MIPI Alliance Specification for Device Descriptor Block (DDB)

Version 0.82.01 – 30 October 2008

MIPI Board Approved 29-Oct-2008

Further technical changes to this document are expected as work continues in the DDB Working Group

NOTICE OF DISCLAIMER

The material contained herein is not a license, either expressly or impliedly, to any IPR owned or controlled by any of the authors or developers of this material or MIPI. The material contained herein is provided on an "AS IS" basis and to the maximum extent permitted by applicable law, this material is provided AS IS AND WITH ALL FAULTS, and the authors and developers of this material and MIPI hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence.

All materials contained herein are protected by copyright laws, and may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of MIPI Alliance. MIPI, MIPI Alliance and the dotted rainbow arch and all related trademarks, tradenames, and other intellectual property are the exclusive property of MIPI Alliance and cannot be used without its express prior written permission.

ALSO, THERE IS NO WARRANTY OF CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THIS MATERIAL OR THE CONTENTS OF THIS DOCUMENT. IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THIS MATERIAL OR THE CONTENTS OF THIS DOCUMENT OR MIPI BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT, SPECIFICATION OR DOCUMENT RELATING TO THIS MATERIAL, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Without limiting the generality of this Disclaimer stated above, the user of the contents of this Document is further notified that MIPI: (a) does not evaluate, test or verify the accuracy, soundness or credibility of the contents of this Document; (b) does not monitor or enforce compliance with the contents of this Document; and (c) does not certify, test, or in any manner investigate products or services or any claims of compliance with the contents of this Document. The use or implementation of the contents of this Document may involve or require the use of intellectual property rights ("IPR") including (but not limited to) patents, patent applications, or copyrights owned by one or more parties, whether or not Members of MIPI. MIPI does not make any search or investigation for IPR, nor does MIPI require or request the disclosure of any IPR or claims of IPR as respects the contents of this Document or otherwise.

Questions pertaining to this document, or the terms or conditions of its provision, should be addressed to:

MIPI Alliance, Inc.
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854
Attn: Board Secretary

Contents

Figures

Tables

MIPI Alliance Specification for Device Descriptor Block (DDB)

1 Introduction

This MIPI Device Descriptor Block specification defines Services to transfer descriptor and configuration data between Devices on a MIPI Interconnect. This document is used with other MIPI Alliance specifications as part of a complete system design.

The descriptor and configuration data, hereafter referred to as DDB Data, is comprised of several fields with each field representing a single data item.

The DDB specification allows for three levels of conformity, called Level 1, Level 2, and Level 3. Level 1 provides access to basic Device descriptor data fields as defined in this document. Level 2 provides getting and setting of DDB Data fields, using a sequence of bytes as access model for the Device's fields. Level 3 provides the same functionality as Level 2, but uses a field ID-based access model. Level 2 and Level 3 include the Level 1 functionality. Level 2 and Level 3 are functional alternatives that may be supported individually or in combination.

For a given Device, a manufacturer may choose whether or not to support DDB. In addition, the manufacturer may choose to support only DDB Level 1, to support Level 2, to support Level 3 or to support both Level 2 and Level 3. For example, a simple Device such as a MEMS microphone may only need to convey limited information such as its manufacturer ID and device class to other Devices; in this case providing Level 1 suffices. A more complex Device such as a display module may need to provide additional descriptor data such as the display resolution and configuration information such as color depth to other Devices; this additional information can be provided by DDB Level 2 or by DDB Level 3.

1.1 Scope

This document defines DDB Level 1 and Level 2 Services, Service Access Points (SAPs), Level 1 descriptor data fields, and the Level 2 data model framework for DDB Data. The DDB Data fields may be defined in a MIPI Interface specification, in a separate Device class specification, or in a manufacturer's Device specification and are outside the scope of this document.

This document also defines a reference protocol that may be used in the definition of the mapping of DDB Services onto the various MIPI Interfaces. The actual mapping is defined in separate, Interface specific, specifications and is outside the scope of this document.

1.2 Purpose

The DDB specification provides a common set of Services to dynamically read (get) descriptor and configuration data from Devices and to write (set) configuration data to Devices regardless of the Device type or the underlying Interconnect.

Descriptor data can be used in the Device discovery process to identify the available Devices and their class, e.g. microphone, display module, modem. Descriptor data can also be used to dynamically detect variability between Devices of the same class, e.g. the actual resolution of a display, allowing the user of a Device to adjust its use of the Device to be in accordance with its capabilities or properties. This is useful when dealing with second sourcing, dealing with evolution of Devices and for building product families.

2 Terminology (informative)

The MIPI Alliance has adopted Section 13.1 of the *IEEE Standards Style Manual*, which dictates use of the words “shall”, “should”, “may”, and “can” in the development of documentation, as follows:

> The word *shall* is used to indicate mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).
>
> The use of the word *must* is deprecated and shall not be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.
>
> The use of the word *will* is deprecated and shall not be used when stating mandatory requirements; *will* is only used in statements of fact.
>
> The word *should* is used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (*should* equals *is recommended that*).
>
> The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted*).
>
> The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

All sections are normative, unless they are explicitly indicated to be informative.

Numbers are decimal unless otherwise indicated. A prefix of 0x indicates a hexadecimal number, while a prefix of 0b indicates a binary number. Unless stated otherwise, all values are unsigned integers.

Throughout the document, in all bytes and octets, the most significant bit is bit 7.

2.1 Definitions

DDB Data: Device description or configuration data fields that are accessible through a DDB Service.

DDB Level 1 Data: The basic Device descriptor data as defined in this document. See section 5.2.1.1.

DDB Level 2 Data: DDB Data that is accessible through the GET-DDB-LEVEL2 or SET-DDB--LEVEL2 Service. DDB Level 2 Data field definitions are outside the scope of this document.

DDB Protocol Data Unit (DDB-PDU): A unit of data exchanged between DDB Service Providers consisting of DDB layer Protocol Control Information and any payload.

Device: An addressable entity on an Interconnect.

Interaction: A communication between a Requestor and a Provider.

Interconnect: A data transport layer Interface.

Interface: The protocols, signaling characteristics, commands, clocking signals, register models, application program interfaces and data structures to the extent they enable interoperation, interconnection or communication between integrated circuits (even if located on the same die).

Primitive: see Service Primitive.

Protocol Control Information (PCI): Information exchanged between DDB layers in different devices to coordinate their joint-operation.

Provider: A Device that provides DDB data access to Devices on the Interconnect.

Request Message: A DDB-PDU with a MessageType of REQ_MESSAGE. See section 6.1.3.

Requestor: A Device that requests DDB data access from Devices on the Interconnect.

Response Message: A DDB-PDU with a MessageType of RESP_MESSAGE. See section 6.1.3.

Service: A capability of the DDB layer and the layers beneath it.

Service Access Point (SAP): The point at which a DDB Service is provided by the DDB layer to Service Users.

Service Primitive: An abstract, and implementation independent, representation of an interaction between a DDB Service User and a DDB Service Provider.

Service Provider: An abstract representation of the DDB layer and any underlying Interconnect that provides the DDB Service.

Service User: An abstract representation of an entity in a system that uses the DDB Service.

Slice: Contiguous portion of mapped DDB Level 2 Data.

2.2 Service Primitive Naming

This document uses an OSI-like naming convention in the definition of Service Primitives:

<service-primitive> ::= <primitive-name> ([<parameter-list>])

<primitive-name> ::= <service-name> . <primitive-type>

<service-name> ::= GET-DDB-LEVEL1 | GET-DDB-LEVEL2 | SET-DDB-LEVEL2

<primitive-type> ::= request | indication | response | confirm

<parameter-list> ::= <parameter> { , <parameter> }*

<parameter> ::= <Service control information> | <Service User data>

2.3 Acronyms

DDB Device Descriptor Block

IC Integrated Circuit

MIPI	Mobile Industry Processor Interface
OSI	Open Systems Interconnection
PCI	Protocol Control Information
PDU	Protocol Data Unit
SAP	Service Access Point

3 References

[IEEE01] 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*, ISBN 978-1-5593-7653-2, Institute of Electrical and Electronics Engineers, 1 January 2003

[IETF01] RFC3629, *UTF-8, a transformation format of ISO 10646*, <http://www.ietf.org/rfc/rfc3629.txt>, The Internet Society, November 2003

[IETF02] RFC2781, *UTF-16, an encoding of ISO 10646*, <http://www.ietf.org/rfc/rfc2781.txt>, The Internet Society, February 2000

[ITUT01] ITU-T Recommendation X.200 (7/94), *Information technology - Open Systems Interconnection - Basic Reference Model: The basic model*, <http://www.itu.int/rec/T-REC-X/en>, International Telecommunication Union, 7 November 1997

[ITUT02] ITU-T Recommendation X.210 (11/93), *Information technology - Open systems interconnection - Basic Reference Model: Conventions for the definition of OSI services*, <http://www.itu.int/rec/T-REC-X/en>, International Telecommunication Union, 30 November 1994

[MIPI01] MIPI Alliance, *Current Members – List of all MIPI Manufacturer IDs*, "List of MIPI Manufacturer IDs", <http://www.mipi.org/view_mid.asp>, 10 December 2007

4 Architecture Overview (informative)

The DDB specification defines Services that allow a Device on a MIPI Interconnect (Figure 1) to transfer DDB Data with any Device on the Interconnect that supports DDB. Typical DDB-capable Devices include processor ICs and peripheral ICs.

DDB Data can be classified as descriptor data or configuration data. Descriptor data includes identifying information such as the manufacturer, capability information such as the supported display formats for a display device, and calibration information such as the gain of a particular microphone. Configuration data provides a means to configure the device in a standard way. Typically, descriptor data is read-only (i.e. get) while configuration data is read-write (i.e. get and set).

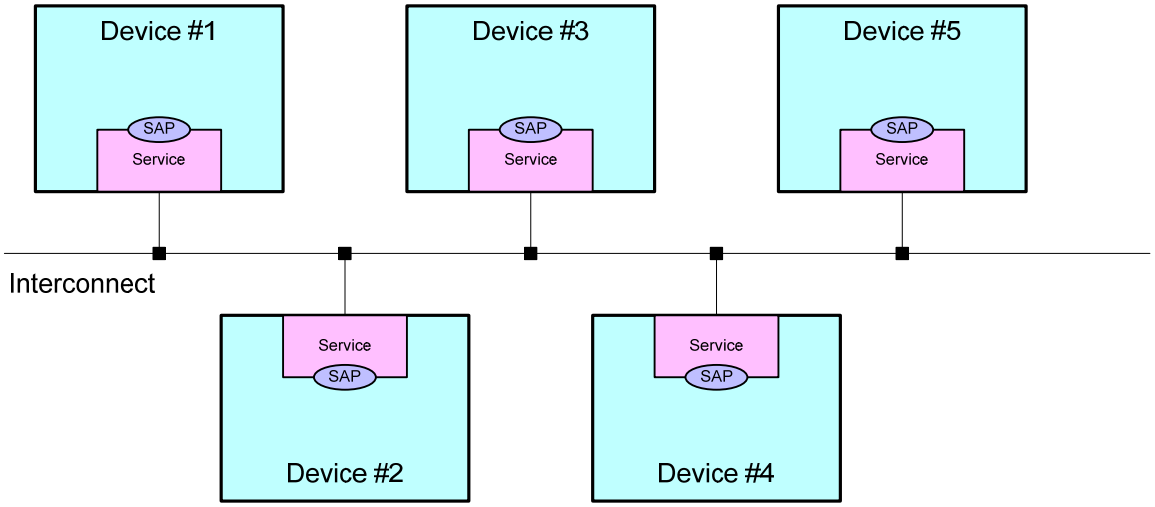


Figure 1 Devices on an Interconnect

DDB is built on an OSI-style layered model [ITUT01], similar to many network systems. The DDB Services are contained within a layer so that DDB is independent of the Service User and can provide the same Services no matter the specific underlying Interconnect(s), as shown in Figure 2.

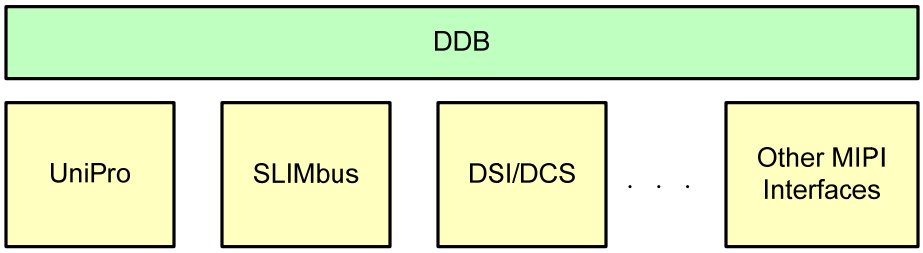


Figure 2 Mapping DDB to MIPI Interfaces

The Service User accesses DDB Services through DDB Service Access Points (SAPs). The DDB layer accesses the underlying Interconnect through the Interconnect's SAPs. The DDB layered model with SAPs is shown in Figure 3.

The Interaction between the DDB Service User and the DDB layer is defined using four types of Service Primitives: request, indication, response, and confirm [ITUT02].

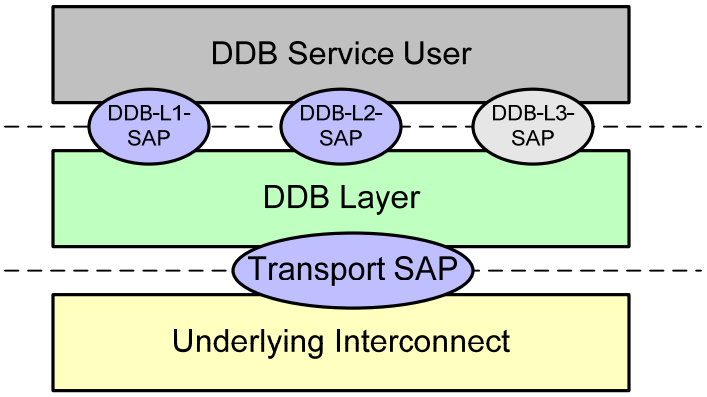


Figure 3 DDB Layer Overview

A Device initiating a DDB Data exchange is called the Requestor and a Device providing or acting on the DDB Data is called the Provider. Requestor and Provider are roles played by a Device. Any Device may act as a Requestor, a Provider, or both.

Figure 4 shows a typical Interaction and the relationship between the SAP, the Service Primitives and the Requestor and Provider roles. Service User R in Device A (the Requestor) accesses a DDB Service by generating a request Primitive, passing the ID of Device B (the Provider) and any DDB Data (in case of a set Primitive) as parameters. The DDB layer in the Requestor communicates the Service request to the DDB layer in the Provider through the underlying Interconnect. The DDB layer in the Provider signals Service User P by generating an indication Primitive that Device A requested the specific Service, passing any relevant data as parameters. Service User P consumes (in case of a set Primitive) or produces (in case of a get Primitive) the data and generates a response Primitive with any DDB Data (in case of a get Primitive) to the Provider's DDB layer. This layer in turn forwards the response through the underlying Interconnect to the DDB layer in the Requestor. Finally, the DDB layer in the Requestor signals Service User R by generating a confirm Primitive passing any data and the status as parameters.

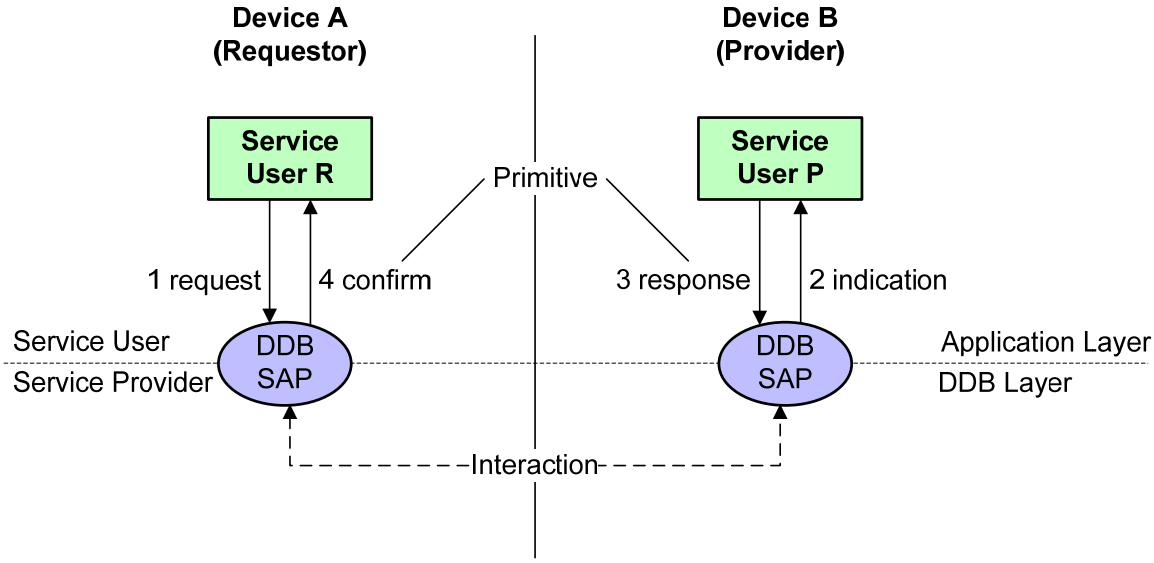


Figure 4 Service Primitives and Roles

5 DDB Services

The DDB Services describe the capability of the DDB layer and any underlying Interconnect to exchange DDB Data between Devices on that Interconnect. Separate Services are defined for DDB Level 1 get and for DDB Level 2 get and set.

Each DDB Service is defined in terms of parameterized Service Primitives that are available at a Service Access Point. There are four types of Service Primitives: request, indication, response, and confirm. For each Primitive the parameters, when it is generated, the effect on receipt, and any additional requirements are defined.

5.1 Generic Service Elements

This section contains elements that are common to DDB Services on multiple levels.

5.1.1 General Service Parameters

Some parameters are common to Services on multiple levels; these are defined in the following sections.

5.1.1.1 RequestorID

The RequestorID shall uniquely identify an Interconnect endpoint reachable in the scope of the Provider's Service User.

Table 1 RequestorID Format

Name	Size, bits	Description
RequestorID	32	ID to identify an Interconnect endpoint reachable in the scope of the Service User.

5.1.1.2 ProviderID

The ProviderID shall uniquely identify an Interconnect endpoint reachable in the scope of the Requestor's Service User. A valid ProviderID is obtained through a mechanism that is outside the scope of this document.

Table 2 ProviderID Format

Name	Size, bits	Description
ProviderID	32	ID to identify an Interconnect endpoint reachable in the scope of the Service User.

5.1.1.3 TransactionID

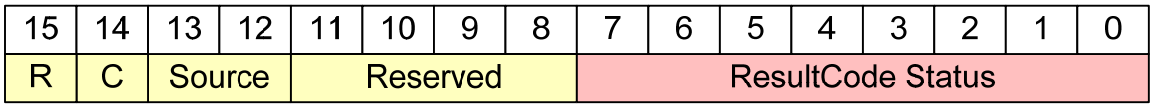
The TransactionID shall be used to link a confirm Primitive to a specific request Primitive and to link a response Primitive to a specific indication Primitive. The Service Provider and the Provider's Service User shall treat the TransactionID as an opaque value.

311 **Table 3 TransactionID Format**

Name	Size, bits	Description
TransactionID	8	The TransactionID is used to link a confirm Primitive to a request Primitive and to link a response Primitive to an indication Primitive.

312 #### 5.1.1.4 ResultCode

313 The ResultCode is a 16-bit value with flags indicating the source of the result and a status value indicating
314 the result status. Figure 5 defines the ResultCode format.



315

316 **Figure 5 ResultCode Format**

317 The ResultCode flag descriptions and their values are defined in Table 4.

318 **Table 4 ResultCode Flags**

Bits	Description
15	Reserved (R). Must be zero.
14	Component (C) 0b0: Provider 0b1: Requestor
[13:12]	Source 0b00: Not Applicable 0b01: Transport Layer 0b10: DDB Layer 0b11: Service User
[11:8]	Reserved. Must be zero.

319 Table 5 defines the values for the ResultCode status; values not listed are reserved and shall not be used.

320 **Table 5 ResultCode Status Values**

ResultCode Status	Value	Description
RESULT_OK	0x00	No error occurred.
ERROR_NOT_SUPPORTED	0x01	The Device does not support this Service.
ERROR_INVALID_ID	0x02	The ProviderID is not valid.
ERROR_NO_RESPONSE	0x03	A transmission error occurred or no response was received from the Device within a time-out interval. The time-out value depends on the underlying Interconnect and shall be defined in a separate document.
ERROR_UNKNOWN	0x04	An unspecified error occurred.

ResultCode Status	Value	Description
ERROR_BUSY	0x05	An element of the implementation is busy and cannot process the request. The number of outstanding requests may be limited by any element of the implementation.
ERROR_INVALID_SLICE	0x06	The Slice of the DDB Level 2 Data defined by Offset and RequestLength (in case of a GET-DDB-LEVEL2 Service) or the length of DdbL2Data (in case of a SET-DDB-LEVEL2 Service) is not valid for the Device.
ERROR_UNSUPPORTED_LENGTH	0x07	The RequestLength (in case of a GET-DDB-LEVEL2 Service) or the length of DdbL2Data (in case of a SET-DDB-LEVEL2 Service) exceeds the capabilities of the Device.
ERROR_INVALID_VALUE	0x08	A field is being set with an invalid value.

5.1.2 Error Reporting

The Service Provider reports errors to the Requestor's Service User through a confirm Primitive.

The Provider's Service User reports errors to the Service Provider through a response Primitive.

The Requestor's Service Provider shall report the ERROR_NO_RESPONSE and ERROR_INVALID_ID conditions and any result codes reported by the Provider. The Requestor's Service Provider may report the ERROR_BUSY and ERROR_UNKNOWN conditions.

The Provider's Service Provider may report the ERROR_NOT_SUPPORTED, ERROR_BUSY, ERROR_UNKNOWN and ERROR_UNSUPPORTED_LENGTH conditions.

The Provider's Service User may report the ERROR_BUSY, ERROR_UNKNOWN, ERROR_INVALID_SLICE, ERROR_UNSUPPORTED_LENGTH, and ERROR_INVALID_VALUE conditions.

5.2 DDB Level 1 Service

The DDB Level 1 functionality consists of a single Service, GET-DDB-LEVEL1. The GET-DDB-LEVEL1 Service provides a mechanism for Requestor Devices to obtain the basic Device descriptor data as defined by the DdbL1Data fields from Provider Devices.

This section defines the DDB Level 1 Data and the Service Primitives used to provide the DDB Level 1 Service.

5.2.1 DDB Level 1 Specific Service Parameter

There is one DDB Level 1 specific Service parameter, DdbL1Data.

5.2.1.1 DdbL1Data

The DdbL1Data is a structured parameter containing the basic Device descriptor data fields. The fields of this parameter are defined in Table 6.

343 **Table 6 DdbL1Data Fields**

Item	Size, bits	Description
Revision	8	The revision of the DDB specification supported by this Device encoded as major-version * 0x10 + minor-version. For this version (v1.0) the value shall be 0x10.
Level	8	Eight bit field indicating the DDB support: b0: DDB Level 1 support. Shall be 0b1. b1: DDB Level 2 get support. Shall be 0b1 if and only if the GET-DDB-LEVEL2 Service is supported. If b1 is set, then b0 shall also be set. b2: DDB Level 2 set support. Shall be 0b1 if and only if the SET-DDB-LEVEL2 Service is supported. If b2 is set, both b0 and b1 shall also be set. b[7:3]: Reserved. Shall be 0b00000.
DeviceClass	16	The device class ID of the Device as specified by the MIPI Alliance. If the Device does not conform to a specified device class, the value shall be 0.
ManufacturerID	16	The manufacturer ID of the Device's manufacturer as specified by the MIPI Alliance [MIPI01].
ProductID	16	The product ID as specified by the Device manufacturer.
Length	16	The length of any DDB Level 2 data. For Devices supporting only DDB Level 1, the value shall be 0. For Devices supporting DDB Level 2, the value shall be the size of the available DDB Level 2 data in bytes.

344 ### 5.2.2 GET-DDB-LEVEL1 SAP

345 The parameters used by the Service Primitives are described in sections 5.1.1 and 5.2.1.

346 The GET-DDB-LEVEL1 Primitives are covered in this section and are listed in Table 7.

347 **Table 7 DDB-L1-SAP Primitives**

Service Primitive	request	indication	response	confirm
GET-DDB-LEVEL1	5.2.2.1	5.2.2.2	5.2.2.3	5.2.2.4

348 #### 5.2.2.1 GET-DDB-LEVEL1.request

349 The specification of this Primitive is:

350 GET-DDB-LEVEL1.request (ProviderID, TransactionID)

351 **When Generated**

352 The Requestor's DDB Service User shall generate this Service Primitive to obtain the DdbL1Data of the
353 Device given in ProviderID.

354 The Service User shall provide TransactionID to associate the corresponding confirm Primitive with this
355 request.

Effect on Receipt

Receipt of this Service Primitive shall cause the generation of a single corresponding confirm Primitive.

Additional Requirements

If the ProviderID is not valid in the context of the Requestor's Service User, the Requestor's Service Provider shall generate a corresponding confirm Primitive with ResultCode status value ERROR_INVALID_ID and ResultCode flags set to Requestor DDB Layer (Component = Requestor, Source = DDB Layer) or to Requestor Transport Layer (Component = Requestor, Source = Transport Layer) according to the component that detected the error.

If the Requestor's Service Provider detects a transmission error on the Interconnect, it may generate a corresponding confirm Primitive with ResultCode status value ERROR_NO_RESPONSE and ResultCode flags set to Requestor Transport Layer (Component = Requestor, Source = Transport Layer).

If, after a time-out value as specified by the underlying Interconnect's DDB Service to protocol mapping, the Requestor's Service Provider has not obtained the necessary data to generate a corresponding confirm Primitive, it shall generate a corresponding confirm Primitive with ResultCode status value ERROR_NO_RESPONSE and ResultCode flags set to Requestor DDB Layer (Component = Requestor, Source = DDB Layer).

5.2.2.2 GET-DDB-LEVEL1.indication

The specification of this Primitive is:

GET-DDB-LEVEL1.indication (RequestorID, TransactionID)

When Generated

The DDB Service Provider may generate this Primitive as a result of a GET-DDB-LEVEL1.request Primitive generated on the Device identified by RequestorID.

The Service Provider shall provide TransactionID to associate the corresponding response Primitive with this indication.

Effect on Receipt

The Provider's DDB Service User shall generate a single corresponding response Primitive using the given RequestorID and TransactionID.

Additional Requirements

None.

5.2.2.3 GET-DDB-LEVEL1.response

The specification of this Primitive is:

GET-DDB-LEVEL1.response (RequestorID, TransactionID, DdbL1Data, ResultCode)

When Generated

The Provider's DDB Service User shall generate this Service Primitive as a result of a GET-DDB-LEVEL1.indication Primitive.

RequestorID shall be identical to RequestorID of the corresponding indication Primitive.

TransactionID shall be identical to TransactionID of the corresponding indication Primitive.

ResultCode shall contain one of the ResultCode status values listed in Table 8. If the ResultCode status is RESULT_OK, the ResultCode flags shall be set to Provider Not Applicable (Component = Provider, Source = Not Applicable). If the ResultCode status is not RESULT_OK, the ResultCode flags shall be set to Provider Service User (Component = Provider, Source = Service User).

If the ResultCode status is RESULT_OK, DdbL1Data shall contain the basic descriptor data of this Device.

Table 8 GET-DDB-LEVEL1.response ResultCode Status Values

ResultCode Status	Comment
RESULT_OK	DdbL1Data contains the basic descriptor data of this Device
ERROR_BUSY	DdbL1Data is undefined
ERROR_UNKNOWN	

Effect on Receipt

Receipt of this Service Primitive shall cause the generation of a single corresponding confirm Primitive on the Device identified by RequestorID, except when such a Service Primitive has already been generated; for example, due to a time-out.

Additional Requirements

None.

5.2.2.4 GET-DDB-LEVEL1.confirm

The specification of this Primitive is:

GET-DDB-LEVEL1.confirm (ProviderID, TransactionID, DdbL1Data, ResultCode)

When Generated

The DDB Service Provider shall generate this Service Primitive as a result of a GET-DDB-LEVEL1.request Primitive.

ProviderID shall be identical to ProviderID of the corresponding request Primitive.

TransactionID shall be identical to TransactionID of the corresponding request Primitive.

ResultCode shall contain one of the ResultCode status values listed in Table 9. If the ResultCode status is RESULT_OK, the ResultCode flags shall be set to Provider Not Applicable (Component = Provider, Source = Not Applicable). If the ResultCode status is not RESULT_OK, the ResultCode flags shall indicate the source of the error.

If the ResultCode status is RESULT_OK, DdbL1Data shall contain the basic descriptor data of the Device identified by ProviderID; this data shall originate from the Device identified by ProviderID.

Table 9 GET-DDB-LEVEL1.confirm ResultCode Status Values

ResultCode Status	Comment
RESULT_OK	DdbL1Data contains the basic descriptor data of the Device identified by ProviderID
ERROR_BUSY	DdbL1Data is undefined
ERROR_INVALID_ID	
ERROR_NO_RESPONSE	
ERROR_NOT_SUPPORTED	
ERROR_UNKNOWN	

Effect on Receipt

The Requestor's Service User is provided with DdbL1Data.

Additional Requirements

None.

5.3 DDB Level 2 Services

The DDB Level 2 functionality consists of two Services, GET-DDB-LEVEL2 and SET-DDB-LEVEL2. The GET-DDB-LEVEL2 Service provides a mechanism for Service Users to get DDB data from Provider Devices. The SET-DDB-LEVEL2 Service provides a mechanism for Service Users to set DDB data on Provider Devices. GET-DDB-LEVEL2 may be supported without supporting SET-DDB-LEVEL2. However, GET-DDB-LEVEL2 shall be supported if SET-DDB-LEVEL2 is supported.

This section defines the DDB Level 2 Data Model and Service Primitives used to provide the DDB Level 2 Services.

5.3.1 DDB Level 2 Data Model

The DDB Level 2 Services present the Provider’s DDB Data fields as a sequence of bytes. The first byte of the DDB Level 2 data shall have index zero as indicated in Figure 6.

Fields may be separated by filler bytes in the byte sequence, e.g. to provide field alignment to a word or some other boundary or to allow for future extension. A get or set operation on a filler byte shall not cause an error. The filler byte shall be transferred between Provider and Requestor and shall be included in the RequestLength or ResponseLength value. The value of a filler byte is not defined.

The sequence of bytes can be accessed at any offset and with any length of data transfer. Field mapping specifications may limit the supported offsets and data lengths in order to allow for future extensions, preserve data integrity, align to address boundaries, or any other specification or implementation reason.

The mapping of DDB data fields onto a DDB Level 2 byte sequence is outside the scope of this document.

5.3.1.1 Data Types

The DDB Level 2 Data should be encoded according to the types listed in Table 10.

445 **Table 10 DDB Level 2 Data Type Encoding**

Type	Size, bits	Description
INT8	8	Signed 8-bit integer
UINT8	8	Unsigned 8-bit integer
INT16	16	Signed 16-bit integer
UINT16	16	Unsigned 16-bit integer
INT32	32	Signed 32-bit integer
UINT32	32	Unsigned 32-bit integer
INT64	64	Signed 64-bit integer
UINT64	64	Unsigned 64-bit integer
FLOAT	32	Single precision floating-point number encoding according to [IEEE01].
DOUBLE	64	Double precision floating-point number encoding according to [IEEE01].
STRING	fixed	Strings are encoded with a fixed reserved length in bytes as indicated in the device data sheet. Character encoding should use zero terminated UTF-8 [IETF01] or UTF-16 [IETF02]. A single character is encoded as a STRING with length equal to the character length in bytes.

446 #### 5.3.1.2 Field Mapping

447 A single multi-byte field should be mapped to consecutive bytes in big-endian format with the most
448 significant byte having the lowest index. The device data sheet should identify the byte arrangement for
449 multi-byte values in the DDB Level 2 Data. See Figure 6 for an example mapping.

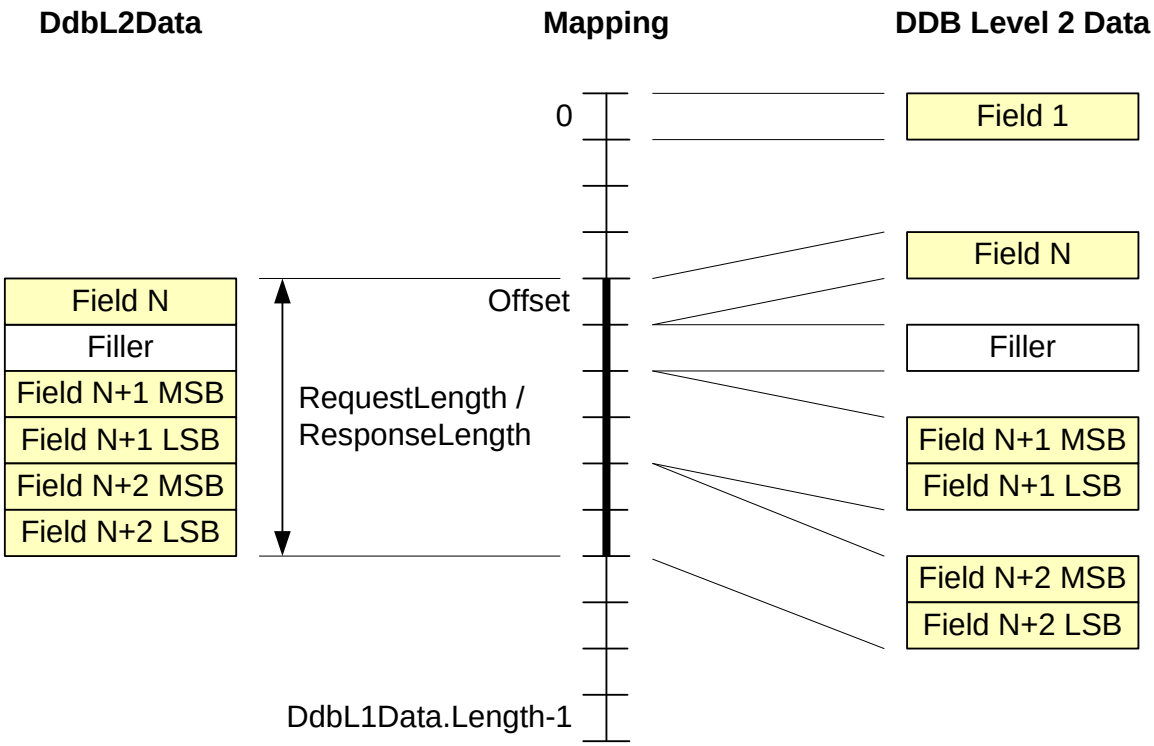


Figure 6 Level 2 Data Model and Mapping Example

5.3.2 DDB Level 2 Specific Service Parameters

The DDB Level 2 specific Service parameters are Offset, RequestLength, ResponseLength, and DdbL2Data.

5.3.2.1 Offset

Offset specifies the index of the first byte of the mapped DDB Level 2 data to be processed.

Table 11 DDB Level 2 Offset Format

Name	Size, bits	Description
Offset	16	The index of the first byte of the mapped DDB Level 2 data to be processed.

5.3.2.2 RequestLength

RequestLength specifies the number of bytes to be processed in the GET-DDB-LEVEL2 Service.

Table 12 DDB Level 2 RequestLength Format

Name	Size, bits	Description
RequestLength	16	The number of DDB Level 2 data bytes to be processed in the GET-DDB-LEVEL2 Service.

5.3.2.3 ResponseLength

ResponseLength specifies the number of bytes that have been processed by the Service User in the SET-DDB-LEVEL2 Service. ResponseLength may be smaller than the DdbL2Data length, in which case only the first ResponseLength bytes have been processed starting from the Offset position.

Table 13 DDB Level 2 ResponseLength Format

Name	Size, bits	Description
ResponseLength	16	The number of bytes that have been processed in the SET-DDB-LEVEL2 Service.

5.3.2.4 DdbL2Data

DdbL2Data represents a slice of the Provider's mapped DDB Level 2 data.

5.3.3 GET-DDB-LEVEL2 SAP

The parameters used by the GET-DDB-LEVEL2 Service Primitives are defined in sections 5.1.1 and 5.3.2.

The GET-DDB-LEVEL2 Service Primitives are defined in this section and are listed in Table 14.

Table 14 GET-DDB-LEVEL2 Primitives

Service Primitive	request	indication	response	confirm
GET-DDB-LEVEL2	5.3.3.1	5.3.3.2	5.3.3.3	5.3.3.4

5.3.3.1 GET-DDB-LEVEL2.request

The specification of this Primitive is:

GET-DDB-LEVEL2.request (ProviderID, TransactionID, Offset, RequestLength)

When Generated

The Requestor's DDB Service User shall generate this Service Primitive to obtain DDB Level 2 data of the Device given in ProviderID.

The Service User shall provide TransactionID to associate the corresponding confirm Primitive with this request.

Effect on Receipt

Receipt of this Service Primitive shall cause the generation of a single corresponding GET-DDB-LEVEL2.indication Primitive on the Device given in ProviderID.

Additional Requirements

If the ProviderID is not valid in the context of the Requestor's Service User, the Requestor's Service Provider shall generate a corresponding confirm Primitive with ResultCode status value ERROR_INVALID_ID and ResultCode flags set to Requestor DDB Layer (Component = Requestor, Source = DDB Layer) or to Requestor Transport Layer (Component = Requestor, Source = Transport Layer) according to the component that detected the error.

If the Requestor's Service Provider detects a transmission error on the Interconnect, it may generate a corresponding confirm Primitive with ResultCode status value ERROR_NO_RESPONSE and ResultCode flags set to Requestor Transport Layer (Component = Requestor, Source = Transport Layer).

If the Service Provider detects that the Device identified by ProviderID does not support the GET-DDB-LEVEL2 Service, it shall generate a corresponding confirm Primitive with ResultCode status value ERROR_NOT_SUPPORTED. The ResultCode flags shall be set to Provider DDB Layer (Component = Provider, Source = DDB Layer) or Requestor DDB Layer (Component = Requestor, Source = DDB Layer) according to the component that detected the error.

If, after a time-out value as specified by the underlying Interconnect's Service to protocol mapping, the Requestor's Service Provider has not obtained the necessary data to generate a corresponding confirm Primitive, it shall generate a corresponding confirm Primitive with ResultCode status value ERROR_NO_RESPONSE and ResultCode flags set to Requestor DDB Layer (Component = Requestor, Source = DDB Layer).

5.3.3.2 GET-DDB-LEVEL2.indication

The specification of this Primitive is:

GET-DDB-LEVEL2.indication (RequestorID, TransactionID, Offset, RequestLength)

When Generated

The DDB Service Provider shall generate this Service Primitive as a result of a GET-DDB-LEVEL2.request Primitive generated on the Device identified by RequestorID.

The Service Provider shall provide TransactionID to associate the corresponding response Primitive with this indication.

Offset shall be equal to Offset of the corresponding request Primitive.

RequestLength shall be equal to RequestLength of the corresponding request Primitive.

Effect on Receipt

The Provider's DDB Service User shall process, i.e. get, its DDB Level 2 data starting at Offset for RequestLength bytes and shall subsequently generate a single corresponding GET-DDB-LEVEL2.response Primitive using the given RequestorID and TransactionID.

Additional Requirements

The mapped DDB Data shall be processed in order of increasing index. Each byte shall be processed exactly once. If the Service User is aware of the field boundaries, then the Service User should access the field atomically so that when the field is processed all bytes of the field are consistent and correct.

The Provider's Service User shall stop processing at the first error condition encountered and indicate the error condition in the ResultCode status of the corresponding response Primitive.

If Offset exceeds the Provider's DDB Data Length as given in the DdbL1Data Length field, the Provider's Service User shall not process any data and indicate the ResultCode status ERROR_INVALID_SLICE in the corresponding response Primitive.

If Offset is the index of a byte other then the first byte in a field or the sum of Offset and RequestLength minus one is not the last byte in a field, the Provider's Service User may process no data at all and indicate the ResultCode status ERROR_INVALID_SLICE in the corresponding response Primitive.

If the sum of Offset and RequestLength exceeds the DDB Level 2 data length as given in the DdbL1Data Length field, the Provider's DDB Service User shall either process no data at all and return a zero length DdbL2Data parameter in the corresponding response Primitive or process as many bytes of data as are available and return only the processed bytes as the DdbL2Data parameter in the corresponding response Primitive. In both cases the Service User shall indicate the ResultCode status ERROR_INVALID_SLICE in the corresponding response Primitive.

If RequestLength exceeds the Device's capabilities, the Service User shall process as many bytes of data as the Device can transfer and indicate the ResultCode status ERROR_UNSUPPORTED_LENGTH in the corresponding response Primitive.

5.3.3.3 GET-DDB-LEVEL2.response

The specification of this Primitive is:

GET-DDB-LEVEL2.response (RequestorID, TransactionID, DdbL2Data, ResultCode)

When Generated

The Provider's DDB Service User shall generate this Service Primitive as a result of a corresponding GET-DDB-LEVEL2.indication Primitive.

RequestorID shall be identical to RequestorID of the corresponding indication Primitive.

TransactionID shall be identical to TransactionID of the corresponding indication Primitive.

DdbL2Data shall be set to a Slice of the Provider's mapped DDB Data starting at Offset as given in the corresponding indication Primitive and with a length equal to the number of bytes processed. If no bytes were processed then DdbL2Data shall have a length of zero.

ResultCode shall contain one of the ResultCode status values listed in Table 15. If the ResultCode status is RESULT_OK, the ResultCode flags shall be set to Provider Not Applicable (Component = Provider, Source = Not Applicable). If the ResultCode status is not RESULT_OK, the ResultCode flags shall be set to Provider Service User (Component = Provider, Source = Service User).

Table 15 GET-DDB-LEVEL2.response ResultCode Status Values

ResultCode	Comment
RESULT_OK	DdbL2Data contains valid data.
ERROR_BUSY	DdbL2Data contains either valid data or has a length of zero.
ERROR_UNKNOWN	
ERROR_INVALID_SLICE	
ERROR_UNSUPPORTED_LENGTH	

Effect on Receipt

Receipt of this Service Primitive shall cause the generation of a single corresponding GET-DDB-LEVEL2.confirm Primitive on the Device identified by RequestorID, except when such a Service Primitive has already been generated; for example due to a time-out.

Additional Requirements

None.

5.3.3.4 GET-DDB-LEVEL2.confirm

The specification of this Primitive is:

GET-DDB-LEVEL2.confirm (ProviderID, TransactionID, DdbL2Data, ResultCode)

When Generated

The DDB Service Provider shall generate this Service Primitive as the result of a corresponding GET-DDB-LEVEL2.response Primitive generated on the Device identified by ProviderID.

TransactionID shall be identical to TransactionID of the corresponding request Primitive.

DdbL2Data shall be equal to DdbL2Data of the corresponding response Primitive.

ResultCode shall contain one of the ResultCode status values in Table 16. If the ResultCode status is RESULT_OK, the ResultCode flags shall be set to Provider Not Applicable (Component = Provider, Source = Not Applicable). If the ResultCode status is not RESULT_OK, the ResultCode flags shall indicate the source of the error.

Table 16 GET-DDB-LEVEL2.confirm ResultCode Status Values

ResultCode	Comment
RESULT_OK	DdbL2Data contains valid data.
ERROR_BUSY	DdbL2Data will either contain valid data or have a length of zero.
ERROR_INVALID_ID	
ERROR_NO_RESPONSE	
ERROR_NOT_SUPPORTED	
ERROR_UNKNOWN	
ERROR_INVALID_SLICE	
ERROR_UNSUPPORTED_LENGTH	

Effect on Receipt

The Requestor's Service User is provided with DdbL2Data.

If the ResultCode status is ERROR_NO_RESPONSE, ERROR_BUSY or ERROR_UNKNOWN, no assumptions shall be made on whether DDB Level 2 data has been processed at the Device identified by ProviderID or not.

Additional Requirements

When generation of this Primitive is not caused by a corresponding response Primitive, DdbL2Data shall have a length of zero.

5.3.4 SET-DDB-LEVEL2 SAP

The parameters used by the Service Primitives are described in sections 5.1.1 and 5.3.1.

The SET-DDB-LEVEL2 Primitives are covered in this section and are listed in Table 17.

Table 17 SET-DDB-LEVEL2 Primitives

Service Primitive	request	indication	response	confirm
SET-DDB-LEVEL2	5.3.4.1	5.3.4.2	5.3.4.3	5.3.4.4

5.3.4.1 SET-DDB-LEVEL2.request

The specification of this Primitive is:

SET-DDB-LEVEL2.request (ProviderID, TransactionID, Offset, DdbL2Data)

When Generated

The Requestor's DDB Service User shall generate this Service Primitive to set DDB Level 2 Data on the Device given in ProviderID to DdbL2Data.

The Service User shall provide TransactionID to associate the corresponding confirm Primitive with this request.

Effect on Receipt

Receipt of this Service Primitive shall cause the generation of a single corresponding SET-DDB-LEVEL2.indication Primitive on the Device given in ProviderID.

Additional Requirements

If the ProviderID is not valid in the context of the Requestor's Service User, the Requestor's Service Provider shall generate a corresponding confirm Primitive with ResultCode status value ERROR_INVALID_ID and ResultCode flags set to Requestor DDB Layer (Component = Requestor, Source = DDB Layer) or to Requestor Transport Layer (Component = Requestor, Source = Transport Layer) according to the component that detected the error.

If the Requestor's Service Provider detects a transmission error on the Interconnect, it may generate a corresponding confirm Primitive with ResultCode status value ERROR_NO_RESPONSE and ResultCode flags set to Requestor Transport Layer (Component = Requestor, Source = Transport Layer).

If the Service Provider detects that the Device identified by ProviderID does not support the SET-DDB-LEVEL2 Service, it shall generate a corresponding confirm Primitive with ResultCode status value ERROR_NOT_SUPPORTED. The ResultCode flags shall be set to Provider DDB Layer (Component = Provider, Source = DDB Layer) or Requestor DDB Layer (Component = Requestor, Source = DDB Layer) according to the component that detected the error.

If, after a time-out value specified by the underlying Interconnect's DDB Service to protocol mapping, the Requestor's Service Provider has not obtained the necessary information required to generate a corresponding confirm Primitive, it shall generate a corresponding confirm Primitive with ResultCode status value ERROR_NO_RESPONSE and ResultCode flags set to Requestor DDB Layer (Component = Requestor, Source = DDB Layer).

5.3.4.2 SET-DDB-LEVEL2.indication

The specification of this Primitive is:

SET-DDB-LEVEL2.indication (RequestorID, TransactionID, Offset, DdbL2Data)

When Generated

The DDB Service Provider shall generate this Service Primitive as a result of a corresponding SET-DDB-LEVEL2.request Primitive generated on the Device identified by RequestorID.

The Service Provider shall provide TransactionID to associate the corresponding response Primitive with this indication.

Offset shall be equal to Offset of the corresponding request Primitive.

DdbL2Data shall be equal to DdbL2Data of the corresponding request Primitive.

Effect on Receipt

The Provider's DDB Service User shall process DdbL2Data, i.e. set the Slice of mapped DDB Data starting at Offset for as many bytes as contained in DdbL2Data to the values given in DdbL2Data, and shall subsequently generate a corresponding SET-DDB-LEVEL2.response Primitive using the given RequestorID and TransactionID.

Additional Requirements

The DdbL2Data shall be processed in order of increasing index. Each byte shall be processed exactly once. If the Service User is aware of the field boundaries, then the Service User should access the field atomically so that when the field is processed all bytes of the field are consistent and correct.

The Provider's Service User shall stop processing at the first error condition encountered and indicate the error condition in the ResultCode status of the corresponding response Primitive.

If the value set to a field is not valid for that field, the Provider's Service User may stop processing, in which case it shall indicate the ResultCode status ERROR_INVALID_VALUE in the corresponding response Primitive.

If Offset exceeds the Provider's DDB Data Length as given in the DdbL1Data Length field, the Provider's Service User shall not process any data and indicate the ResultCode status ERROR_INVALID_SLICE in the corresponding response Primitive.

If Offset is the index of a byte other then the first byte in a field or the sum of Offset and the length of DdbL2data minus one is not the last byte in a field, the Provider's Service User may process no data at all and indicate the ResultCode status ERROR_INVALID_SLICE in the corresponding response Primitive.

If the sum of Offset and the length of DdbL2data exceeds the DDB Level 2 data Length as given in the DdbL1Data Length field, the Provider's DDB Service User shall either process no data at all and set ResponseLength to zero in the corresponding response Primitive or process as many bytes of data as are available and report the number of processed bytes in ResponseLength of the corresponding response Primitive. In both cases the Service User shall indicate the ResultCode status ERROR_INVALID_SLICE in the corresponding response Primitive.

5.3.4.3 SET-DDB-LEVEL2.response

The specification of this Primitive is:

SET-DDB-LEVEL2.response (RequestorID, TransactionID, ResponseLength, ResultCode)

When Generated

The Provider's DDB Service User shall generate this Service Primitive as a result of a corresponding SET-DDB-LEVEL2.indication.

RequestorID shall be identical to RequestorID of the corresponding indication Primitive.

TransactionID shall be identical to TransactionID of the corresponding indication Primitive.

ResponseLength shall indicate the number of bytes processed.

ResultCode shall contain one of the ResultCode status values listed in Table 18. If the ResultCode status is RESULT_OK, the ResultCode flags shall be set to Provider Not Applicable (Component = Provider, Source = Not Applicable). If the ResultCode status is not RESULT_OK, the ResultCode flags shall be set to Provider Service User (Component = Provider, Source = Service User).

Table 18 SET-DDB-LEVEL2.response ResultCode Status Values

ResultCode status	Comment
RESULT_OK	DDB Level 2 data has been set
ERROR_BUSY	DDB Level 2 data has been set as indicated by ResponseLength
ERROR_UNKNOWN	
ERROR_INVALID_SLICE	
ERROR_INVALID_VALUE	

Effect on Receipt

Receipt of this Service Primitive shall cause the generation of a single corresponding SET-DDB-LEVEL2.confirm Primitive on the Device identified by RequestorID, except when such a Service Primitive has already been generated; for example due to a time-out.

Additional Requirements

None.

5.3.4.4 SET-DDB-LEVEL2.confirm

The specification of this Primitive is:

SET-DDB-LEVEL2.confirm (ProviderID, TransactionID, ResponseLength, ResultCode)

When Generated

The DDB Service Provider shall generate this Service Primitive as the result of a corresponding SET-DDB-LEVEL2.response Primitive generated on the Device identified by ProviderID.

TransactionID shall be identical to TransactionID of the corresponding request Primitive.

ResponseLength shall be identical to ResponseLength of the corresponding response Primitive.

ResultCode shall contain one of the ResultCode status values listed in Table 19. If the ResultCode status is RESULT_OK, the ResultCode flags shall be set to Provider Not Applicable (Component = Provider, Source = Not Applicable). If the ResultCode status is not RESULT_OK, the ResultCode flags shall indicate the source of the error.

Table 19 SET-DDB-LEVEL2.confirm ResultCode Status Values

ResultCode	Comment
RESULT_OK	No error occurred
ERROR_BUSY	An error occurred
ERROR_INVALID_ID	
ERROR_NO_RESPONSE	
ERROR_NOT_SUPPORTED	
ERROR_UNKNOWN	
ERROR_INVALID_SLICE	
ERROR_INVALID_VALUE	
ERROR_UNSUPPORTED_LENGTH	

Effect on Receipt

The Requestor's Service User is notified of how many bytes have been set.

If the ResultCode status is ERROR_NO_RESPONSE, ERROR_BUSY or ERROR_UNKNOWN, no assumptions shall be made on whether DDB Level 2 data has been processed at the Device identified by ProviderID or not.

Additional Requirements

When generation of this Primitive is not caused by a corresponding response Primitive, ResponseLength shall be zero.

6 DDB Protocol

The DDB protocol defines how the Requestor and Provider parts of the DDB layer interact to realize the DDB Services. The DDB Layer uses an asymmetric, connectionless protocol to transfer control information and DDB data between a Provider and a Requestor on an Interconnect. The protocol follows the model shown in Figure 7.

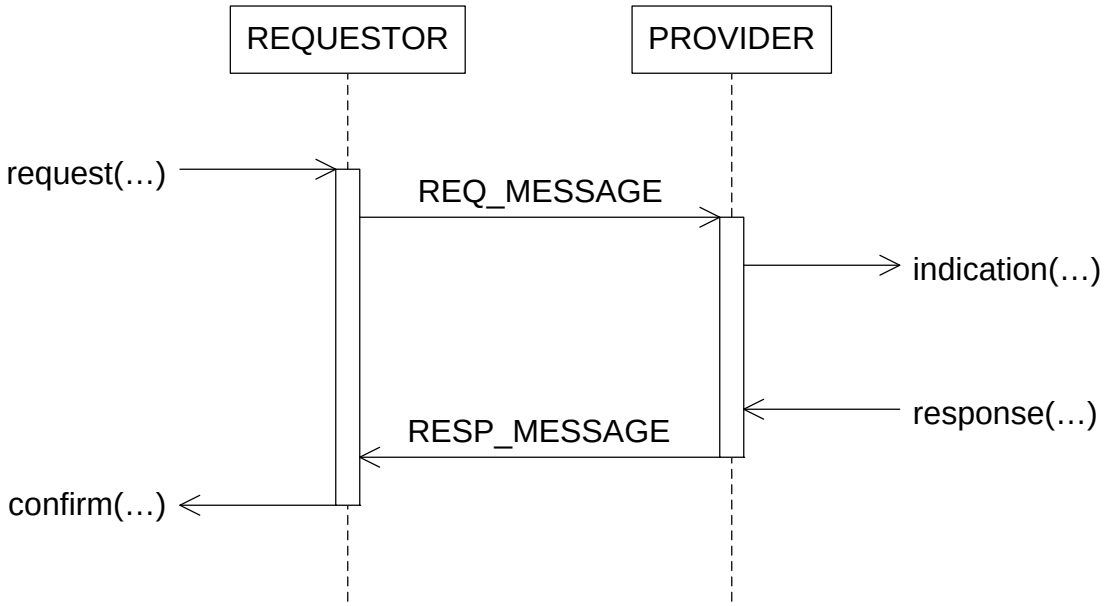


Figure 7 Service Primitive to Protocol Relationship

This document defines the DDB high-level protocol that may be mapped to MIPI Interfaces. Such mappings define how DDB protocol messages (PDUs) are transported using the MIPI Interface specific Services.

The DDB protocol is optional. Therefore, a MIPI Interface may specify any other mechanism to realize the DDB Services for that specific MIPI Interface. However, if the DDB protocol is used it shall be implemented as defined in this section.

6.1 Generic Protocol Elements

The DDB Protocol is a collection of related Service specific protocols. This section describes the elements that are common to all DDB Service protocols: the protocol model, the underlying Interconnect requirements, the DDB-PDU format, and the error handling.

6.1.1 Requestor and Provider Protocol Model

In the DDB Requestor and Provider protocol model, a Requestor shall initiate an Interaction with a Provider by sending a single DDB-PDU (section 6.1.3) with a MessageType of REQ_MESSAGE (Request Message) to that Provider. The Request Message contains all the data needed by the Provider to perform the request.

The Provider shall process the Request Message and shall finalize the Interaction by sending a single DDB-PDU with a MessageType of RESP_MESSAGE (Response Message) to the Requestor. The Response Message contains all the data needed by the Requestor.

A Requestor may have up to 256 simultaneous Interactions with a single Provider, and may have simultaneous Interactions with multiple Providers. A Provider shall be capable of processing at least one Request Message at a time.

6.1.2 Underlying Interconnect Requirements

The underlying Interconnect shall be capable of indicating the source of a DDB-PDU to the recipient.

The underlying Interconnect shall be capable of transferring a DDB-PDU to a given endpoint on that Interconnect.

The underlying Interconnect shall guarantee error-free transfer or shall detect and indicate transfer errors.

6.1.3 DDB-PDU Format

The DDB-PDU is composed of two parts, the Protocol Control Information (PCI) and the payload, and shall have the format defined in Figure 8.

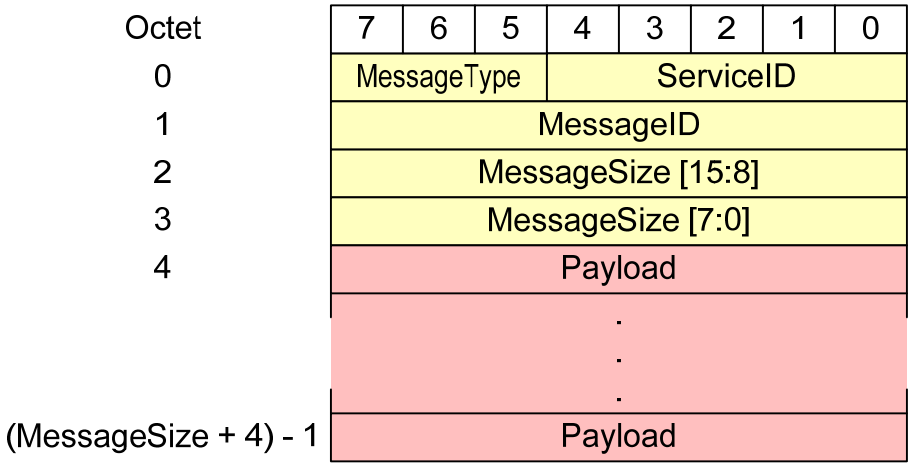


Figure 8 DDB-PDU Format

The PCI contains four fields: MessageType, ServiceID, MessageID and MessageSize. The PCI fields are described in sections 6.1.3.1, 6.1.3.2, 6.1.3.3 and 6.1.3.4. The payload depends on the MessageType and ServiceID values and is specified in the protocol sections.

6.1.3.1 MessageType Values

The MessageType shall be one of the values listed in Table 20. Values not listed in the table are reserved. A Device shall not send a DDB-PDU with a reserved value.

Table 20 MessageType Values

Value	MessageType	Description
0x00	REQ_MESSAGE	Request Message
0x01	RESP_MESSAGE	Response Message

6.1.3.2 ServiceID Values

The ServiceID shall be one of the values listed in Table 21. Values not listed in the table are reserved. A Device shall not send a DDB-PDU with a reserved value.

Table 21 ServiceID Values

Value	Service Name	Description
0x01	GET-DDB-LEVEL1	See section 5.2.2
0x02	GET-DDB-LEVEL2	See section 5.3.3
0x03	SET-DDB-LEVEL2	See section 5.3.4

6.1.3.3 MessageID

The MessageID links a Response Message to a particular Request Message. The Requestor shall provide the MessageID within the Request Message. The Provider shall set the MessageID in a Response Message to the MessageID contained in the corresponding Request Message.

The Requestor's DDB Service Provider may use the TransactionID of the request Primitive for the MessageID in the DDB-PDU. The Provider's DDB Service Provider may use the MessageID of the DDB-PDU as TransactionID for the indication Primitive.

6.1.3.4 MessageSize

The MessageSize gives the size of the DDB-PDU in octets as an unsigned integer number. The number of message octets shall be the same as the value of the MessageSize field.

6.1.4 Error Handling

The DDB protocol only supports error detection; error recovery is the responsibility of the Service User.

If the DDB Layer receives a DDB-PDU with an unknown MessageType, it shall ignore the DDB-PDU.

If the DDB Layer receives a DDB-PDU with MessageType REQ_MESSAGE and an unknown ServiceID, the DDB Layer should report the error to the Requestor using a corresponding Response Message with ResultCode status value ERROR_NOT_SUPPORTED and ResultCode flags set to Provider DDB Layer (Component = Provider, Source = DDB Layer).

On receipt of a DDB-PDU with MessageType REQ_MESSAGE, the DDB Layer may internally detect the ERROR_ BUSY and/or ERROR_UNKNOWN conditions. If either of these error conditions is detected, it should report the error to the Requestor using a corresponding Response Message with the detected ResultCode status value and ResultCode flags set to Provider DDB Layer (Component = Provider, Source = DDB Layer).

If the DDB Layer receives a DDB-PDU with MessageType RESP_MESSAGE and a combination of ServiceID and MessageID that do not belong to an ongoing Transaction, it shall ignore the DDB-PDU.

If the Requestor's DDB Layer does not receive a Response Message within the time-out value specified in the Interconnect Mapping for that Interconnect, it shall report ERROR_NO_RESPONSE to the Service User. The ResultCode flags shall be set to Requestor DDB Layer (Component = Requestor, Source = DDB Layer).

6.1.4.1 Interaction with the Interconnect

If the Requestor's DDB Layer detects an invalid ProviderID value when receiving a request primitive, or receives an indication from the Transport layer of an invalid ProviderID condition when sending a REQ_MESSAGE, it shall abort the Transaction and shall report Result Status ERROR_INVALID_ID with ResultCode flags set to Requestor DDB Layer (Component = Requestor, Source = DDB Layer) or Requestor Transport Layer (Component = Requestor, Source = Transport Layer) according to the component that detected the error to the Service User in the corresponding confirm primitive.

The Provider’s DDB Layer may detect an invalid RequestorID value when receiving a response primitive. In this case it should not generate a RESP_MESSAGE.

The Provider’s DDB Layer shall ignore ERROR_INVALID_ID and transport error conditions indicated by the Interconnect.

6.2 DDB Protocol Support for Level 1 Service

The DDB protocol shall be used to realize the DDB-LEVEL-1 Service.

6.2.1 Relation to Service Primitives

The receipt of a GET-DDB-LEVEL1.request Primitive may cause the sending of a GET-DDB-LEVEL1 Request Message.

The receipt of a GET-DDB-LEVEL1 Request Message may cause the generation of a GET-DDB-LEVEL1.indication Primitive.

The receipt of a GET-DDB-LEVEL1.response Primitive shall cause the sending of a GET-DDB-LEVEL1 Response Message.

The receipt of a Response Message shall generate a GET-DDB-LEVEL1.confirm Primitive, except when such a Service Primitive has already been generated; for example due to a time-out. See section 5.2.1.1.

6.2.2 GET-DDB-LEVEL1 Payloads

This section describes the Request Message and Response Message DDB-PDU payloads for the GET-DDB-LEVEL1 service.

6.2.2.1 GET-DDB-LEVEL1 Request Message Payload

There shall be no payload in the Request Message for GET-DDB-LEVEL1 service.

6.2.2.2 GET-DDB-LEVEL1 Response Message Payload

The payload of the Response Message shall contain the ResultCode (section 5.1.1.4) and, if the ResultCode Status is RESULT_OK, the DdbL1Data (section 5.2.1) and shall be transferred in the order specified in Figure 9.

Octet	7	6	5	4	3	2	1	0
0	ResultCode Flags							
1	ResultCode Status							
2	Revision							
3	Level							
4	DeviceClass [15:8]							
5	DeviceClass [7:0]							
6	ManufacturerID [15:8]							
7	ManufacturerID [7:0]							
8	ProductID [15:8]							
9	ProductID [7:0]							
10	Length [15:8]							
11	Length [7:0]							

Figure 9 GET-DDB-LEVEL1 Response Message Payload Format

6.3 DDB Protocol support for Level 2 Services

The DDB protocol shall be used to realize the DDB-LEVEL-2 Services.

6.3.1 Relation to Service Primitives

The receipt of a valid DDB Level 2 request Primitive shall cause the sending of a corresponding DDB Level 2 Request Message.

The receipt of a DDB Level 2 Request Message shall cause the generation of a corresponding DDB Level 2 indication Primitive.

The receipt of a DDB Level 2 response Primitive shall cause the sending of a corresponding DDB Level 2 Response Message.

The receipt of a DDB Level 2 Response Message shall generate a corresponding DDB Level 2 confirm Primitive, except when such a Service Primitive has already been generated; for example, due to a time-out. See sections 5.3.3.1 and 5.3.4.1 for more information.

6.3.2 GET-DDB-LEVEL2 Payloads

This section defines the Request Message and Response Message DDB-PDU payloads for the GET-DDB-LEVEL2 service.

6.3.2.1 GET-DDB-LEVEL2 Request Message Payload

The payload of the Request Message shall contain the Offset (section 5.3.2.1) and the RequestLength (section 5.3.2.2) and shall be transferred in the order as specified in Figure 10.

Octet	7	6	5	4	3	2	1	0
0	Offset [15:8]							
1	Offset [7:0]							
2	RequestLength [15:8]							
3	RequestLength [7:0]							

Figure 10 GET-DDB-LEVEL2 Request Message Payload Format

6.3.2.2 GET-DDB-LEVEL2 Response Message Payload

The payload of the Response Message shall contain the ResultCode (section 5.1.1.4), the DataLength of the DdbL2Data and the DdbL2Data values (section 5.3.2.4) and shall be transferred in the order as defined in Figure 11.

DdbL2Data shall be transferred in the order of increasing index values.

Octet	7	6	5	4	3	2	1	0
0	ResultCode Flags							
1	ResultCode Status							
2	DataLength [15:8]							
3	DataLength [7:0]							
4	DdbL2Data							
	.							
	.							
	.							
(DataLength + 4) - 1	DdbL2Data							

Figure 11 GET-DDB-LEVEL2 Response Message Payload Format

6.3.3 SET-DDB-LEVEL2 Payloads

6.3.3.1 SET-DDB-LEVEL2 Request Message Payload

The payload of the Request Message shall contain the Offset (section 5.3.2.1), the DataLength of the DdbL2Data and the DdbL2Data values (section 5.3.2.4) and shall be transferred in the order as defined in Figure 12.

DdbL2Data shall be transferred in the order of increasing index values.

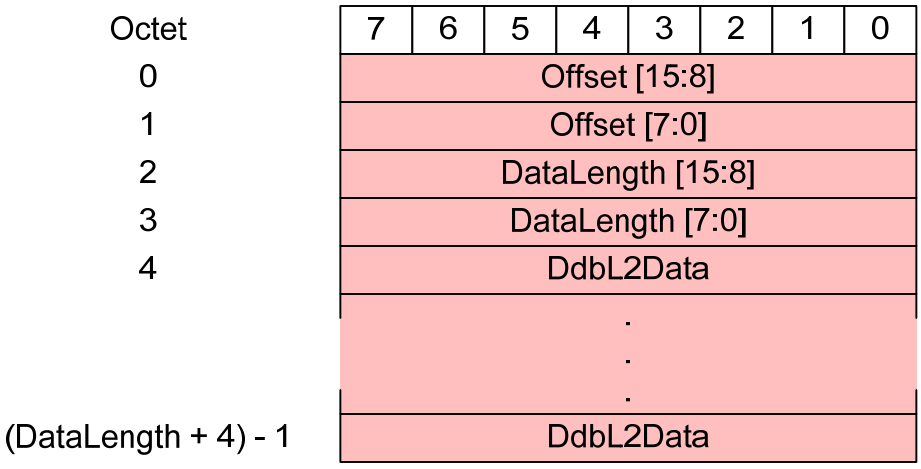


Figure 12 SET-DDB-LEVEL2 Request Message Payload Format

6.3.3.2 SET-DDB-LEVEL2 Response Message Payload

The payload of the Response Message shall contain the ResultCode (section 5.1.1.4) and the ResponseLength (section 5.3.2.3) and shall be transferred in the order as defined in Figure 13.

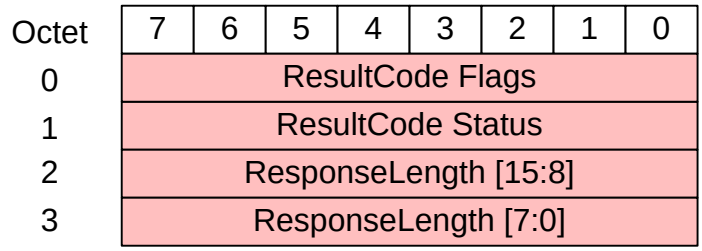


Figure 13 SET-DDB-LEVEL2 Response Message Payload Format

Annex A DDB Error Flow (informative)

Figure 14 shows the simplified communication flow from the request Primitive initiated by the Requestor's Service User through the Service Providers and the Transport to the Provider's Service User. Note, not all error conditions are shown in the diagram.

In cases where there is a ResultCode denoting an error, the ResultCode flags are set to indicate the source of that error. This information is intended as a system debugging aid – the Requester Service User may ignore the flag bits. The ResultCode flags set depend on the activity lane in Figure 14 where the error is detected, and use the values listed in Table 4.

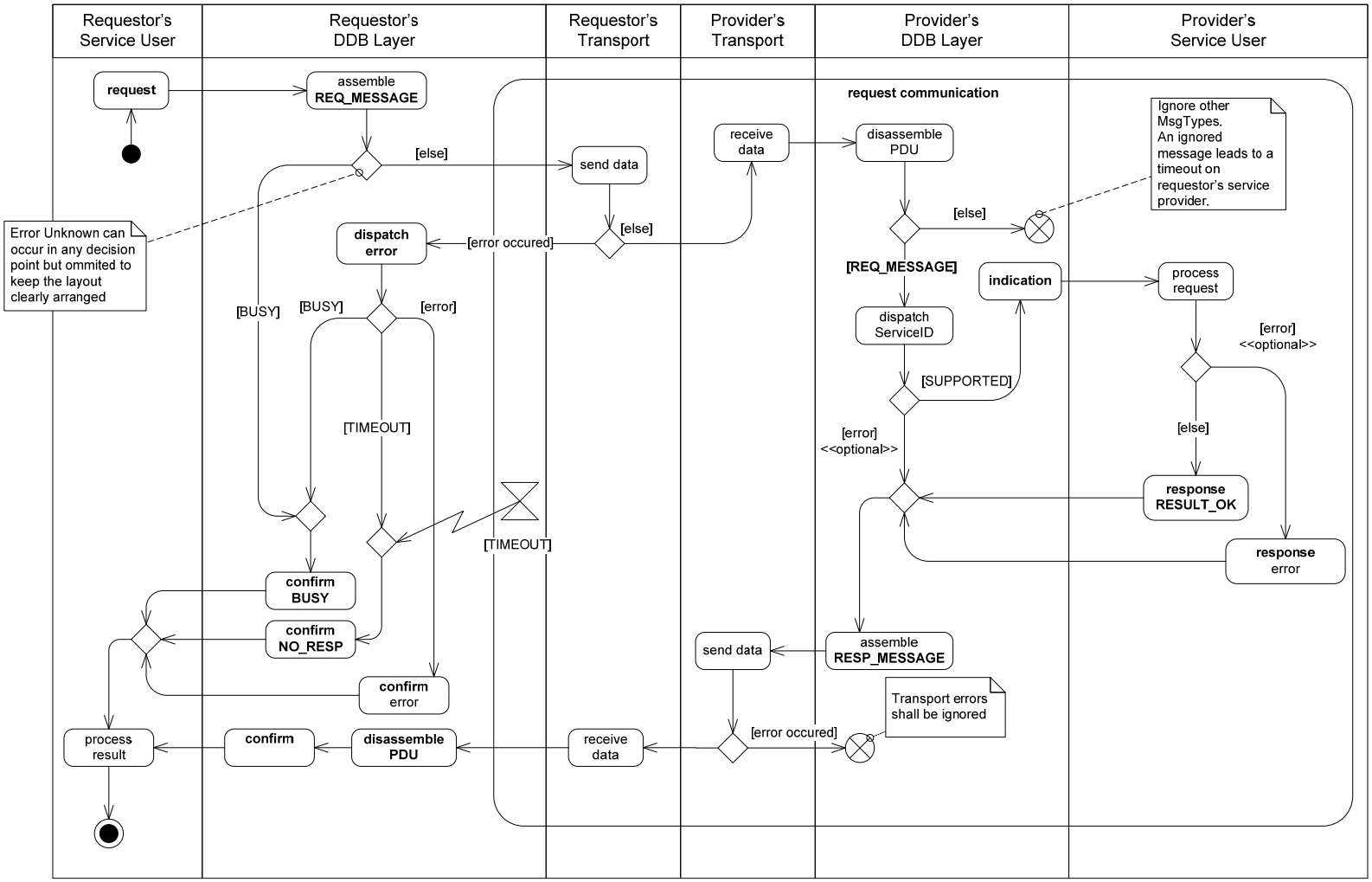


Figure 14 DDB Error Flow