if": {
            "properties": {
              "type": {
                "const": "utime"
              }
            },
            "required": [
              "type"
            ]
          },
          "then": {
            "properties": {
              "value": {
                "type": "string",
                "pattern": "^[0-9]{4}-(?:[0-2][0-9]{2}|3[0-5][0-9]|36[0-6])T(?:[01][0-
9]|2[0-3]):[0-5][0-9]:[0-5][0-9]\\.[0-9]{1,9}Z$",
                "description": "The UTC time string in ISO 8601 ordinal date format as
follows: YYYY-DDDThh:mm:ss[.nnnnnnnnn]Z, where: YYYY is four digits of year, DDD is day
of the year (001 through 365), character T denotes beginning of the Time field, hh is
hours (00 through 23), mm is minutes (00 through 59), ss is seconds (00 through 59),
[.nnnnnnnnn] is an optional 1-9 digit fractional seconds in nanoseconds resolution,
character Z denotes Zulu time which represents time zone UTC+00:00, equivalent to
Greenwich Mean Time (GMT)"
              }
            }
          }
        },
        {
          "if": {
            "properties": {
              "type": {
                "const": "hex_value"
              }
            },
            "required": [
```

```
              "type"
            ]
          },
          "then": {
            "properties": {
              "value": {
                "type": "string",
                "pattern": "^[0-9A-Fa-f]+/([0-9]|[1-9][0-9]*)$",
                "description": "Hexadecimal digits followed by / and bit length (e.g.,
FAF320/24 or 0/0)"
              }
            }
          }
        }
      ],
      "additionalProperties": false
    }
  }
}
```

INDEX