Car Connectivity Consortium

MirrorLink®

Android-Specific Specifications

Version 1.1.2
(CCC-TS-056)

VERSION HISTORY

Version	Date	Comment
1.1.0	30 April 2014	Approved Version
1.1.1	14 May 2014	Approved Errata Version
1.1.2	11 February 2015	Approved Errata Version

LIST OF CONTRIBUTORS

Pichon, Ed — E-Qualus (for CCC)
Brakensiek, Jörg — Microsoft Corporation
Sorin Basca (Editor) — RealVNC Ltd

TRADEMARKS

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, the Developer Agreement found on the Developer Portal ("Developer Agreement") and the CCC LLC Limited Liability Company Agreement (the "LLC Agreement").

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

FOR MEMBERS AND DEVELOPERS

CCC LLC hereby grants each Member and Developer a right to use and to make verbatim copies of the Specification for the purposes of implementing the technologies specified in the Specification in their products ("Implementing Products") under the terms of the LLC Agreement or Developer Agreement, as appropriate (the "Purpose"). No other license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

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.

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 PROPERTY RIGHTS 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 reserves the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

Each Member or Developer, as appropriate, (i) 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, (ii) 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, and (iii) acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations or licenses**.**

FOR DEVELOPERS ONLY

Any use of the Specification not in compliance with the terms of this Legal Notice and the Developer Agreement is prohibited and any such prohibited use may result in termination of the Developer Agreement and in other liability as permitted by the Developer Agreement or by applicable law to the CCC LLC or any of its Members for patent, copyright and/or trademark infringement. Developers are not permitted to make available or distribute this Specification or any copies thereof to any third party.

FOR MEMBERS ONLY

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

This Specification may not be provided to any third party other than to Affiliates of Members (as defined in the LLC 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 LLC 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 LLC Agreement.

TABLE OF CONTENTS

1

TERMS AND ABBREVIATIONS

ACMS	Application Certification Management System
ML	MirrorLink
OCSP	Online Certificate Status Protocol
UPnP	Universal Plug and Play
AIDL	Android Interface Definition Language

1 ABOUT

This document provides the elements of the MirrorLink specification that apply only to Android MirrorLink Server devices. MirrorLink Server devices that use the Android Operating System MUST comply with the requirements listed in this document.

The specification lists a series of requirements, either explicitly or within the text, which are mandatory elements for 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 [1].

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 PLATFORM-SPECIFIC SPECIFICATION CONCEPT OVERVIEW

In order to support third-party applications within a MirrorLink session, the MirrorLink protocols require certain information be provided to the MirrorLink Server's software (the MirrorLink "Stack") and to the Application Certification Management System (ACMS), and that certain functionality be exposed to those applications (the Common API). In order to prevent fragmentation of the application ecosystem, simplify implementation for MirrorLink Server Device developers, and to increase the number of devices that a given application can be run on, these systems should be common to a given mobile device platform. The goal being that a MirrorLink application written for Android, for example, should be able to run on all MirrorLink-certified Android devices. It is understood that differences of versions and hardware capabilities limit the ability to ensure cross-device compatibility – however the intent is that MirrorLink should not create additional barriers to such cross-compatibility.

This document contains the requirements for MirrorLink Server devices that utilize the Android OS.

3 APPLICATION IDENTIFIER

As described in the Application Certificate Handling specification [5], each application must have a unique application identifier (App ID) that is provided to the Application Certification Management System (ACMS) via the HTTP GET and OCSP requests sent to it by the MirrorLink Server device. This AppID needs to be unique for that application, and change whenever the application is modified. For Android devices, the App ID is generated using the below method. Source code that implements the below algorithm is provided in Appendix B – App ID Generation Code.

3.1 Format

The application ID for an Android application MUST be a URL-safe base-64 encoding [9] of a SHA-256 digest. As the digest of SHA-256 is 32 bytes long the application ID will therefore be a string of 43 URL-safe base-64 characters.

The application ID MUST not place any padding characters at the beginning or end of the encoding. This ensures all implementations will generate an identical application ID and prevents the need to escape any characters when querying the Application Certificate Management System (ACMS).

3.2 Calculation

The data to be hashed MUST be the concatenation of the following in the order specified:

1. String encoding of Android package name provided in AndroidManifest.xml [10].[10]
2. Big endian 8 byte integer representing the version code provided in AndroidManifest.xml [11].
3. The string “startManifestMain”
4. For each attribute in the main section of the APK manifest (META-INF/MANIFEST.MF), sorted lexicographically by unicode code point of the attribute name:
    a. String encoding of the full attribute name
    b. String encoding of the attribute value
5. The string “endManifestMain”
6. For each file named in the main section of the APK manifest (META-INF/MANIFEST.MF), sorted lexicographically by unicode code point of the file path:
    a. The string “startFile”
    b. Skip this file if the path is “assets/self-signed.ccc.crt”
    c. String encoding of the file path
    d. For each attribute in the file section of the APK manifest, sorted lexicographically by unicode code point of the attribute name:
        i. String encoding of the full attribute name
        ii. String encoding of the attribute value
    e. The string “endFile”

The encoding of strings MUST be a big-endian 4-byte integer specifying the length in bytes followed by the UTF-8 encoding of the string.

Example: The string “Hi Σ” would be encoded as the following bytes:

```
0x00 0x00 0x00 0x05          (Length of UTF-8 string, 5 bytes)
0x48 0x69 0x20 0xCE 0xA3     (UTF-8 encoding of string)
```

Therefore an example APK for an application with the following properties:

- Package name of “com.example.a”
- Version code of 124
- Containing the following files
    - assets/image.png
    - assets/self-signed.ccc.crt
    - META-INF/CERT.RSA

- META-INF/CERT.SF
- META-INF/MANIFEST.MF
- AndroidManifest.xml
- classes.dex
- resources.arsc

- A META-INF/MANIFEST.MF containing the following:

```
Manifest-Version: 1.0
Created-By: 1.0 (Android)

Name: classes.dex
SHA1-Digest: eEcd5Q6I3GDCkZ7gAAsC0dB8KxU=

Name: resources.arsc
SHA1-Digest: IEBCJEW4Ws8uS/ML7RgEqwGYvtU=

Name: assets/image.png
SHA1-Digest: gHoXviEAT+JdNMqsvo/vERe231Y=

Name: assets/self-signed.ccc.crt
SHA1-Digest: r/HlpR8FRHOKcsF4o5PFSeV32yk=

Name: AndroidManifest.xml
SHA1-Digest: /czwhhK4YJmcwq9E/qHoB26/K90=
```

Would have an application ID which is the SHA-256 digest of the concatenation of the following data:

```
"com.example.a"
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x7C
"startManifestMain"
"Created-By"
    "1.0 (Android)"
"Manifest-Version"
    "1.0"
"endManifestMain"
"startFile"
"AndroidManifest.xml"
    "SHA1-Digest"
          "/czwhhK4YJmcwq9E/qHoB26/K90="
"endFile"
"startFile"
"assets/image.png"
    "SHA1-Digest"
          "gHoXviEAT+JdNMqsvo/vERe231Y="
"endFile"
"startFile"
"classes.dex"
    "SHA1-Digest"
          "eEcd5Q6I3GDCkZ7gAAsC0dB8KxU="
"endFile"
"startFile"
"resources.arsc"
    "SHA1-Digest"
          "IEBCJEW4Ws8uS/ML7RgEqwGYvtU="
"endFile"
```

where the strings are encoded using the method described above. Please note that the quotation marks are not included in the calculation, they are here just to indicate the exact string which is going to be hashed.

To see a working example for calculating the AppID please check the Android Application ID Generator [14] which contains a sample APK. The README file provided contains the expected application ID.

Please note that the Mirror Link Server SHOULD reject any APK which contains the same filename twice. An APK with the same filename multiple times could be used to mask some unwanted data, therefore the Server MUST treat it as untrusted.

3.2.1 Shared UIDs

In Android APKs are allowed to shared UIDs. Each APK must be treated as an application as a whole and if two or more APK use the same shared UID, each of them will be considered an application in its own right. Each of those APKs must be certified independently and each of them will have a unique application ID.

When an application connects to a service of the MirrorLink Server it MUST provide the package name of the APK it lives in. The server then MUST validate that the package name has the UID of the process making the request. This is because when an IPC request is being made, in Android, there is no way to tell the package from which the request came, only the UID can be retrieved.

3.2.2 ROM applications

Application identifiers of apps built-into the device firmware (aka ROM) MAY be extracted by the MirrorLink Server from the self-signed certificate instead of being dynamically generated at runtime. This may be necessary due to firmware optimizations, which change the structure of an APK thus making it difficult to maintain a unique application identifier for the same application across different firmware variations. The application identifier of the optimized application within the self-signed certificate MUST match the application identifier of the same, un-optimized application.

The application MUST contain a self-signed certificate, following the requirements in section 4.1. The MirrorLink Server MUST have an integrity check method for build-in ROM applications to verify the integrity of such build-in applications, before granting the exemption. Integrity check MAY be done by validating that the ROM application was signed with the same ROM signature key (platform private key).

If an updated version of a ROM-based application is installed as a standard APK, the MirrorLink Server MUST treat it as an updated version of the ROM application, MUST cease making OCSP requests in relation to the ROM-based version, and MUST NOT list the ROM-based version in the UPnP Application Listings. Updated versions of built-in applications installed as a standard APK MUST be treated like normal MirrorLink applications.

3.3 APK Validation

An APK which contains files outside of the ‘META-INF’ directory with no MANIFEST.MF entries will be rejected by the Android platform. If a file is present in the MANIFEST.MF but does not a have a *-Digest entry then this will also be rejected by the Android platform. This is because both of these are requirements for a valid JAR signature [12].

An application ID MUST NOT be generated by an ID generation tool, for an APK with files outside of the ‘META-INF’ directory with no MANIFEST.MF entry. An application ID also MUST NOT be generated for an APK containing files without a ‘*-Digest’ entry in the MANIFEST.MF.

An application ID MUST NOT be generated also when the same filename is present more than once in the APK. This is to avoid any attempts to hide files within the APK. The MirrorLink Server is responsible itself for rejecting such applications.

A MirrorLink server MAY validate the signature of the APK in addition to the APK signature being validated by the Android platform.

4 APPLICATION INFORMATION

As described in section 4.1 of the Handling of Application Certificates specification [5], an application SHOULD come with a MirrorLink developer self-signed certificate Application Certificate for the purposes of providing the information needed by the MirrorLink Server to list the application in the A_ARG_TYPE_AppList as described in Table 2.2 of [1], and to control interaction with the ACMS as described in [5]. This section describes how the Application Certificate is generated and included with the application for Android MirrorLink Server devices, as well as any alternate methods for providing that information.

4.1 Self-Signed Application Certificates

The MirrorLink servers running on an Android platform MUST allow for Self-Signed Application Certificates as described in Section 4.1 of [5].

4.1.1 Application Certificate Self-Signing Process

To create a MirrorLink aware application a Self-Signed X.509 Certificate MUST be created. The certificate MUST include the CCC extension headers as described in section 3.1 of [5]. The key used MUST be a 2048-RSA key and the hashing must be done using SHA-256 or SHA-512 signature algorithms.

The self-signed certificate MUST be signed with a key pair generated by the developer. The Jarsigner utility included in the SDK, as described in [13], SHOULD be used to sign the Self-Signed Application Certificate.

4.1.2 Self-Signed Application Certificate Installation

The Self-Signed Certificate MUST be placed in a file called “self-signed.ccc.crt”. The certificate file MUST be placed in the assets folder of the APK containing the application.

The MirrorLink server SHALL use the “self-signed.ccc.crt” file as the self-signed Application Certificate as described in Section 4.1 of [5].

4.2 Alternative to Self-Signed Application Certificates

The Android platform SHALL NOT use an alternate method for providing the information contained in the Self-Signed Application Certificates as described in Section 4.1 of [5].

4.3 Particulars of the Android Application Certificates

This section describes any items which are in the Application Certificates that must be adapted to the Android OS.

4.3.1 Platform version

The platform version in the certificates MUST match the constants defined in Build.VERSION_CODES. So for example if an application is available for Ice Cream Sandwich devices, the constant value of Build.VERSION_CODES.ICE_CREAM_SANDWICH should be used, which is the integer 14. The string representation MUST be decimal, signed and without any leading 0, or white spaces.

4.3.2 Icon URLs

The Icon URL field MUST be left empty, therefore the iconList element MUST be omitted from the Application Certificate. The MirrorLink Server MUST use the icon provided with the Activity which accepts the LAUNCH and TERMINATE MirrorLink-specific Intents. See 6.1.1.

Having the URL field empty means that in future versions this can be used without a break in compatibility.

The omission of the iconList element is allowed, since it is optional. In [5], chapter 5.3.2, A_ARG_TYPE_AppList, the Server is mandated to copy all the entries from the Application Certificate into the matching A_ARG_TYPE_AppList entries, while leaving the missing entries to the default values. This does not apply to the iconList for the Android Servers, as it is easiest to provide the icon data through the Android specific means.

4.4 Localized strings for the entries in the application certificate

The application certificate provides various strings without any localization information. In order to support localized versions of the strings, the application will provide them as string resources as follows:

Certificate element	Resource name
n**ame**	“com.mirrorlink.app.certificate.name”
providerName	“com.mirrorlink.app.certificate.providerName”
providerURL	“com.mirrorlink.app.certificate.providerURL”
description	“com.mirrorlink.app.certificate.description”

Each resource may contain localized versions. The Server will retrieve the resource string for the locale of the device.

5 DEVELOPMENT CERTIFICATES

Android MirrorLink Server devices will support the development of MirrorLink Applications. This section of the specification covers those aspects of the MirrorLink specification that address development of applications as described in the Handling of Application Development Certificates specification [7].

5.1 Device ID

All Android MirrorLink Servers SHOULD use the IMEI (or MEID/ESB for CDMA devices) as provided by the TelephonyManager's getDeviceId method as the Device ID. If an IMEI (or MEID/ESB) is not available, the Android MirrorLink Server SHALL use the serial number as reported by the android.os.SystemProperties ro.serialno property.

See Section 2 of the Handling of Application Developer Certificates Document [5] for more information on the Device ID.Application Development Certificates

The Android platform uses Application Developer Certificates as described in Sections 3 and 4 of the Handling of Application Developer Certificates Document [5]. These certificates differ from the Self-Signed Application Certificate by the inclusion of an additional extension containing the Developer ID, which can be obtained from the ACMS. If the Android MirrorLink Server determines that the self-signed application certificate contains a developer ID extension, it will be treated as an Application Developer Certificate.

Application Developer Certificates follow the same signing and installation requirements for Self-Signed Application Certificates, as described above.Developer ID Entry

The Android MirrorLink server MAY provide a mechanism for the user of the device to enter one or more developer IDs into the server, as part of functionality available as part of the "developer mode" menu. (In Android 4.2, this can be reached by tapping on the Build Number entry 7 times in the About Phone screen.) The Android MirrorLink server MUST attempt to retrieve the Developer ID certificate for all entered Developer IDs using the Device ID as described above.

If the MirrorLink server does not provide a mechanism for entering the Device IDs, the Android MirrorLink server MUST attempt to retrieve the Developer ID certificate for all Developer IDs reported in all Application Development Certificates available on the device.

See Section 4 of the Handling of Application Developer Certificates Document [Insert Reference Here] for more information on requesting Developer ID certificates.

5.2 Manual Developer ID Certificate Revocation Checks

The Android MirrorLink server SHALL allow for the user of the phone to immediately request the Developer ID Certificate associated with the Developer ID. This capability SHOULD be provided as part of the "developer mode" menu as described above.

6 COMMON API IMPLEMENTATION

All MirrorLink Servers using the Android platform MUST implement the Common API as described below.

6.1 API Overview

Before defining the actual API an overview of the mechanisms used in it should be considered:

- The API is based on the "Android Interface Definition Language" (AIDL). This should provide the necessary flexibility and extensibility of the API. Also it means that there is no need to maintain and distribute binaries to the application developers.
- Applications access the MirrorLink server by communicating with an Android service. The package name for the service MUST be unique across all Android MirrorLink servers. The name used for the package is: com.mirrorlink.android.service.

6.1.1 Android MirrorLink Server Requirements

- MirrorLink Servers MUST allow access for all applications that send the "com.mirrorlink.android.service.BIND" Intent, and that have the "com.mirrorlink.android.service.ACCESS_PERMISSION" permission. This permission SHOULD be user-grantable on the MirrorLink Server. If more than 3 seconds elapse from the launch of the application to receipt of the com.mirrorlink.android.service.BIND, then the MirrorLink server SHOULD mark the application as not MirrorLink aware and remove it from the application list. If the application responds later on, the server SHOULD add it back to the list.
- The MirrorLink server MUST respond to the Launch request successfully if the application has been found and the intent has been sent to it. If the application responds back through the Common API, then the server MUST send an Application Status Update notifying the client that the application is now in Foreground.
- The MirrorLink server MUST consider an application that unregistered from all the Services as terminated. It MAY consider it as terminated as soon as it unregisters from the Connection Manager Service (the only mandatory one for the duration of the application run), but MAY also opt to wait until it unregisters from all Services.
- The structures used in the API are defined using android.os.Bundle. This allows adding fields to the structures without affecting compatibility. Each field will be referred to by name string. All Android MirrorLink Servers MUST use these structures without modification.
- The MirrorLink Server MAY examine support for the com.mirrorlink.android.app.LAUNCH and com.mirrorlink.android.app.TERMINATE Intents in establishing which applications are MirrorLink aware (the server can ask the Package Manager to return all the applications that support the Intent).
- The MirrorLink Server MUST use the com.mirrorlink.android.app.LAUNCH and com.mirrorlink.android.app.TERMINATE Intents to Launch, or Terminate a MirrorLink aware application.
- If an application crashes during a callback, the Android OS will cause the callback to raise an exception on the MirrorLink Server side. The MirrorLink Server MUST catch that exception to detect the crash.
- The MirrorLink server MUST listen to the binderDied() callback to detect if an application has crashed outside of a callback.All the callbacks to the applications MUST be asynchronous. This is in order to avoid the application hanging and keeping the server blocked. If a callback requires a response this would be made by a call to the server. If the server wishes to declare the callback as "timed-out" it MAY have use a timer to do so, but it MUST wait at allow at least 3 seconds for the application to make the response.
- The MirrorLink server MUST be implemented for at least Android API level 14 (Android 4.0).
- The MirrorLink server MUST not allow an application to bind to a service concurrently more than once. If a second bind happens, the Server must throw java.lang.IllegalStateException.

6.1.2 Application Requirements

For an application to be MirrorLink aware there need to be some requirements that must be met:

- The application MUST listen to the com.mirrorlink.android.app.LAUNCH and com.mirrorlink.android.app.TERMINATE Intents. It MUST use the android.intent.category.DEFAULT category to filter on these intents. There MUST be only one Activity that receives each of these Intents. The same Activity MAY receive both intents, but equally the Application MAY have separate Activities for each event.
- To avoid TERMINTE triggering a move of the Application into the foreground it is recommended that the Application receives it into an Activity which has the following properties: android:excludeFromRecents=”true”; android:noHistory=”true”; android:theme=”@android:style/Theme.NoDisplay”.
- The application MUST have the com.mirrorlink.android.service.ACCESS_PERMISSION permission.
- The application, when started, either by the MirrorLink server, or by the user from a Launcher on the device, MUST bind within 3 seconds to the MirrorLink Service by sending the com.mirrorlink.android.service.BIND Intent.
- The application MUST use the Connection Manager Service from the Common API and MUST register a listener for its callbacks. It MUST stay registered for the entire duration while it is running. If the application unregisters from the Connection Manager Service, then the Server MAY consider it as not running anymore. The Server MAY also consider the application as running if it is still registered to at least one Service, even though that Service might not be the Connection Manager Service.
- The application MUST provide its package name when accessing any of the MirrorLink Services. This is in order to correctly identify the APK where the call originates from when shared UIDs are used.
- An application MUST NOT access more than once concurrently the same MirrorLink Service. There MUST be only one connection between an application and a Service.
- The application MUST not unbind from the MirrorLink server for the entire duration of it running. The unbinding MUST be made only when the application is closed.
- The application MUST call the unregister() method on any of the Managers that it bound to when it no longer needs to use them. This is to notify the MirrorLink Server that the application is no longer using one of the services.
- It SHOULD react and respond to callbacks as soon as it is reasonably possible.

6.2 Common API Library Definition

This section should define how the Common API library will be identified, so that all applications can access the API.

6.3 Data Type Definitions

For the data types defined please refer to the API javadoc files **Error! Reference source not found.**. The ata types refer back to the relevant section in the Common API Template [6].

Please note that the Common API template defines quite a few data types (for example all the variants of integers). To keep it simpler for the Java platform the types used will be: Boolean, int, Float, Double, String, typeName, typeName[] and Object.

6.4 Data Structure Definitions

For the data structures defined please refer to the API javadoc files **Error! Reference source not found.**. he data types refer back to the relevant section in the Common API Template [6].

6.5 Common API Elements

To access the Common API an application needs to send a com.mirrorlink.android.service.BIND Intent. Once the "service bind" callback is received, the application will get a reference to an ICommonAPIService object. From this the application can get the API level of the service, or retrieve the objects representing the different services (managers) provided.

The API level is not the same as the Android API level, it reflects the MirrorLink version. This document describes the API level 1 elements.

The services available are (together with the requirements for the Server and Application supporting them):

Service	Interface name	Retrieved via	MirrorLink Server
MirrorLink Device Info	IDeviceInfoManager	getDeviceInfoManager()	MUST
Certification Information	ICertificationManager	getCertificationManager()	MUST
Connection Information	IConnectionManager	getConnectionManager()	MUST
Display Related Features	IDisplayManager	getDisplayManager()	MUST
Event Related Features	IEventMappingManager	getEventMappingManager()	MUST
Context Information Related Features	IContextManager	getContextManager()	MUST
Device Status Features	IDeviceStatusManager	getDeviceStatusManager()	MUST
(CDB) Data Services	IDataServicesManager	getDataServicesManager()	MAY
Notifications	INotificationManager	getNotificationManager()	MAY

Note: If the MirrorLink server does not support a data service with an obligation of "MAY", the methods will respond with null. This is the only way for an application to find out if a service is available or not.

For information on all the services and calls associated please have a look at the API javadoc files [8]. The data types refer back to the relevant section in the Common API Template [6]. One difference from the Common API template is the fact that the callbacks already contain the information needed for the application. There is no need to call a Get function in response to a callback notifying of a change in some MirrorLink Server/connection parameters. The new values are passed in the arguments of the callbacks. This is to avoid unnecessary extra IPC calls and to mirror the Android way of doing these callbacks.

Also please note that the service interfaces are called "Managers" and callback interfaces "Listener". This is done to follow the Android practice.

As the MirrorLink Server MUST use the Android OS standard mechanisms for key events and virtual keyboards, the services for those are not needed here. For more details please see 7.

7 MIRRORLINK EVENTS

7.1 Touch Events Injections

The touch events MUST be injected into the Android system by the MirrorLink server and the application SHOULD receive them as appropriate. This allows for the system to decide which activity gets to handle the events.

7.2 Key Events Injections

The key events MUST be injected into the Android system by the MirrorLink server and the application SHOULD receive them as appropriate. This allows for the system to decide which activity gets to handle the events.

The MirrorLink server MAY choose to handle of some events that would otherwise result in an undesired framebuffer being displayed. For example it can decide to handle the receiving of the back key event, or home key event, itself if it does not want to allow navigation away from an application.

7.3 Key Event Mapping

MirrorLink Key Events, injected into the MirrorLink Server Device are mapped in Table 1 to the following Platform Specific Key Events. The MirrorLink Server MUST re-map all the Key Events marked with YES and no other ones outside this set (none of the entries in the rows marked with NO may be mapped). The mappings are based on Android API level 14.

Some key events have no Android correspondent, but they are quite important for the MirrorLink Clients, so custom values have been used (i.e. KeyEvent.KEYCODE_BUTTON_*). For example the Knob diagonal shifts have been mapped to custom buttons. If future version of the Android OS will start to cover the key codes, this can be updated in future versions of the API, as both the applications and the server will know the API level implementations of each other. The only issue that can be caused by using custom buttons is if another subsystem uses the same custom buttons to mean something different and the application is using that subsystem as well as the MirrorLink Server. In that case it would be up to the application to ensure correct behavior.

Cate-gory	Mnemonic	MirrorLink Key Symbol Value	Platform Key Symbol Value	Map-ping Support	Adver-tise-ment
Knob Keys	Knob_2D_n_shift_right	0x3000 00n0	KeyEvent.KEYCODE_DPAD_RIGHT	YES	
	Knob_2D_n_shift_left	0x3000 00n1	KeyEvent.KEYCODE_DPAD_LEFT	YES	
	Knob_2D_n_shift_up	0x3000 00n2	KeyEvent.KEYCODE_DPAD_UP	YES	
	Knob_2D_n_shift_up_right	0x3000 00n3	KeyEvent.KEYCODE_BUTTON_1	YES	
	Knob_2D_n_shift_up_left	0x3000 00n4	KeyEvent.KEYCODE_BUTTON_2	YES	
	Knob_2D_n_shift_down	0x3000 00n5	KeyEvent.KEYCODE_DPAD_DOWN	YES	

Cate-gory	Mnemonic	MirrorLink Key Symbol Value	Platform Key Symbol Value	Map-ping Support	Adver-tise-ment
	Knob_2D_n_shift_down_right	0x3000 00n6	KeyEvent.KEYCODE_BUTTON_3	YES	
	Knob_2D_n_shift_down_left	0x3000 00n7	KeyEvent.KEYCODE_BUTTON_4	YES	
	Knob_2D_n_shift_push	0x3000 00n8	KeyEvent.KEYCODE_DPAD_CENTER	YES	
	Knob_2D_n_shift_pull	0x3000 00n9		NO	
	Knob_2D_n_rotate_x	0x3000 00nA	KeyEvent.KEYCODE_BUTTON_L1	YES	
	Knob_2D_n_rotate_X	0x3000 00nB	KeyEvent.KEYCODE_BUTTON_R1	YES	
	Knob_2D_n_rotate_y	0x3000 00nC	KeyEvent.KEYCODE_BUTTON_L2	YES	
	Knob_2D_n_rotate_Y	0x3000 00nD	KeyEvent.KEYCODE_BUTTON_R2	YES	
	Knob_2D_n_rotate_z	0x3000 00nE	KeyEvent.KEYCODE_TAB	YES	
	Knob_2D_n_rotate_Z	0x3000 00nF	KeyEvent.KEYCODE_TAB with KeyEvent.META_SHIFT_ON	YES	
ITU Keys	ITU_Key_0	0x3000 0100	KeyEvent.KEYCODE_0	YES	
	ITU_Key_1	0x3000 0101	KeyEvent.KEYCODE_1	YES	
	ITU_Key_2	0x3000 0102	KeyEvent.KEYCODE_2	YES	
	ITU_Key_3	0x3000 0103	KeyEvent.KEYCODE_3	YES	
	ITU_Key_4	0x3000 0104	KeyEvent.KEYCODE_4	YES	
	ITU_Key_5	0x3000 0105	KeyEvent.KEYCODE_5	YES	
	ITU_Key_6	0x3000 0106	KeyEvent.KEYCODE_6	YES	
	ITU_Key_7	0x3000 0107	KeyEvent.KEYCODE_7	YES	
	ITU_Key_8	0x3000 0108	KeyEvent.KEYCODE_8	YES	
	ITU_Key_9	0x3000 0109	KeyEvent.KEYCODE_9	YES	
	ITU_Key_Asterix	0x3000 010A	KeyEvent.KEYCODE_STAR	YES	

Cate-gory	Mnemonic	MirrorLink Key Symbol Value	Platform Key Symbol Value	Map-ping Support	Adver-tise-ment
	ITU_Key_Pound	0x3000 010B	KeyEvent.KEYCODE_POUND	YES	
Device Keys	Device_Phone_call	0x3000 0200	KeyEvent.KEYCODE_CALL	YES	
	Device_Phone_end	0x3000 0201	KeyEvent.KEYCODE_ENDCALL	YES	
	Device_Soft_left	0x3000 0202	KeyEvent.KEYCODE_SOFT_LEFT	YES	
	Device_Soft_middle	0x3000 0203		NO	
	Device_Soft_right	0x3000 0204	KeyEvent.KEYCODE_SOFT_RIGHT	YES	
	Device_Application	0x3000 0205	KeyEvent.APP_SWITCH	YES	
	Device_Ok	0x3000 0206	KeyEvent.KEYCODE_ENTER	YES	
	Device_Delete	0x3000 0207	KeyEvent.KEYCODE_DEL	YES	
	Device_Zoom_in	0x3000 0208	KeyEvent.ZOOM_IN	YES	
	Device_Zoom_out	0x3000 0209	KeyEvent.ZOOM_OUT	YES	
	Device_Clear	0x3000 020A	KeyEvent.CLEAR	YES	
	Device_Forward	0x3000 020B		NO	
	Device_Backward	0x3000 020C	KeyEvent.KEYCODE_BACK	YES	
	Device_Home	0x3000 020D	KeyEvent.KEYCODE_HOME	YES	
	Device_Search	0x3000 020E	KeyEvent.KEYCODE_SEARCH	YES	
	Device_Menu	0x3000 020F	KeyEvent.KEYCODE_MENU	YES	
Func-tion Keys	Function_Key_0	0x3000 0300	KeyEvent.KEYCODE_F1	YES	
	Function_Key_1	0x3000 0301	KeyEvent.KEYCODE_F2	YES	
	…	…		…	
	Function_Key_11	0x3000 0311	KeyEvent.KEYCODE_F12	YES	
	Function_Key_12	0x3000 0312		No	

Cate-gory	Mnemonic	MirrorLink Key Symbol Value	Platform Key Symbol Value	Map-ping Support	Adver-tise-ment
	…	…		…	
	Function_Key_255	0x3000 03FF		NO	
Multi-media Keys	Multimedia_Play	0x3000 0400	KeyEvent.KEYCODE_MEDIA_PLAY_PAUSE	YES*	
	Multimedia_Pause	0x3000 0401	KeyEvent.KEYCODE_MEDIA_PLAY_PAUSE	YES*	
	Multimedia_Stop	0x3000 0402	KeyEvent.KEYCODE_MEDIA_STOP	YES	
	Multimedia_Forward	0x3000 0403	KeyEvent.KEYCODE_MEDIA_FAST_FORWARD	YES	
	Multimedia_Rewind	0x3000 0404	KeyEvent.KEYCODE_MEDIA_REWIND	YES	
	Multimedia_Next	0x3000 0405	KeyEvent.KEYCODE_MEDIA_NEXT	YES	
	Multimedia_Previous	0x3000 0406	KeyEvent.KEYCODE_MEDIA_PREVIOUS	YES	
	Multimedia_Mute	0x3000 0407	KeyEvent.KEYCODE_VOLUME_MUTE	YES*	
	Multimedia_Unmute	0x3000 0408	KeyEvent.KEYCODE_VOLUME_MUTE	YES*	
	Multimedia_Photo	0x3000 0409	KeyEvent.KEYCODE_CAMERA	YES	

Table 1: Key Event Mapping

* - please note that on Android only a toggle key event is available, while on MirrorLink there are two separate key event (for example for play/pause or mute/unmute).

7.4 Virtual Keyboard

The virtual keyboard will be supported by the MirrorLink Server through the standard Android OS. Therefore the application will not need any specific means to retrieve information about the virtual keyboard support.

8 REFERENCES

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

[2] IETF, RFC 3281, "An Internet Attribute Certificate Profile for Authorization", April 2002, http://www.ietf.org/rfc/rfc3281.txt

[3] Car Connectivity Consortium, "MirrorLink - Application Server Service", Version 1.1; CCC-TS-024

[4] Car Connectivity Consortium, "MirrorLink – VNC based Display and Control", Version 1.1, CCC-TS-010

[5] Car Connectivity Consortium, "MirrorLink – Handling of Application Certificates", Version 1.1, CCC-TS-036

[6] Car Connectivity Consortium, "MirrorLink – Common API", Version 1.1, CCC-TS-038

[7] Car Connectivity Consortium, "MirrorLink – Handling of Application Developer Certificates", Version 1.1.0, CCC-TS-044

[8] CCC "MirrorLink Android Common API" javadoc

[9] IETF, RFC 4648, "The Base16, Base32, and Base64 Data Encodings", October 2006. http://www.ietf.org/rfc/rfc4648.txt

[10] Android <manifest> package documentation. http://developer.android.com/guide/topics/manifest/manifest-element.html#package

[11] Android <manifest> android:versionCode documentation http://developer.android.com/guide/topics/manifest/manifest-element.html#vcode

[12] JAR File Specification. http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html

[13] Signing Your Applications. http://developer.android.com/tools/publishing/app-signing.html

[14] Car Connectivity Consortium, Android Application ID Generator

APPENDIX A – HEADER FILES

For the aidl and java files please see the sample MirrorLink Android Common API code provided.

APPENDIX B – APP ID GENERATION CODE

The following source is code for generating Android App IDs, released under the Apache License, 2.0.

```
/*
 * AppIdGenerator.java - Class for generating application ID from an APK/jar.
 *
 * Copyright (C) 2013 RealVNC Ltd.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.mirrorlink.android;

import java.io.File;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.io.IOException;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Enumeration;
import java.util.Set;

import java.util.jar.Attributes;
import java.util.jar.JarFile;
import java.util.jar.JarEntry;

import java.security.MessageDigest;
import java.security.DigestInputStream;
import java.security.NoSuchAlgorithmException;

public class AppIdGenerator {

    /* Exception thrown if the APK doesn't appear to be signed, so is
     * lacking digest information. */
    static public class NotSignedException extends Exception {
        public NotSignedException(String path) {
            super("File has no signature: " + path);
        }
    };

    /* Lookup table for base64 encoding */
    static final private int[] BASE64_LOOKUP = new int[] {
        'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
        'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b',
        'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
        'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2', '3',
        '4', '5', '6', '7', '8', '9', '-', '_'
    };

    /* Hashing algorithm to use to generate the app ID */
    public static final String HASH_ALGORITHM = "SHA-256";
    /* Location of MirrorLink certificate file to ignore when generating app ID */
    public static final String ML_CERT_PATH = "assets/self-signed.ccc.crt";
    /* Start of META-INF path location */
    public static final String META_INF_PATH = "META-INF/";

    /* Marker for the start of the main section of the manifest */
```

```
    public static final String START_MANIFEST_MAIN = "startManifestMain";
    /* Marker for the end of the main section of the manifest */
    public static final String END_MANIFEST_MAIN = "endManifestMain";
    /* Marker for the start of a file */
    public static final String START_FILE = "startFile";
    /* Marker for the end of a file */
    public static final String END_FILE = "endFile";


    /* Base64 encodes a binary array. */
    static private String base64Encode(byte[] hashOutput) {
        /* Convert hash value to base64 string */
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < hashOutput.length; i += 3) {
            /* Each base64 character is 6 bits, so encode 3 bytes at a time to 4 characters
*/
            int value = (((int)hashOutput[i]) & 0xff) << 16;
            int outputChars = 2;
            if (i + 1 < hashOutput.length) {
                value |=  (((int)hashOutput[i + 1]) & 0xff) << 8;
                ++outputChars;
            }
            if (i + 2 < hashOutput.length) {
                value |=  (((int)hashOutput[i + 2]) & 0xff);
                ++outputChars;
            }
            for (int j = 0; j < 4; ++j) {
                if (outputChars > 0) {
                    /* Take each 6 bits, beginning with the most significant one (out of
the 3 bytes
                     * being encoded), and lookup the base64 code point */
                    int index = (value >> (6*(3-j))) & 0x3F;
                    sb.appendCodePoint(BASE64_LOOKUP[index]);
                } else {
                    /* omit the padding '=' signs */
                }
                --outputChars;
            }
        }
        return sb.toString();
    }

    /* Converts a long into a sequence of 8 big-endian bytes */
    static private byte[] longToBytes(long v) {
        byte[] ret = new byte[8];
        for (int i = 0; i < ret.length; ++i) {
            ret[i] = (byte)((v >> (8 * (7 - i))) & 0xff);
        }
        return ret;
    }

    /* Comparator for JarEntry for sorting by lexicographic ordering
     * of unicode code points for name */
    static private final Comparator<JarEntry> ENTRY_COMPARATOR = new Comparator<JarEntry>()
{
        public int compare(JarEntry lhs, JarEntry rhs) {
            return lhs.getName().compareTo(rhs.getName());
        }
    };

    /* Adds a string to the digest with a size header
     *
     * Prepend with length of values to prevent two different
     * attributes hashing to the same information. E.g.  Foo -> Bar
     * and FooB -> ar. */
    static private void addStringToDigest(String value, MessageDigest md)
        throws UnsupportedEncodingException {
        byte[] valueBytes = value.getBytes("UTF-8");
        md.update(longToBytes(valueBytes.length));
        md.update(valueBytes);
    }

    /* Adds the manifest meta-data from the provided entry to the digest.
```

```
     *
     * Returns true if at least one attribute ending in -Digest is present. */
    static private boolean addAttribsToDigest(Attributes attribs, MessageDigest md)
        throws UnsupportedEncodingException {
        boolean ret = false;
        /* Sort the attributes before adding them to ensure consistency */
        ArrayList<String> names = new ArrayList<String>();
        for (Object obj : attribs.keySet()) {
            names.add(obj.toString());
        }
        Collections.sort(names);
        for (String attrib : names) {
            String value = attribs.getValue(attrib);
            if (value == null) {
                /* Shouldn't occur as all attributes should be
                 * present, treat as empty */
                value = "";
            }
            if (attrib.endsWith("-Digest")) {
                ret = true;
            }
            addStringToDigest(attrib, md);
            addStringToDigest(value, md);
        }
        return ret;
    }

    /* Calculates the app ID fro the given package name, version code and file. */
    static public String calculateAppId(String pkgName, long versionCode, File file)
        throws UnsupportedEncodingException, NoSuchAlgorithmException,
               IOException, NotSignedException {
        MessageDigest md = MessageDigest.getInstance(HASH_ALGORITHM);
        /* Add the package name as UTF-8. */
        addStringToDigest(pkgName, md);
        /* Add the version code. */
        md.update(longToBytes(versionCode));
        /* Add the contents of the APK */
        JarFile jar = new JarFile(file);

        /* Add the main section in the manifest (with the marker strings) */
        addStringToDigest(START_MANIFEST_MAIN, md);
        addAttribsToDigest(jar.getManifest().getMainAttributes(), md);
        addStringToDigest(END_MANIFEST_MAIN, md);

        /* Add all the files inside the jar into the hash. However
         * skipping ML_CERT_PATH and ensuring consistency by
         * processing filenames in alphabetical order (defined by
         * unicode code points).
         *
         * The contents of the META-INF directory is skipped as it is
         * added to the digest later on.
         *
         * Each file is marked by the specified markse strings. */
        ArrayList<JarEntry> entries = new ArrayList<JarEntry>();
        for (Enumeration<JarEntry> e = jar.entries(); e.hasMoreElements(); ) {
            entries.add(e.nextElement());
        }
        Collections.sort(entries, ENTRY_COMPARATOR);
        for (JarEntry entry : entries) {
            if (entry.getName().equals(ML_CERT_PATH) ||
                    entry.getName().startsWith(META_INF_PATH)) {
                continue;
            }
            addStringToDigest(START_FILE, md);
            addStringToDigest(entry.getName(), md);
            if (!addAttribsToDigest(entry.getAttributes(), md)) {
                throw new NotSignedException(entry.getName());
            }
            addStringToDigest(END_FILE, md);
        }

        byte[] hashOutput = md.digest();
        return base64Encode(hashOutput);
```

```
    }

    public static void main(String[] args) {

        /* Simple command line validation */
        if (args.length != 3) {
            System.out.println("Usage: java com.mirrorlink.android.AppIdGenerator <Package
Name> <Version Code> <APK filename>");
            return;
        }
        String pkgName = args[0];
        /* This will fail for very large version codes as long is signed */
        long versionCode = Long.parseLong(args[1]);
        File file = new File(args[2]);
        try {
            String appID = calculateAppId(pkgName, versionCode, file);
            System.out.println("App ID for " + pkgName + " version " + versionCode + " is
" + appID);
        } catch (NotSignedException e) {
            System.out.println("APK does not appear to be signed");
            e.printStackTrace();
        } catch (Throwable e) {
            System.out.println("Failed to generate app ID");
            e.printStackTrace();
        }
    }
}
```