Car Connectivity Consortium

MirrorLink®

Service Binary Protocol

Version 1.1.2
(CCC-TS-018)

Copyright © 2011-2013 Car Connectivity Consortium LLC

All rights reserved

Confidential

VERSION HISTORY

Version	Date	Comment
1.1	31 March 2012	Approved Version
1.1.1	24 September 2012	Approved Errata Version
1.1.2	20 December 2012	Approved Errata Version

LIST OF CONTRIBUTORS

Park, Keun-Young	Nokia Corporation
Benesch, Matthias	Daimler
Brakensiek, Jörg (Editor)	Nokia Corporation
Fernahl, Dennis	Carmeq (for Volkswagen AG)
Hrabak, Robert	General Motors
Kim, Jungwoo	LG Electronics
Kim, Mingoo	LG Electronics
Tom, Alfred	General Motors

TRADEMARKS

MirrorLink is a registered trademark of Car Connectivity Consortium LLC
Bluetooth is a registered trademark of Bluetooth SIG Inc.
RFB and VNC are registered trademarks of RealVNC Ltd.
UPnP is a registered trademark of UPnP Forum.
Other names or abbreviations used in this document may be trademarks of their respective owners.

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

LEGAL NOTICE

The copyright in this Specification is owned by the Car Connectivity Consortium LLC ("CCC LLC"). Use of this Specification and any related intellectual property (collectively, the "Specification"), is governed by these license terms and the CCC LLC Limited Liability Company Agreement (the "Agreement").

Use of the Specification by anyone who is not a member of CCC LLC (each such person or party, a "Member") is prohibited. The legal rights and obligations of each Member are governed by the Agreement and their applicable Membership Agreement, including without limitation those contained in Article 10 of the LLC Agreement.

CCC LLC hereby grants each Member a right to use and to make verbatim copies of the Specification for the purposes of implementing the technologies specified in the Specification to their products ("Implementing Products") under the terms of the Agreement (the "Purpose"). Members are not permitted to make available or distribute this Specification or any copies thereof to non-Members other than to their Affiliates (as defined in the Agreement) and subcontractors but only to the extent that such Affiliates and subcontractors have a need to know for carrying out the Purpose and provided that such Affiliates and subcontractors accept confidentiality obligations similar to those contained in the Agreement. Each Member shall be responsible for the observance and proper performance by such of its Affiliates and subcontractors of the terms and conditions of this Legal Notice and the Agreement. No other license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Any use of the Specification not in compliance with the terms of this Legal Notice, the Agreement and Membership Agreement is prohibited and any such prohibited use may result in termination of the applicable Membership Agreement and other liability permitted by the applicable Agreement or by applicable law to CCC LLC or any of its members for patent, copyright and/or trademark infringement.

THE SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS, AND COMPLIANCE WITH APPLICABLE LAWS.

Each Member hereby acknowledges that its Implementing Products may be subject to various regulatory controls under the laws and regulations of various jurisdictions worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of Implementing Products. Examples of such laws and regulatory controls include, but are not limited to, road safety regulations, telecommunications regulations, technology transfer controls and health and safety regulations. Each Member is solely responsible for the compliance by their Implementing Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Implementing Products related to such regulations within the applicable jurisdictions.

Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses**.**

NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS. ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTYRIGHTS OR FOR NONCOMPLIANCE WITH LAWS, RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. BY USE OF THE SPECIFICATION, EACH MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST CCC LLC AND ITS MEMBERS RELATED TO USE OF THE SPECIFICATION.

CCC LLC reserve the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

Copyright © 2011. CCC LLC.

Approved

TABLE OF CONTENTS

Approved

1 Terms and Abbreviations

CDB	Common Data Bus
HU	Automobile Head-unit (this term may be used interchangeably with the term “Terminal Mode client”)
IP	Internet Protocol
SBP	Service Binary Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
UPnP	Universal Plug-and-Play

Approved

MirrorLink is a trademark of the Car Connectivity Consortium LLC

Bluetooth is a registered trademark of Bluetooth SIG Inc.

UPnP is a registered trademark of UPnP Implementers Corporation.

Other names or abbreviations used in this document may be trademarks of their respective owners.

1 INTRODUCTION

This specification describes top level architecture of MirrorLink data service, a mechanism to exchange meaningful data between MirrorLink Server and Client [1]. One example of such data is fuel level. By providing such data to applications, data service will lead into the creation of new applications utilizing the exchanged data. Although data service ultimately targets application development, this specification will focus on the interface between MirrorLink server and client. Providing such API to application layer SHOULD be done by each platform providers.

Following key requirements were considered in the design of data service:

- Minimizing resource requirements to support very low-end head-unit
- Allow easy creation and implementation of new data service
- Protect end-user's data
- Data service SHOULD be symmetric: Both MirrorLink server and client SHOULD be able to provide data service and subscribe to the available services from the other side.

MirrorLink data service is composed of three layers: Common Data Bus (CDB) [2], Service framework (Service Binary Protocol), and service provider/subscriber. This specification will cover top level architecture for all three layers and will cover Service framework layer in further detail. There will be a separate specification for individual services.

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 [3].

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.)

Approved

2 MIRRORLINK DATA SERVICE ARCHITECTURE

This chapter provides architectural overview of MirrorLink Data service.

2.1 Overall Architecture

MirrorLink Data service is composed of CDB, service framework and service provider / subscriber. CDB is the underlying multiplexing layer which also provides service discovery feature. On top of it, service framework layer allows implementing a new service in easier way by providing common abstraction for service provider / subscriber.

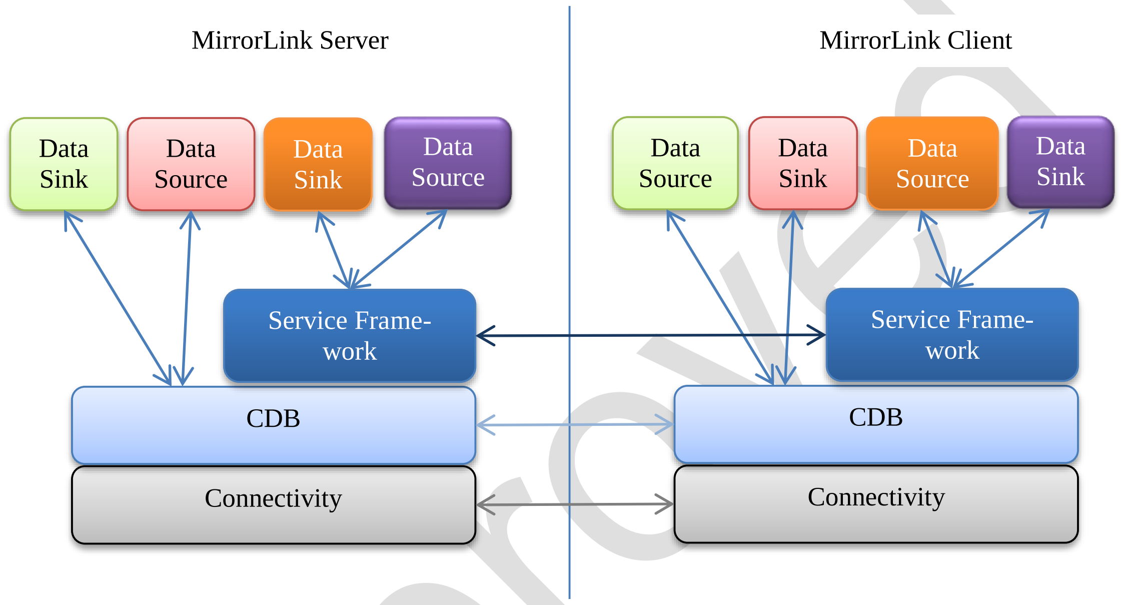

Figure 1 Top level architecture of MirrorLink data service

Figure 1 shows the top level architecture of MirrorLink data service. Underlying connectivity can be a TCP/IP session on top of physical connectivity like USB, WLAN, and Bluetooth. Besides TCP/IP, it will be also possible to run MirrorLink data service 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.

On top of the connectivity layer, the CDB layer is located. CDB relies on the connectivity layer to provide TCP like connection oriented session, and all other layers above rely on the CDB to provide communication interface.

Above the CDB can be the service framework layer or data source (service provider) / data sink (service subscriber) layer depending on the data service used. Service framework layer implements common features for individual data services to allow creating a new service easier. Some data service MAY decide not to use the service framework to re-use existing protocols or to reduce the additional overhead caused by the framework. It is highly RECOMMENDED for any new data service to consider using the service framework first. If that approach does not work, accessing directly to CDB layer can be considered. Some data service MAY open its own TCP/IP session, but such use case is outside the scope of this specification.

On top of data service framework can be service provider (data source) or service subscriber (data sink). Each data source can support up to one data sink: Zero data sink means the service is not used. As there can be only one data sink for each data source, it is up to each side of MirrorLink connection (MirrorLink server and MirrorLink client) to make sure that the service can be shared across multiple applications if necessary. Depending on the implementation, there can be one system component which can work as a data sink and can provide received data to all interested applications. Another implementation MAY allow only one application to get the data. How such access control is implemented is outside the scope of this specification, but

Approved

it is RECOMMENDED to allow multiple applications to access the data unless that data is meaningful only for selected applications.

2.2 Version convention

CDB and service framework layers are bound under the same version number provided from CDB. In other words, the service framework layer does not have separate version number. CDB version number will be updated when there is an update in CDB or service framework layer. This specification, version 1.1 goes together with version 1.1 CDB specification.

All version numbers in MirrorLink data service are composed of major version number and minor version number. A change in the major version number means incompatibility with the previous major version. A change in the minor version guarantees compatibility with the previous minor version. This policy should be maintained across all the layers of MirrorLink data service.

2.3 Starting Data Service

MirrorLink data service requires CDB as underlying layer. And to use data service, CDB should be launched by via UPnP application launch mechanism [4]. Note that there is no separate application for data service, and launching CDB is enough. More details on discovering CDB services can be found from section 2.2 of CDB specification [2]. Note that MirrorLink client needs to launch the CDB with right version number.

Once CDB is started, all available services can be discovered by using CDB ServicesRequest and ServicesSupproted messages. Then service client, either from MirrorLink client or server side, can ask service server to start the service via CDB StartService message. For details, check the CDB specification.

2.4 Data Service Security with Device Attestation Protocol

CDB can support payload encryption by using a pre-arranged session key. In the current MirrorLink architecture, the session key can be acquired after attestation of CDB in MirrorLink server side by utilizing Device Attestation Protocol [1]. The application public key generated from the attestation of CDB is the session key used for encrypting / decrypting CDB payload in MirrorLink client side. MirrorLink server will use matching private key to encrypt / decrypt CDB payload. Note that this key can be generated per each MirrorLink connection, and MirrorLink client MUST not re-use the key from the previous connections.

Approved

3 SERVICE FRAMEWORK: SERVICE BINARY PROTOCOL (SBP)

As a basic data representation mechanism in the service framework layer, CCC members have preferred binary version compared to XML mainly for performance reason. Due to that, a new binary protocol for service framework, SBP (Service Binary Protocol) was defined. Even if the service framework is based on binary protocol, it is important to allow easy service definition and future extendibility. To allow future extension, the concept of identifying each member variable by unique ID is used.

Big-endian is used for all data types. The protocol does not guarantee data alignment for compact data representation, and in most cases, data should be re-constructed from byte stream. Due to that, there is no big advantage of having little-endian instead of big-endian.

SBP assumes lossless data delivery through CDB layer. Due to that, there is no separate data integrity check, but still there can be mal-formed SBP payloads due to implementation error. Such error will be checked inside SBP.

Due to the time constraint for MirrorLink 1.1 specification, decision was made to focus on basic features in this version of specification. Following features will be addressed in this version:

- Getting and setting data
- Subscribing to a data

Following features will be added in later revisions:

- Remote Procedure Call feature
- Details about authentication. Command is defined, but specific details will be added later.
- Meta-data description
- Interface Description Language: In this draft, style similar to C++ is used for convenience, but it is not formally defined.

3.1 Service Description Example

Service description can be done by defining data objects including member variables. Mechanism for subscribing the data objects will be explained later. Let's assume an example service with the name of "com.mirrorlink.sensor_example". The name is used to uniquely identify the service in CDB layer.

Following figure shows data objects defined in the service.

```
/* com.mirrorlink.sensor_example, version 1.0 */
/** @UID: 0xD6804B4A @max_subscription_rate: 50Hz */
Object accelerometer {
  STRUCTURE accel_data {
    FLOAT x; /// @unit: m/s^2 @mandatory @UID: 0x150A2CB3
    FLOAT y; /// @unit: m/s^2 @mandatory @UID: 0x150A2CB4
    TIME time; /// @mandatory @UID: 0x00A0FDB2
  };
  STRUCTURE_ARRAY<accel_data> data; /// @UID: 0x144A776F
};
/** @UID: 0xD73DFF88 @writable @control: accelerometer */
Object accelerometer_control {
  BOOLEAN filterEnabled; /// @UID: 0x2B230C64 @optional: false
  INT samplingRate; /// @UID: 5F2BF0EC
```

Approved

```
};
/** @UID: 0x41F75401 @max_subscription_rate: 1Hz
Object thermometer {
   INT temperature; /// @UID: 0x9D28234F @unit: Celsius
};
```

Figure 2: Example Service Description

A service can be composed of one or more Objects. The example service is composed of three Objects: accelerometer, accelerometer_control and thermometer. Javadoc style [5] is used to document each object. Each object can be individually accessed by using Get, Set or Subscribe command. Details of these commands will be presented in later sections.

The accelerometer object has one member variable: data. The "data" is an array of STRUCTURE accel_data which has three members: acceleration in x direction, acceleration in y direction, and time. Note that STRUCTURE_ARRAY<XYZ> means an array of STRUCTURE XYZ. Similarly, ARRAY<XYZ> represents an array of basic type (non-STRUCTURE, non-ARRAY) XYZ. The example subscription also shows that the accelerometer object allows the maximum subscription rate of 50Hz with maximum sampling rate of 100Hz. Due to the difference in rates, one data notification can include multiple samples. Note that /** */ and /// is used for comments and additional information as in Javadoc. All objects allow reading data, but writing is allowed selectively. "accelerometer_control" object allow writing as @writable tag shows. Member variables can be either mandatory or optional. Member variables are mandatory by default, and optional member can be specified with @optional tag which can also include the specification of default value when the member variable is not present. For example, in the accelerometer_control object, filterEnabled is optional with default value of false.

Note that accelerometer_control object can be used to control the behavior of accelerometer object.

An Object can inherit other Objects or STRUCTUREs. Then all member variables defined in parents Objects/STRUCTURREs are available in the child Object. A STRUCTURE can also inherit other Objects or STRUCTUREs to re-use the already defined data layout. An example, an Object "A" inheriting a STRUCTURE "a" can be expressed as "Object A inherits STRUCTURE a {};".

3.2 Data representation

Data in SBP is represented in the following way using Extended Backus-Naur Form (EBNF) [6].

EBNF	Form No	Matching data_type
data = data_type, value |	1	BOOLEAN,BYTE, SHORT,INT,LONG, FLOAT,DOUBLE
data_type, no_elements, {value} |	2	BYTES, STRING
data_type, element_data_type, no_elements, {value} |	3	ARRAY
data_type, no_elements, { data_with_UID }, END |	4	STRUCTURE
data_type, no_elements, {data[1]}, END;	5	STRUCTURE_ARRAY
data_with_UID = UID, data;	-	-

Table 1: Binary representation of data in EBNF

1: Each data MUST have the same STRUCTURE type, and thus only data with form 4 can be placed.

The following table describes symbols used in EBNF description of data.

Approved

Category	Size	Description
data_type	U8	Tell the type of data.
UID	U32	Unique identifier of data. Hash value of data's name is used as UID.
value	8, 16, 32, 64 bits	Raw data without any addition. Size depends on the data_type.
no_elements	U32	Number of elements contained in the array, array of structure, or structure.
element_data_type	U8	data_type of elements contained in the array. This data_type can be only BOOLEAN, SHORT, INT, LONG, FLOAT, or DOUBLE. Putting other data_type MUST be treated as irrecoverable error.
END	U8	Special character (0x81) used for terminating STRUCTURE or STRUCTURE_ARRAY for checking data integrity.
data_with_UID	-	UID, data pair binding UID with data.

1 Table 2: Description of symbols

2 The following table shows all the data types with matching EBNF description to represent the data.

Name	data_type	Form	Description
BOOLEAN	0x82	1	U8, true (non-zero) or false (0).
BYTE	0x83	1	8 bits, signed integer
SHORT	0x84	1	16 bits, signed integer
INT	0x85	1	32 bits signed integer
LONG	0x86	1	64 bits signed integer
FLOAT	0x87	1	32 bits value, IEEE754-1985 single-precision
DOUBLE	0x88	1	64 bits value, IEEE754-1985 double-precision
BYTES	0x90	2	Array of BYTE
STRING	0x91	2	Array of UTF16 characters. Each character takes 2 bytes (UTF16).
ARRAY	0xA0	3	Array of basic data types (BOOLEAN, SHORT, INT, LONG, FLOAT, and DOUBLE)
STRUCTURE	0xA1	4	Generic container for heterogeneous data as in structure in C language. Note that STRUCTURE can nest another STRUCTURE inside, but creating too many depths can increase processing overhead.
STRUCTURE_ARRAY	0xA2	5	Array for the STRUCTURE of the same type. Note that this data type is not necessarily efficient in the amount of data point of view as the same meta-data is repeated for all child elements. If reducing the amount of data is important, other data type should be considered.

3 Table 3: List of data_type

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

Form column shows how each data_type can be represented in binary format. For example, ARRAY has form 3, which corresponds to Form No 3 in Table 1.

Besides what is listed above, in service description, pseudo data_type of TIME can be used. TIME is a 64 bit signed integer (LONG) with the meaning of time in milliseconds since 1970-01-01-00:00 in UTC or relative time in milliseconds depending on how it is defined in each service. Note that TIME is only used in service description level, and in SBP protocol level, TIME is always delivered as LONG.

Usage of data_type not defined in Table 4 MUST be treated as irrecoverable error.

Sequence for the placement of child elements MUST follow the service description. For example, in the example service description of Figure 2, STRUCTURE accel_data has three data members: x, y, and time. When this STRUCTURE is transmitted under SBP, the order of data MUST be x, y, and time as defined in the description. Following table shows how it will be in binary representation.

High-level	STRUCTURE accel_data {
	FLOAT x; /// @unit: m/s^2 @mandatory @UID: 0x150A2CB3
	FLOAT y; /// @unit: m/s^2 @mandatory @UID: 0x150A2CB4
	TIME time; /// @mandatory @UID: 0x00A0FDB2 };
	STRUCTURE accel_data data; / @UID: 0x144A776F */**

Binary Description	UID: “data”, data_type: 0xA1(STRUCTURE), no_elements: 3, UID: “x”, data_type: 0x87(FLOAT), value: 0, UID: “y”, data_type: 0x87(FLOAT), value: 0, UID: “time”, data_type: 0x86(LONG), value: 0, END
Binary (data_with_UID)	0x144A776F, 0xA1, 0x00000003, 0x150A2CB3, 0x87, 0x00000000, 0x150A2CB4, 0x87, 0x00000000, 0x00A0FDB2, 0x86, 0x0000000000000000, 0x81

Table 5: Example for the sequence of member variables in binary representation

If some data members are OPTIONAL, it is allowed to skip that member. But in that case, each service description should either define the default value or should provide a relevant mechanism for SBP Sink to know if some members are present or not. The latter can be done by providing an additional member variable or an Object containing such information.

3.3 Command representation

Compared to data representation, there is only one type of EBNF for command which is presented below.

```
command = command_type, payload_length, UID, packet_id, value, no_elements, {data_with_UID}, END_C;
```

Table 6: Binary representation of command in EBNF

Following table describes symbols used in EBNF of command.

Category	Size	Description
command_type	U8	Tell the type of command.
payload_length	U32	Total length of command including END_C – 5 (command_type + payload_length)
UID	U32	Unique identifier of object. Hash value of object’s name is used as UID.
packet_id	U16	Unique identifier for each packet. Value of “0” means do not care.

Approved

value	U32	Command specific value.
no_elements	U32	Number of child data elements contained in this command.
data_with_UID	-	UID, data pair as defined in Table 1
END_C	U8	Special character (0xB0) used for terminating a command.

1 Table 7: Description of symbols

2 Following table shows summary of defined commands.

Name	command_type	UID	value	Description
Get	0xB1	Object	0	Reads an object once. Depending on the object, this operation can take time.
Set	0xB2	Object	0	Write to the Object. Depending on the object, this operation can take time.
Subscribe	0xB3	Object	Subscription type and interval (ms)	Get multiple notifications for the Object. Object data is sent later depending on subscription type and interval. Subscription type takes MSB 8 bits, and subscription interval takes remaining LSB 24 bits from 32bits “value”.
Cancel	0xB4	Object	command_type	Cancels the currently active command (Get, Set, and Subscribe) with the given command_type.
AliveRequest	0xB5	0	0	Request the SBP Source to send AliveResponse.
AliveResponse	0xB6	0	0	Reply for AliveRequest.
AuthenticationChallenge	0xB7	Object or Function UID	authentication method	Authentication challenge when an Object which requires authentication is accessed. Data passed are defined by each authentication method. In this version, only service specific authentication is allowed, and service specific authentication MUST use the authentication method value of 0x80000000.
AuthenticationResponse	0xB8	Object or Function UID	Error code	Authentication response for the challenge. For the current version of the specification, AuthenticationResponse MUST return Feature not supported error code except for service specific authentication.
Response	0xB9	Object / Function	Error code	This is a response for Get, Subscribe, Cancel, and AuthenticationResponse.
Reserved commands	0xBA to 0xBF	-	-	This range is reserved for next update for Call and other features. For the current version of specification, SBP Source MUST return a Response with “Feature not supported” error code when command in these ranges is received.

3 Table 8: List of commands

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

Currently following subscription types are defined:

Subscription type	Description
0x0, regular interval	Send update in regular interval with interval (ms) specified in 24bits subscription interval. When the interval is smaller than what SBP Source supports, SBP Source SHOULD return error.
0x1, on change	Send update when there is a change. When there is too frequent changes, SBP Source can decide to either drop some updates or combine multiple updates into single one if data structure allows it.
0x2, automatic	It is up to SBP Source to decide either to choose regular interval or on change. SBP Source can choose the optimal notification mechanism for the requested Object.

Table 9: subscription_type in Subscribe command

The SBP Source MUST support Get, Set, Subscribe, Cancel, and AliveRequest commands. Except for the AliveRequest command, when the current service does not support these commands for the given object, SBP Source MUST return Response with "Feature not supported" error code.

The SBP Sink MUST support Response, and AuthenticationResponse commands.

Note that each command can access one Object as a whole. Each Object can include member variable with different data types like STRUCTURE, but it cannot include another Object. STRUCTURE can also include member variable, but unlike Object, STRUCTURE cannot be individually accessed by command. A STRUCTURE can be only accessed as a member variable of an Object or Objects.

Note that SBP Sink and SBP Source MUST recognize all the commands defined. If a specific command is not supported by a specific Object, SBP Sink and SBP Source MUST return proper error code like "Feature not supported" or "Write not allowed". It is up to each service to decide if specific command is supported for the specific Object, but Get and Cancel command MUST be supported unless specified otherwise in the service specification.

3.4 Command Sequences

This section shows how commands are related with each other by showing sequences and examples. Note that each command sequence MUST share the same packet_id. Any AuthenticationChallenge-AuthenticationResponse phase, which is actually a sequence, following a Get/Set/Subscribe command, MUST have the same packet_id as the first command.

A Cancel command sequence for a pending Get/Set/Subscribe command MUST have a different packet_id as the pending command.

A command sequence with wrong packet_id MUST be ignored by both SBP Sink and SBP Source.

All command sequences are initiated by the SBP Sink. Upon receiving the initial command, the SBP Source SHOULD send reply within 5 seconds. If the SBP Source fails to send a reply within 5 seconds, the SBP Sink SHOULD treat that request as an error, like notifying the upper layer with time-out error, and following responses from the SBP Source, which arrive later, can be ignored. Depending on the command, some operation like Get and Set can take more time and MAY NOT be completed within 5 seconds. In that case, SBP Source SHOULD send Response message with error code of "continue" (Response-Continue from now on). Then SBP Source can spend more time. Note that the SBP Source can send multiple of Response-Continue in some regular interval until the requested operation is completed. For example, if a Set request for an object takes 11 seconds, SBP Source can send two Response-Continue at 4 seconds and 8 seconds later. Then at 11 seconds, SBP Source will send the final reply. In this example, SBP Source is sending Response-Continue earlier than the 5 seconds time-out as it can take time for the message to arrive to the SBP Sink.

SBP Sink SHOULD NOT send the same command to the same object until the currently active command sequence is completed. SBP Source MUST return "Command already pending" error code upon detecting such situation.

Approved

As all commands are processed in asynchronous way, SBP Source needs to guarantee that the certain number of command sequences can be processed at a time. It is up to each service to define the maximum number of active commands that MUST be supported, and SBP Source SHOULD guarantee at least that number of active commands. In the case when SBP Source cannot process a new command due to resource limitation, SBP Source MUST return an error code, “no more session”. Note that cancelling existing active command SHOULD work always as it is not adding a new active command.

Note that, in all the examples in this section, original name is shown as UID, but actual data carried is hash value of the name rather than the name itself.

3.4.1 Get, [{Response-Continue}], Response

Get is used to fetch an Object data once. For protected object, OPTIONALLY, authentication can be requested in the middle. When the authentication stage is included, service SBP Source will send AuthenticationChallenge message and SBP Sink MUST respond with AuthenticationResponse. After the OPTIONAL authentication, Response comes from SBP Source. There can be multiple Response-Continue message in the middle if it takes time to get the requested data.

Following table shows example of the data exchange for the example service.

1. SBP Sink: command_type: Get, payload_length , UID: “accelerometer”, packet_id: 1, 0, 0, END_C
2. SBP Source: command_type: Response, payload_length, UID: “accelerometer”, packet_id: 1, value 0 (OK), …, END_C

Table 10: Example of Get command sequences

Get command can be sent without first subscribing the object. Note that the SBP Sink can send Get, Set, and Subscribe commands without having dependency on any other commands. For example, SBP Sink can send Set command without sending Subscribe or Get command beforehand.

3.4.2 Set, [{Response-Continue}], Response

Set is used to set an object to the desired state. Set operation is similar to writing to hardware registers which will trigger some action. As it is the case with hardware register, successful write does not necessarily mean that the object, when read back, will have the same value as requested via Set.

The example below shows the case where SBP Source is sending Response-Continue message once as the process took some time.

1. SBP Sink: command_type: Set, payload_length , UID: “accelerometer_control”, packet_id: 2, 0, 2, member data, END_C
2. SBP Source: command_type: Response, payload_length , UID: “accelerometer_control”, packet_id:2, value: continue error, 0, END_C
3. SBP Source: command_type: Response, payload_length, UID: “accelerometer_control”, packet_id: 2, value 0 (OK), 0, END_C

Table 11: Example of Set command sequence

3.4.3 Subscribe, {Response-OK/NOK}, [{Response}]

Subscribe command is used to request asynchronous notification for the object. Notification can be requested either in regular interval or on change of data. Depending on the service, sending data in regular interval or on change MAY NOT make sense, and in that case, only one mechanism will be supported. SBP Source SHOULD answer to the Subscribe request from SBP Sink within 5 seconds by sending Response command, which just tells if the subscription request is successfully accepted or not. If SBP Source fails to send initial Response within 5 seconds, SBP Sink SHOULD treat it as recoverable error. If SBP Sink treat it as an error, SBP Sink MUST send Cancel command to cancel the current subscription. Between Subscribe and first Response, OPTIONALLY, there can be an authentication stage. Once the subscription request is successfully accepted, there can be multiple Response command from SBP Source which delivers the requested data. Note

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

that the 2nd Response with data can take time depending on the data, and there is no 5 seconds limitation for the Response.

An example of the Subscribe sequence is presented below.

1. SBP Sink: command_type: Subscribe, payload_length , UID: “thermometer”, packet_id: 3, type: 0, interval: 1000 (1Hz), 0, END_C
2. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 3, value 0 (OK), 0, END_C
3. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 3, value 0, 1, UID: “temperature”, data_type: INT, value: 0, END_C
4. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 3, value: error, 0, END_C

Table 12: Example of Subscribe command sequences

In the last part of the sequence, the SBP Source is sending Response with an error message. Such error message stops the currently active subscription and the SBP Sink needs to send Subscribe command again to get notification if the problem is temporary.

3.4.4 Cancel, Response

Cancel command stops the currently active Get, Set, or Subscribe command. Upon receiving this command, SBP Source SHOULD cancel the processing for the requested command. Following example shows how the thermometer object, subscribed before, can be cancelled.

1. SBP Sink: command_type: Subscribe, payload_length , UID: “thermometer”, packet_id: 3, type: 0, interval: 1000 (1Hz), 0, END_C
2. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 3, value 0 (OK), 0, END_C
3. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 3, value 0, 1, UID: “temperature”, data_type: INT, value: 0, END_C
4. SBP Sink: command_type: Cancel, payload_length , UID: “thermometer”, packet_id: 4, value: Subscribe, 0, END_C
5. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 4, value 0 (OK), 0, END_C
6. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 3, value 0x1000000B (Successfully cancelled), 0, END_C

Table 13: Example of Cancel command sequences (Subscribe)

The following example shows how a get request to the thermometer object is cancelled.

1. SBP Sink: command_type: Get, payload_length , UID: “thermometer”, packet_id: 3, 0, 0, END_C
2. SBP Sink: command_type: Cancel, payload_length , UID: “thermometer”, packet_id: 4, value: Get, 0, END_C
3. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 4, value 0 (OK), 0, END_C
4. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 3, value 0x1000000B (Successfully cancelled), 0, END_C

Table 14: Example of Cancel command sequences (Get)

The following example shows, how a set request to the thermometer object is cancelled.

1. SBP Sink: command_type: Set, payload_length , UID: “accelerometer_control”, packet_id: 3, 0, 2, member data, END_C
2. SBP Source: command_type: Response, payload_length , UID: “accelerometer_control”, packet_id:3, value: continue error, 0, END_C

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

3. SBP Sink: command_type: Cancel, payload_length , UID: “thermometer”, packet_id: 4, value: Set, 0, END_C
4. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 4, value 0 (OK), 0, END_C
5. SBP Source: command_type: Response, payload_length , UID: “thermometer”, packet_id: 3, value 0x1000000B (Successfully cancelled), 0, END_C

Table 15: Example of Cancel command sequences (Set)

Note that the SBP Source will be able to detect, whether the SBP Sink has canceled a GET, SET or a SUBSCRIBE command, from the value entry in the Cancel command.

After sending the Response for the Cancel command, SBP Source SHOULD NOT send Response messages with data for the requested command any more. SBP Sink MUST treat such situation as recoverable error and MUST ignore such response as sending a response for cancelled command can happen due to the asynchronous nature of command-response.

3.4.5 AuthenticationChallenge, AuthenticationResponse

SBP Source can send AuthenticationChallenge after receiving Get/Set/Subscribe command for an object which requires authentication. The current specification does not define any authentication mechanism, and version 1.0 service SBP Source SHOULD NOT use this command unless service specific authentication is defined. But it can happen that a SBP Source with newer SBP version sends the AuthenticationChallenge command to the version 1.0 SBP Sink. In that case, the SBP Sink MUST reply with AuthenticationResponse with the “Feature not supported” error code. Then the SBP Source SHOULD send Response message to the original command with authentication failed error code. Note that upon receiving AuthenticationChallenge command, SBP Sink SHOULD send AuthenticationResponse within 5 seconds. If SBP Sink fails to send the AuthenticationResponse within 5 seconds, the SBP Source MUST send Response with “Authentication failed” error code. Some SBP Source MAY terminate the CDB session after sending the Response for failed authentication, but it is up to each service to define such behavior.

3.4.6 AliveRequest, AliveResponse

AliveRequest is used by SBP Sink to check if the SBP Source is alive or not. As it is the case with other commands, upon receiving this command, SBP Source SHOULD reply with AliveResponse within 5 seconds. Failure to do that MUST be interpreted as an irrecoverable error by SBP Sink.

3.5 Hash as UID

In SBP, hash value of object name or member variable name is used as a UID. Due to this, each service description SHOULD make sure that there is no conflict in hash value.

- All object names MUST have unique hash value inside the service. Hash conflict across different services does not matter.
- Each member variable inside an object with the same depth MUST have unique UID inside that object. UID conflict with member variable of other object does not matter.

If there is a conflict in hash value, either name SHOULD be changed into something else or character like”_” SHOULD be appended. To avoid conflict in UID, it is RECOMMENDED to include UID value in the specification of each service.

Following pseudo-code, which is adopted from ETCH protocol [7] shows the algorithm to calculate hash value for a name given as an 8-bits character string:

```
int hash(char array name) /* name is an array of U8 character */
  int hash = 5381; /* int is 32 bit signed */
```

Approved

```
foreach char c in name
        h6 = hash<<6;
        hash = (h6<<10) + h6 - hash + c;
return hash;
```

Figure 3: Pseudo code for calculating hash

3.6 Error handling

This section describes how error SHOULD be handled. Error can be classified into irrecoverable error and recoverable error. Additionally, there are errors which SHOULD be just ignored.

3.6.1 Irrecoverable error

Irrecoverable error is an error when integrity of the other side cannot be trusted any more. Both SBP Sink and SBP Source, upon receiving this kind of error MUST terminate the current session of the service in CDB level either by sending CDB StopService message or by sending CDB ServiceResponse message with error code of “service reset” as defined in the CDB specification.

3.6.1.1 Unknown data type

Adding new data type breaks compatibility with old version of service framework as the size of data cannot be determined. This situation SHOULD have been avoided by checking version number in UPnP stage or CDB stage, but if this case happens due to other reason like lost synchronization, it MUST be treated as an irrecoverable error. Another case when this error can happen is the wrong element_data_type in ARRAY data type. Array data type can have element_data_type of BOOLEAN, SHORT, INT, LONG, FLOAT, and DOUBLE. Setting other data type MUST be treated as irrecoverable error.

If SBP Source detects this error for the command received from SBP Sink, SBP Source SHOULD send Response command with error code to notify the SBP Sink about the error, if sending Response is allowed in the current command sequence. In case of SBP Sink, when unknown data type is detected for the command received from SBP Source, the SBP Sink MUST terminate the current session immediately.

3.6.1.2 Wrong END: END check failure for form 4, 5 or command

STRUCTURE, STRUCTURE_ARRAY, or command does not terminate with END/END_C after receiving the expected number of elements. END and END_C are used to check the integrity of data payload, and when END/ENC_C are not discovered in the expected location, it MUST be treated as critical error. This case is treated as irrecoverable error as the data included cannot be trusted any more. As it is the case for unknown data type, upon detecting this error, SBP Source SHOULD send Response command with error code before terminating the session. This error is different from the case when either side sends more child elements than what was specified. In that case, no_child_elements will still match with total number of child elements, and it is not an irrecoverable error.

3.6.2 Recoverable error

Recoverable error is an error which SHOULD be replied with error code if sending reply is possible in the command sequence. If command sequence does not allow sending reply, this error SHOULD be ignored. All error code not marked as irrecoverable can be considered as a recoverable error. Complete list of recoverable errors is presented in Table 16.

3.6.2.1 Unknown Object UID

This error happens when UID for the command sent from either side is unknown. SBP Source MUST reply with error code, “unknown UID”. When the SBP Sink receives unknown UID as part of response to a command, the SBP Sink SHOULD ignore it.

Approved

3.6.2.2 Unknown command type

When SBP Sink sends an unknown command, SBP Source MUST reply with a Response with the error code of “Unknown Command’. When SBP Source sends a command with unknown type, the SBP Sink SHOULD ignore it.

3.6.2.3 Unsupported feature

When any SBP endpoint sends a command of an unsupported feature, the other SBP endpoint MUST reply with a Response with the error code of “Feature not supported”.

3.6.3 Error to ignore

Some errors are to be ignored as ignoring such case allows future extension without breaking compatibility.

3.6.3.1 Unknown UID for member variable

This situation can happen when a new member variable is added to a service. Even if one side does not support the latest version with the new member variable, the other side can still send the object data with new member variable. Upon receiving such member variable with unknown UID, either side MUST just ignore the child element and SHOULD proceed to the next element in the received object data. This alleviates the need to send different versions of objects depending on the service version.

3.6.4 Error code definition

Following table gives the list of error code defined. Error code in the range of 0x1 to 0x0fffffff is allocated for irrecoverable error. Error code from 0x10000000 to 0x3fffffff is allocated for recoverable error. As new error code can be added in the future, range of the error code SHOULD be checked first rather than checking individual error code.

Error code	Description
0	OK, no error
0x1	Unknown data_type: data_type is unknown. This error is irrecoverable.
0x2	Wrong END: BYTES, STRING, ARRAY, STRUCTURE, STRUCTURE_ARRAY, or command does not terminate with END/END_C after receiving the expected number of children. Or END/END_C is found in wrong place. This error is irrecoverable.
0x3	Wrong element_data_type. This is the case when SBP Source has set wrong data type as element_data_type of array. As the size of child data cannot be predicted, this error is irrecoverable.
0x4	UID and type does not match. The type bound with UID does not match with the type actually transferred.
0x01000000	Irrecoverable error in either in SBP Source or SBP Sink side due to implementation specific reason like no memory.
0x10000000	Continue. SBP Source needs more time to process the request. This is error code for Response-Continue message.
0x10000001	Unknown UID: unrecognized object UID for the service.
0x10000002	Feature not supported.
0x10000003	Wrong subscription interval. Error code for Subscribe command.
0x10000004	Wrong subscription type. The type is not supported by the service.
0x10000005	Missing mandatory data. Mandatory member variable is missing.

Approved

0x10000006	Not available. The requested data is currently unavailable.
0x10000007	Authentication failed.
0x10000008	Command already pending. Error code when the same command is sent again before the previous one is completed.
0x10000009	Command not pending. Error code for Cancel command when the command is not pending.
0x1000000A	No more session. SBP Source cannot support new commands until currently active commands are completed.
0x1000000B	Command successfully cancelled. When a cancel request is successful, this error code SHOULD be returned. Note that sending OK response for cancel command will mean successful completion of the command, not the cancellation. Packet_id MUST be the same as the original GET, SET or SUBSCRIBE command.
0x1000000C	Write not allowed. The Object does not allow writing. This is the error SBP Source MUST return when Set command is sent to an Object which does not support writing.
0x1000000D	Unknown command. This is the error code to respond when a command not defined in Table 8 is received.
0x11000000	Recoverable error in SBP Sink or SBP Source side due to implementation specific reason.
0x40000000 to 0x4fffffff	Reserved for service specific error code. Each service can define a new error code in this range.

Table 16: List of error code

3.7 Authentication mechanism

The purpose of authentication is for SBP Source to verify if SBP Sink has valid permission to access the resource. MirrorLink CDB already has mechanism to restrict access to selected applications. But there MAY be service specific needs to have additional authentication.

Also note that authentication does not protect the data from eavesdropping. To protect against the eavesdropping, the whole service can be encrypted using CDB's payload encryption mechanism.

The current version of specification does not have any built-in mechanism for authentication. Details on how service specific authentication SHOULD be done will be defined later. If necessary, each service can define its own authentication mechanism.

3.8 Support of optional Objects

The current version of SBP does not have mechanism to list available Objects or to retrieve meta-data. It is up to each service to pass necessary meta-data information. SBP Sink can check if an optional Object is supported or not by trying to Get or Subscribe the Object. If the Object is not supported, SBP Source MUST return "unknown UID" error.

3.9 Version listing and selection

This is supported in CDB level. Once CDB StartService message, which includes version selection, is received, the version is maintained while the session is maintained.

Approved

3.10 Initialization Sequence

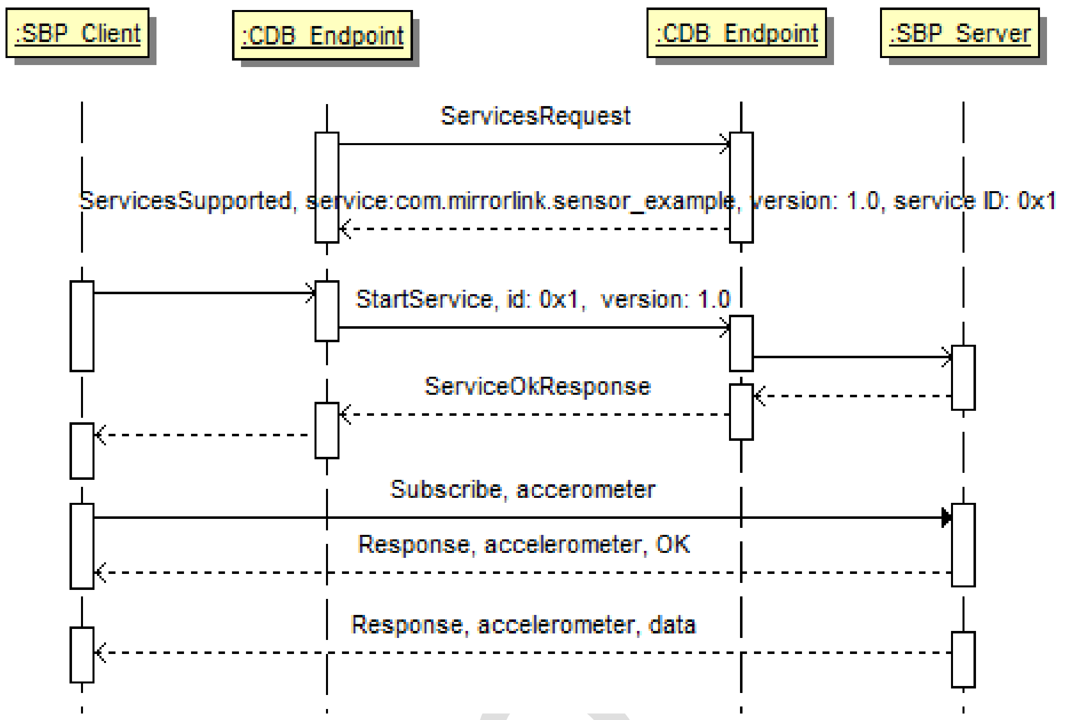
:SBP Client
:CDB Endpoint
:CDB Endpoint
:SBP Server
ServicesRequest
ServicesSupported, service:com.mirrorlink.sensor_example, version: 1.0, service ID: 0x1
StartService, id: 0x1, version: 1.0
ServiceOkResponse
Subscribe, accerometer
Response, accelerometer, OK
Response, accelerometer, data

Figure 4: Example starting sequence of CDB/SBP

The above figure shows initialization sequence of SBP with CDB.

1. CDB Endpoint in the SBP Sink side requests the list of supported services by sending ServicesRequest message. Before the step, SBP Source MAY register itself to CDB endpoint, but that step is not shown.
2. The message is replied with ServicesSupported message which shows the example service, com.mirrorlink.sensor_example.
3. The availability of the service is discovered by / informed to SBP Sink in platform specific mechanism. SBP Sink requests the start of the service to SBP Source via CDB StartService message with preferred version of 1.0.
4. The request succeeds and ServiceOkResponse is received in CDB layer. All subsequent messages are SBP messages which are delivered via CDB ServicePayload message.
5. The SBP Sink requests the subscription of accelerometer object via Subscribe command.
6. The SBP Source returns OK with Response command to notify that the subscription is successful.
7. Later, when a data is available, the SBP Source sends the accelerometer data via Response command.

3.11 Other topics

This section covers topics that were frequently asked.

3.11.1 Extending a service

Adding a new data member to existing object is an easy way to extend existing service without breaking compatibility with already deployed counter-parts.

Compatibility can break when the data type of an existing member variable is changed. In such case, there SHOULD be a change in major version number.

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

3.11.2 Payload fragmentation

Each SBP command MUST be delivered by one or more than one CDB ServicePayload messages. CDB layer can do the optimization of combining multiple ServicePayload into one TCP packet, but such concatenation SHOULD NOT happen in service framework level. If a command is too big to fit into single CDB Payload message, it MUST be fragmented into multiple CDB ServicePayload message. Even in that case, data of two different commands MUST not be mixed in one CDB ServicePayload message. Either side, upon detecting such payload, MUST handle it as irrecoverable error. Support of the fragmentation allows data services to exchange bigger data than 8KB. Fragmented payload can have arbitrary size, but the first payload MUST include command_type, payload_length, UID, packet_id, value, and no_elements. As a result, fragmentation can happen only right before data, right after data, or in data.

3.11.3 Inheritance

In the service description, inheritance can be used to avoid defining the same type again and again. Each service description using inheritance needs to make sure that the final data generated can be bounded. For example, a loop in inheritance relationship will create a data with infinite length which cannot be used. Each service description also needs to make sure that all member variables, whether it is from inheritance or not, have unique UID.

3.11.4 Shutdown clean-up and reconnection

When a SBP session is closed due to normal shut-down or shut-down caused by irrecoverable error, SBP Source SHOULD close all active commands by itself. If the service is still available in CDB service list, and the SBP Sink requests the service again, SBP Source MUST guarantee that the previous shut-down does not prevent the current session's normal operation.

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

4 REFERENCES

[1] Car Connectivity Consortium, “MirrorLink – Core Architecture”, version 1.1; CCC-TS-032

[2] Car Connectivity Consortium, “MirrorLink – Common Data Bus”, version 1.1; CCC-TS-016

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

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

[5] http://en.wikipedia.org/wiki/Javadoc

[6] http://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_Form

[7] https://cwiki.apache.org/ETCH/home.html

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

APPENDIX A BINARY REPRESENTATION (DATA_WITH_UID) EXAMPLE

This chapter will give examples of various data and command representation in binary forms.

High-level	**INT aaa = 1;** /* 32bit signed integer */
Binary Description	UID: “aaa”, data_type: 0x85(INT), value: 1 (0x00000001)
Binary	0x27E6B6DC, 0x85, 0x0000001

Table 17 Binary representation of INT

High-level	**BYTES bbb = {1, 2, 3, 4};** /* byte array of 4 bytes */
Binary Description	UID: “bbb”, data_type: 0x90(BYTES), no_elements: 4, value: 1 2 3 4
Binary	0x2865C69D, 0x90, 0x00000004, 0x1, 0x2, 0x3, 0x4

Table 18 Binary representation of BYTES

High-level	**ARRAY<INT> ccc = {1, 2, 3, 4};** /* Array of 32bit signed integer, 4 elements */
Binary Description	UID: “ccc”, data_type: 0xA0(ARRAY), element_data_type: 0x85(INT), no_elements: 4, value: 0x00000001 0x00000002 0x00000003 0x00000004
Binary	0x28E4D65E, 0xA0, 0x85, 0x00000004, 0x00000001, 0x00000002, 0x00000003, 0x00000004

Table 19 Binary representation of INT ARRAY

High-level	STRUCTURE str{
	INT a;
	INT b;
	};
	STRUCTURE str s = {1, 2}; /* a = 1, b = 2 */

Binary Description	UID: “s”, data_type: 0xA1(STRUCTURE), no_elements: 2, UID: “a”, data_type: 0x85(INT), value: 1, UID: “b”, data_type 0x85(INT), value: 2, END
Binary	0x150A2CAE, 0xA1, 0x00000002, 0x150A2C9C, 0x85, 0x00000001, 0x150A2C9D, 0x85, 0x00000002, 0x81

Table 20 Binary representation of STRUCTURE

High-level	STRUCTURE str{
	INT a;
	INT b;
	};
	STRUCTURE_ARRAY<str> s_array = {{1, 2}, {3,4}};

Approved

Copyright © 2011-2013 Car Connectivity Consortium LLC. All rights reserved.
Confidential

	/* 1st STRUCTURE: a = 1, b = 2 2nd STRUCTURE: a = 3, b = 4 */
Binary Description	UID: “s_array”, data_type: 0xA2(STRUCTURE_ARRAY), no_elements: 2, data_type: 0xA1(STRUCTURE), no_elements: 2, UID: “a”, data_type: 0x85(INT), value: 1, UID: “b”, data_type 0x85(INT), value: 2, END, data_type: 0xA1(STRUCTURE), no_elements: 2, UID: “a”, data_type: 0x85(INT), value: 3, UID: “b”, data_type 0x85(INT), value: 4, END, END
Binary	0xBFCB5248, 0xA2, 0x00000002, 0xA1, 0x00000002, 0x150A2C9C, 0x85, 0x00000001, 0x150A2C9D, 0x85, 0x00000002, 0x81, 0xA1, 0x00000002, 0x150A2C9C, 0x85, 0x00000003, 0x150A2C9D, 0x85, 0x00000004, 0x81, 0x81

Table 21 Binary representation of STRUCTURE_ARRAY

High-level	Object Obj1{
	STRUCTURE str member;
	};
	Obj1.member = {1, 2}; /* a = 1, b = 2 */
	Set Obj1

Binary Description	command_type: 0xB2(Set), payload_length: 39, UID: “Obj1”, packet_id: 1, value: 0, no_elements: 1, UID: “member”, data_type: 0xA1(STRUCTURE), no_elements: 2, UID: “a”, data_type: 0x85(INT), value: 1, UID: “b”, data_type 0x85(INT), value: 2, END, END_C
Binary	0xB2, 0x0000002B, 0x43AF649F, 0x0001, 0x00000000, 0x00000001, 0xF19C0ABF, 0xA1, 0x00000002, 0x150A2C9C, 0x85, 0x00000001, 0x150A2C9D, 0x85, 0x00000002, 0x81, 0xB0

Table 22 Binary representation of Call command

Approved