GlobalPlatform Technology

Broker Interface

Card Specification v2.3 – Amendment J

Version 0.0.0.14

Public Review

November 2019

Document Reference: GPC_SPE_157

Contents

1 Introduction 6
1.1 Audience 6
1.2 IPR Disclaimer 6
1.3 References 6
1.4 Terminology and Definitions 6
1.5 Abbreviations and Notations 7
1.6 Revision History 7

2 Overview 8

3 Specification Amendments 9
3.1 Broker Interface API 9
3.2 Broker Application Requirements 10
3.3 Application Requirements 11

4 Broker Interface Usage for CDCVM 12
4.1 Overview 12
4.2 Parameter Values for the *CDCVMBrokerListener* Interface 14
  4.2.1 CDCVM Event 14
  4.2.2 CDCVM Information 16
  4.2.3 CDCVM Environment 16
  4.2.4 CDCVM Device Identifier 17
  4.2.5 CDCVM Security Features 18
  4.2.6 CDCVM Security Assurance 18
  4.2.7 CDCVM Discretionary Data 19
4.3 CDCVM Verified on the Secure Element 19
  4.3.1 CDCVM Request 19
4.4 Parameter Values for the *CDCVMBrokerCallbackRequest* Interface 20

Tables

Table 1-1: Normative References.......6
Table 1-2: Abbreviations.......7
Table 1-3: Revision History.......7
Table 3-1: DGI Used to Personalize Broker Application Whitelist.......11
Table 4-1: List of Values for the CDCVM Event Parameter.......15
Table 4-2: List of Values for the CDCVM Info Parameter.......16
Table 4-3: List of Values for the CDCVM Environment Parameter (Byte 1).......16
Table 4-4: List of Values for the CDCVM Environment Parameter (Byte 2).......17
Table 4-5: List of Values for the CDCVM Device Parameter.......17
Table 4-6: List of Values for the CDCVM Security Parameter.......18
Table 4-7: List of Values for the CDCVM Security Assurance Parameter.......18
Table 4-8: List of Values for the CDCVM Discretionary Parameter.......19
Table 4-9: List of Values for the CDCVM Request Parameter.......19

Figures

Figure 2-1: Interactions Enabled by the Broker Interface API 8
Figure 4-1: Example of Process Flow for Push and Pull Mode 13

1 Introduction

In the mobile payment arena, some of the functions and verifications that used to be controlled by Financial Institutions, Payment Network Operators, Identity and Access Control Authorities have been shifted to new entities, e.g. Mobile Network Operators, Original Equipment Manufacturers, and third-party Wallet Providers. These new entities typically have their own applet in the Secure Element to process such functions, e.g. user verification or proxy communication with remote servers.

This document assists Service Providers with the development of products supporting Applications that interact with third-party Broker Application(s) through a Shareable Interface Object as defined in Java Card ([JCCE]).

1.1 Audience

This amendment is intended primarily for payment software developers, Secure Element providers, MNOs, OEMs, and third-party wallet providers for the development of Broker Applications and Applications that use Broker Application services.

It is assumed that the reader is familiar with smart card technology, smart card production, and in particular familiar with the GlobalPlatform Card Specification [GPCS].

1.2 IPR Disclaimer

Attention is drawn to the possibility that some of the elements of this GlobalPlatform specification or other work product may be the subject of intellectual property rights (IPR) held by GlobalPlatform members or others. For additional information regarding any such IPR that have been brought to the attention of GlobalPlatform, please visit https://globalplatform.org/specifications/ip-disclaimers/. GlobalPlatform shall not be held responsible for identifying any or all such IPR, and takes no position concerning the possible existence or the evidence, validity, or scope of any such IPR.

1.3 References

Table 1-1: Normative References

Standard / Specification	Description	Ref
GlobalPlatform Card Specification	GlobalPlatform Technology Card Specification v2.3	[GPCS]
GPCS Amendment C	GlobalPlatform Technology Contactless Services, Card Specification v2.3 – Amendment C, v1.2	[Amd C]
Java Card Platform, Classic Edition 3.0.5	Java Card Platform, Classic Edition 3.0.5	[JCCE]

1.4 Terminology and Definitions

Technical terms used in this document are defined in [GPCS].

1.5 Abbreviations and Notations

Table 1-2: Abbreviations

Abbreviation	Meaning
AID	Application IDentifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
CDCVM	Consumer Device Cardholder Verification Method
CREL	Contactless Registry Event Listener
CRS	Contactless Registry Services
CVM	Cardholder Verification Method
DGI	Data Grouping Identifier
MNO	Mobile Network Operator
OEM	Original Equipment Manufacturer
PIN	Personal Identification Number
REE	Rich Execution Environment
SIO	Shareable Interface Object
TEE	Trusted Execution Environment

1.6 Revision History

GlobalPlatform technical documents numbered *n*.0 are major releases. Those numbered *n*.1, *n*.2, etc., are minor releases where changes typically introduce supplementary items that do not impact backward compatibility or interoperability of the specifications. Those numbered *n*.*n*.1, *n*.*n*.2, etc., are maintenance releases that incorporate errata and precisions; all non-trivial changes are indicated, often with revision marks.

Table 1-3: Revision History

Date	Version	Description
June 20, 2017	0.2	CSWG update
August 2, 2017	0.0.0.3	CSWG update
January 25, 2018	0.0.0.4	CSWG update / Reworked Java Card API
January 2018	0.0.0.5	Committee Review
November 2018	0.0.0.10	Member Review
April 2019	0.0.0.11	Post-Member Review
November 2019	0.0.0.14	Public Review
TBD	1.0	Public Release

2 Overview

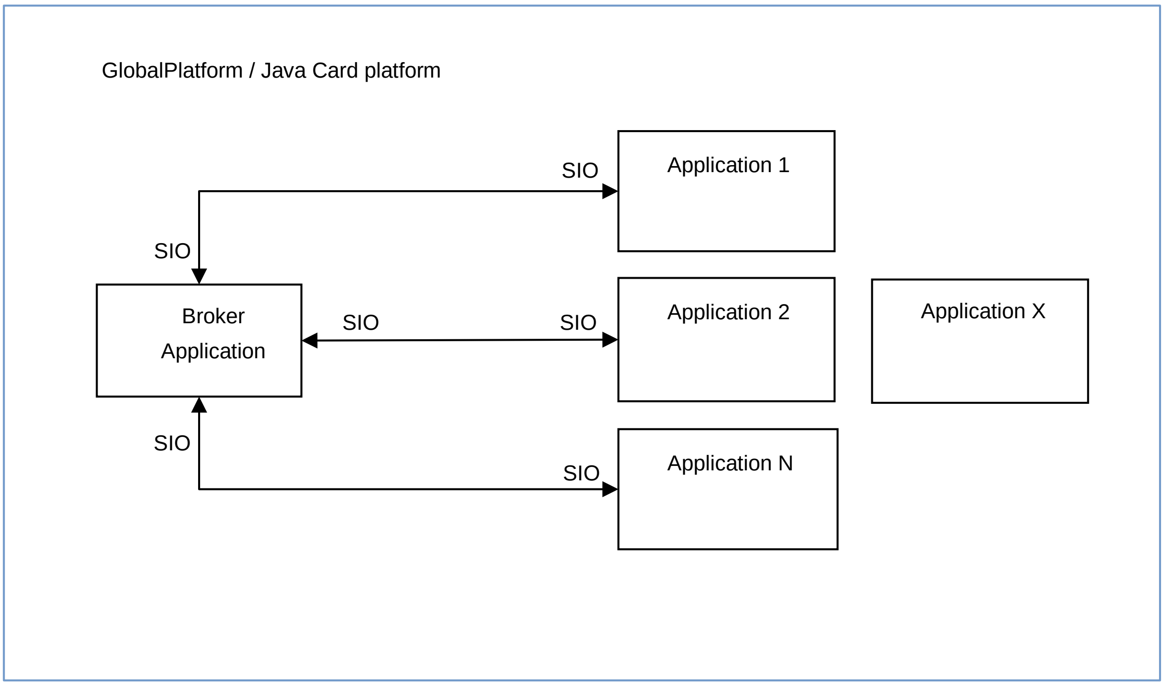
This amendment introduces a new Broker Interface API which provides means for a Broker Application to interact with and supply information to one or more other Applications. A Broker Application receives processed data or receives data which it processes for use by other Applications. In a typical scenario, such information is previously received by the Broker Application from another entity (either on-secure element or off-secure element). The Broker Interface API allows Applications to gather details about this information and how it was collected and consequently evaluate the trust it can place into it. The Broker Application may also request Applications to perform some actions. Several Broker Applications may be present and running on the same Secure Element.

The Broker Interface API (i.e. its Executable Load File) may be loaded onto a GlobalPlatform-compliant platform using procedures described in sections 11.5 and 11.6 of [GPCS].

The AID of its Executable Load File shall be: 'A000000151 08'

Figure 2-1 shows an example of a Broker Application sharing information with other Applications using Shareable Interface Objects (SIO) implementing the Broker Interface API:

Figure 2-1: Interactions Enabled by the Broker Interface API



3 Specification Amendments

This section describes the Broker Interface API and the specific requirements for Broker Applications and Applications wishing to interact with Broker Applications. This specification focuses on a particular Broker Application, but nothing prevents several Broker Applications from interacting with several Applications through the Broker Interface.

3.1 Broker Interface API

The Broker Interface API defines the following Java Card interfaces that enable interactions between Broker Application(s) and other Applications:

- *BrokerListener*

  Interface implemented by an Application that wishes to listen to events, receive and respond to requests sent from Broker Application(s).

- *CDCVMBrokerListener*

  Extension of the *BrokerListener* interface including the definitions specific to the Consumer Device Cardholder Verification Method (CDCVM) use case.

- *BrokerCallbackRequest*

  Interface implemented by a Broker Application allowing Applications to request an immediate notification (i.e. through the *BrokerListener* interface) if the Broker Application is currently aware of the occurrence of a particular type of event.

These interfaces enable the interaction scenarios known as Push and Pull Modes.

Interactions according to the Push Mode require Applications wishing to receive data from a Broker Application to provide a Shareable Interface Object (SIO) implementing the *BrokerListener* interface. In this scenario, the Broker Application shall send events and/or requests to Applications, via the Application's SIO, using the following methods:

- *onBrokerEvent(event, info, environment, device, security, assurance, discretionary)*
- *initBrokerRequest(request)*
- *updateBrokerRequest(inBuffer, inOffset, inLength)*
- *processBrokerRequest(inBuffer, inOffset, inLength, outBuffer, outOffset)*

Interactions according to the Pull Mode require the Broker Application to provide an SIO implementing the *BrokerCallbackRequest* interface. In this scenario, an Application requests the Broker Application to send details of available events via the Application's SIO using the following method:

- *requestCallback(event)*

...where *event* specifies the type of event about which the Application requires information.

In Pull Mode, Applications shall also provide an SIO implementing the *BrokerListener* interface (like in Push Mode) that will be used by the Broker Application to perform the callback. Support for Pull Mode implicitly requires Push Mode to be supported.

3.2 Broker Application Requirements

In both Push and Pull Mode, a Broker Application sends events and requests to Applications. An Application may also respond to a request from a Broker Application.

A Broker Application interacts with an Application using the *BrokerListener* SIO, which is implemented by the Application. In order to retrieve this SIO, the Broker Application calls the *JCSystem.getAppletShareableInterfaceObject(serverAID, parameter)* method, providing the Application’s AID and with *parameter* set to *BrokerListener.SIO_BROKER_LISTENER*. The Broker Application shall not store the reference to this SIO.

If a Broker Application wishes to provide support for Pull Mode, it shall implement a *BrokerCallbackRequest* SIO. This SIO may be made available to other Applications as a Global Service or as a uniquely registered Global Service. When a Global Service is uniquely registered, Applications don't need to know the AID of the Application providing it. Applications wanting to use a non-uniquely registered Global Service need to know (i.e. need to be provisioned with) the AID of the Broker Application providing the service.

In order to provide access to the *BrokerCallbackRequest* SIO of a Broker Application as a Global Service, the following steps shall be followed:

- The Broker Application shall be installed with the Global Service privilege.
- During the installation of the Broker Application, the Service Name '88xx' (with 'xx' different from '00'; see NOTE below) shall be specified as part of the System Install Parameters (sub-tag 'CB' Global Service Parameters).
- The Broker Application shall implement the *Applet.getShareableInterfaceObject(clientAID, parameter)* method in order to return a *GlobalService* SIO, where *parameter* is set to the predefined constant *GPSystem.GLOBAL_SERVICE_IDENTIFIER*. The Broker Application may optionally check the received *clientAID* if it wishes to restrict access to its Global Services to certain Applications.
- This *GlobalService* SIO shall implement the *getServiceInterface(clientRegistryEntry, sServiceName, baBuffer, sOffset, sLength)* in order to return a *BrokerCallbackRequest* SIO when *sServiceName* is correctly set to '88xx' (with 'xx' different from '00'; see NOTE below).

In addition, if the Broker Application wishes to uniquely register this Global Service, it shall use the *GPRegistryEntry.registerService(sServiceName)* method with *sServiceName* set to '88xx' (with 'xx' different from '00'; see NOTE below).

NOTE**:** To register a *CDCVMBrokerCallbackRequest* SIO, the suitable service name would be *CDCVMBrokerCallbackRequest.SERVICE_BROKER_CDCVM* (i.e. '8801'). See details in the Javadoc API documentation.

Only a single Broker Application on the card may uniquely register this Global Service. From this point on, no other Application will be able to uniquely register a Global Service with a Service Name of '88xx' until this Broker Application is deleted or deregisters this Global Service.

3.3 Application Requirements

An Application that wishes to receive events or requests from a Broker Application shall implement the *Applet.getShareableInterfaceObject(clientAID, parameter)* method in order to return its *BrokerListener* SIO to the Broker Application:

- *clientAID* identifies the specific Broker Application;
- *parameter* is set to *BrokerListener.SIO_BROKER_LISTENER*.

The Application may restrict access to specific Broker Applications. For this purpose, the Application may implement a Broker Application Whitelist, in which case it shall reject any request to retrieve its *BrokerListener* SIO coming from a Broker Application that is not referenced in such a list. If supported by the Application, this whitelist shall be stored using a STORE DATA command and DGI '4F55'.

Table 3-1: DGI Used to Personalize Broker Application Whitelist

DGI	Description
'4F55'	Sequence of tags '4F', each containing the AID of a Broker Application that can be trusted by the Application.

If an Application wishes to use Pull Mode, it shall retrieve the *BrokerCallbackRequest* SIO provided as a Global Service by a Broker Application. To do so, the Application shall first obtain the Broker Application’s *GlobalService* SIO using the *GPSystem.getService(serverAID, sServiceName)* method:

- With *serverAID* set to *null*, if the Broker Application uniquely registered its Global Service.
- With *serverAID* set to a particular Broker Application’s AID if that Broker Application did not uniquely register its Global Service. If the Application stores a Broker Application Whitelist (DGI '4F55'), then it may only use an AID present in this whitelist.
- With *sServiceName* set to '88xx' (with 'xx' different from '00'; see NOTE below).

In order to retrieve the Broker Application’s *BrokerCallbackRequest* SIO, the Application shall then invoke the *GlobalService* SIO's *getServiceInterface(clientRegistryEntry, sServiceName, baBuffer, sOffset, sLength)* method with the same *sServiceName*. The Application shall not store the reference to this *BrokerCallbackRequest* SIO.

NOTE: To retrieve a *CDCVMBrokerCallbackRequest* SIO, the suitable service name would be *CDCVMBrokerCallbackRequest.SERVICE_BROKER_CDCVM* (i.e. '8801'). Also, the *BrokerCallbackRequest* SIO initially retrieved shall be explicitly casted to *CDCVMBrokerCallbackRequest* by the caller Application. See details in the Javadoc API documentation.

As support for Pull Mode by a Broker Application is optional (and indeed the specific Broker Application may or may not be present), the Application should check that it successfully retrieved the SIO.

4 Broker Interface Usage for CDCVM

The term CDCVM, standing for Consumer Device Cardholder Verification Method, may actually apply to a wide variety of use cases, where the capture and verification of a CVM value may be implemented in environments other than the one hosting the relying Application (i.e. the Application relying on the CVM verification result). Examples of such use cases include:

- SE hosted by a smartphone: A PIN, pattern, or fingerprint is captured and verified by the host device, then the verification result is transmitted to a Broker Application in the SE.
- SE hosted by a contact or contactless card: A fingerprint is captured using a sensor on the card, then transmitted to the hosted SE for verification, then the verification result is transmitted to a Broker Application (internally to the SE).
- SE connected to a remote server: A CVM verification result is transmitted to a Broker Application in the SE, through an OTA secure channel.
- SE part of a FIDO solution: A CVM verification result is transmitted to a Broker Application in the SE, through channels as defined by FIDO.

4.1 Overview

The CDCVM verification results can be broadcasted through a Broker Application to one or several other Applications. This can be achieved in two modes:

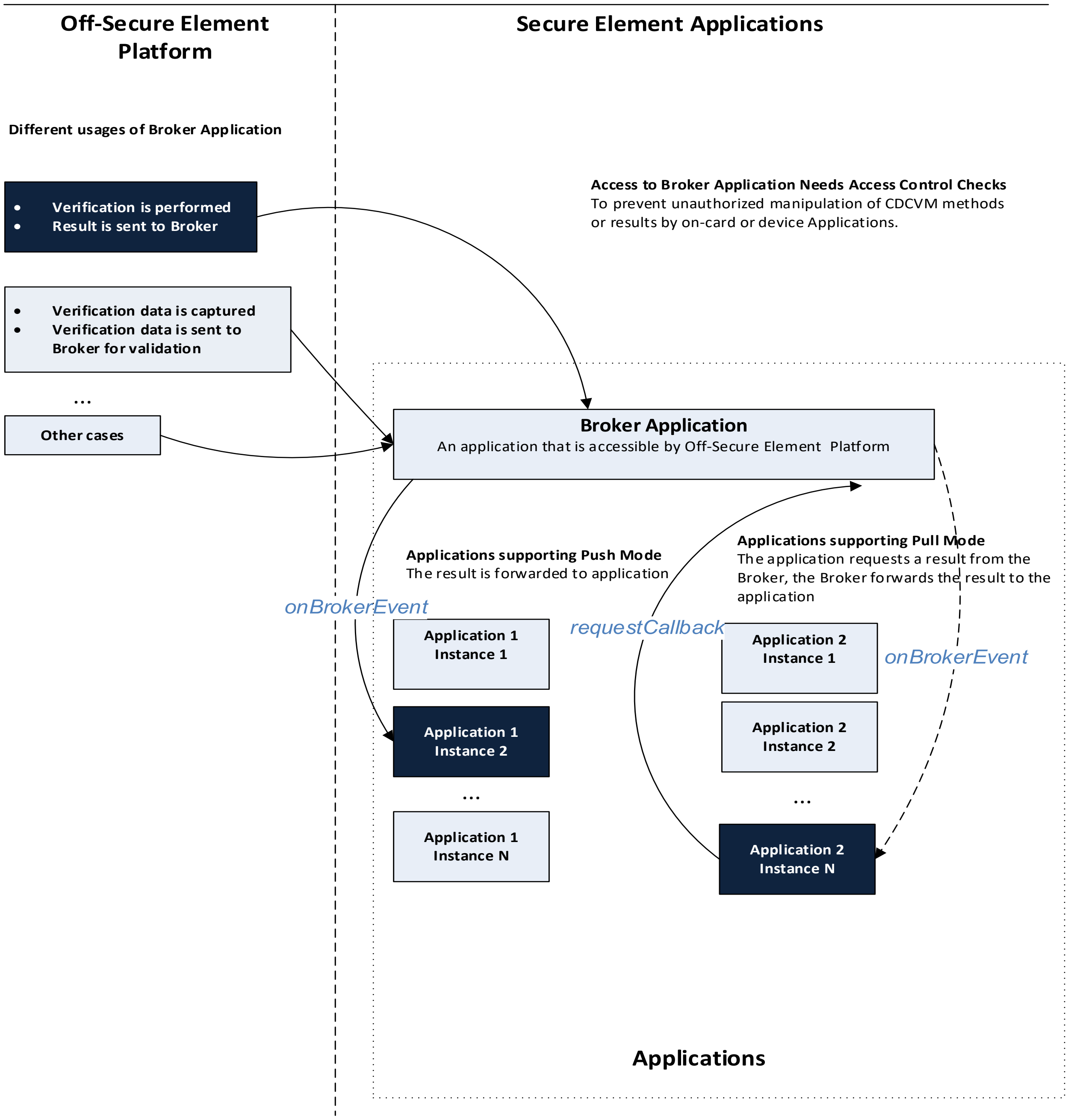
- Push Mode: The CDCVM results are broadcasted to all Applications.

  For example, the Broker Application may be implemented as a CRS Application (as defined in GlobalPlatform Contactless Services, Amendment C [Amd C]). This would allow it to access the Contactless Registry, track the list of Applications activated over the contactless interface, and push the CDCVM results to them.

  For example, the Broker Application may also be implemented as a CREL Application (as defined in [Amd C]). This would allow it to push the CDCVM results to its associated Applications (which are activated over the contactless interface).
- Pull Mode: The CDCVM results, if available, may be queried by Applications.

Figure 4-1 shows an example of the process flow for Push and Pull Modes.

Figure 4-1: Example of Process Flow for Push and Pull Mode


Off-Secure Element Platform
Secure Element Applications
Different usages of Broker Application
• Verification is performed
• Result is sent to Broker
• Verification data is captured
• Verification data is sent to Broker for validation
...
Other cases
Access to Broker Application Needs Access Control Checks
To prevent unauthorized manipulation of CDCVM methods or results by on-card or device Applications.
Broker Application
An application that is accessible by Off-Secure Element Platform
Applications supporting Push Mode
The result is forwarded to application
onBrokerEvent
Application 1 Instance 1
Application 1 Instance 2
...
Application 1 Instance N
requestCallback
Applications supporting Pull Mode
The application requests a result from the Broker, the Broker forwards the result to the application
Application 2 Instance 1
onBrokerEvent
Application 2 Instance 2
...
Application 2 Instance N
Applications

The set of Applications known to and interacting with the Broker Application is out of scope of this specification.

The Broker Application is able to transmit the following information:

- CDCVM type used for cardholder verification (fingerprint, pattern, password, PIN, etc.)
- CDCVM value, if the verification is to be performed on the Secure Element
- CDCVM outcome / status (verified, failed, blocked, not performed, not applicable, error, etc.)

- Form factor of the device where the CDCVM was captured (e.g. mobile device, wearable, etc.)
- Runtime environment where the CDCVM was captured (e.g. REE, TEE, undefined, etc.)
- Runtime environment where the CDCVM was verified (e.g. REE, TEE, undefined, etc.)
- Type of security protecting the transmission of data between the runtime environment where the CDCVM was captured and the runtime environment where it was verified (secure channel, no secure channel, specific interface, etc.)
- Security assurance level
- Request to verify a CVM (i.e. on-card CVM verification use cases)

4.2 Parameter Values for the *CDCVMBrokerListener* Interface

This section describes the values that the Broker Application may use for the different parameters used when invoking the *CDCVMBrokerListener.onBrokerEvent(event, info, environment, device, security, assurance, discretionary)* method. These values are reflected by the constants defined in the *CDCVMBrokerListener* interface.

4.2.1 CDCVM Event

This section describes the possible values for the *event* parameter.

Table 4-1: List of Values for the CDCVM Event Parameter

CDCVM Event	CDCVM Type	Description
1st byte	2nd byte	
'00'	'00'	No CDCVM performed
'01'		CDCVM verification succeeded
'02'		CDCVM verification failed
'03'		CDCVM blocked
'04'		CDCVM verification succeeded – consumed (e.g. used by some Application in a transaction)
'05'		Error
'FF'		CDCVM verification result – invalidated (e.g. after a timeout)
'06' to 'FE'		RFU
	'10' to '1F'	Passcode (including password)
	'10'	Default
	'11'	PIN (digits)
	'12'	PIN (digits, scrambled PIN pad)
	'13'	Password (any character)
	'20' to '2F'	Fingerprint
	'20'	Default
	'30' to '3F'	Pattern
	'30'	Default
	'40' to '4F'	Facial biometric
	'40'	Default
	'50' to '5F'	Iris biometric
	'50'	Default
	'60' to '6F'	Voice biometric
	'60'	Default
	'70' to '7F'	Vein recognition (e.g. palm)
	'70'	Default
	'80' to 'FE'	RFU
	'FF'	All CDCVM types

The Application may receive information indicating a CDCVM type in any of the ranges specified in Table 4-1 (2nd byte). Each value range corresponds to a general CDCVM type, but the interpretation of specific CDCVM type values is out of scope of this version of the specification.

The event "CDCVM verification result – invalidated" indicates that any previously recorded occurrence of event "CDCVM verification succeeded" shall be discarded. An application shall be notified of this event only once. After that, until the Broker Application becomes aware of any new CDCVM event, the Broker Application shall send notification of the event "No CDCVM performed" to any Application requesting information about the last CDCVM event (i.e. in Pull Mode).

4.2.2 CDCVM Information

This section describes the possible values for the *info* parameter.

Table 4-2: List of Values for the CDCVM Info Parameter

Value		Description
1st byte	2nd byte	
'XY'	--	X: Try Limit (0 to 15), with the following meaning: 0 – The Try Limit is unknown or not applicable or greater than 15. 1..15 – Try Limit value Y: Remaining Tries Counter (0 to 15), with the following meaning: 0 – The CDCVM is known to be blocked (i.e. information shall not conflict with the value of the *event* parameter). 1..14 – Remaining Tries Counter value (CVM not blocked) 15 – The Remaining Tries counter is unknown or not applicable or greater than 14. NOTE: A value of 0 shall be used instead if the CDCVM is known to be blocked.
--	'xx'	RFU

4.2.3 CDCVM Environment

This section describes the possible values for the *environment* parameter, carrying information about where the CDCVM was captured and verified.

Table 4-3: List of Values for the CDCVM Environment Parameter (Byte 1)

Environment where Captured 1st byte	Description
'00'	Unknown or undefined entity
'10' to '1F'	Untrusted Input Hardware (e.g. input managed by Host Device OS)
'10'	Default
'20' to '2F'	Trusted Input Hardware (e.g. input managed by TEE)
'20'	Default
'30' to '3F'	External Trusted Device
'30'	Default
	All other values are RFU

Table 4-4: List of Values for the CDCVM Environment Parameter (Byte 2)

Environment where Verified 2nd byte	Description
'00'	Unknown or undefined entity
'10' to '1F'	Host Device OS
'20' to '2F'	Trusted Execution Environment (TEE)
'30' to '3F'	Host Device Application (running in Rich Execution Environment – REE)
'30'	Default
'31'	Digital Wallet
'40' to '4F'	Remote Server (Cloud)
'50' to '5F'	Application on same Secure Element as Broker Application
'60' to '6F'	Application on another Secure Element
'70' to '7F'	External Trusted Device
	All other values are RFU

4.2.4 CDCVM Device Identifier

This section describes the possible values for the *device* parameter, carrying information about the device on which the CDCVM was captured and verified.

Table 4-5: List of Values for the CDCVM Device Parameter

Device Where Captured	Device Where Verified	Description
1st byte	2nd byte	
'00'	'00'	Unknown
'01' to '0F'	'01' to '0F'	RFU
'10' to '1F'	'10' to '1F'	Host Device
'20' to '2F'	'20' to '2F'	Companion Device (i.e. associated to a primary Host Device)
	'20'	Default
	'21'	Mobile Phone
	'22'	Wearable
	'23'	Computer
	'24'	Tablet
'30' to 'FF'	'30' to 'FF'	RFU

4.2.5 CDCVM Security Features

This section describes the possible values for the *security* parameter, carrying information about how the following communications are secured:

- Between entities responsible for CDCVM capture and CDCVM verification
- Between the entity responsible for CDCVM verification and the Broker Application

Table 4-6: List of Values for the CDCVM Security Parameter

Value		Description
1st byte Security between captured and verified	2nd byte Security between verified and Broker	
'00'	'00'	No security/Unknown
'01'	'01'	Not Applicable (e.g. performed by the same entity, no transmission required)
'02' to '0F'	'02' to '0F'	RFU
'10' to '1F'	'10' to '1F'	Secure Channel
'20' to '2F'	'20' to '2F'	No Secure Channel, dedicated hardware interface
'30' to 'FF'	'30' to 'FF'	RFU

4.2.6 CDCVM Security Assurance

This section describes the possible values for the *assurance* parameter, carrying information about how the CDCVM system can be trusted.

Table 4-7: List of Values for the CDCVM Security Assurance Parameter

Value		Description
1st byte	2nd byte	
'00'	'00'	No Assurance/Unknown
'01' to 'FF'	'01' to 'FF'	RFU

4.2.7 CDCVM Discretionary Data

This section describes the possible values for the *discretionary* parameter.

Table 4-8: List of Values for the CDCVM Discretionary Parameter

Value		Description
1st byte	2nd byte	
'00'	'00'	No discretionary data /Unknown
'01' to '7F'	'01' to '7F'	RFU
'80' to 'FF'	'80' to 'FF'	For Proprietary Usage

4.3 CDCVM Verified on the Secure Element

This section describes the values that the Broker Application may use for the different parameters used when invoking the *CDCVMBrokerListener.initBrokerRequest(request)* method. These values are reflected by the constants defined in the *CDCVMBrokerListener* interface.

4.3.1 CDCVM Request

This section describes the possible values for the *request* parameter.

Table 4-9: List of Values for the CDCVM Request Parameter

Value		Description
1st byte	2nd byte	
'00'		Verify CDCVM
'01' to 'FF'		RFU
	'10'	Passcode (including password)
	'20'	Fingerprint
	'30'	Pattern
	'40'	Facial biometric
	'50'	Iris biometric
	'60'	Voice biometric
	'70'	Vein recognition (e.g. palm)
	'FF'	Any CDCVM type
		All other values RFU

Assigning the first byte of the *request* parameter to “Verify CDCVM” allows the Broker Application to request the Application to verify the CDCVM. The *CDCVMBrokerListener.processBrokerRequest(..)* method shall return with no error if the verification was successful or throw an *ISOException* in case of verification failure.

4.4 Parameter Values for the *CDCVMBrokerCallbackRequest* Interface

This section describes the values that an Application may use for the *request* parameter when invoking the *CDCVMBrokerCallbackRequest.requestCallback(request)* method in a Pull Mode scenario:

- '01': Request last CDCVM event (of specified type)
- '02': Request new CDCVM verification attempt
- '03': Request notification of “CDCVM verification succeeded – consumed” event

These values are also reflected by the constants defined in the *CDCVMBrokerCallbackRequest* interface.