Car Connectivity Consortium

MirrorLink®

Common Data Bus

Version 1.1.3
(CCC-TS-016)

VERSION HISTORY

Version	Date	Comment
1.1	31 March 2012	Approved Version
1.1.1	24 September 2012	Approved Errata Version
1.1.2	05 March 2013	Approved Errata Version
1.1.3	05 November 2013	Approved Errata Version

CONTRIBUTORS

Bose, Raja — Nokia Corporation
Brakensiek, Jörg (Editor) — Nokia Corporation
Park, Keun-Young — Nokia Corporation

LEGAL NOTICE

TABLE OF CONTENTS

TERMS AND ABBREVIATIONS

BT	Bluetooth
CDB	Common Data Bus
IP	Internet Protocol
MMU	Memory Management Unit
OOB	Out-of-Band
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
UPnP	Universal Plug-and-Play

1 INTRODUCTION

This document is part of the MirrorLink specification, which specifies an interface for enabling remote user interaction of a mobile device via another device. This specification is written having a car head-unit to interact with the mobile device in mind, but it will similarly apply for other devices, which do provide a colored display, audio input/output and user input mechanisms.

This specification describes the Common Data Bus, a data exchange mechanism which can be utilized by applications running on the MirrorLink platform. The Common Data Bus is a simple extensible transport mechanism which:

- Provides a common bus between two end-points for bi-directional exchange of data over the same transport medium.
- Provides a light-weight way to host services on a client.
- Useful for applications which do not have heavy bandwidth requirements.
- Avoids use of multiple network socket connections for each independent client-server application thereby reducing processing and memory overhead and improving performance.
- Has low implementation overhead.
- Adheres to a simple yet functionally complete protocol.
- Is application agnostic.
- Makes no assumption about APIs utilized by either end-point to communicate locally with their own hosted applications.

In this document we cover the details of the Common Data Bus system architecture, message types, messaging protocol, examples of services which can utilize it and error recovery.

The specification lists a series of requirements, either explicitly or within the text, which are mandatory elements for a compliant solutions. Recommendations are given, to ensure optimal usage and to provide suitable performance. All recommendations are optional.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are following the notation as described in RFC 2119 [2].

1. MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
2. MUST NOT: This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
3. SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
4. SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
5. MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

2 COMMON DATA BUS ARCHITECTURE

The Common Data Bus (CDB) is a simple, low-bandwidth shared bus, which allows exchanging data between two CDB endpoints, residing in the MirrorLink server and client. The Common Data Bus is fully symmetrical, i.e. services can be provided on both endpoints independently from each other.

2.1 General Architecture

A CDB endpoint can host a CDB Sink endpoint and a CDB Source endpoint. CDB Sink endpoints are subscribing to data services provided from CDB Source endpoints. A CDB source endpoint can provide multiple data services from data sources. A CDB sink endpoint can deliver data from multiple data sources to multiple data sinks. The endpoints are responsible for marshalling and de-marshalling of all the data from multiple applications passing through the common data bus. A CDB data sink subscribes to a service, provided from a data source service. For simplicity, the following figure shows two CDB services, which one data source and data sink each.

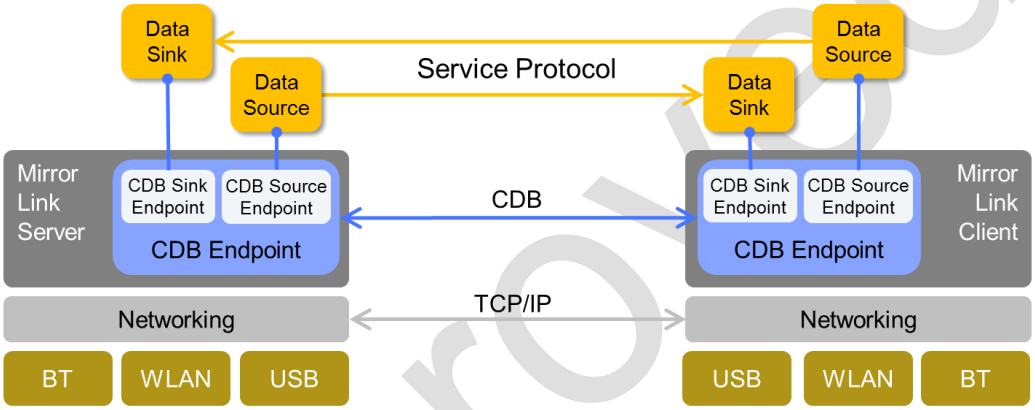

Figure 1: Common Data Bus Architecture with TCP binding

2.2 Bindings

Within this specification, the Common Data Bus has been defined with a TCP binding.

2.2.1 TCP Binding

2.2.1.1 Launching the Common Data Bus

The MirrorLink server, providing Common Data Bus functionality, MUST include the Common Data Bus into its application listing as specified in [1]. The MirrorLink Client MUST identified the Common Data Bus from the application listing as specified in [1].

The Common Data Bus MUST be started from the MirrorLink client, using the UPnP TmApplicationServer:1 service LaunchApplication action [1]. The MirrorLink client's CDB endpoint MUST open a TCP connection to the server's CDB endpoint, using the returned URL from the LaunchApplication action.

The MirrorLink client SHOULD launch the MirrorLink server's CDB endpoint not later than 10s after receiving the first A_ARG_TYPE_AppList response from the MirrorLink server.

The MirrorLink client MUST have launched the MirrorLink server's CDB prior starting the first VNC based remote application.

2.2.1.2 Intentionally Terminating the Common Data Bus

The MirrorLink client and server can terminate the server's Common Data Bus endpoint anytime, using the CDB ByeBye message and the UPnP TmApplicationServer:1 services, as defined in [1].

The CDB endpoint in the MirrorLink client MUST use the following sequence to terminate the CDB operation:

1. Client CDB endpoint MUST send a CDB `ByeBye` message. The CDB client MUST NOT send any further CDB messages, after sending the CDB ByeBye message. The CDB client endpoint SHOULD ignore all incoming CDB messages, after sending a CDB ByeBye message.
2. Server CDB endpoint MUST respond with a CDB ByeBye message.
3. Client CDB endpoint MUST disconnect the TCP connection. The CDB client SHOULD disconnect the TCP connection, if it does not receive the CDB ByeBye message back within 5s.
4. CDB server SHOULD disconnect the TCP connection on detection of the client TCP disconnect or 5s after sending the CDB ByeBye message, whatever comes first.

Client CDB endpoint SHOULD send a UPnP TmApplicationServer:1 service TerminateApplication action for the server CDB endpoint.

The CDB endpoint in the MirrorLink server MUST use the following sequence to terminate the CDB operation:

1. Server CDB endpoint MUST send a CDB `ByeBye` message. The Server CDB endpoint MUST NOT send any further CDB messages after sending the CDB ByeBye message. The CDB endpoint SHOULD ignore all incoming CDB messages, after sending a CDB ByeBye message.
2. Client CDB endpoint MUST disconnect the TCP connection
3. Server CDB endpoint MUST signal the CDB endpoint's termination to the client, if it has subscribed to the TmApplicationServer:1 AppStatusUpdate event.
4. Server CDB endpoint SHOULD disconnect the TCP connection on detection of the client TCP disconnect or 5s after sending the CDB ByeBye message, whatever comes first.

Note: If the CDB is terminated prior to the establishment of the TCP connection, steps 1, 2 and 4 MUST be omitted.

2.2.1.3 Unintentionally Terminating the CDB Session

Unintentional termination of the CDB session MAY happen any time in case of error conditions. In this case the respective CDB server or client endpoint will disconnect the TCP connection. The respective counterpart SHOULD disconnect as well.

If the MirrorLink Client decides to re-establish the CDB session, it MUST follow the steps given in Chapter 2.2.1.1.

To avoid the CDB server or client endpoint being in a TCP TIME-WAIT time-out loop, as a result of an unintentional active disconnect, the TCP socket SHOULD be established using the SO_REUSEADDR option (or similar platform specific variants), allowing the operating system to reuse a port address, even it is currently in the TIME-WAIT state or the CDB server endpoint SHOULD use a different, unaffected port number.

2.2.2 Other Bindings

Besides TCP/IP, it will be also possible to run MirrorLink Common Data Bus on top of other protocol like Bluetooth RFCOMM, but how to discover and establish connection for such configuration is outside the scope of this specification.

2.3 Testing Considerations

If the MirrorLink Client is in a dedicated testing state (as part of the MirrorLink Certification), it MUST launch a new CDB session (either initiated automatically or manually from the user), whenever the CDB server endpoint has intentionally terminated the CDB session.

If the MirrorLink Client is in a dedicated testing state (as part of the MirrorLink Certification), it MUST launch a new CDB session (either initiated automatically or manually from the user), whenever the CDB server endpoint has unintentionally terminated the CDB session.

3 MESSAGE TYPES AND FORMAT

The Common Data Bus (CDB) defines the following messages, which are specified in more detail in the following paragraphs.

- **ServicesRequest:** Requests the list of supported services
- **ServicesSupported:** Provide a list of supported data services
- **StartService:** Request to start a specific service
- **StopService:** Request to stop a specific service
- **ServicePayload:** Deliver service specific payload
- **ServiceResponse:** Response to StartService, StopService, ServicePayload
- **ByeBye:** Terminates the Common Data Bus
- **Ping:** Message to check the connection
- **PingResponse:** Responds to a Ping message

All U16 values are encoded in big endian.

3.1 ServicesRequest

The `ServicesRequest` message is used from the CDB Sink endpoint to request the list of supported services from the CDB Source endpoint.

# bytes	Type	Value	Description
2	U16	0xB101	Message-type
2	U16	2	Payload length
1	U8	1	CDB sink endpoint major version
1	U8	1	CDB sink endpoint minor version

Table 1: ServicesRequest Message

After the Common Data Bus connection is initiated between the 2 CDB endpoints, each CDB sink endpoint can send a `ServicesRequest` message to indicate that it is interested in getting the list of services available at the CDB source endpoint and is also interested in getting updates whenever the list changes.

If a CDB endpoint supports CDB sink functionality, the CDB Sink endpoint MUST send a `ServicesRequest` message within 5 s after the CDB connection has been established.

The CDB Source endpoint MUST respond with a `ServicesSupported` message to the `ServicesRequest` message within 5s. The CDB Sink endpoint SHOULD NOT send a `ServicesRequest` message, if it does not support service subscription.

In case a response is not received within 5s, the CDB Sink endpoint MUST assume that the CDB Source is not providing any service. No further action is required.

3.2 ServicesSupported

The `ServicesSupported` message MUST be used from the CDB Source endpoint to notify the CDB Sink endpoint about the data services it can provide.

# bytes	Type	Value	Description
2	U16	0xB102	Message-type
2	U16	4+M	Payload length
1	U8	1	CDB source endpoint major version
1	U8	1	CDB source endpoint minor version

# bytes	Type	Value	Description
2	U16	N	Total number of Services
M	Array of U8		Array of service descriptions, as defined in Table 3.

Table 2: ServicesSupported Message

The CDB Source endpoint version MUST be equal or smaller than the CDB Sink endpoint's version. Otherwise the CDB Sink endpoint MUST send a `ServiceResponse` message with the Error code 0x0209 in response to the received `ServicesSupported` message.

Each service is described as given in the following table.

# bytes	Type	Value	Description
2	U16		Unique Service ID
1	U8		Major version of service
1	U8		Minor version of service
1	U8	*Bit*	*Service Configuration ('1' enabled, '0' disabled)*
		[0]	Service Encryption Data payload is encrypted with the data provider's session key, exchanged as within the DAP of the server's CDB endpoint.
		[1]	Service Resource Constraints Only one service can be started, in case multiple entries with same service name, but different version, exist in the services supported list.
1	U8	*Bit*	*Service Access Control ('1' enabled, '0' disabled)*
		[0]	Unlimited Data can be provided to any application.
		[1]	CCC-Certified Data can be provided to any application, certified from the Car Connectivity Consortium.
		[2]	Source-Certified Data can be provided to any application, certified from the manufacturer of the source device.
		[3]	Service-Certified Data can be provided to any application, certified from the manufacturer of the source device to access the data service.
1	U8	N	Indicates the length of service name string.
N	Array of U8		Service name string

Table 3: Service Description

Each service MUST have a service ID, which is unique during a CDB session. Service IDs for services provided from the MirrorLink server MUST have the range of 0x4001 to 0x7FFF, and service IDs for services from the MirrorLink client MUST have the range of 0x0001 to 0x3FFF. The most significant bit is reserved for future use and MUST be set to 0. The CDB Source endpoint MUST NOT change the Service ID number during a CDB session.

Setting more than one Service Access Control bit, will aggregate (i.e. OR) the access control conditions. The source grants no access to a data service, if all Service Access Control bits in the above table are set to zero (0). The relationship between the Service Access Control layers is shown in Figure 2.

The manufacturer of the source device MUST be identified from the <manufacturer> entry in the UPnP TmClientProfile:1 SetClientProfile action. If the client does not use the service, or the <manufacturer> entry

is left empty, the Source- and Service-Certified Service Access Control bits are ignored (i.e. assumed to be zero (0)).

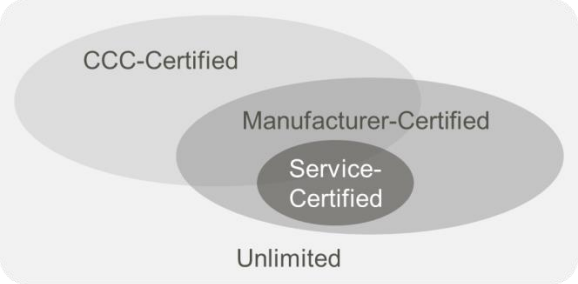

Figure 2: Service Access Control Layers

Service names MUST follow the naming convention *domainName*.*serviceName*, where *domainName* is following the Java namespace convention (e.g. com.daimler). MirrorLink specified services MUST use "com.mirrorlink" as the domain name.

A CDB Source endpoint MUST send a ServicesSupported message for every ServicesRequest message that it receives. However, after the receipt of the first ServicesRequest message, a CDB Source endpoint MUST send a ServicesSupported message, whenever the list of available services changes, even if the other endpoint has not sent any subsequent ServicesRequest messages.

The CDB source endpoint MUST list only one service with identical service name string and major version. Any CDB services MUST be backward compatible with regard to its minor versions.

A ServicesSupported message MUST contain the complete list of services currently available. In case a CDB Source endpoint does not have any services available, it MUST respond with a ServicesSupported message where the Total number of Services (see Table 2) is set equal to 0.

3.3 StartService

The StartService message is used from the CDB Sink endpoint to signal to the CDB Source endpoint to start a specific service and commence putting data packets related to that data service on to the common data bus. The following table specifies the format of the message.

# bytes	Type	Value	Description
2	U16	0xB103	Message-type
2	U16	4	Payload length
2	U16		Service Id
1	U8		Major version of service
1	U8		Minor version of service

Table 4: StartService Message

The service version MUST be equal or smaller than the announced service version from the ServicesSupported message. The CDB Source endpoint MUST respond with a ServiceResponse message with the corresponding Service Id within 5s. Otherwise the CDB Source endpoint MUST respond with a ServiceResponse message (Error Code set to 0x0010 – Response pending) latest every 5s until the final response message is available. The CDB Source endpoint MUST respond with a ServiceResponse message (Error Code set to 0x0201 – Launch failed) if no response can be provided within 2 min.

In case the ServiceResponse messages are not received in time, the CDB Sink endpoint MUST consider the launch finally failed and MUST send a StopService message.

3.4 StopService

The `StopService` message is used from the CDB Sink endpoint to signal to the CDB Source endpoint to stop a specific service and stop putting data packets related to that service on to the common data bus. The following table specifies the format of the message:

# bytes	Type	Value	Description
2	U16	0xB104	Message-type
2	U16	2	Payload length
2	U16		Service Id

Table 5: StopService Message

The CDB Source endpoint MUST respond with a `ServiceResponse` message with the corresponding Service Id within 5s.

In case the `ServiceResponse` message is not received in time, the CDB Sink endpoint MUST consider the termination finally failed. No further action is required.

3.5 ServicePayload

The `ServicePayload` message delivers data from one CDB endpoint to the other one for any of the services, which have been started. The payload can have any data format of its own depending on the available services. The specification of the service payload is done within separate specifications and is out of the scope of this specification. The following table specifies the format of this message:

# bytes	Type	Value	Description
2	U16	0xB105	Message-type
2	U16	3+M	Payload length
2	U16		Service Id
1	U8		Service Access Control
M	Array of U8		Payload for the Service

Table 6: ServicePayload Message

The Service Access Control byte of the `ServicePayload` message MUST be identical to the Service Access Control byte from the respective service in the `ServicesSupported` message.

In case the Service Encryption bit is set to '1' in the Service Configuration bit of the Service listing, the entire Service Access Control byte and the Service Payload MUST be encrypted. Each CDB endpoint MUST use its allocated encryption mechanism and session key. The definition of the encryption mechanism and session key are outside the scope of this specification.

The service payload length M, as defined in Table 6, MUST be equal or smaller than 8,100 bytes. Both MirrorLink client and server MUST be able to receive a service payload up to this size.

Any service using the Common Data Bus, MAY split long payloads into smaller service payloads, fulfilling the service payload length requirement. In such case, it is up to each service to fragment and re-assemble those longer packets.

The CDB Sink endpoint MUST respond with a `ServiceResponse` message with the corresponding Service Id, if the `ServicePayload` message's Service Id is unknown or if the Service Configuration bits have changed.

3.6 ServiceResponse

The `ServiceResponse` message MUST be used from the CDB Source or Sink endpoint to respond to a message send from the CDB Sink or Source endpoint respectively (refer to Table 12 for details). The following table specifies the format of this message:

# bytes	Type	Value	Description
2	U16	0xB107	Message-type
2	U16	4	Payload length
2	U16		Service Id
2	U16		Response Value

Table 7: ServiceResponse Message

The error types are defined in the following table.

Response Value		Description
0x0001	Ok – Service started	Service has been successfully started.
0x0002	Ok – Service stopped	Service has been successfully terminated.
0x0010	Response pending	Response pending for another 5s
0x0101	Warning – Service running	Service Id used in a `StartService` message refers to an already running service.
0x0102	Warning – Service not running	Service Id used in a `StopService` or `ServicePayload` message refers to a not-running service.
0x0201	Error – Service Launch failed	Service cannot be launched CDB Source endpoint MUST delist the service and send an updated `ServicesSupported` message.
0x0202	Error – Service Termination failed	Service cannot be terminated CDB Sink endpoint MUST ignore ServicePayloads from the service in question. CDB Sink endpoint MUST terminate the CDB.
0x0203	Error – Service reset	Service is reset from CDB Source endpoint. Service can be immediately started again using `StartService` message.
0x0204	Error – Service terminated	Service is terminated from CDB Source endpoint Service is (temporarily) unavailable. CDB Source endpoint MUST send an updated `ServicesSupported` message.
0x0205	Error – Unknown Service Id	Service Id used in a `ServicePayload`, `StartService` or `StopService` message does not exist.
0x0206	Error – Wrong Access Control	Access Control in `ServicePayload` is wrong. CDB sink endpoint MUST stop the service.
0x0207	Error – Resource busy	Resource constraint service not started. Service resources are busy.
0x0208	Error – Wrong Sequence Number	Sequence number in PingResponse is wrong. Receiving CDB endpoint MUST terminate CDB.
0x0209	Error – Wrong CDB version number	Wrong version number in the `ServicesRequested` or `ServicesReported` message. CDB MUST be terminated.
0x020A	Error – Wrong Service version number	Invalid version number in the `StartService` message. Service is not started.
0x020B	Error – Unsupported Payload Format	The payload format used in the `ServicePayload` message is not supported from the Data Service.

Response Value		Description
		CDB sink endpoint MUST stop the service.

Table 8: Error Types for ServiceResponse message

3.7 ByeBye

This message MUST be used from either the CDB Source or Sink endpoint to terminate the Common Data Bus endpoint. The following table specifies the format of this message:

# bytes	Type	Value	Description
2	U16	0xB109	Message-type
2	U16	0	Payload length

Table 9: BusError Message

Sending and receiving a `ByeBye` message MUST terminate the Common Data Bus, according chapter 2.2.1.2.

3.8 Ping

The `Ping` message MAY be used from either side, to check if the other side is still alive. The endpoint, receiving a ping message, MUST reply with a `PingResponse` message containing the sequence number. The following tables specify the format of the `Ping` message.

# bytes	Type	Value	Description
2	U16	0xB108	Message-type
2	U16	2	Payload length
2	U16		Sequence number Arbitrary value for distinguishing PingResponse messages.

Table 10: Ping Message

If a Ping message is not replied via a `PingResponse` message within 10 seconds, the Common Data bus MUST be considered non-functional. If either side recognizes the Common Data Bus to be non-functional, it MUST terminate the Common Data Bus as specified in chapter 2.2.1.2.

3.9 PingResponse

The `PingResponse` message MUST be used from either side, to respond to a `Ping` messages. The receiving endpoint SHOULD send the `PingResponse` message, prior sending any other message. The following tables specify the format of the `PingResponse` message.

# bytes	Type	Value	Description
2	U16	0xB106	Message-type
2	U16	2	Payload length
2	U16		Sequence number in reply to a `Ping` message

Table 11: PingResponse Message

4 MESSAGE FLOW

The following table summarizes the CDB Sink and Source Endpoint's message flow, it MUST follow. The sending endpoint is given in brackets in the Message Received column.

Message Received (from)	Message Responses	Comment
ServicesRequest (Sink)	ServicesSupported	-
	ServiceResponse (0x0209)	Wrong CDB version
ServicesSupported (Source)	No response	Wait for StartService
	ServiceResponse (0x0209)	Wrong CDB version
Start Service (Sink)	ServiceResponse (0x0001)	Launch ok
	ServiceResponse (0x0010)	Response pending
	ServiceResponse (0x0101)	Launch failed – Already running
	ServiceResponse (0x0201) ServicesSupported	Launch failed – Cannot launch service; update service list
	ServiceResponse (0x0205)	Launch failed – Unknown ServiceId
	ServiceResponse (0x0207)	Launch failed – resource busy
	ServiceResponse (0x020A)	Launch failed – Invalid version
Stop Service (Sink)	ServiceResponse (0x0002)	Stop ok
	ServiceResponse (0x0102)	Stop failed – Not running
	ServiceResponse (0x0202)	Stop failed – Cannot stop service
	ServiceResponse (0x0205)	Launch failed – Unknown ServiceId
ServicePayload (Sink)	No response	Payload ok
	ServiceResponse (0x0101)	Payload failed – Service not running
	ServiceResponse (0x0203)	Payload failed – Service reset
	ServiceResponse (0x0204) ServicesSupported	Payload failed – Service terminated; update service list
	ServiceResponse (0x0205)	Payload failed – Unknown Service Id
	ServiceResponse (0x0206)	Payload failed – Wrong Access Control
	ServiceResponse (0x020B)	Payload failed – Unsupported format
ServicePayload (Source)	No response	Payload ok
	ServiceResponse (0x0101)	Payload failed – Service not running
	ServiceResponse (0x0205)	Payload failed – Unknown Service Id
	ServiceResponse (0x0206) StopService	Payload failed – Wrong Access Control; stop service
	ServiceResponse (0x020B) StopService	Payload failed – Unsupported format; stop service
ServiceResponse (Sink)	No action	Other response values
	ByeBye	If response value = 0x0208
	ByeBye	If response value = 0x0209
ServiceResponse (Source)	No action	Other response values
	ByeBye	If response value = 0x0202
	StartService or No action	If response value = 0x0203
	StopService	If response value = 0x0206
	ByeBye	If response value = 0x0208

Message Received (from)	Message Responses	Comment
	ByeBye	If response value = 0x0209
	StopService	If response value = 0x020B
ByeBye (Client Endpoint)	No response	Terminate the CDB session
ByeBye (Server Endpoint)	ByeBye	Terminate the CDB session
Ping (Sink / Source)	PingResponse	-
PingResponse (Sink / Source)	No response	Ping ok
	ServiceResponse (0x0208)	Ping failed – Wrong sequence number

Table 12: CDB Message Flow

5 REFERENCES

[1] Car Connectivity Consortium, “MirrorLink - Application Server Service”, Version 1.1; CCC-TS-024

[2] IETF, RFC 2119, Keys words for use in RFCs to Indicate Requirement Levels, March 1997. http://www.ietf.org/rfc/rfc2119.txt