Date: September 2024

OBJECT MANAGEMENT GROUP®

Ground Data Delivery Interface (GDDI)

Version 1.0 – beta 1

OMG Document Number: dtc/24-09-16

Normative Reference: https://www.omg.org/spec/GDDI/

Normative Machine Readable file(s):

https://www.omg.org/members/cgi-bin/doc?space/24-07-02.xml

This OMG document replaces the submission document (space/24-07-01). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome and should be directed to issues@omg.org by October 11, 2024.

You may view the pending issues for this specification from the OMG revision issues web page https://issues.omg.org/issues/lists.

The FTF Recommendation and Report for this specification will be published in June 2025. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

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 Rd, PMB 274, Milford, MA 01757, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see: https://www.omg.org/legal/tm_list.htm. 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.

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:

https://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 https://www.iso.org

1 Scope

The Ground Data Delivery Interface (GDDI) specification defines a lightweight message interface standard to encapsulate spacecraft data and metadata for transfer between ground applications within a common network using a model-driven approach. The purpose of GDDI is to standardize the data/bearer-plane for digital baseband data within satellite ground systems, making it a companion standard to the existing OMG GEMS specification which provides a standard for the control/status-plane. By standardizing the structure of this message interface, GDDI facilitates interoperability between endpoints that agree to use compatible metadata definitions (i.e., types, identifiers/tags, etc.). GDDI is applicable to data and metadata in the digital domain, also known as baseband. This type of data and metadata includes spacecraft commands, telemetry, and mission payload *prior to conversion* to/from signals in the RF domain (i.e., waveform modulation). The scope of GDDI is the baseband domain. Sufficient standards already exist for the RF domain such as ANSI/VITA-49 and IEEE-ISTO DIFI.

Figure 1 below shows the scope of GDDI within a notional satellite ground system. The applicable baseband domain boundary is depicted as well as the non-applicable RF waveform boundary. It also shows an overview of the conceptual model for the GDDI specification with typical baseband data-plane interfaces between the various ground functions composed of varying types of metadata between them. This model provides the necessary elastic flexibility of appending different types of metadata at different points within the data path, dynamically.

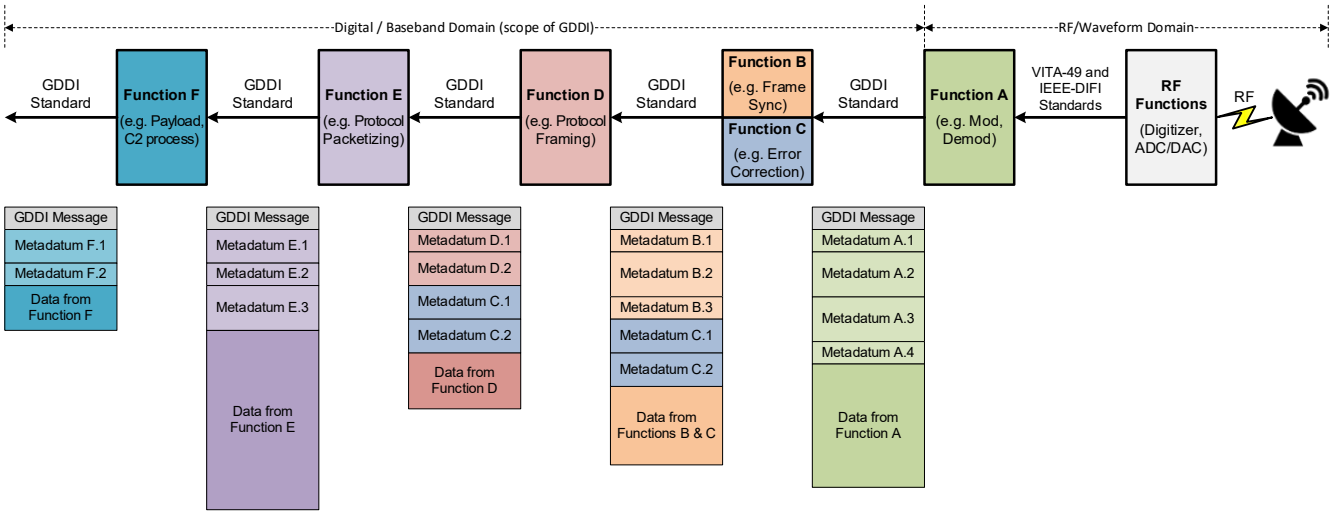

Figure 1 – GDDI Conceptual Model and Scope

The GDDI PIM defines a model of the elastic message structure and behavior that is standard to all GDDI implementations. In the case of GDDI, the platform is defined as the message encoding format and the network transport mechanism used to transfer the messages between endpoints. By defining the platform at these levels, the focus of the GDDI PIM is the message content and behavior, leaving the specifics of defining the message format and transporting that message to the PSMs.

Two GDDI PSMs are defined:

1. The "Encoding Format PSM" uses well-defined CORBA type definitions, encoding formats, and bit/byte ordering that are well suited for many transport platforms. This PSM focuses on reusing a robust standard that allows implementations to lever OMG IDL or other languages and tooling for efficient development, language mapping, maintenance, and portability, all with the goal of promoting interoperability.

2. The "Network Transport PSM" uses the ubiquitous TCP/IP protocol pair for guaranteed transmission and reception of GDDI Messages between endpoints. The rationale for using this transport layer is that TCP provides a proven, efficient, and ensured transport across IP-based networks that is prevalent among satellite ground systems.

Many other platform specific mappings of the GDDI PIM are possible. These potentially include other standard encoding formats such as JSON or XML, as well as network transports such as Stream Control Transmission Protocol (SCTP), Datagram Congestion Control Protocol (DCCP), or Data Distribution Service (DDS).

2 Conformance

A valid implementation requires and consists of everything specified in the PIM. Conformance is based on one encoding PSM and one transport PSM in their entirety.

3 Normative References

The following are normative references that apply to this specification:

- [CORBAINTEROP] CORBA Specification, Version 3.3, Part 2: CORBA Interoperability, OMG Document: formal/2012-11-14
- [GDDIRFP] Ground Data Delivery Interface (GDDI) Request For Proposal (RFP), OMG Document: space/2023-03-01
- [TCP] Transmission Control Protocol, RFC9293, Internet Engineering Task Force (IETF)
- [IP] Internet Protocol, RFC791 (IPv4) and RFC8200 (IPv6), Internet Engineering Task Force (IETF)
- [UTF8] Unicode Transformation Format, RFC3629, ISO/IEC10646-1, UTF-8

4 Terms and Definitions

None.

5 Symbols

None.

6 Additional Information

6.1 Acknowledgments

The following companies submitted this specification:

- Kratos RT Logic, Inc.

The following companies and organizations support this specification:

- Sphinx Defense

7 PIM

7.1 Overview

The GDDI specification defines a standard, Platform Independent Model (PIM) used to encapsulate and transfer data and metadata between endpoints within a satellite ground system. The GDDI model does not presume or try to define a specific system level architecture. Instead, it defines generic concepts such as message structure, metadata, and endpoints that are relatively simple to implement and provides system integrators with common ways to transfer multiple types of space related metadata and data within a ground system.

Being a lightweight specification, the GDDI PIM provides a straightforward **message interface** standard with **simplex transmission between endpoints**. These PIM features promote low encoding/formatting overhead, as well as efficient network throughput and latency performance required for modern ground systems.

The following GDDI PIM sections consist of behavioral use cases and message related classes that allow ground system endpoints to efficiently transfer data and metadata between themselves.

7.2 GDDI Use Cases

Figure 2 depicts the GDDI use cases. These use cases define common interactions and activities associated with defining and using the GDDI data/metadata message interface. Subsequent sections explain each element in this diagram.

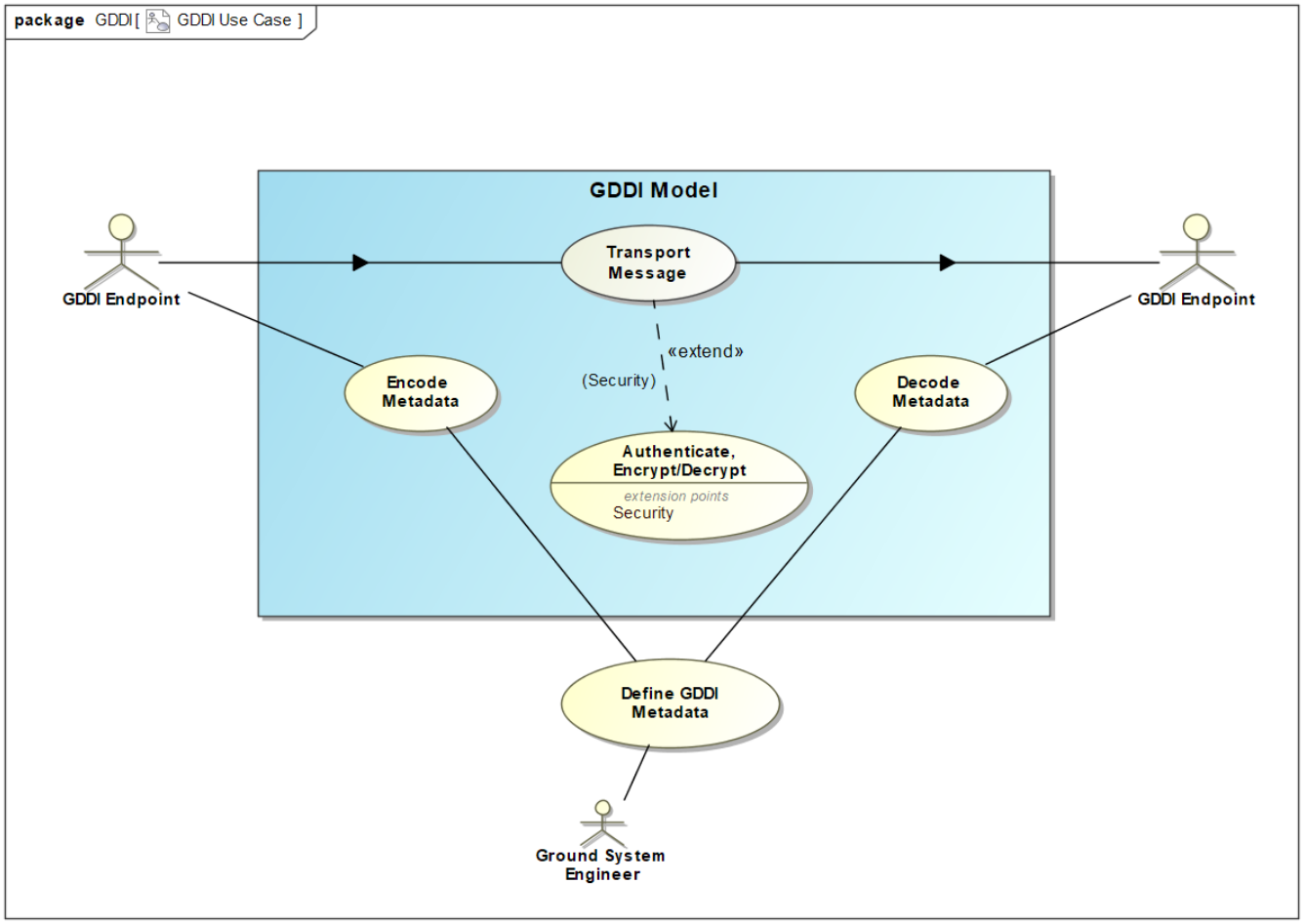

Figure 2 – GDDI Use Cases

7.2.1 GDDI Endpoint

This actor represents an endpoint user of a GDDI Message interface. A GDDI Endpoint can be a sender or a receiver, where the information passed between them is the GDDI Message containing the encoded metadata and data to be transferred. The GDDI Endpoint commonly takes the form of a software application that implements GDDI to encapsulate, encode, and transfer data and metadata to another GDDI Endpoint.

The metadata definitions that this actor sends and receives via the GDDI Message are previously defined by the Ground System Engineer actor. This provides compatibility between endpoints that agree to use the same metadata definitions, allowing them to interoperate.

If an endpoint is both a sender and a receiver, a conformant implementation shall forward all metadata that it is not directly consuming or transforming, and it shall be able to forward messages not intended for itself. Unknown metadata must be able to be included in subsequent derived messages to other components though an implementation may choose to allow it to be disabled per a configuration setting. For metadata that is supported by an implementation, the metadata may be changed by an implementer along with the data/payload of the message.

7.2.1.1 Sender Endpoint

A GDDI sender endpoint assembles, encodes, and transmits a GDDI Message over one of the network transports specified in the Platform Specific Model (PSM) for a given transport.

7.2.1.2 Receiver Endpoint

A GDDI receiver endpoint receives, decodes, and parses a GDDI Message so that it can process its metadata and data. Optionally, this endpoint can forward the received metadata/data to a sender endpoint that may append additional metadata (of different types) or remove metadata, then encode and transmit them via GDDI Message to another endpoint.

7.2.2 Encode Metadata

This use case represents the assembly and encoding of metadata and data into the GDDI Message format. This metadata and data are supplied by the Sender GDDI Endpoint, and the metadata definitions are supplied by the Ground System Engineer actor. The sender endpoint performs this Encode Metadata function per the specific PSM for a given encoding format. After performing metadata encoding, the sender endpoint transmits the encoded GDDI Message to the receiver endpoint via the Transport Message use case. Figure 3 provides an interaction diagram depicting this Encode Metadata use case.

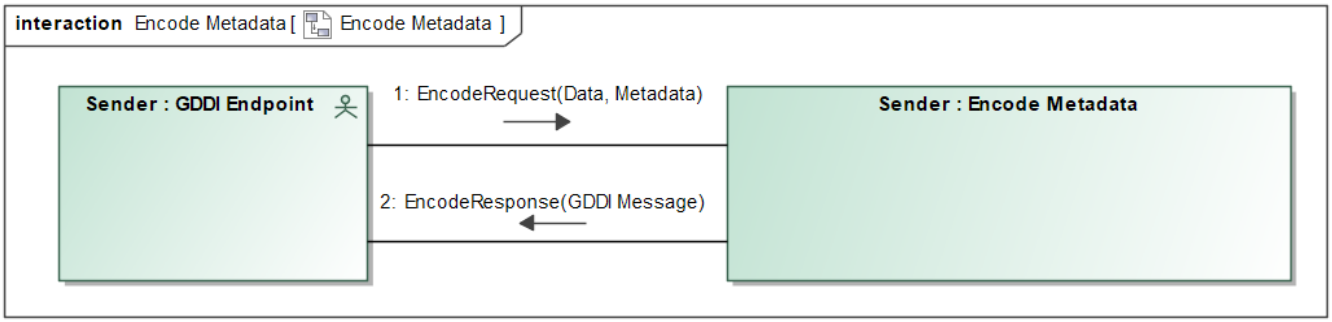

Figure 3 – Encode Metadata Use Case

7.2.3 Decode Metadata

This use case represents the parsing and decoding of the GDDI Message into the original metadata and data for processing by the receiver endpoint. The metadata definitions are supplied by the Ground System Engineer actor. The receiver endpoint performs this Decode Metadata function per the specific PSM for a given format. After performing metadata decoding, the receiver endpoint uses the decoded metadata and data to perform operations and subsequent processing. Figure 4 depicts this Decode Metadata use case.

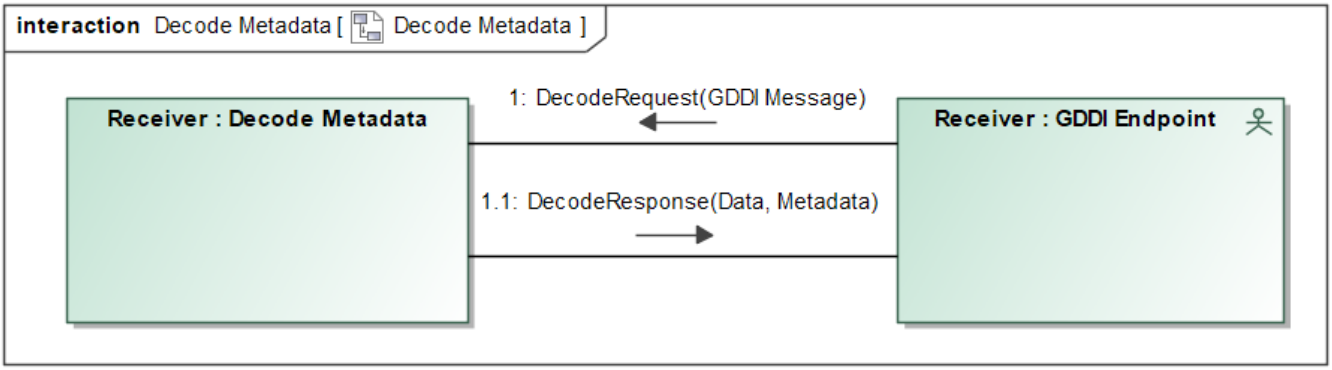

Figure 4 – Decode Metadata Use Case

7.2.4 Transport Message

This use case represents the fundamental GDDI behavior of transferring endpoint data and metadata (encoded into GDDI Messages) via transmission across a network transport channel. The type of transport is mapped from a specific GDDI PSM. The following subsections explain how GDDI supports both unidirectional (simplex) and bidirectional (duplex) message transfer over a supporting transport channel(s).

7.2.4.1 Unidirectional Transfer

Figure 5 shows the unidirectional transfer of a GDDI Message where a single transport channel is used in simplex mode. The sender endpoint is shown using the transport channel to transfer the GDDI Message (containing the metadata and data) to the receiver endpoint.

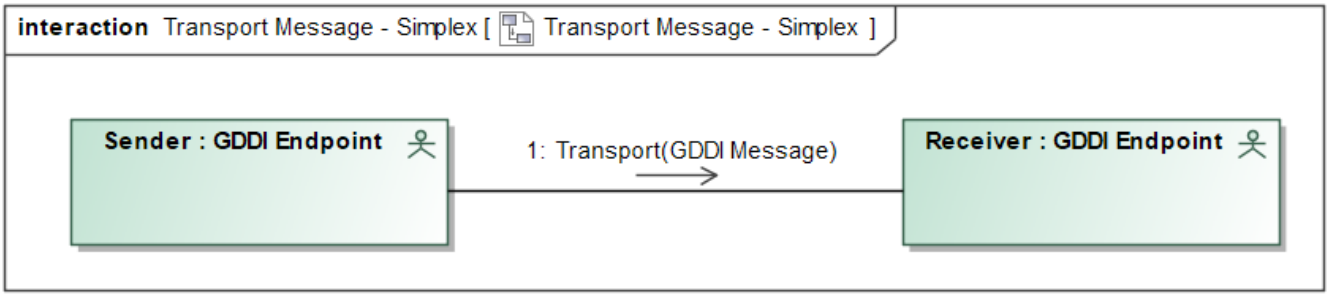

Figure 5 – Unidirectional Transfer Use Case

By specifying simplex transmission and a single transport channel, this Unidirectional Transfer use case forms the foundation for other multidirectional use cases, like the bidirectional use case described in the following sections.

7.2.4.2 Bidirectional Transfer

For bidirectional transfer, the simplex transmission specified in the GDDI PIM is extended to support duplex transmission between GDDI Endpoints. Doing so provides duplex capability where the sender endpoint can send requests concurrently or sequentially to the receiver endpoint while it sends responses back to the sender. Figure 6 shows the bidirectional transfer of GDDI Messages where separate transport channels are used to support duplex mode. This allows a receiver to send messages back to the sender (e.g., for status or display purposes).

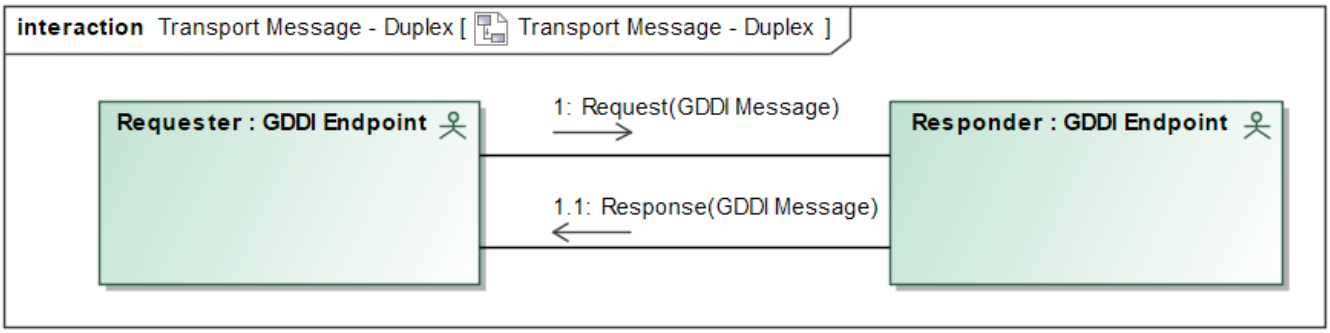

Figure 6 – Bidirectional Transfer Use Case

7.2.5 Ground System Engineer

The role of this actor is to provide GDDI metadata definitions to be used by GDDI Endpoints to encode the metadata into GDDI Messages. The Ground System Engineer commonly takes the form of a system or software developer who selects and/or defines the necessary metadata needed between endpoints within a satellite ground system. These metadata definitions are explained in the Define GDDI Metadata use case below.

7.2.6 Define GDDI Metadata

This use case represents the process where the satellite Ground System Engineer determines the metadata necessary for the GDDI endpoints (within the respective ground system) to communicate between themselves using GDDI Messages. Once the required metadata is determined, the Ground System Engineer then selects existing metadata definitions from the Standard and/or Vendor-Specific Metadata package(s) and may add any new required metadata definitions to either or both packages, as described in the Section 7.4.

These definitions include the metadata types, identifiers/tags, versions, corresponding value lengths, and value typedefs needed to build GDDI Messages in compliance with this specification.

This specification supports both standard and vendor-specific metadata definitions in order to facilitate data interface reuse as well as provide the flexibility to support vendor-specific needs. For example, Standard metadata definitions can be selected for an interface requiring GDDI Types “A” and “B” and Vendor-Specific definitions can be selected for an interface requiring a vendor’s custom Type “C” metadata. Also, vendors can extend standard GDDI Types with their own vendor-specific metadata without impacting existing GDDI endpoints that support the standard Types. This affords flexibility to the Ground System Engineer to design data-plane communications that lever an open standard interface (GDDI) using industry standard metadata definitions, with the ability to use Vendor-Specific metadata definitions when required.

7.3 GDDI Message Design

The central concept of GDDI is the **GDDI Message** used to transfer data and metadata. This construct consists of a **GDDI Header** containing the necessary attributes for defining the message envelope, like Sync Marker, Version, Total Length, etc. The GDDI Message also contains the raw **Data/Payload** carrying the baseband data whose encoding format can be included as GDDI metadata, or agreed upon between sender and receiver endpoints. Most importantly, the GDDI Message contains multiple *types* of metadata, grouped into *blocks* of related metadata called the **Type Block**, also referred to as the Type-Tag-Length-Value (TTLV) construct.

Within a given *type* of metadata, there can be one-to-many metadata **Values**, each identified with a metadata **Tag** and a **Length** (of the value); when combined is referred to as the Tag-Length-Value **(TLV) Triplet**. This multi-dimensional metadata elasticity is the crux of the GDDI concept, as depicted above in Figure 1 where multiple types of metadata are appended at various points across the data chain, with a dynamic number of metadata values per type. The TLV triplet Length (of value) field supports not only primitive metadata types, but larger more complex metadata values.

Figure 7 below provides the complete UML model of the GDDI Message, its Header, its Type Block, and its Data/Payload.

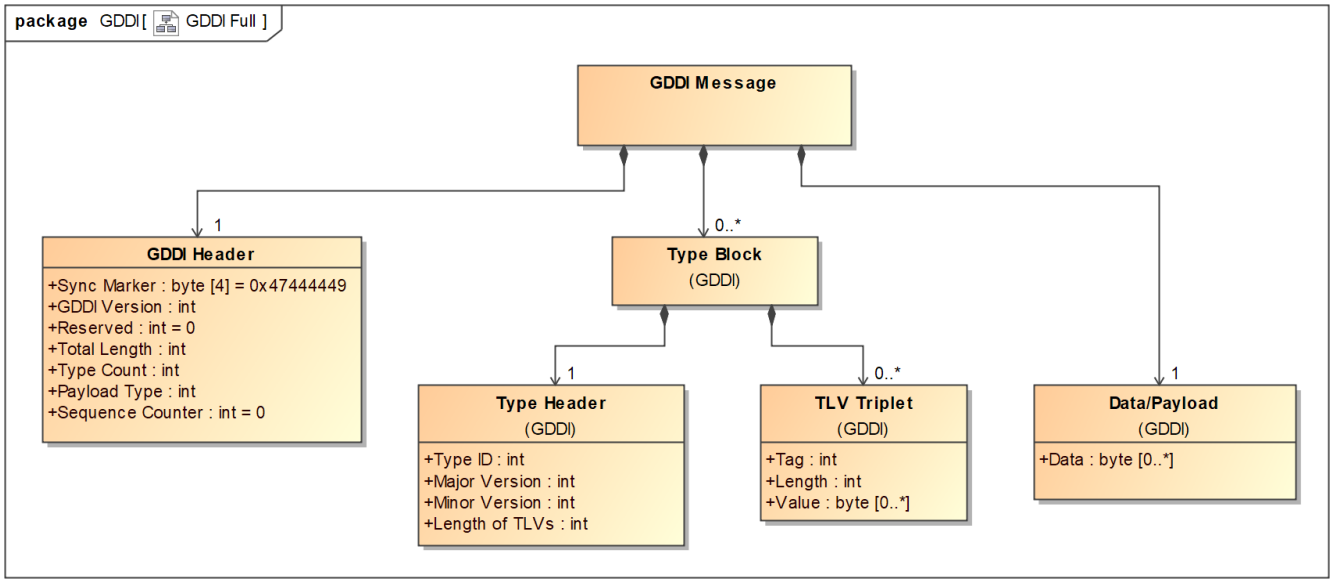

Figure 7 – GDDI Message UML Class Diagram

7.3.1 GDDI Message

This message is the primary data construct transferred between GDDI Endpoints. The GDDI Message is a concrete class composed of three other classes: the GDDI Header, Type Block, and Data/Payload. The encoding of the composed GDDI Message fields is specified by a GDDI Encoding PSM. This message structure provides the basis for the PSM encoding. The transport of GDDI Messages is specified by another GDDI PSM, described herein. The case where only Data/Payload needs to be transferred is supported by appending no Type Blocks to the GDDI Message; this provides a minimal message encapsulation when no metadata is needed but is not the nominal use case for GDDI.

7.3.2 GDDI Header

A single GDDI Header is used to envelope the subsequent contents of a GDDI Message. This header consists of the following attributes that are encoded prior to transport per the selected PSM:

- **Sync Marker**: This attribute consists of the message-leading byte pattern for receivers to find and extract GDDI Messages from stream-based transports, such as TCP/IP which provides a byte stream with no encapsulation. This byte pattern shall be set to the following hexadecimal values: 0x47 0x44 0x44 0x49 (i.e., the ASCII character values: “G” “D” “D” “I”, respectively). These values shall be fixed for all interfaces using GDDI Messages.

- **GDDI Version**: This attribute represents the *numeric version* of the GDDI Message structure for a *given* Encoding PSM. This provides separate versioning per Encoding PSM to allow for changes to the encoded field formats as well as for PIM-level changes to the overall structure of the message (which would increment the version for all encoding PSMs). For example, GDDI Version for Encoding PSM#1 could be 4 while for Encoding PSM#2 it could be 0. The GDDI Version attribute is zero-based, where version value of 0 indicates logical version 1. Note: this GDDI Version attribute does not correlate with the document version of this GDDI specification. Supported GDDI Versions are explicitly stated in each Encoding PSM.

- **Reserved**: This attribute is a placeholder for future interface use and shall be assigned to all zero bits in all PSMs.

- **Total Length**: This integer attribute represents the length (in octets) of the entire encoded GDDI Message. This attribute, combined with the Sync Marker, is used by GDDI receiver endpoints to determine the end of the received GDDI Message so that it can be extracted from various stream-oriented transports like TCP. This attribute is also used in combination with the *Type Count* and *Length of TLVs* attributes to determine the size of the Data/Payload.

- **Type Count**: This attribute represents the *number of Type Blocks* contained in the GDDI Message. It is used by GDDI receiver endpoints to determine the last Type Block in the message so it can locate the beginning of the Data/Payload.

- **Payload Type**: This attribute indicates which of the Types (within the current GDDI Message) contains the primary metadata describing the Data/Payload being transferred, i.e., the Type ID of the most recent data transformation.

- **Sequence Counter**: This attribute is incremented by one for each successive GDDI Message transmitted from a given Sender GDDI Endpoint to its corresponding Receiver GDDI Endpoint. This numeric attribute starts at the value zero and increments by one for each GDDI Message transmitted. It is used by the receiver to detect any missing or out-of-order messages if the underlying PSM does not guarantee them.

7.3.3 Type Block

A Type Block represents a specific category of metadata for a given ground system processing function. A GDDI Endpoint can append zero or more Type Blocks to a GDDI Message to fully describe the data as it transits multiple processing functions across the ground system. A Type Block consists of a Type Header and zero or more TLV Triplets, per below. There is no required or fixed order for Type Blocks, so receivers shall handle any ordering, i.e., if a Type is not needed or is unknown, it must skip to the next Type using the Length of TLVs attribute provided in the Type Header.

7.3.4 Type Header

The Type Header is used to start the beginning of a Type Block and describes the block's contents. This header consists of the following attributes that are encoded prior to transport per the selected PSM:

- **Type ID**: This numeric attribute is a unique identifier for a given metadata type and represents the category of all subsequent TLV triplets in the corresponding Type Block identified by this Type ID. See section 7.3.6 for how this attribute is used to support standard and vendor-specific Type Blocks.

- **Major Version**: This integer attribute represents the *incompatibility* portion of the combined Major.Minor version. When a Major Version increments, it means that existing TLV(s) for that Type ID have been modified and/or deleted, making the new Major version incompatible with previous Major versions.

- **Minor Version**: This integer attribute represents the *compatibility* portion of the combined Major.Minor version. When a Minor Version increments, it means that new TLV(s) for that Type ID have only been added, making the new Minor version backward compatible with previous Minor versions of the same Major version.

- **Length of TLVs**: This integer attribute represents the length (in octets) of all TLV triplets in the corresponding Type Block; excludes the length of the Type Header. Receiver endpoints use the Type ID and Length of TLVs to process or ignore TLVs for a given Type. A zero Length of TLVs shall be allowed where the Type Block includes a Type Header with no TLVs; this is to support the use case where *optional* TLVs are not transmitted, but the sender wants to indicate presence of that Type to the receiver.

7.3.5 Tag-Length-Value (TLV) Triplet

The Tag-Length-Value Triplet is the foundational construct of the GDDI Message. The Tag member identifies the individual metadatum, the Length member indicates the length of the metadatum Value. And the Value member provides the actual value of the metadatum. There is no required or fixed order for TLVs within the Type Block, so receivers shall handle any ordering, i.e., if a Tag is not needed or is unknown, it must skip to the next Tag using the Length attribute in the TLV triplet. The TLV Triplet consists of the following three attributes that are encoded prior to transport per the selected PSM:

- **Tag**: This numeric attribute is an identifier for a given metadatum item, referred to as a TLV triplet. Tag numbers are not required to be unique, where multiple instances of the same tag within a message are allowed and shall be handled accordingly by implementations. This allows the same Tag number to be repeated within a given Type to provide multiple Values for a common Tag, e.g. multiple "IP address" Tags could exist for a Type that represents all IP addresses for a certain host, each with a different Value. Unique Tag numbers are also supported to provide groupings of different yet related metadata for a given Type. The semantics regarding Tag uniqueness are determined and set for each Type ID a-priori so that GDDI receiver endpoints can properly parse and decode the TLV triplets. See section 7.3.6 for how this attribute is used to support standard and vendor-specific TLVs.
- **Length**: This numeric attribute indicates the length in octets of the Value attribute. This provides flexibility to support small or large Values while maintaining a lightweight and elastic metadata interface. A zero Length shall be allowed to support the use case of Tags without Values, e.g. A "middle name" Tag could have no Value.
- **Value**: This attribute is an array of bytes representing the metadatum value for the given Tag. The length of this value is specified by the Length attribute of the TLV. The format and data type of this value attribute is known based on the specific Tag representing it, and maps to one of the definitions in the selected Encoding PSM.

7.3.6 Standard and Vendor-Specific Metadata

A GDDI message may carry metadata containing one or more of the following:

1. *Standard-Only* Type Block(s) containing TLV(s) with concrete definitions from the Standard Metadata Package as described in section 7.4.1, i.e., contains no vendor-specific metadata.
2. A *standard* Type Block containing standard TLV(s) *followed by* one or more vendor-specific TLV(s). This is how vendors can *extend* a standard Type, i.e., combined metadata from Standard *and* Vendor-specific packages.
3. *Vendor-Only* Type Block(s) containing TLV(s) with concrete definitions from a *Vendor-Specific* Metadata Package as described in section 7.4.2, i.e., contains vendor-only Type(s) with a single vendor's metadata per Type. These Type Blocks use a single reserved **Vendor-Only Type ID** (prescribed in the Encoding PSM) to indicate that the Type Block only contains vendor-specific metadata.

For cases #2 and #3 above, vendor-specific TLV(s) shall be preceded with a **Vendor-Identifying TLV** consisting of a reserved **Vendor ID Tag**, a fixed Length, and a **Vendor ID Value** containing a unique identifier of that specific vendor. This Vendor ID Tag and Length are prescribed in the Encoding PSM, and the *Vendor ID Values* are prescribed in a separate repository/website as referenced in section 7.4.2 of this specification. Also refer to section 8.2.2 for an example GDDI message carrying metadata for all three of the above cases.

7.3.7 Data/Payload

The Data/Payload portion of the GDDI Message is an array of bytes containing the variable-length payload data described by the attached metadata in the GDDI Header and Type Block(s). The format and encoding of this payload data can be included as GDDI metadata, or agreed upon between endpoints.

The length of this data/payload is determined by the GDDI Header Total Length less the lengths of all Type Blocks, where the length of each Type Block is determined by the fixed length of the encoded Type Header and the encoded Length of TLVs attribute.

In the case where the usable length of the data/payload is not an integral number of octets (bytes), a TLV can be used to specify a bit-length. For example, a TLV could be specified to represent the "Length in Bits", where the data length could be 15 bits carried within a 2 byte Data/Payload field, i.e., only the first 15 most significant bits would be valid.

7.4 GDDI Package Design

GDDI specifies the message structure and interaction between GDDI Endpoints. It does not specify the exact metadata definitions (i.e., types, identifiers/tags, versions, corresponding value lengths, and value typedefs) for any given interface. That is beyond the scope of this specification. Instead, GDDI delineates the approach to use when defining specific metadata; as well as how that metadata is structured and encoded in GDDI Messages for transport between endpoints.

When first defining a given interface between GDDI Endpoints, the Ground System Engineer provides the concrete metadata definitions that are compliant with the PIM and PSM used. To represent this interaction, GDDI specifies two notional packages: the Vendor-Specific Metadata package, containing definitions of concrete metadata that meet the GDDI specification but are custom to a given vendor, and the Standard Metadata package also containing concrete metadata definitions but are standardized among multiple vendors to ensure interoperability among them. While not included in this document, it is envisioned that the Standard package eventually becomes a machine-readable, publicly available XML metadata dictionary hosted in a separate repository linked from this specification or OMG website. Each Vendor package (maintained by the respective vendors) may take a similar XML dictionary approach. Both packages contain the concrete metadata definitions, including Type IDs, Type Versions, Tags, corresponding Value lengths, etc**Error! Reference source not found.**. Figure 8 shows the GDDI packages.

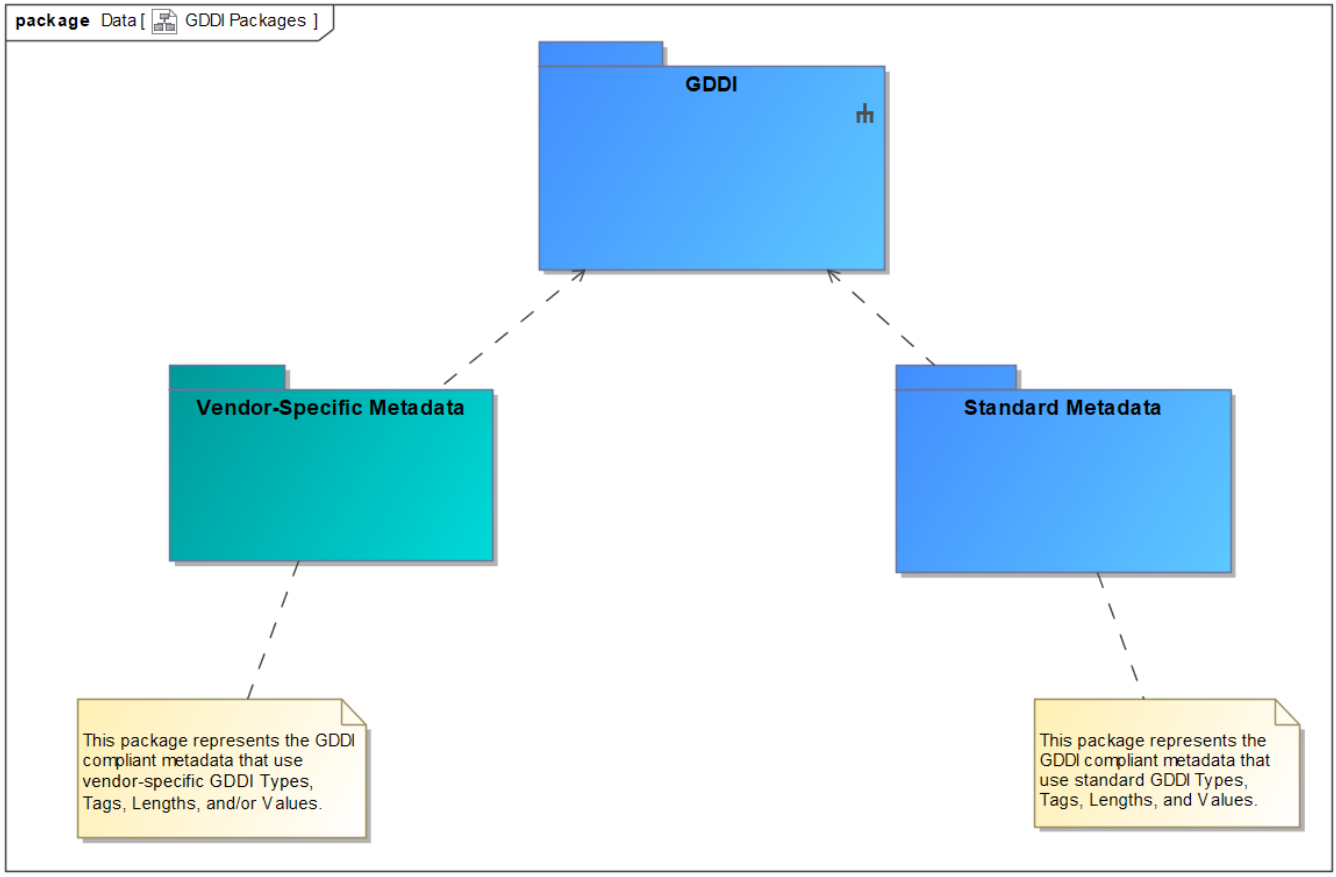

Figure 8 – GDDI Packages

When supporting the Standard Metadata package, a GDDI implementation enables customers to easily configure the implementation for interoperable communication of baseband metadata and data following the GDDI model. When GDDI implementations use Vendor-Specific packages, interoperability between different vendors is not guaranteed.

Metadata definitions supported by a GDDI implementation shall be available via one or both of these packages which can be distributed by the vendor via product documentation, an API provided by a GDDI implementation, or via XML dictionary.

Refer to the following subsections for more details on GDDI metadata definitions (packages) and the planned use of XML data dictionaries.

7.4.1 Standard Metadata Package

As mentioned in the GDDI Package Design, it is envisioned that the Standard Metadata package eventually becomes a machine-readable XML metadata dictionary available in a separate repository linked from this specification or OMG website (e.g., wiki or GitHub repository).

This dictionary will provide the concrete definitions for Standard GDDI metadata. A *concrete* definition is an explicit assignment of a given PIM attribute or supporting property, e.g., Type ID = 7, where '7' is the concrete definition.

This allows GDDI implementers and users to retrieve these standard definitions from the referenced external repository. Concrete metadata definitions provide Type-Tag-Length-Value (TTLV) information needed for two GDDI endpoints to interoperate.

Concrete definitions for the following normative attributes shall be provided by the Standard Metadata Package (i.e., via metadata dictionary), and used to populate GDDI Message fields for a given interface as specified in the PIM and the Encoding PSM. It is preferred that Type Name and Tag Name carry well-known values to facilitate machine-readability. Optional definitions are enclosed in brackets "[]" and *italics text* is informational:

- **Type ID**
- **Type Name**
- *Per Type:*
    - **Major Version**
    - **Minor Version**
    - *Per TLV Triplet:*
        - **Tag** (ID)
        - **Tag Name**
        - **Length** (of Value)
        - **Value Type** (per the Encoding PSM)
        - **[Value Units]**
        - **[Tag Description]**

In addition to the Standard metadata dictionary provided via OMG-managed site, GDDI implementations may supply these standard metadata definitions in an Interface Control Document (ICD) and/or provide an API to retrieve them programmatically from a running GDDI-enabled application.

The following Table 1 is not normative but illustrates an example of a Standard Metadata Dictionary in a generic form. This example dictionary contains concrete metadata definitions for five standard Types (IDs 1 – 5). To provide the flexible/elastic interface, GDDI messages can transfer one or more Types depending on the use case.

Table 1. Example GDDI Standard Metadata Dictionary

Type ID	Type Name	Major Ver	Minor Ver	Tag (ID)	Tag Name	Length (of Value)	Value Type	Value Units	Tag Description
1	Raw	1	0	1	Sequence Number	2	unsigned short	count	Sequence Number used to detect out-of-order and/or lost data
				2	Data Rate	8	double	bits per sec	Bit Rate (in bits per second) of the underlying Raw data
				3	Time Stamp Seconds	4	unsigned long	sec	Time Stamp: Number of seconds since 00:00:00 Jan 1, 1970 (UTC), minus leap seconds
				4	Time Stamp Nanosec	4	unsigned long	nsec	Time Stamp: Nanoseconds representing the fractional portion of the timestamp
				5	Data Length	8	unsigned long long	bits	Length in bits of the Raw data payload
2	Frame	1	2	1	FrameSync Lock State	1	octet	enum	Lock State of the Frame: 0=stopped, 1=search, 2=verfiy, 3=lock, 4=check, 5=noSignal
				2	Bits Slipped	2	short	bits	Bits slipped: negative value slipped to the left, positive value slipped to the right
				3	Time Stamp Seconds	4	unsigned long	sec	Time Stamp (POSIX): Number of seconds since 00:00:00 Jan 1, 1970 (UTC), minus leap seconds
				4	Time Stamp Nanosec	4	unsigned long	nsec	Time Stamp (POSIX): Nanoseconds representing the fractional portion of the timestamp
				5	Frame Length	4	unsigned long	bits	Length in bits of the Frame data payload
3	FEC	1	1	1	Bits Corrected	2	unsigned short	bits	Forward Error Correction (FEC) Bits Corrected in this frame
				2	Uncorrectable	1	boolean	true/false	Forward Error Correction (FEC) Uncorrectable Frame flag
				3	ASM Vector	0	octet array	bytes	Attached Sync Marker (*Length of Value of zero indicates this TLV is variable-length*)
4	CCSDS Transfer Frame	2	0	1	Spacecraft Identifier	2	unsigned short	ID	SpaceCraft ID from the received CCSDS Data Link Transfer Frame Header
				2	Virtual Channel ID	1	octet	ID	Virtual Channel ID from the received CCSDS Data Link Transfer Frame Header
				3	CRC Error	1	boolean	true/false	Indicates if CRC error(s) for this CCSDS Transfer Frame have been detected
5	IP Addrs	1	3	1	IP Address	0	string	address	IPv4 address in dot notation *(Length of Value of zero indicates this TLV is variable-length)*
				1	IP Address	0	string	address	IPv4 address in dot notation *(Length of Value of zero indicates this TLV is variable-length)*
				1	IP Address	0	string	address	IPv4 address in dot notation *(Length of Value of zero indicates this TLV is variable-length)*

7.4.2 Vendor-Specific Metadata Package

To support the Vendor-Specific Metadata package per the GDDI Package Design, vendors may extend the metadata of a given *Standard Type* by adding their own TLV definitions and/or create their own *Vendor Types* with vendor-specific TLV definitions. These options are discussed in section 7.3.6.

This capability provides vendor flexibility to on-ramp and/or innovate while using the standard GDDI interface. It allows unknown/unexpected vendor metadata to be easily parsed or skipped without impacting existing GDDI implementations.

Vendors may use a similar metadata dictionary approach as shown in Table 2 below (not normative). This example dictionary contains one additional column named "Vendor ID" to identify the vendor's TLV triplet(s) per Type Block.

For the first Type Block with Type ID 2, Vendor IDs 11 and 22 have *extended* the Standard "Frame" type with their own vendor metadata. Note how the first TLV for a given vendor's metadata must be the *Vendor-Identifying TLV* with Tag 255, Length of 1, and a Value equal to the *Vendor ID* as described in section 7.3.6.

The next two Type Blocks use Type ID 255 (a reserved value per the Encoding PSM) to indicate that the subsequent TLV(s) for each Type Block only contain metadata for the vendor specified by the first *Vendor-Identifying TLV*. In this example, these two Type Blocks contain metadata for Vendor IDs 33 and 44, respectively.

Vendor ID Values containing unique identifiers of each vendor used to support vendor-specific metadata will be provided in a separate repository linked from this specification or OMG website.

Table 2. Example GDDI Vendor-Specific Metadata Dictionary

Type ID	Type Name	Major Ver	Minor Ver	Tag (ID)	Tag Name	Length (of Value)	Vendor ID	Value Type	Value Units	Tag Description
2	Frame	1	2	255	Vendor ID	1	11	octet	ID	Vendor ID (*255 is the reserved Tag ID used to indicate Vendor ID in its value*)
				1	Frame Data Inverted	1	11	boolean	true/false	Indicates if the frame data was detected to be inverted based on the sync pattern
				2	Frame Antenna Name	0	11	string	name	Name of the ground antenna site that sourced this frame data
				255	Vendor ID	1	22	octet	ID	Vendor ID (*255 is the reserved Tag ID used to indicate Vendor ID in its value*)
				1	Frame Sync Lost	1	22	boolean	true/false	Indicates if the frame synchronization was lost since the previous frame
255	Vend33	1	0	255	Vendor ID	1	33	octet	ID	Vendor ID (*255 is the reserved Tag ID used to indicate Vendor ID in its value*)
				1	Vendor33 Meta X	4	33	float	widgets	Vendor33 description for this metadatum
				2	Vendor33 Meta Y	8	33	unsigned long long	widgets	Vendor33 description for this metadatum
255	Vend44	1	0	255	Vendor ID	1	44	octet	ID	Vendor ID (*255 is the reserved Tag ID used to indicate Vendor ID in its value*)
				1	Vendor44 Meta A	2	44	short	widgets	Vendor44 description for this metadatum
				2	Vendor44 Meta B	0	44	octet array	widgets	Vendor44 description for this metadatum

Vendor-Specific metadata definitions may be supplied via a metadata dictionary (preferably a similar XML format as the standard metadata dictionary), an Interface Control Document (ICD) and/or an API to retrieve them programmatically from a running GDDI-enabled application.

7.5 GDDI Security

Security has been considered for this specification, though it does not apply at the PIM level. Specific security controls are not specified by the GDDI PIM because it does not define a network transport. See the Security section 9.2 in the PSM-TCP/IP Network Transport for details of how the PSM supports security.

7.6 GDDI Internationalization

GDDI supports international use as there are no specific regional type definitions in the PIM. Encoding PSM section 8.1.1 references section 7.10.2.6 of [CORBAINTEROP] which states that strings are encoded using UTF-8, allowing for the representation of characters within the Unicode character set that are supported internationally. All other GDDI-specified types are internationally supported.

8 PSM-CORBA Encoding

This Encoding PSM specifies Data Type Definitions and CORBA Encoding Formats for all GDDI Message attributes other than the Data/Payload as described in Section 7.3.7. Additionally, CORBA provides established floating-point encoding formats via the ANSI/IEEE 754 Standard. By levering this subset of CORBA, this PSM provides encoding formats that are interoperable among many processing platforms as well as across a variety of transport mechanisms.

8.1 PIM to PSM Mapping

For the GDDI PIM to PSM (CORBA Encoding) mapping, each attribute in the GDDI Message and its containing members shall adhere to the following mapping rules, which shall be considered normative. All ranges and attribute semantics are described in the corresponding PIM attributes, unless otherwise stated.

8.1.1 Attributes

PIM *attributes* map to encoded PSM *fields*. Each attribute called out in the GDDI Message UML Class Diagram in Figure 7 shall map to one of the CORBA Encoding Formats called out in Table 2. The subsequent "Format" sections specify exactly which Data Type/CORBA Encoding is used for each field that is mapped from a PIM attribute.

The *Value* attribute in the TLV Triplet shall also map to one of the below Data Type definitions, or may be omitted if a Value is not required, i.e., an empty Value. Refer to Section 8.1.5.3 for details.

Table 3. Attribute to CORBA Mapping

PIM Attribute Data Type	PSM-CORBA Encoding Format (Size, Bit Ordering)
short, ushort, long, ulong, long long, unsigned long long	Per Section 9.3.1.2 of [CORBAINTEROP]
float, double	Per Section 9.3.1.3 of [CORBAINTEROP]
octet	Per Section 9.3.1.4 of [CORBAINTEROP]
boolean	Per Section 9.3.1.5 of [CORBAINTEROP]
string	Per Section 7.10.2.6 of [CORBAINTEROP]
octet array	Per Section 9.3 (octet stream) of [CORBAINTEROP]

All above data types shall be encoded with big-endian ordering per the referenced CORBA section prior to transport.

To provide the required flexibility and compact efficiency, enumerations can use any of the integer types above.

Since GDDI provides an interface that carries data and metadata over an octet stream, GDDI Message attributes with the above types shall be *adjacently packed on octet boundaries* within the octet stream and aligned on boundaries as described in Section 9.3.1.1 and Table 9.1 of [CORBAINTEROP].

8.1.2 GDDI Header Format

A fully encoded GDDI Header shall adhere to the format in Figure 9 with the specified CORBA encodings and octet lengths for each field mapped from the corresponding PIM attribute. The GDDI Header is the first member of the GDDI Message, hence is transmitted first over the selected transport. Transmission starts with the Most Significant Bit (MSB) 0 of the first byte[0] Sync Marker field as shown. Additional detail is provided in the subsections below the figure.

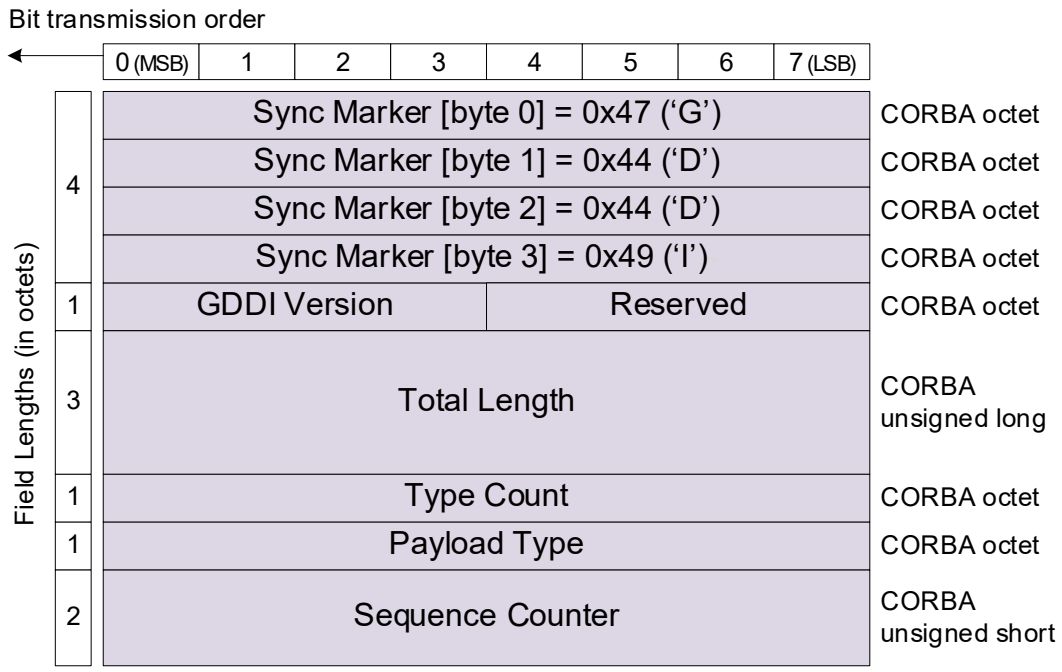

Figure 9 – GDDI Header: CORBA Encoded Format

8.1.2.1 Sync Marker

This field is directly mapped from the PIM attribute whose type is a byte array of length 4. Each byte is encoded as a CORBA octet, which are considered as unsigned 8-bit integer values per [CORBAINTEROP]. The required values of each byte in this array directly map from the default value of this PIM attribute.

8.1.2.2 GDDI Version

For this PSM, the GDDI Version integer attribute from the PIM maps to an unsigned 4-bit integer that resides in the most significant four bits of the fifth octet in the encoded GDDI Header, as shown in the above figure. The range for this field shall be 0 to 15, meaning 16 unique versions per Encoding PSM.

The currently supported GDDI Versions for this CORBA-Encoding PSM are:

- 0 (representing Message Version 1)

Note: New versions shall be added to this list whenever this PSM encoding format changes or the PIM structure changes.

8.1.2.3 Reserved

This integer PIM attribute maps to an unsigned 4-bit integer that resides in the least significant four bits of the fifth octet in the encoded GDDI Header, as shown in the above figure. This field is reserved for future use and shall be assigned to all zero bits.

8.1.2.4 Total Length

This integer PIM attribute maps to a CORBA unsigned long whose most significant 8-bits are unused, resulting in a 24-bit (3 octet) field as shown in the above figure. When encoding this field, a CORBA unsigned long shall be populated with the Total Length of the GDDI Message, then after formatting into big-endian order, the MSB octet 0 is discarded and octets 1, 2, and 3 of the Big-Endian *long* (per [CORBAINTEROP] Figure 9.1) shall be placed into the Total Length field. The range for this field shall be 12 to 16,777,215. The minimum length equates to only the 12-byte GDDI Header being transferred with no Type Blocks and a zero length Data/Payload (supported but not a nominal use case).

8.1.2.5 Type Count

This integer PIM attribute maps to the CORBA octet type and is encoded as an unsigned 8-bit integer value per Section 9.3.1.4 of [CORBAINTEROP]. The range for this field shall be 0 to 255. Nominally Type Count will be 1 or greater, however zero is allowed for the rare case where only Data/Payload needs to be transferred with no Type Blocks.

8.1.2.6 Payload Type

This integer PIM attribute maps to the CORBA octet type and is encoded as an unsigned 8-bit integer value per Section 9.3.1.4 of [CORBAINTEROP]. The range for this attribute shall be 0 to 254. The Payload Type shall only be set to zero when Type Count is zero, otherwise it shall be 1 to 254. Payload Type 255 is reserved.

8.1.2.7 Sequence Counter

This integer PIM attribute maps to the CORBA unsigned short type and is encoded as an unsigned 16-bit integer value per Section 9.3.1.2 of [CORBAINTEROP]. This field is ordered as a Big-Endian *short* per [CORBAINTEROP] Figure 9.1. The range for this attribute shall be 0 to 65,535, hence is incremented for each transmitted GDDI message by modulo-65,536.

8.1.3 Type Block

The Type Block is composed of one Type Header and 0 to 255 TLV Triplets. The following two sections describe the encoding formats of these Type Block components.

8.1.4 Type Header Format

A fully encoded Type Header shall adhere to the format in Figure 10 with the specified CORBA encodings and octet lengths for each field mapped from the corresponding PIM attribute. Additional encoding details of each field are provided below.

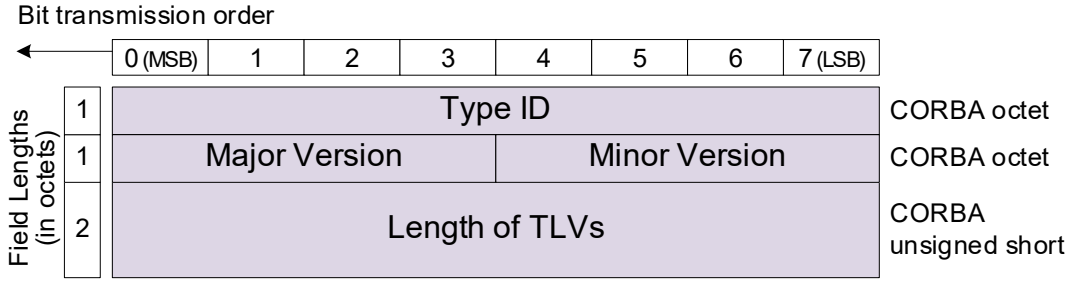

Figure 10 – Type Header: CORBA Encoded Format

8.1.4.1 Type ID

This integer PIM attribute maps to the CORBA octet type and is encoded as an unsigned 8-bit integer value per Section 9.3.1.4 of [CORBAINTEROP]. The range for this attribute shall be 1 to 254, where Type IDs 0 and 255 are reserved. Reserved **Type ID 255** shall be used by GDDI implementations for the **Vendor-Only Type ID** as described in section 7.3.6.

8.1.4.2 Major Version

For this PSM, the Type's Major Version integer attribute from the PIM maps to an unsigned 4-bit integer that resides in the most significant four bits of the first octet in the encoded Type Header, as shown in the above figure. The range for this attribute shall be 0 to 15, meaning 16 unique Major versions per Type.

The Major Version for each Type is maintained in the Standard or Vendor-Specific Metadata Package per Section 7.4.

8.1.4.3 Minor Version

For this PSM, the Type's Minor Version integer attribute from the PIM maps to an unsigned 4-bit integer that resides in the least significant four bits of the first octet in the encoded Type Header, as shown in the above figure. The range for this attribute shall be 0 to 15, meaning 16 unique Minor versions for a given Major Version, per Type.

The Minor Version for each Type is maintained in the Standard or Vendor-Specific Metadata Package per Section 7.4.

8.1.4.4 Length of TLVs

This integer PIM attribute maps to the CORBA unsigned short type and is encoded as an unsigned 16-bit integer value per Section 9.3.1.2 of [CORBAINTEROP]. This field is ordered as a Big-Endian *short* per [CORBAINTEROP] Figure 9.1. The range for this field shall be 0 to 65,535, where a zero Length of TLVs is described in the corresponding PIM attribute.

8.1.5 TLV Triplet Format

A fully encoded TLV Triplet shall adhere to the format in Figure 11 with the specified CORBA encodings and octet lengths for each field mapped from the corresponding PIM attribute. Additional encoding details of each field are provided below.

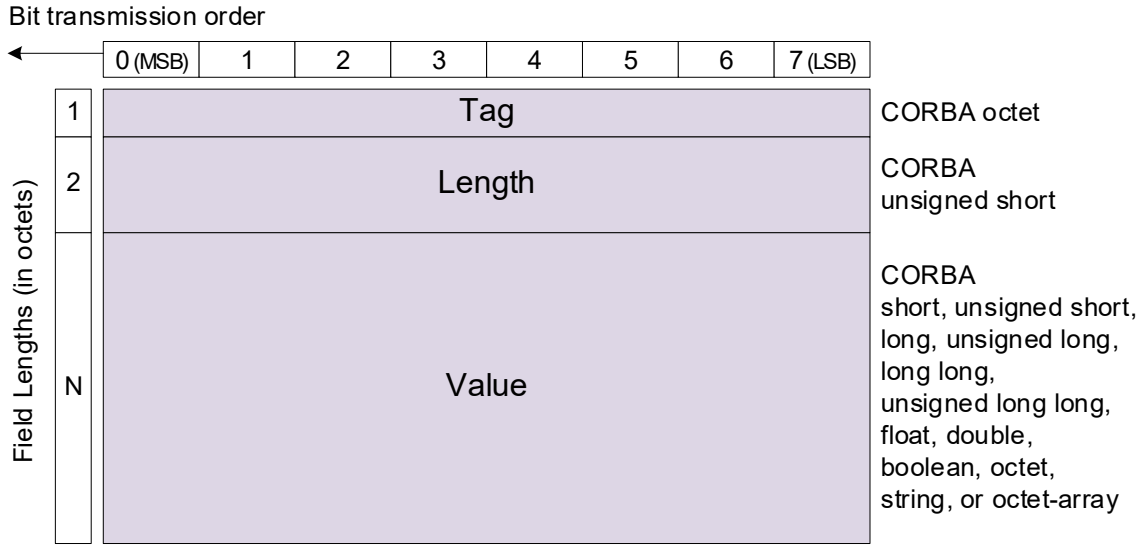

Figure 11 – TLV Triplet: CORBA Encoded Format

8.1.5.1 Tag

This integer PIM attribute maps to the CORBA octet type and is encoded as an unsigned 8-bit integer value per Section 9.3.1.4 of [CORBAINTEROP]. The range for this attribute shall be 1 to 254, where Tags 0 and 255 are reserved. Reserved **Tag 255** shall be used by GDDI implementations for the **Vendor ID Tag** as described in section 7.3.6.

8.1.5.2 Length

This integer PIM attribute maps to the CORBA unsigned short type and is encoded as an unsigned 16-bit integer value per Section 9.3.1.2 of [CORBAINTEROP]. This field is ordered as a Big-Endian *short* per [CORBAINTEROP] Figure 9.1.

The range of this 16-bit integer is 0 to 65,531 representing the length in octets of the TLV's Value. A zero length indicates that the Value field does not exist. The maximum range of 65,531 octets is derived from the maximum "Length of TLVs" field which is a CORBA ushort (65,535 octets) less 4 octets for the encoded length of a Type Header.

Lengths from 1 to 8 bytes are used for common primitive value types, i.e., octet to long long. Lengths larger than 8 bytes are used for string and octet array values.

8.1.5.3 Value

This PIM attribute of type byte with multiplicity of [0..*] shall map to this PSM in one of two methods, as follows:

- Value maps to one of the Data Types/CORBA Encodings in Table 2 as indicated by the corresponding Tag within the given Type. The Value field's length N is 1 to 8 octets for primitive types or may be larger than 8 bytes for string and octet-array types, with length N of 1 to 65,531 octets.

- A TLV with an empty Value shall map as a valid Tag, a zero Length (of value), and no Value field encoded. The Value field's length N is 0 octets, i.e., no Value exists in the encoded GDDI Message.

8.1.6 Data/Payload Format

This PIM attribute of type byte with multiplicity of [0..*] shall map to this PSM in one of two methods, as follows:

- Data/payload shall map as an octet array whose encoding format can be included as GDDI metadata or agreed upon between the sender and receiver GDDI endpoints.
- When no data/payload is required to be transferred between GDDI endpoints, the Total Length field shall only include the lengths of the GDDI Header and Type Block(s) to indicate zero octets for the Data/Payload field.

8.1.7 GDDI Message Format

A full GDDI message shall consist of the above elements per Figure 12, which shows a basic GDDI use case. The number of Types, the number and size of the TLV Triplets, and the length of Data/Payload is determined for each specific use case.

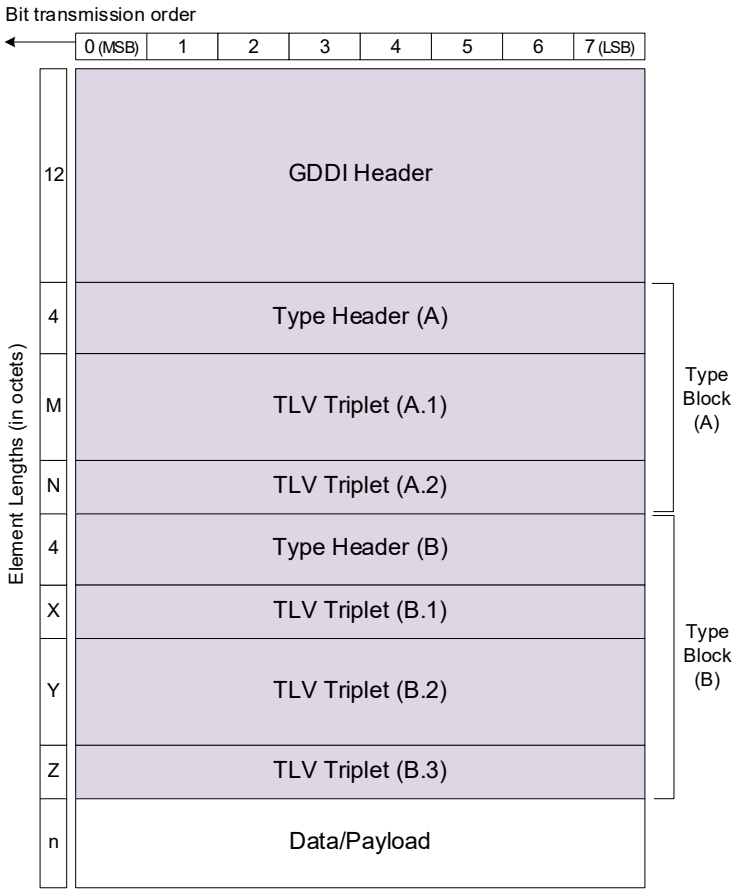

Figure 12 – Complete GDDI Message

8.2 GDDI Message Examples

The diagrams in the following subsections are intended to be an aid in comprehension of this encoding PSM as well as the overall GDDI metadata approach. They are not normative.

8.2.1 Example GDDI Message with Standard Metadata

Figure 13 depicts an example GDDI message with generic PSM-CORBA encodings of *standard* metadata. It is not normative. This diagram shows all constructs of the GDDI message: the GDDI Header, three example Type Blocks (labelled 'A', 'B', and 'C', delineated with brackets on the left side), each Type Block containing several example TLV Triplets of varying lengths, and a variable-length Data/payload. This example helps to show the elastic metadata concept, and the structure where *concrete* metadata definitions (from the metadata dictionary) are to be applied within an encoded message. It also shows how the GDDI message provides a structure that can be readily parsed by Cross Domain Solutions (CDS) via the self-describing length and other supporting fields.

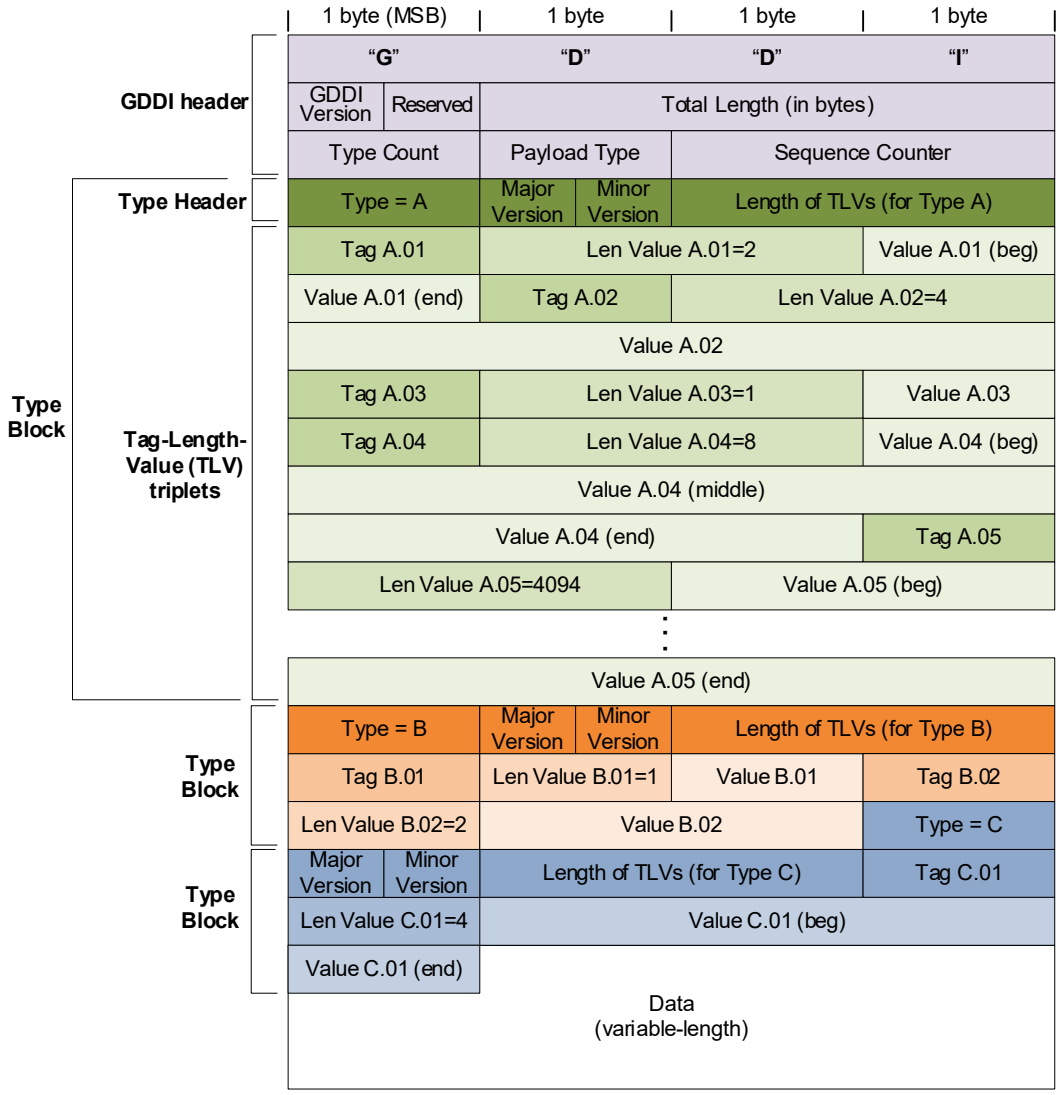

Figure 13 – Example GDDI Message with Standard Metadata

8.2.2 Example GDDI Message with Vendor-Specific Metadata

Figure 14 depicts an example GDDI message with generic PSM-CORBA encodings of *vendor-specific* metadata. It is not normative. This diagram shows how a GDDI message containing standard metadata can be extended with vendor-specific metadata either by: 1) adding it to a standard GDDI Type, or 2) adding completely new Vendor-Only Type Block(s).

In the diagram, standard metadata is shown by "Type A" fields in black text, and vendor-specific metadata is shown by "Type A" fields in red and purple text, where two different vendors (vendor ID 11 and vendor ID 22) extend Type A with their own metadata TLV triplets as described in section 7.3.6, #2.

The orange and blue Type Blocks use the Vendor-Only Type ID (255) combined with the Vendor-Identifying TLV to indicate the vendor-only metadata/TLVs for both vendorID 33 and vendor ID 44 as described in section 7.3.6, #3.

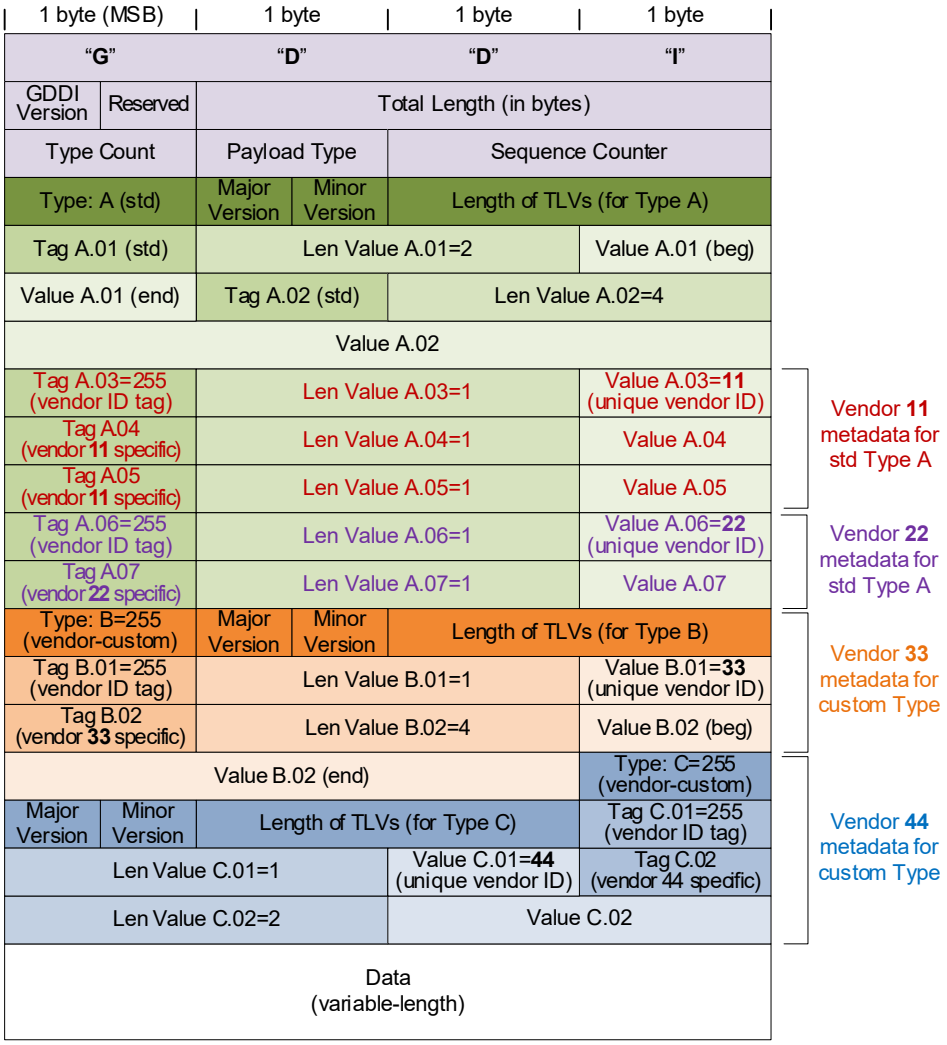

Figure 14 – Example GDDI Message with Vendor-Specific Metadata

9 PSM-TCP/IP Network Transport

This network transport PSM specifies a TCP/IP-based formatting for all message types supported by GDDI. Implementations that follow the details of this PSM can send and receive GDDI-formatted messages in compliance with the GDDI specification.

9.1 PIM to PSM Mapping

TCP is a stream-oriented protocol and provides built-in message delivery assurance which allows for the transport of GDDI messages to be reliably delivered with high Quality of Service (QoS). Further, TCP is broadly implemented in existing hardware and software utilized in ground systems, enabling it to be an implementation that can be quickly realized by implementations and compatible with incorporation into new and existing satellite ground systems.

Within the TCP/IP Network Transport PSM, GDDI CORBA/IEEE data type-encoded messages are inserted over TCP/IP. Implementations may choose to utilize additional transport features, such as Transport Level Security (TLS) to provide additional security control. However, any additional features shall be able to be turned off.

The CORBA/IEEE Types and Encoding, as specified in Section 8 shall be used for GDDI Header and TTLV Fields in the GDDI Message. The data/payload encoding format may be specified via GDDI metadata or agreed upon between endpoints.

The GDDI specification defines that each message is prepended by “G” “D” “D” “I” magic bytes. These four octets serve as a marker for the beginning of a new message and implementing clients can seek to find where to read a new message and to begin extraction of a new message from the octet stream. It is expected that data is octet-aligned. Within the TCP/IP PSM, this seeking is performed against the TCP/IP network stream after any TLS-transport decryption has taken place (if encryption is enabled).

The representation of all data types and formats shall adhere to the encoding types as specified in the PSM-CORBA Encoding, detailed in the previous section of this specification. As such, all multi-octet data types are to be represented in big endian format (as shown in [CORBAINTEROP] Section 9.3.1). In addition, the size and bit ordering across the network shall comply with [CORBAINTEROP] Section 9.3.1.

9.2 Security

The TCP/IP PSM is designed to support integration with security controls that may exist in a network infrastructure but not to require them. This allows for the greatest flexibility in supporting implementations of this specification within existing TCP/IP-based systems and is compatible with current/future technology architectures that may improve on security capabilities built into TLS and secure network transport. As such, the specific versions of TLS, ciphers, encryption mechanisms/configuration are not specified. However, it is recommended that implementations follow the latest security controls available while allowing for security controls to be disabled or adjusted to support maximum flexibility.

9.2.1 Encryption

Encryption is recommended to be supported by PSMs, optional within the TCP/IP PSM. Implementations shall allow for encryption, but it must be able to be disabled. Encryption may be directly implemented using a native implementation (such as via OpenSSL) or it could be implemented by the infrastructure hosting the software elements. Several options that could be utilized by the infrastructure include the use of a service mesh or an IPSec tunnel (such as a Virtual Private Network) while the application itself is implemented with data connections unencrypted.

9.2.2 Authentication

When the TCP/IP PSM is implemented using TLS/SSL over TCP (e.g., OpenSSL or via a service mesh), authentication can be controlled by way of trusted certificates or a trusted certificate root, as built into the standard TLS certificate implementation. While implementing TLS is optional, if implemented, implementations should support specifying certificate options, to include the client certificate, certificate trust chain/trusted certificates, and certificate revocation lists.

This page intentionally left blank.