ETSI TS 103 544-22 V1.3.1 (2019-10)

Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 22: Android Specific Specifications enabling AIDL-based MirrorLink® Applications

Reference

RTS/ITS-98-22

Keywords

interface, ITS, PAS, smartphone

Contents

Intellectual Property Rights 5
Foreword 5
Modal verbs terminology 5
1 Scope 6
2 References 6
2.1 Normative references 6
2.2 Informative references 7
3 Definition of terms, symbols and abbreviations 7
3.1 Terms 7
3.2 Symbols 7
3.3 Abbreviations 7
4 Platform-Specific Specification Concept Overview 8
5 Application Identifier 8
5.1 General 8
5.2 Format 8
5.3 Calculation 9
5.3.1 General 9
5.3.2 Shared UIDs 10
5.3.3 ROM applications 11
5.4 APK Validation 11
6 Application Information 11
6.1 General 11
6.2 Self-Signed Application Certificates 11
6.2.1 General 11
6.2.2 Application Certificate Self-Signing Process 12
6.2.3 Self-Signed Application Certificate Installation 12
6.3 Particulars of Android Application Certificates 12
6.3.1 General 12
6.3.2 Platform version 12
6.3.3 Icon URLs 12
6.4 Localized strings for the entries in the application certificate 12
7 Development Certificates 13
7.1 General 13
7.2 Device ID 13
7.3 Application Development Certificates 13
7.4 Developer ID Entry 13
7.5 Manual Developer ID Certificate Revocation Checks 14
8 MirrorLink API Implementation 14
8.1 General 14
8.2 API Overview 14
8.2.1 General 14
8.2.2 Android MirrorLink Server Requirements 14
8.2.3 Application Requirements 16
8.3 MirrorLink API Library Definition 17
8.3.1 Data Type Definitions 17
8.3.2 Data Structure Definitions 17
8.3.3 MirrorLink API Elements accessible from the Service 18
8.3.4 MirrorLink API Elements accessible from the Content Provider 19
8.3.4.1 General 19
8.3.4.2 0xF0xx ' MirrorLink API Information mapping 19
8.3.4.3 0x02xx ' Certification Information mapping 19
8.3.4.4 0x03xx ' Connection Information mapping 20

8.3.4.5 0x0Cxx ' Actions mapping 20
8.3.5 Modules dependencies 20
8.3.5.1 General 20
8.3.5.2 General Android MirrorLink Service access 20
8.3.5.3 General Android Content Provider Service access 21
8.3.5.4 Actions Module write access 21
8.3.5.5 DataService Module Source Service Access 21
8.3.5.6 Launch of Actions 21
8.3.5.7 MirrorLink Actions 21
8.3.6 Context Information Lifetime 22
8.3.7 Return values outside of a MirrorLink session 22

9 MirrorLink Events 23
9.1 General 23
9.2 Touch Events Injections 23
9.3 Key Events Injections 23
9.4 Key Event Mapping 23
9.5 Virtual Keyboard 24

10 Platform Limitations 25
10.1 CCC-TS-010-VNC Based Display and Control (ETSI TS 103 544-2) 25
10.1.1 Server Event Configuration Message 25
10.1.2 Handling of Overlays 25
10.1.3 Handling of Applications Seeking Foreground Status 25
10.1.4 Removal of audio from non-certified applications 25

Annex A (informative): App ID Generation Code 26

Annex B (informative): Authors and Contributors 30

History 31

Intellectual Property Rights

Essential patents

Trademarks

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Intelligent Transport Systems (ITS).

The present document is part 22 of a multi-part deliverable. Full details of the entire series can be found in part 1 [i.1].

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

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

The present document provides the elements of the MirrorLink specification that apply only to Android MirrorLink Server devices.

The API javadoc files contained in the archive CCC-TS-065_Mirrorlink_API-Level2-AIDL-files__v138.zip, contained in ts_10354422v010301p0.zip, are an integral part of the present document.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI TS 103 544-14 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 14: Application Certificates".

[2] ETSI TS 103 544-15 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink® ; Part 15: Application Programming Interface (API) Level 1 & 2".

[3] ETSI TS 103 544-16 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 16: Application Developer Certificates".

[4] IETF RFC 4648: "The Base16, Base32, and Base64 Data Encodings", October 2006.

NOTE: Available at http://www.ietf.org/rfc/rfc4648.txt.

[5] ETSI TS 103 544-9 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 9: UPnP Application Server Service".

[6] ETSI TS 103 544-2 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 2: Virtual Network Computing (VNC) based Display and Control".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TS 103 544-1 (V1.3.1): "Publicly Available Specification (PAS); Intelligent Transport Systems (ITS); MirrorLink®; Part 1: Connectivity".

[i.2] Android <manifest> package documentation.

NOTE: Available at http://developer.android.com/guide/topics/manifest/manifest-element.html#package.

[i.3] Android <manifest> android:versionCode documentation.

NOTE: Available at http://developer.android.com/guide/topics/manifest/manifest-element.html#vcode.

[i.4] JAR File Specification.

NOTE: Available at http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html.

[i.5] Signing Your Applications.

NOTE: Available at http://developer.android.com/tools/publishing/app-signing.html.

[i.6] Car Connectivity Consortium: "Android Application ID Generator".

[i.7] Common Intents.

NOTE: Available at https://developer.android.com/guide/components/intents-common.html.

[i.8] Managing audio focus.

NOTE: Available at https://developer.android.com/guide/topics/media-apps/audio-focus.html.

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ACMS	Application Certification Management System
AIDL	Android Interface Definition Language
API	Application Programming Interface
APK	Android PacKage
CDB	Common Data Bus

CDMA	Code-Division Multiple Access
HTTP	HyperText Transfer Protocol
IMEI	International Mobile Equipment Identity
IPC	Inter-Process Communication
JAR	Java ARchive
ML	MirrorLink
OCSP	Online Certificate Status Protocol
OS	Operating System
ROM	Read-Only Memory
RSA	Rivest–Shamir–Adleman
SBP	Service Binary Protocol
SDK	Software Development Kit
UID	Unique IDentifier
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
VNC	Virtual Network Computing

4 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 MirrorLink 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.

The present document contains the requirements for MirrorLink Server devices that utilize the Android OS. MirrorLink Server devices that use the Android Operating System shall comply with the requirements listed in the present document.

5 Application Identifier

5.1 General

As described in the Application Certificate Handling specification [1], each application shall 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 Annex A.

5.2 Format

The application ID for an Android application shall be a URL-safe base-64 encoding [4] 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 shall 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).

5.3 Calculation

5.3.1 General

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

1) String encoding of Android package name provided in AndroidManifest.xml [i.2].
2) Big endian 8-byte integer representing the version code provided in AndroidManifest.xml [i.3].
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 shall be a big-endian 8-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 where the strings are encoded using the method described above:

```
"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"
```

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 [i.6], which contains a sample APK. The README file provided contains the expected application ID.

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 shall treat it as untrusted.

5.3.2 Shared UIDs

In Android APKs are allowed to share UIDs. Each APK shall 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 shall 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 shall provide the package name of the APK it lives in. The server then shall 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.

5.3.3 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 shall match the application identifier of the same, un-optimized application.

The application shall contain a self-signed certificate, following the requirements in clause 6.2. The MirrorLink Server shall 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 shall treat it as an updated version of the ROM application, shall cease making OCSP requests in relation to the ROM-based version, and shall not list the ROM-based version in the UPnP Application Listings. Updated versions of built-in applications installed as a standard APK shall be treated like normal MirrorLink applications.

5.4 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 [i.4].

An application ID shall 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 shall not be generated for an APK containing files without a '*-Digest' entry in the MANIFEST.MF.

An application ID shall 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.

6 Application Information

6.1 General

As described in clause 6.1 of the Handling of Application Certificates specification [1], 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 4.3 of [5], and to control interaction with the ACMS as described in [1]. This clause 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.

6.2 Self-Signed Application Certificates

6.2.1 General

The MirrorLink servers running on an Android platform shall allow for Self-Signed Application Certificates as described in clause 6.1 of [1].

6.2.2 Application Certificate Self-Signing Process

To create a MirrorLink aware application a Self-Signed X.509 Certificate shall be created. The certificate shall include the MirrorLink Extension Headers as described in clause 5.2.1 of [1]. The key used shall be a 2048-RSA key and the hashing shall be done using SHA-256 or SHA-512 signature algorithms.

The self-signed certificate shall be signed with a key pair generated by the developer. The Jarsigner utility included in the SDK, as described in [i.5], should be used to sign the Self-Signed Application Certificate.

6.2.3 Self-Signed Application Certificate Installation

The Self-Signed Certificate shall be placed in a file called "self-signed.ccc.crt". The certificate file shall 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 clause 6.1 of [1].

6.3 Particulars of Android Application Certificates

6.3.1 General

The following clauses 6.3.2 and 6.3.3 describe any items which are in the Application Certificates that need to be adapted to the Android OS.

6.3.2 Platform version

The platform version in the certificates shall 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 shall be decimal, signed and without any leading 0, or white spaces.

6.3.3 Icon URLs

The Icon URL field shall be left empty; therefore, the *iconList* element shall be omitted from the Application Certificate. The MirrorLink Server shall use the icon provided with the Activity which accepts the LAUNCH and TERMINATE MirrorLink-specific Intents (see clause 8.2.2) or the icon provided through the Actions module when applications are making use of variants.

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 platform specific. In [1], clause 7.3.2, *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.

6.4 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.

Table 1

Certificate element	Resource name
Name	"com.mirrorlink.app.certificate.name"
providerName	"com.mirrorlink.app.certificate.providerName"
providerURL	"com.mirrorlink.app.certificate.providerURL"

Certificate element	Resource name
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.

7 Development Certificates

7.1 General

Android MirrorLink Server devices will support the development of MirrorLink Applications. Clause 7 of the present document covers those aspects of the MirrorLink specification that address development of applications as described in the Handling of Application Development Certificates specification [3].

7.2 Device ID

Device IDs are the IMEI/IMEISV number or version 5 UUID derived from an equivalent unique identifier of the MirrorLink Server devices on which development applications can be used.

All Android MirrorLink Servers should use the IMEI (or MEID/ESB for CDMA devices) as provided by the TelephonyManager's getDeviceId method or the ANDROID_ID as provided by the Settings.Secure.getString(<content resolver>, Settings.Secure.ANDROID_ID) method as seed of the Device ID. If an IMEI (or MEID/ESB) or ANDROID_ID is not available, the Android MirrorLink Server shall use the serial number as reported by the android.os.SystemProperties ro.serialno property or any other read-only secure property uniquely identifying a device.

See clause 5.2.3.2 of the Handling of Application Developer Certificates Document [3] for more information on the Device ID.

7.3 Application Development Certificates

The Android platform uses Application Development Certificates as described in clauses 5.1 and 7 of the Handling of Application Developer Certificates Document [3]. 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 Development Certificate.

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

7.4 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 shall 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 Developer IDs, the Android MirrorLink server shall attempt to retrieve the Developer ID certificate for all Developer IDs reported in all Application Development Certificates available on the device.

A MirrorLink server only implementing API Level 1, shall allow developers having entered their Developer ID in the device to consult its DEVICE ID for inclusion in their Application Developer Certificates.

See clause 6 of the Handling of Application Developer Certificates Document [3] for more information on requesting Developer ID certificates.

7.5 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.

8 MirrorLink API Implementation

8.1 General

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

8.2 API Overview

8.2.1 General

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) and on the concept of content provider Contract classes. 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 and an Android Content Provider. The package name for the service and content provider should be unique across all Android MirrorLink servers. The recommended name used for the packages are: "com.mirrorlink.android.service" and "com.mirrorlink.android.provider".
- Constants used when interacting either with the MirrorLink Service, ContentProvider, Intents or specific Bundle will be regrouped in a single java source file with one distinct class per module, or structure, being supported.
- Whenever possible, android.os.Bundle objects will be used to exchange complex data structures between the MirrorLink Service and the Application.

8.2.2 Android MirrorLink Server Requirements

- MirrorLink Servers shall allow access to the MirrorLink Service for all applications that bind using the "com.mirrorlink.android.service.BIND" family of Intents, 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 bind request, 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.
- MirrorLink servers shall allow access to the MirrorLink ContentProvider for all applications listing the "com.mirrorlink.android.provider" in their certificate xml <serviceList/>, using the "com.mirrorlink.android.provider" authority and using the "com.mirrorlink.android.provider. MIRRORLINK_PROVIDER_ACCESS_PERMISSION" permission. This permission should be user-grantable on the MirrorLink Server.
- MirrorLink servers shall support the use of android.database.ContentObserver by MirrorLink aware applications to monitor underlying changes made to the MirrorLink ContentProvider datasets.
- Application can use the appropriate "com.mirrorlink.android.service.BIND" action to specify the MirrorLink API level they wish to bind to. If a MirrorLink Server does not support a particular API level, it shall fail the bind operation.

- In response to the default "com.mirrorlink.android.service.BIND" Intent, MirrorLink servers shall return the Android MirrorLink API Service Level 1 binder from the bind request.
- In response to a "com.mirrorlink.android.service.BIND.<api_level>" Intent, MirrorLink Servers shall return the corresponding MirrorLink API Service Level binder from the bind request or null if the requested API Level is not supported.
- MirrorLink servers shall allow access to the Android MirrorLink Service and Android MirrorLink ContentProvider at all times, inside and outside of a MirrorLink Connection.
- MirrorLink servers shall keep the Android MirrorLink Service alive as long as there is at least one MirrorLink Aware application bound to it.
- The MirrorLink server shall 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 MirrorLink Service API, then the server shall send an Application Status Update notifying the client that the application is now in Foreground.
- The MirrorLink server shall 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 service API and in Intent Extras 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 shall use these structures without modification.
- The structures used in the content provider are defined using a distinct Contract Class for each module supported by the content provider. This allows adding new modules or fields to structures without affecting compatibility. Each field will be referred to by name string. All Android MirrorLink Servers shall use these Contract Classes without modifications.
- 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 shall use the "com.mirrorlink.android.app.LAUNCH" and "com.mirrorlink.android.app.TERMINATE" Intents to Launch, or Terminate a MirrorLink aware application.
- The MirrorLink Server shall use the "com.mirrorlink.android.app.variant.LAUNCH" and "com.mirrorlink.android.app.variant.TERMINATE" Intents to Launch or Terminate a variant exposed by a MirrorLink aware application using the Action Manager Module. The MirrorLink Server shall provide the action detailed information for the variant being launched using a "com.mirrorlink.android.app.variant.EXTRA_ACTION" Bundle extra.
- 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 shall catch that exception to detect the crash.
- The MirrorLink server shall listen to the binderDied() callback to detect if an application has crashed outside of a callback. All the callbacks to the applications shall 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 shall wait at allow at least 3 seconds for the application to make the response.
- The MirrorLink server shall be implemented for at least Android API level 14 (Android 4.0).
- The MirrorLink server shall not allow an application to bind to a service concurrently more than once. If a second bind happens, the Server shall throw java.lang.IllegalStateException.

- MirrorLink servers supporting API Level 2 shall broadcast the "com.mirrorlink.android.CONNECTION_STATE" Intent reflecting changes in the state of the MirrorLink session. The status of the MirrorLink session will reflect the value of the MirrorLink Session Established variable (Function reference 0x0301) defined by the MirrorLink API and will be accessible using the "com.mirrorlink.android.EXTRA_SESSION_ESTABLISHED" Boolean extra. MirrorLink servers shall protect the broadcasted intent using the "com.mirrorlink.android.service.ACCESS_PERMISSION" permission.

- The MirrorLink server should update the state the MirrorLink state notification no later than a second after the state of a MirrorLink Session has changed.

8.2.3 Application Requirements

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

- The application shall listen to the "com.mirrorlink.android.app.LAUNCH" and "com.mirrorlink.android.app.TERMINATE" Intents. It shall use the "android.intent.category.DEFAULT" category to filter on these intents. There shall 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.

- Upon receiving the TERMINATE intent, the application shall go or stay in the background.

- 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 shall have the com.mirrorlink.android.service.ACCESS_PERMISSION permission.

- Applications that publish actions shall include "com.mirrorlink.api.actions" in the <serviceList/> element of their certificate xml extension.

- Applications that act as a source of data for DataServices shall include "{service-name-as-specified}.source" in the <serviceList/> element of their certificate xml extension.

- Applications that act as a data sink for a service should include "{service-name-as-specified}" in the <serviceList/> element of their certificate xml extension.

- When advertising variants, the application shall listen to the "com.mirrorlink.android.app.variant.LAUNCH" and "com.mirrorlink.android.app.TERMINATE" Intents. It shall use the "android.intent.category.MIRRORLINK" category to filter these intents. There shall be only one Activity that receives each of these intents and it is the responsibility of this Activity to execute the action indicated within the "com.mirrorlink.android.app.variant.EXTRA_ACTION" Bundle extra.
  The application shall make use of the Activity.onNewIntent() override in order to receive updated Intents from multiple variant launch.

- The application should not bind to the Android MirrorLink Service outside of an established MirrorLink connection. The application shall use a BroadcastReceiver or a ContentObserver to monitor the status of the MirrorLink connection.

- The application, when started using the "com.mirrorlink.android.app.LAUNCH" or "com.mirrorlink.android.app.variant.LAUNCH" intent, either by the MirrorLink server, or by the user from a Launcher on the device, shall bind within 3 seconds to the Android MirrorLink Service by sending the appropriate bind Intent, corresponding to the Api Level they which to make use of.

- The application shall use the Connection Manager Module from the MirrorLink API and shall register a listener for its callbacks. It shall stay registered for the entire duration while it is running. If the application unregisters from the Connection Manager Module, 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 Module, even though that Service might not be the Connection Manager Module.
- The application shall register a listener for every module it intends to access through the Android MirrorLink Service.
- The application shall 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 shall not access more than once concurrently the same MirrorLink Service. There shall be only one connection between an application and a Service.
- The application shall not unbind from the MirrorLink server for the entire duration of it running during a MirrorLink session. The unbinding shall be made only when the application is terminated.
- The application shall 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.
- The Application should react and respond to callbacks as soon as it is reasonably possible.
- The application shall not request platform permissions from the end user when in a MirrorLink session. The application shall try to get all necessary permissions when the application is launched out of MirrorLink. The application shall otherwise remain fully functional to the exclusion of features requiring unacquired permissions.
- If an application makes use of audio playback or audio announcements the following applies:
  - Multimedia applications should use exclusive focus during a MirrorLink session and stop playing or duck when losing the audio focus.
  - The application shall keep the MirrorLink server informed updated with its audio output status using the 0x0803 ' Audio Context Information/IContextManager.SetAudioContext() api.
  - Multimedia applications shall pause playback when blocked and should resume playback when unblocked.
  - Applications generating announcement shall dismiss the current notification when blocked but should not wait to be un-blocked before generating the next announcement.

NOTE: More information on how to manage the audio focus can be found in [i.8].

8.3 MirrorLink API Library Definition

8.3.1 Data Type Definitions

For the data types defined please refer to the API javadoc files [i.4]. The data types refer back to the relevant section in the MirrorLink API Template [2].

The MirrorLink 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.

8.3.2 Data Structure Definitions

For the data structures defined please refer to the API javadoc files [i.4]. The data types refer back to the relevant section in the MirrorLink API Template [2].

8.3.3 MirrorLink API Elements accessible from the Service

To access the MirrorLink API Android MirrorLink Service an application shall use the "com.mirrorlink.android.service.BIND" family of Intents. Once the "service bind" callback is received, the application will get a reference to an ICommonAPIService (Api Level 1) or IMirrorLinkService_2 (Api Level 2) object. From this the application can get the API level of the service, or retrieve the objects representing the different modules (managers) provided.

To access the MirrorLink API Android MirrorLink ContentProvider, an application shall use a ContentResolver using the "com.mirrorlink.android.provider" authority. From this, the application can get access to the API level of the content provider, the state of the MirrorLink session and the datasets representing the different modules (content) provided.

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

The modules available through the Android MirrorLink Service are (together with the requirements for the Server and Application supporting them).

Table 2

Module	Interface name	Retrieved via	Obligation
MirrorLink Device Info	IDeviceInfoManager	getDeviceInfoManager()	Shall
Certification Information	ICertificationManager	getCertificationManager()	Shall
Certificate Information (Level 2)	ICertificationManager_2	getCerticationManager()	Shall (Level 2 onwards)
Connection Information	IConnectionManager	getConnectionManager()	Shall
Display Related Features	IDisplayManager	getDisplayManager()	Shall
Event Related Features	IEventMappingManager	getEventMappingManager()	Shall
Context Information Related Features	IContextManager	getContextManager()	Shall
Device Status Features	IDeviceStatusManager	getDeviceStatusManager()	Shall
(CDB) Data Services	IDataServicesManager	getDataServicesManager()	Conditional (Level 1) Shall (Level 2)
Notifications	INotificationManager	getNotificationManager()	Conditional (Level 1) Shall (Level 2)

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.

MirrorLink Server based on Android shall implement the MirrorLink API in compliance to the API javadoc files contained in the archive CCC-TS-065_Mirrorlink_API-Level2-AIDL-files__v138.zip, contained in ts_10354422v010301p0.zip, which accompanies the present document. The data types refer back to the relevant section in the MirrorLink API Template [2]. One difference from the MirrorLink 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.

The service interfaces are called "Managers" and callback interfaces "Listeners". This is done to follow the regular Android practice.

8.3.4 MirrorLink API Elements accessible from the Content Provider

8.3.4.1 General

The modules available through the Android MirrorLink Content Provider (together with the requirements for the Server and Application supporting them) are:

Table 3

Module	Content Path	Mime Type	Obligation
CommonAPI	/apilevel/	vnd.com.mirrorlink.android.provider.apilevel	Shall
Connection Information	/connection/	vnd.com.mirrorlink.android.provider.connection	Shall
Certification Information	/certificates/	vnd.com.mirrorlink.android.provider.certificate	Shall
Actions	/actions/	vnd.com.mirrorlink.android.provider.action	Shall

Clauses 8.3.4.2 to 8.3.4.5 document the mapping between specific MirrorLink API methods and the corresponding ContentResolver query, insert, update, delete, notifyChange, getType main methods.

8.3.4.2 0xF0xx ' MirrorLink API Information mapping

Table 4

Resolver	MirrorLink API	Selection
Query	0xF001 ' MirrorLink API Version	None
Query	0xF003 ' Server Device Identifier	None
Insert	Not supported	n/a
Update	Not supported	n/a
Delete	Not supported	n/a
Notify	Not supported	n/a
getType	0xF002 ' MirrorLink API Module Available	n/a

8.3.4.3 0x02xx ' Certification Information mapping

Table 5

Resolver	MirrorLink API	Selection
Query	0x0206 ' Get Any Application Certification Status	appId
Query	0x0207 ' Get Any Application Certifying Entities	appId
query	0x0208 ' Get Any Application Certification Information	appId, entity
Insert	Not supported	n/a

Resolver	MirrorLink API	Selection
Update	Not supported	n/a
Delete	Not supported	n/a
Notify	00x0209 ' Get Certified Applications List Changed Callback	

8.3.4.4 0x03xx ' Connection Information mapping

Table 6

Resolver	MirrorLink API	Cursor
Query	0x0301 ' Established MirrorLink Connection	None
Insert	Not supported	n/a
Update	Not supported	n/a
Delete	Not supported	n/a
Notify	0x0302 ' Established MirrorLink Connection Callback	n/a

8.3.4.5 0x0Cxx ' Actions mapping

Table 7

Resolver	Common API	Selection
Insert	0x0C01 ' Create Application Actions	None
Update	0x0C02 ' Update Application Actions	actionId
Query	0x0C03 ' Retrieve Application Actions	appId actionType
Delete	0x0C04 ' Delete Application Action	actionId
Notify	0x0C07 ' Application Actions Changed Callback	None

8.3.5 Modules dependencies

8.3.5.1 General

This clause describes the dependencies that exist between each module and other part of the MirrorLink protocol, like the application certificate for example.

Whenever possible, the documentation pertaining to each specific module will be provided in javadoc format together with the Android MirrorLink Service AIDL and Android MirrorLink Content Provider Contract classes.

8.3.5.2 General Android MirrorLink Service access

The Android MirrorLink Service is guarded by the "com.mirrorlink.android.service.ACCESS_PERMISSION". Additionally, a server may restrict access to the Android MirrorLink Service to applications sporting a valid certificate.

8.3.5.3 General Android Content Provider Service access

The Android MirrorLink Content Provider is guarded by the "com.mirrorlink.android.provider. MIRRORLINK_PROVIDER_ACCESS_PERMISSION". Additionally, a server may restrict access to the Android MirrorLink Service to applications sporting a valid certificate.

8.3.5.4 Actions Module write access

Only applications having the "com.mirrorlink.android.provider" pseudo service identifier listed in their certificate <serviceList> entry are allowed to publish actions to other MirrorLink aware applications.

Only applications having the "com.mirrorlink.android.app.variant" pseudo service identifier listed in their certificate <serviceList> entry are allowed to publish variants to the application listing.

8.3.5.5 DataService Module Source Service Access

Only applications having the "{service-name}.source" pseudo service identifier listed in their certificate <serviceList> entry are allowed to contribute data to MirrorLink Data Services implemented as source by the MirrorLink server and consumed through a Sink by the MirrorLink client, for those services that it also lists in their certificate <serviceList> entry.

A MirrorLink Server shall only advertise a MirrorLink SBP Source to a MirrorLink client if there is at least one application installed on the server claiming support for the corresponding SBP source in its certificate.

8.3.5.6 Launch of Actions

On Android, and with the exception of variants, Actions shall be mapped to an Intent where the Intent action is the action type, the Intent package is the publishing application's package name and the Intent category is set to "com.mirrorlink.category.MIRRORLINK". Any Action Parameters shall be passed using the Intent Extra property and related methods.

Any Application publishing an action shall add the corresponding intent filter to the Activity implementing the action. A MirrorLink Server shall deny the addition of any action which cannot be resolved to an activity belonging to the publishing application.

Any Application launching an action shall use the startActivity mechanism using a fully qualified Intent, as detailed above.

8.3.5.7 MirrorLink Actions

Whenever possible MirrorLink actions defined in the MirrorLink API specification shall be matched to their natural Android counterpart. As Intents shall be used by MirrorLink Servers and Applications to invoke a MirrorLink Action and query the result of an Action invocation each MirrorLink Action Type shall be matched to its standard Android counterpart if existing, as provided in following mapping table and specified in the API javadoc files contained in the archive CCC-TS-065_Mirrorlink_API-Level2-AIDL-files__v138.zip, contained in ts_10354422v010301p0.zip, which accompanies the present document.

Whenever possible, the Android Developer Documentation section on Common Intents [i.7] applies when using these intents.

Table 8

MirrorLink Action Type	Android Symbol	Android Constant Value
ACTION_SET_ALARM	android.provider.AlarmClock. ACTION_SET_ALARM	android.intent.action.SET_ALARM
ACTION_SET_TIMER	android.provider.AlarmClock.ACTION_SET_TIME	android.intent.action.SET_TIMER
ACTION_DISMISS_ALARM	android.provider.AlarmClock. ACTION_DISMISS_ALARM	android.intent.action.DISMISS_ALARM
ACTION_SNOOZE_ALARM	android.provider.AlarmClock. ACTION_SNOOZE_ALARM	android.intent.action.SNOOZE_ALARM

MirrorLink Action Type	Android Symbol	Android Constant Value
ACTION_DIAL	android.content.Intent.ACTION_DIAL	android.intent.action.DIAL
ACTION_CALL	android.content.Intent.ACTION_CALL	android.intent.action.CALL
ACTION_TEXT	android.content.Intent.ACTION_SENDTO	android.intent.action.SENDTO
ACTION_PLAY_MEDIA	android.provider.MediaStore.INTENT_ ACTION_MEDIA_PLAY_FROM_SEARCH	android.media.action. MEDIA_PLAY_FROM_SEARCH
ACTION_PLAY	android.media.intent.action.PLAY	android.media.intent.action.PLAY
ACTION_PAUSE	android.media.intent.action.PAUSE	android.media.intent.action.PAUSE
ACTION_NEXT	Not applicable	com.mirrorlink.android.media.NEXT
ACTION_PREVIOUS	Not applicable	com.mirrorlink.android.media.action.PREVIOUS
ACTION_SHUFFLE	Not applicable	com.mirrorlink.android.media.action.SHUFFLE
ACTION_REPEAT	Not applicable	com.mirrorlink.android.media.action.REPEAT
ACTION_MUTE	Not applicable	com.mirrorlink.android.media.action.MUTE
ACTION_RESPONSE_ RESUME	android.media.intent.action.RESUME	android.media.intent.action.RESUME
ACTION_STOP	android.media.intent.action.STOP	android.media.intent.action.STOP
ACTION_OPEN_ APPLICATION	android.content.Intent.ACTION_MAIN	android.intent.action.MAIN
ACTION_RESPONSE_YES	Not applicable	com.mirrorlink.android.action.response.yes
ACTION_RESPONSE_NO	Not applicable	com.mirrorlink.android.action.response.no
ACTION_RESERVE_TAXI	Not applicable	com.mirrorlink.android.action. RESERVE_TAXI_RESERVATION
ACTION_RESPONSE_ FREE_FROM_SPEECH	Not applicable	com.mirrorlink.android.action.response.speech
ACTION_RESPONSE_ FREE_FROM_AUDIO	Not applicable	com.mirrorlink.android.action.response.audio
ACTION_RESPONSE_ RETRY	Not applicable	com.mirrorlink.android.action.response.retry
ACTION_CREATE_NOTE	Not applicable	com.mirrorlink.android.app.CREATE_NOTE
ACTION_NONE	Not applicable	com.mirrorlink.android.action.NONE
ACTION_VARIANT	android.content.Intent. ACTION_CREATE_SHORTCUT	android.intent.action.CREATE_SHORTCUT
ACTION_MIRRORLINK_ HOME_SCREEN	Not applicable	com.mirrorlink.android.homecreen.HOME
ACTION_MIRRORLINK_ APP_LIST	Not applicable	com.mirrorlink.android.homescreen.APPLIST
ACTION_MIRRORLINK_ MUSIC	Not applicable	com.mirrorlink.android.homescreen.MUSIC

8.3.6 Context Information Lifetime

The context information provided by a MirrorLink aware application shall be persisted and used by the MirrorLink server for the lifetime of the application's connection to the Android MirrorLink Service, independent of the underlying status of the MirrorLink connection to a MirrorLink client.

8.3.7 Return values outside of a MirrorLink session

A MirrorLink Server may return undefined implementation specific values outside of a MirrorLink session.

9 MirrorLink Events

9.1 General

As the MirrorLink Server shall use the Android OS standard mechanisms for key events and virtual keyboards, these modules are not exposed using the MirrorLink API Service or Content Provider. Instead, standard Android Event injections mechanism are used to translate between MirrorLink events and native Android events.

9.2 Touch Events Injections

The touch events shall 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.

9.3 Key Events Injections

The key events shall 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.

9.4 Key Event Mapping

MirrorLink Key Events, injected into the MirrorLink Server Device are mapped in Table 9 to the following Android Specific Key Events. The MirrorLink Server shall re-map all the Key Events marked with YES to standard Android Key Events 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 behaviour. It is worth noting that only the first knob (Knob_2D_0) events are mapped, all others are left unmapped or reserved for future use.

Table 9: Key Event Mapping

Set	MirrorLink Key Symbol	Symbol Value	Platform Key Symbol Value (KeyEvent.*)	Map
Knob Keys	Knob_2D_0_shift_right	0x3000 0000	KEYCODE_DPAD_RIGHT	YES
	Knob_2D_0_shift_left	0x3000 0001	KEYCODE_DPAD_LEFT	YES
	Knob_2D_0_shift_up	0x3000 0002	KEYCODE_DPAD_UP	YES
	Knob_2D_0_shift_up_right	0x3000 0003	KEYCODE_BUTTON_1	YES
	Knob_2D_0_shift_up_left	0x3000 0004	KEYCODE_BUTTON_2	YES
	Knob_2D_0_shift_down	0x3000 0005	KEYCODE_DPAD_DOWN	YES
	Knob_2D_0_shift_down_right	0x3000 0006	KEYCODE_BUTTON_3	YES
	Knob_2D_0_shift_down_left	0x3000 0007	KEYCODE_BUTTON_4	YES
	Knob_2D_0_shift_push	0x3000 0008	KEYCODE_DPAD_CENTER	YES
	Knob_2D_0_shift_pull	0x3000 0009		NO
	Knob_2D_0_rotate_x	0x3000 000A	KEYCODE_BUTTON_L1	YES
	Knob_2D_0_rotate_X	0x3000 000B	KEYCODE_BUTTON_R1	YES
	Knob_2D_0_rotate_y	0x3000 000C	KEYCODE_BUTTON_L2	YES
	Knob_2D_0_rotate_Y	0x3000 000D	KEYCODE_BUTTON_R2	YES
	Knob_2D_0_rotate_z	0x3000 000E	KEYCODE_TAB	YES
	Knob_2D_0_rotate_Z	0x3000 000F	KEYCODE_TAB with KeyEvent.META_SHIFT_ON	YES

Set	MirrorLink Key Symbol	Symbol Value	Platform Key Symbol Value (KeyEvent.*)	Map
ITU Keys	ITU_Key_0	0x3000 0100		NO
	ITU_Key_1	0x3000 0101		NO
	ITU_Key_2	0x3000 0102		NO
	ITU_Key_3	0x3000 0103		NO
	ITU_Key_4	0x3000 0104		NO
	ITU_Key_5	0x3000 0105		NO
	ITU_Key_6	0x3000 0106		NO
	ITU_Key_7	0x3000 0107		NO
	ITU_Key_8	0x3000 0108		NO
	ITU_Key_9	0x3000 0109		NO
	ITU_Key_Asterix	0x3000 010A		NO
	ITU_Key_Pound	0x3000 010B		NO
Device Keys	Device_Phone_call	0x3000 0200	KEYCODE_CALL	YES
	Device_Phone_end	0x3000 0201	KEYCODE_ENDCALL	YES
	Device_Soft_left	0x3000 0202	KEYCODE_SOFT_LEFT	YES
	Device_Soft_middle	0x3000 0203		NO
	Device_Soft_right	0x3000 0204	KEYCODE_SOFT_RIGHT	YES
	Device_Application	0x3000 0205	APP_SWITCH	YES
	Device_Ok	0x3000 0206	KEYCODE_ENTER	YES
	Device_Delete	0x3000 0207	KEYCODE_DEL	YES
	Device_Zoom_in	0x3000 0208	ZOOM_IN	YES
	Device_Zoom_out	0x3000 0209	ZOOM_OUT	YES
	Device_Clear	0x3000 020A	CLEAR	YES
	Device_Forward	0x3000 020B		NO
	Device_Backward	0x3000 020C	KEYCODE_BACK	YES
	Device_Home	0x3000 020D	KEYCODE_HOME	YES
	Device_Search	0x3000 020E	KEYCODE_SEARCH	YES
	Device_Menu	0x3000 020F	KEYCODE_MENU	YES
Function Keys	Function_Key_0	0x3000 0300	KEYCODE_F1	YES
	Function_Key_1	0x3000 0301	KEYCODE_F2	YES
	…	…		…
	Function_Key_11	0x3000 0311	KEYCODE_F12	YES
	Function_Key_12	0x3000 0312		NO
	…	…		…
	Function_Key_255	0x3000 03FF		NO
Multimedia Keys	Multimedia_Play	0x3000 0400	KEYCODE_MEDIA_PLAY_PAUSE	YES*
	Multimedia_Pause	0x3000 0401	KEYCODE_MEDIA_PLAY_PAUSE	YES*
	Multimedia_Stop	0x3000 0402	KEYCODE_MEDIA_STOP	YES
	Multimedia_Forward	0x3000 0403	KEYCODE_MEDIA_FAST_FORWARD	YES
	Multimedia_Rewind	0x3000 0404	KEYCODE_MEDIA_REWIND	YES
	Multimedia_Next	0x3000 0405	KEYCODE_MEDIA_NEXT	YES
	Multimedia_Previous	0x3000 0406	KEYCODE_MEDIA_PREVIOUS	YES
	Multimedia_Mute	0x3000 0407	KEYCODE_VOLUME_MUTE	YES*
	Multimedia_Unmute	0x3000 0408	KEYCODE_VOLUME_MUTE	YES*
	Multimedia_Photo	0x3000 0409	KEYCODE_CAMERA	YES

NOTE: On Android only a toggle key event is available, while on MirrorLink there are two separate key event (for example as in play/pause or mute/unmute).

Other MirrorLink Key Events, that are not injected in the MirrorLink Server Device using standard Android key Events can be mapped to actions published by applications using the Action Manager module.

9.5 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.

10 Platform Limitations

10.1 CCC-TS-010-VNC Based Display and Control (ETSI TS 103 544-2)

10.1.1 Server Event Configuration Message

As the Android Operation System does not support ITU key bindings, MirrorLink Servers on Android shall not advertise ITU key support in their Server Event Configuration message, in accordance to the mapping table defined in clause 7.

10.1.2 Handling of Overlays

Overlays on Android can be classified in 4 broad classes:

1) Toast messages.

2) Overlay views.

3) System dialogs.

4) Application dialogs.

Android does not support native detection and dismissal of class 1 or 2 overlays, and offer limited support to an application (i.e. not to a system services) to detect class 3 or 4 overlays by way of detecting a loss of focus.

As such, MirrorLink Servers on Android are not required to perform overlay detection and dismissal.
This limitation can be mitigated in part by the fact MirrorLink aware application on android should be able to detect and dismiss some system overlays, but not all.

10.1.3 Handling of Applications Seeking Foreground Status

Android does not support actively preventing an application to be brought in the foreground so MirrorLink Servers on Android cannot be expected to prevent any application to be brought in the foreground by the operation system. MirrorLink Servers however, shall detect application coming into the foreground and alter the VNC context information accordingly.

10.1.4 Removal of audio from non-certified applications

Besides Audio Focus Management, the Android operating system does not offer any APIs allowing to either detect applications actively contributing to the audio stream or prevent applications from contributing to the audio stream.

Thus, MirrorLink Servers on Android cannot be expected to be able to remove effectively audio contributed by non-certified applications to the audio stream.

Annex A (informative): 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 does not 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 from 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();
        }
    }
}
```

Annex B (informative): Authors and Contributors

The following people have contributed to the present document:

Rapporteur:	Dr. Jörg Brakensiek, E-Qualus (for Car Connectivity Consortium LLC)
	Laurent Cremmer, RealVNC Ltd.
Other contributors:	Ed Pichon, E-Qualus (for Car Connectivity Consortium LLC)
	Sorin Basca, RealVNC Ltd

History

Document history		
V1.3.0	October 2017	Publication
V1.3.1	October 2019	Publication