RN00263

TapLinx Android SDK

Rev. 3.0 — 13 November 2025 **Release notes**

1 General points

The TapLinx Android SDK consists of an Android library that can be used to generate APDUs to communicate with NXP Smartcards, leveraging the Android SDK's transceive functionality. It supports Android versions from 4.1 to 15. Note that NFC transceive operations in Android depend on the middleware provided by OEMs.

Figure 1. TapLinx

2 Release contents

2.1 Version history

Table 1. Version history

Release date	Version
3 November 2025	5.0.0
11 July 2025	4.1.0
14 February 2025	4.0.0
28 June 2023	3.1.0
26 November 2022	3.0
05 August 2022	2.0
13 January 2022	1.9
27 January 2020	1.8
09 August 2019	1.7
21 March 2019	1.6
12 July 2018	1.5
16 November 2017	1.4
22 May 2017	1.3
08 February 2017	1.2
17 January 2017	1.1

2.2 User guide

Refer to UG10044 - Starting development with TapLinx Android SDK.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

3 New features

Version 5.0.0

1. Added full support for Mifare DUOX.

Version 4.1.0

1. Added limited support of MIFARE DUOX.

Version 4.0.0

1. Added support for ICODE 3 and ICODE 3TT Cards.
2. Added Limited Support for MIFARE DUOX - Card Detection and ECC Originality Check.

Version 3.1.0:

1. Added support to identify the manufacturing site of MIFARE DESFire EV3 and MIFARE DESFire EV3C with Get_Fab_Identifier API.
2. Added support for MIFARE DESFire EV3 [16K] variant.

Version 3.0.0:

1. Offline License Key Length for Offline License Verification is changed from 256 bytes to 2048 bytes for the security reasons.

Note:

1. *Existing customers (Using TapLinx Android SDK) will only be affected if they upgrade to TapLinx V3.0.0 and continue to use their earlier downloaded Offline License Key of 256 bytes length.*
2. *If the existing customers (Using TapLinx Android SDK only) upgrade to TapLinx 3.0.0 they must also regenerate the new Offline License Key of 2048 bytes from mifare.net for Offline License Verification feature to work.*

Version 2.0:

1. Added support for MIFARE DESFire EV2 card and MIFARE DESFire EV2J/Pay Cards.
2. Added support for Android 12.
3. Bug Fixes

Version 1.9:

1. Added support for MIFARE DESFire EV3C card.
2. Added support for NTAG22x DNA cards, NTAG22x NDA SD cards.
3. Added support for MIFARE Ultralight AES cards.

Version 1.8:

1. Added support for MIFARE DESFire EV3.
2. Added support for NTAG 5 Boost, NTAG 5 Switch and NTAG 5 Link.
3. Added support for MIFARE Plus EV2 card.

Version 1.7:

1. Added support for DESFire Ev2 proximity check.
2. Added support for DESFire Light NDEF format.

Version 1.6:

1. Introduced Offline Local License Verification.
2. Added support for DESFire EV2 XL.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Version1.5:

NA

Version 1.4:

1. Implemented a method for getting the current version of TapLinx.
2. Implemented authentication using predictable challenge for DESFireEV1.

Version 1.3:

1. Added support for INTAG213TagTamper.

Version 1.2:

NA

Version 1.1:

1. Added support for MIFARE Identity.
2. Added support for NTAG413 DNA.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

4 Enhancements

Version 5.0.0:

NA

Version 4.1.0:

NA

Version 4.0.0:

NA

Version 3.1.0:

NA.

Version 3.0.0:

NA.

Version 2.0:

NA.

Version 1.9:

1. Enhancements made for PLUS EV2 and DESFire EV2 to receive the MAC response.

Version 1.8:

NA.

Version 1.7:

NA.

Version 1.6:

1. Enhanced app registration by adding Offline Local License Verification, where user can use the TapLinx library without verifying the license Online, provided the offline verification key is genuine.
2. Replaced Google Analytics by Firebase Analytics.

Version 1.5:

1. Added ISO14443-L4 support to Identify for CID,NAD is supported in RATS response.
2. Added Command exchange as per ISO14443-L4.
3. Added Implement Sample application using ISO14443-L4 implementation and using Plus EV1 card.
4. TapLinx- Value of Mirror Page and Mirror Byte is set to 0 if bytes required calculation is 0 in NTAG21X.

Version 1.4:

1. Add support for predictable challenge for DESFire EV1.
2. ISO File Id should be accepted Uniformly all the APIs and cards.
3. Refactoring of Sample Application source code.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Version 1.3:

1. Large Frame Size support in DESFire EV2 Family cards.
2. Updated clone detection for newly identified clone cards.
3. Split IDESFireEV2#getKeySetVersion in to two APIs IDESFireEV2#getAllKeySetVersion and IDESFire EV2#getKeyVersionFromKeySet.

Version 1.2:

NA

Version 1.1:

1. Getter methods are added to the FileSettings and its derived classes.
2. Strict length error check is added on responses from all the cards in the detection logic.
3. Symmetric originality check in DESFireEV2 is tested.
4. New ISO Select file API is added which returns FCI.
5. NTAG213 and Ultralight detection speed improvement.
6. FileSettings base class made abstract & hence users cannot create it directly.
7. Added support for MIFARE Identity and NTag413 DNA in sample application.
8. Merge InvalidArgumentException & UsageException to single exception and updated the javadoc suitably.
9. As NTAG413 DNA is a derivative of DESFire in spite of being named as NTAG413, and hence this will be under com.nxp.nfclib.desfire package. So import the desfire package to use the NTAG413 DNA.
10. “public void enableReaderMode(final int presenceCheckDelay, NfcAdapter.ReaderCallback readerCallback, int flags) throws NxpNfcLibException” API is added in the NxpNfcLib class to set the custom presence check delay on certain devices where presence check happens too often and cause issues in the session based communication with advanced cards like DESFire EV1 and EV2.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

5 Bug fixes

Version 5.0.0:

1. Resolved issue with NTAG 5 cards where full memory could not be utilized for writing NDEF data.

Version 4.1.0:

NA

Version 4.0.0:

1. For NTAG5 cards, interface object is not updated on every tap within the same session is fixed.
2. Crashes on specific devices while detecting MIfare Plus SL1 Cards is Fixed

Version 3.1.0:

1. For DESFireEV2 card → Failures in DESFireEV2ChangeKeyEV2Factory API on DESFire EV2 2 k, 8 k and 32 k cards is fixed.
2. For MIFARE Classic 1k card → Failure in UnitTestCases1 is fixed.

Version 3.0:

1. NTAG223DNASD → Issue with setCTTCurrentTrimValue() and getCTTCurrentTrimValue() API's is Fixed.

Version 2.0:

1. For NTAG223DNASD and NATG224DNASD → ShowStoredTTStatus and ShowActualTTStatus API's are updating on a wrong byte.
2. For NTAG223DNA and NATG223DNASD → BlockLocking of Incorrect Pages id Fixed.
3. NTAG223DNA and NATG223DNASD → Locking of Incorrect Pages id Fixed.
4. For DESFire EV2J card → Issue with ApplicationKeySettings API.
5. For DESFire EV2J card → Issue in LinearRecordFile_PICC_AES_TestCase API is fixed.
6. For DESFireEV2J Card → Issue in NativeChainingReadWriteDataTestFactory_ISO API is fixed.
7. Issue with TapLinx continuing to send Additional Frames even if there is an error response returned during first frame is fixed.
8. For DESFire EV2/EV3 card → LengthError in read data, record, getValue, getFreeMemory commands is fixed.
9. For DESFire EV3 and DESFire EV3C cards → Issue in getFileCounter API is fixed.
10. For DESFire EV3 and DESFire EV3C → SDMCtrRetAccessByte missing issue fixed.
11. For NTAG424 DNA card → Issue with ReadWriteNDEFTestCases Failure is fixed.
12. ForPlusEV2 Card → Failure in SL1GetVersion and Read Signature commands is fixed.

Version 1.9:

1. For DESFire EV2 → CommitTransaction API to return TMC and TMV values.
2. For DESFire EV3 → Issue with ISO and AES Auth Types when using getFileSettings.
3. For DESFire EV2 → Issue with Read Records API.
4. For DESFire EV2 → Reading Large data from Record File.
5. Plus EV1 2 k cards getting detected as PlusSL0 Card.
6. Classic Family cards are detected multiple times on Android 10.

Version 1.8:

1. Bug fixes with Proximity check on DESFire Ev2.
2. DESFire EV2 Read Signature API fails in the authenticated state (Native, ISO or AES).
3. Integrity error in Change Key settings using DESFire EV2 secure messaging.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

4. For DESFire Ev2, Failed to append MAC while reading record file in Enciphered/MACed communication mode.
5. For DESFire Ev2, getFreeMemory() API fails after Secure Messaging & AuthenticateNonFirst is restricted by Key Length in DESFire EV2.
6. Bug fix for Read/Write NDEF and FormatT2T API in NTagI2CPlus and all T2T cards.
7. Bug fix on boundary values for Linear record API.
8. Fixed the Analytics performance issue on Authentication for DESFire EV2.

Version 1.7:

1. Offline registration with TapLinx for Android 21 is failed – resolved.

Version 1.6:

1. Key diversification for AES with diversification input as 15 bytes is not valid – resolved.
2. User is unable to write NDEF message for MIFARE Ultralight card – resolved.
3. DESFire - Predictable Challenge AuthCounter is not getting increased – resolved.
4. Issue in isoSelect API (For Plus EV1) - resolved.

Version 1.5:

1. In a TapLinx Desktop Java Sample App, connected reader name is not displayed - resolved.
2. In a TapLinx Desktop Java Sample App, No Error message is shown when Reader is not connected ad execute button is clicked- reader - resolved.
3. In a TapLinx Desktop Java Sample App, Application doesn't respond/hangs when card is not available on reader and execute button is clicked - resolved.
4. In a TapLinx Desktop Java Sample application --> Response is shown as null when EV1 card is placed on reader and In Plus EV1 SL3 tab is selected. – resolved.
5. In a TapLinx Desktop Java Sample App, Log communication doesn’t change when user is navigated from one tab to another.
6. Andriod sample app after second tap null pointer exception is thrown – resolved.
7. Finalize keyset operation is failling after change key with integrity error – resolved.
8. In DESFire EV2, Linear and Cyclic Records Write /Read fails with Boundary Error - resolved.
9. TapLinx - WriteData on DESFireEV1 for file whose size is greater than 0xFF is failing - resolved.
10. TapLinx - Can not communicate with Mifare Plus EV1 SL1 with any sector moved to Sl1SL3Mix mode.
11. TapLinx - WriteData on DESFireEV1 for file whose size is greater than 0xFF is failing.
12. TapLinx - NTAG 413 DNA Interaction Counter not visible in Read flow.

Version 1.4:

1. Issue related to changeKeySettings at PICC level for DESFire EV1/2 cards in which key settings bitmap wrongly generated - Resolved.
2. Verified support for NTAG413 DNA and some bugs fixes related to file settings - Resolved.
3. Key Diversification for AES128, 2k3DES and 3k3DES verified and issue related to padding - Resolved.
4. For NTAG413 DNA getNFCCounter was failing in TapLinx - Resolved.
5. For DESFire EV2, ChangeKey with diversified key giving integrity error - Resolved.
6. For DESFire EV2, Credit/Debit operations are failing after EV2 non-first authentication - Resolved.
7. For DESFire EV2, Read/Write data is giving length error in EV2 secure messaging - Resolved.
8. For DESFire EV2 ISO-Chaining was not working for writeData/Record commands - Resolved.
9. AuthStatus is not lost when new tag gets connected - Resolved.
10. For DESFire EV2, getValue() always sent with communicationMode = MACed - Resolved.
11. For DESFire, ISO select is failing with null pointer exception - Resolved.
12. For DESFire EV2, creating Transaction MAC File in DAM application returns length error - Resolved.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

13. For DESFire EV2, not able to add Multiple file access rights while creating file - Resolved.
14. For DESFire, If Card or application is configured for 2k3DES and if user pass 16 bytes in authentication sometime invalid key length exception is thrown - Resolved.
15. Get Free Memory issue - Resolved.
16. TapLinx-Static Key authentication for getting Card UID for Diversified keys - Resolved.
17. For DESFire, file number issue - Resolved.
18. For DESFire, change Key issue in 3k3DES - Resolved.
19. For DESFire EV2, bug in transcieve command - Resolved.
20. For NTAG I2C plus, in fastWrite(Data) API session register were not getting updated- Resolved.
21. For DESFire, Illegal Command Code using Predictable Challenge - Resolved.

Version 1.3:

1. Add Failure Exceptions in ICODE DNA.
2. ChangeKey issue fixed for application level in DESFire EV2 card.
3. getKeySetVersion Method does not return the correct version fixed.
4. DESFire EV1/EV2 setConfiguration bug fixes.
5. IDESFireEV1.getFreeMemory() return invalid values fixed.
6. Plus SL1 do originality check should accept SL1CardAuthKey to perform originality check.
7. Misbehavior on wrong password on an Ultralight EV1.
8. Improvement for misusing TapLinx for attacks on an Ultralight EV1.
9. Mifare DESFire EV1/2 auth/read/identify problems on Samsung Galaxy S7/S7 Edge.

Version 1.2:

1. Issue related to DESFire EV1 standard file communication in encrypted mode resolved.

Version 1.1:

1. TapLinx sending an annoying popup if NFC is off is removed.
2. Desfire getting detected as Plus on certain devices is fixed.
3. Bug found in MISMART Application in EV2 secure messaging is fixed.
4. Fixed create cyclic record file creation API in DESFire.
5. Fixed ChangeKeySettings API for DESFire EV2 at application level.
6. In DESFireEV2, isoSelectFile does not returns FCI template.
7. In DESFireEV1, readNDEF is failing with exception format exception in ISO CommandSet mode.
8. NtagFactory#isNTAG203x API is fixed.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

6 API signature changes

Version 5.0.0:

Table 2. Version 5.0.0 API signature changes

Class	Version 4.1.0	Version 5.0.0
MifareDUOX	writeData(int fileNo, byte[] crlVersion, int offset, byte[] data, byte[] crlSignature)	writeData(int fileNo, byte[] crlVersion, int offset, byte[] data, boolean is CRLSignatureRequired, PrivateKey ca RootKey)
MifareDUOX	writeData(int fileNo, byte[] crlVersion, int offset, byte[] data, byte[] crlSignature, CommunicationType communication Type)	writeData(int fileNo, byte[] crlVersion, int offset, byte[] data, boolean is CRLSignatureRequired, PrivateKey caRootKey, CommunicationType communicationType)
MifareDUOX	changeMifareDUOXFileSettings(int fileNo, MifareDUOXFile.DUOXFile Settings duoxFileSettings)	changeFileSettings(int fileNo, Mifare DUOXFile.DUOXFileSettings duoxFile Settings)
MifareDUOX	setConfigurationCrypto APIManagement(CommunicationType commMode, byte accessCondition)	setConfigurationForCrypto APIManagement(CommunicationType commMode, byte accessCondition)

Version 4.1.0:

Table 3. Version 4.1.0 API signature changes

Class	Version 4.0.0	Version 4.1.0
MifareDUOX	doOriginalityCheck()	doOriginalityCheck(File caFile)

Version 4.0.0:

NA

Version 3.1.0:

NA

Version 3.0.0:

Table 4. Version 3.0.0 API signature changes

Class	Version 2.0	Version 3.0.0
NTAG223DNAStatusDetect	int getCTTCurrentTrimValue()	Change in return type from CTTCurrent TrimValue to int.
NTAG223DNAStatusDetect	void setCTTCurrentTrimValue(int)	Change in signature from CTTCurrent TrimValue to int.
NTAG224DNAStatusDetect	int getCTTCurrentTrimValue()	Change in return type from CTTCurrent TrimValue to int.
NTAG224DNAStatusDetect	void setCTTCurrentTrimValue(int)	Change in signature from CTTCurrent TrimValue to int.
UltralightAES	void setNegativePwdLimit(byte)	Method was inherited from com.nxp. nfclib.ultralight.UltralightEV1, but is now defined locally.

Version 2.0:

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 5. Version 2.0 API signature changes

Class	Version 1.9	Version 2.0
NTAG224DNA	void authenticate (IKeyData, KeyType)	void authenticate(IKeyData, NTAGKey Type)
NTAG224DNA	void programKey (KeyType, byte[])	void programKey(NTAGKeyType, byte[])
UltralightAES	void authenticate (IKeyData, KeyType)	void authenticate(IKeyData, Ultralight KeyType)
UltralightAES	void programKey (IKeyData, KeyType)	void programKey(UltralightKeyType, byte[])

Version 1.9:

Table 6. Version 1.9 API signature changes

Class	Version 1.8	Version 1.9
desfire.EV3PICCConfigurationSettings	void setVcTypeIdentifier(EV3VCType Identifier)	void setVCTypeIdentifier(VirtualCard TypeIdentifier)

Version 1.8:

Table 7. Version 1.8 API signature changes

Class	Version 1.7	Version 1.8
IDESFireEV2	byte[] isoSelect(byte[], IKeyData)	Change in return type from void to byte[].
ValueFileConfigurationStructure	ValueFileConfigurationStructure get Instance(int, int, int, int, boolean, boolean)	Change from final to non-final.

Version 1.7:

NA

Version 1.6:

NA

Version 1.5:

Table 8. Version 1.5 API signature changes

Class	Version 1.3	Version 1.5
INTag413DNA	boolean verifySecureDynamic MessagingMac(byte[], byte[], byte[], byte[], byte[])	Change in signature from (byte[], byte[], byte[], IKeyData, byte[]) to (byte[], byte[], byte[], byte[], byte[]).

Version 1.4:

NA

Version 1.3:

Table 9. Version 1.3 API signature changes

Class	Version 1.3	Version 1.2
IplusEV1SL1	byte[] activateLayer4()	Void activateLayer4()

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 9. Version 1.3 API signature changes*...continued*

Class	Version 1.3	Version 1.2
EV1ApplicationKeySettings	EV1ApplicationKeySettings(byte, byte)	EV1ApplicationKeySettings(byte[])
IDESFireEV1	byte getKeyVersion(int)	Byte[] getKeyVersion(int)

Version 1.2:

NA

Version 1.1:

Table 10. Version 1.1 API signature changes

Class	Version 1.0	Version 1.1
DESFireFile.RecordFileSettings	public RecordFileSettings (final FileType type, final CommunicationType comSettings, final byte readAccess, final byte writeAccess, final byte readWriteAccess, final byte changeAccess, final int recordSize, final int maxNrOfRecords, final int currNoOfRecords)	public RecordFileSettings (final FileType type, final CommunicationType comSettings, final byte readAccess, final byte writeAccess, final byte readWriteAccess, final byte changeAccess, final int recordSize, final int maxNumberOfRecords, final int currentNumberOfRecords, final byte numberOfAdditionalAccess Conditions, final byte[] numberOf AdditionalAccessRights)
DESFireFile.StdDataFileSettings	public StdDataFileSettings(final CommunicationType comSettings, final byte readAccess, final byte writeAccess, final byte readWriteAccess, final byte changeAccess, final int fileSize, final byte nrAddARS, final byte[] addAccessRights)	public StdDataFileSettings(final CommunicationType comSettings, final byte readAccess, final byte writeAccess, final byte readWriteAccess, final byte changeAccess, final int fileSize, final byte numberOfAdditionalAccess Conditions, final byte[] numberOfAdditionalAccess Rights)
DESFireFile.BackupDataFileSettings	public BackupDataFileSettings(final CommunicationType comSettings, final byte readAccess, final byte writeAccess, final byte readWriteAccess, final byte changeAccess, final int fileSize, final byte nrAddARS, final byte[] addAccessRights)	public BackupDataFileSettings(final CommunicationType comSettings, final byte readAccess, final byte writeAccess, final byte readWriteAccess, final byte changeAccess, final int fileSize, final byte numberOfAdditionalAccess Conditions, final byte[] numberOfAdditionalAccess Rights)
IPlusEV1SL1	void sectorSwitchToSL3(final IKeyData key, final int sectorCount,	void sectorSwitchToSL3(final IKeyData sectorSwitchKey, final Map<Integer,

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 10. Version 1.1 API signature changes*...continued*

Class	Version 1.0	Version 1.1
	final Map<Integer, IKeyData> keyBBlockNumberMap)	IKeyData> keyBBlockNumberMap);
IPlusEV1SL1	void sectorSwitchToSL1SL3Mix(final IKeyData key, final int sectorCount, final Map<Integer, IKeyData> blockNumberToKeyMap)	void sectorSwitchToSL1SL3Mix(final IKeyData sectorSwitchKey, final Map<Integer, IKeyData> blockNumberToKeyMap)
EV1ApplicationKeySettings	public Builder setAuthentication RequiredForFileManagement(final boolean authenticationRequiredFor FileManagement)	public Builder setAuthentication RequiredForApplicationManagement(final boolean authenticationRequiredFor ApplicationManagement)
EV1PICCConfigurationSettings	public void setPICCConfigurations(final boolean useRandomID, final boolean disableFormatCommand)	public void setRandomIDAndFormat Configuration(final boolean useRandomID, final boolean disableFormatCommand)
EV2PICCConfigurationSettings	public void setPICCConfigurations(final boolean authVCMandatory, final boolean pcMandatory)	public void setVCAnd PICConfigurations(final boolean authVCMandatory, final boolean pcMandatory)
DESFireEV1	byte[] getCardUID()	byte[] getManufacturerUID()

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

7 Removed/Obsolete APIs

Version 5.0.0:

Table 11. Version 5.0.0 Removed/Obsolete APIs

Class	API	Alternative API
MifareDUOX	setConfigurationToEnableISOSelectFileECCBasedAuthentication(CurveID curveID)	setConfigurationForSecureMessagingConfiguration(boolean enableISOSelectFileFastUIDRetrieval, boolean enableFreeAccessOverI2C, boolean enableEVChargingCardUnilateralAuth, boolean enableISOSelectFileIntegratedECCBasedAuthentication, boolean disableChainedWriting, CurveID curve ID, byte appUIDEncKey)
MifareDUOX	setConfigurationToEnableEVChargingCardUnilateralAuth()	setConfigurationForSecureMessagingConfiguration(boolean enableISOSelectFileFastUIDRetrieval, boolean enableFreeAccessOverI2C, boolean enableEVChargingCardUnilateralAuth, boolean enableISOSelectFileIntegratedECCBasedAuthentication, boolean disableChainedWriting, CurveID curve ID, byte appUIDEncKey)
MifareDUOX	setConfigurationToEnableISOSelectFileFastUIDRetrieval(byte appUIDEncKey)	setConfigurationForSecureMessagingConfiguration(boolean enableISOSelectFileFastUIDRetrieval, boolean enableFreeAccessOverI2C, boolean enableEVChargingCardUnilateralAuth, boolean enableISOSelectFileIntegratedECCBasedAuthentication, boolean disableChainedWriting, CurveID curve ID, byte appUIDEncKey)

Version 4.1.0:

NA

Version 4.0.0:

NA

Version 3.1.0:

Table 12. Version 3.1.0 Removed/Obsolete APIs

Class	API	Alternative API
IPlusEV1	IPlusEV1SL3 getSL3SectorHelper()	Gets the Plus SL3 object which can be used to do SL3 operations on sectors that have been moved to SL3.
IPlusEV2	void proximityCheck(IKeyData, int)	1. Prepare Proximity Check. 2. Proximity Check. 3. Verify Proximity Check.

Version 3.0.0:

NA

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Version 2.0:

Table 13. Version 2.0 Removed/Obsolete APIs

Class	API
IDESFireEV3	void updateRecord(int, int, int, byte[])
IDESFireEV3	void updateRecord(int, int, int, byte[])

Version 1.9:

Table 14. Version 1.9 Removed/Obsolete APIs

Class	API	Alternative API
ntag.INTag213215216	void setMemProtectionAndPwd VerificationForReadWriteAccess(byte, boolean) Now deprecated.	Use enablePasswordProtection or enableAesProtection API instead.

Version 1.8:

Table 15. Version 1.8 Removed/Obsolete APIs

Class	API	Alternative API
DESFireFile.FileSettings	DESFireFile.FileSettings(FileType, CommunicationType, byte, byte, byte, byte)	Base class Constructor.
DESFireFile.FileType	byte mValue	holds the file mType enum value.
EV1ApplicationKeySettings.	Builder byte appKeyChangeAccess Right	
EV1ApplicationKeySettings.	Builder boolean appKeySettings Changeable	
EV1ApplicationKeySettings.	Builder boolean appMasterKey Changeable	
EV1ApplicationKeySettings.	Builder boolean authenticationRequired ForDirectoryConfigurationData	
EV1ApplicationKeySettings.	Builder boolean authenticationRequired ForFileManagement	
EV1ApplicationKeySettings.	Builder boolean isoFileIdentifierPresent	
EV1ApplicationKeySettings.	Builder KeyType keyTypeOfApplication Keys	
EV1ApplicationKeySettings.	Builder int maxNumberOfApplication Keys	

Version 1.7:

NA

Version 1.6:

Table 16. Version 1.6 Removed/Obsolete APIs

Class	API	Alternative API
INTAG5	getNTAG5(CustomModules)	No alternative

Version 1.5:

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

NA

Version 1.4:

Table 17. Version 1.4 Removed/Obsolete APIs

Class	API	Alternative API
IDESFireEV2	void authenticateEV2NonFirst (int, IKey Data, byte[])	void authenticateEV2NonFirst(int, IKey Data)

Version 1.3:

Table 18. Version 1.3 Removed/Obsolete APIs

Class	API	Alternative API
IMIFAREIdentity	byte[] commitTranscation()	byte[] commitTransaction()
IDESFireEV2	byte[] getKeySetVersion()	byte[] getKeyVersionFromKeySet(), byte[] getAllKeySetVersion()

Version 1.2:

NA

Version 1.1:

Table 19. Version 1.1 Removed/Obsolete APIs

Class	API	Alternative API
IICodeDNA	readBuffer(), void challenge()	No alternative
IUltralightEV1	byte readVCSL()	No alternative
LibraryManager	void setLogger(final ILogger logger)	No alternative

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

8 Added class/APIs

Version 5.0.0:

Table 20. Version 5.0.0 Added class/APIs

Class	API Syntax	Description
MifareDUOX	setConfigurationFor PICCConfiguration(boolean useLegacy RandomID, boolean enableRandomID, boolean disableFormatCommand)	Set Configuration for PICC Configuration useLegacyRandomID - Enable legacy random ID compatible with DESFire EV1 (UID0 = 0x80). enableRandomID - Enable random ID. See Bit 5 for UID0 configuration. Once enabled,Random ID cannot be disabled anymore, but the UID0 configuration can still be adapted. disableFormat Command - Disable Format at PICC level only, once disabled cannot be enabled again.
MifareDUOX	setConfigurationForApplication Configuration(boolean disableFormat Command)	Set Configuration for Application Configuration
MifareDUOX	setConfigurationForDefaultKeys Update(byte[] piccAppDefaultKeyData, byte piccAppDefaultKeyVersion)	Set Configuration for Default Keys Update
MifareDUOX	setConfigurationForATSUpdate(byte[] userDefinedATS)	Set Configuration for ATS Update
MifareDUOX	setConfigurationForSAKUpdate(byte sak1, byte sak2)	Set Configuration for SAK Update
MifareDUOX	setConfigurationForSecureMessaging Configuration(boolean enable ISOSelectFileFastUIDRetrieval, boolean enableFreeAccessOver I2C, boolean enableEVCharging CardUnilateralAuth, boolean enable ISOSelectFileIntegratedECCBasedAut hentication, boolean disableChained Writing, CurveID curveID, byte app UIDEncKey)	Set Configuration for Secure Messaging Configuration
MifareDUOX	setConfigurationForCapabilityData(byte VCTID, byte PDCap1_2, byte PDCap2_ 2, byte PDCap2_5, byte PDCap2_6)	Set Configuration for Capability Data.
MifareDUOX	setConfigurationForVCIIDOrApplication ISODFNameRenaming(byte[] VCIIDor DFName, boolean isDelegated Application, byte[] DAMMac)	Set Configuration for VCIID or Application ISO DFName Renaming.
MifareDUOX	setConfigurationForATQAUpdate(byte[] userDefinedATQA)	Set Configuration for ATQA update.
MifareDUOX	setConfigurationFor NFCManagement(boolean disable ECDSASign, boolean disable ECCBasedCardUnilateralAuthenticati on,boolean disableECCBasedMutual AndReaderUnilateralAuthentication,	Set Configuration for NFC Management. Default value is all protocols supported in the manufacturing features selection map are enabled.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
	boolean disableAESBasedSymmetric Authentication)	
MifareDUOX	setConfigurationForI2 CManagement(byte i2cTarget,boolean disableECDSASign,boolean disableECCBasedCardUnilateral Authentication,boolean disable ECCBasedMutualAndReaderUnilateral Authentication,boolean disable AESBasedSymmetricAuthentication)	Set Configuration for I2C Management. Default value is all protocols supported in the manufacturing features selection map are enabled.
MifareDUOX	setConfigurationFor GPIOManagement(GPIO1Co nfiguration gpio1Configuration, GPIO2 Configuration gpio2Configuration,byte manageGPIOAccessCondition,byte readGPIOAccessCondition, byte[] AID,byte addManageGPIOAccess Condition,byte addReadGPIOAccess Condition)	Set Configuration GPIO Management
MifareDUOX	setConfigurationForCrypto APIManagement(CommunicationType commMode, byte accessCondition)	Set Configuration for Crypto API Management
MifareDUOX	manageGPIOForOutputMode(int GPIONo, boolean pauseOrRelease NFC,GPIOControl gpioControl, byte[] NFCPauseRespData,Communication Type communicationType)	Manages the GPIO output if GPIOx Mode is set to output.
MifareDUOX	manageGPIOForPowerDownstream Mode(int GPIONo,GPIO1Configuration. PowerDownstream power Downstream,boolean execute Measurement,boolean enablePower Downstream,CommunicationType communicationType)	Manages the GPIO output if GPIOx Mode is set to Power downstream.
MifareDUOX	manageGPIOForPowerDownstream ModeAutoWTX(int GPIONo, boolean enableAutoWTX,CommunicationType communicationType)	Manages the GPIO output if GPIOx Mode is set to Power downstream.
MifareDUOX	setConfigurationForAuthentication CounterAndLimitConfiguration(boolean enableAuthCtrOptionAuthEV2 First,byte[] authCtrLimit)	Set Configuration for Authentication Counter and Limit Configuration.
MifareDUOX	setConfigurationForHALTAndWake UpConfiguration(boolean enable GpioWakeUp,byte i2cWakeUp AddressByte,byte i2cSDAWakeUp Cycles,boolean enableI2CTarget AddressWakeUp,boolean enable I2CSDACycleWakeUp,boolean enableNFCFieldWakeUpFrom Halt,byte rdacSetting,boolean	Set Configuration for HALT and Wake-up Configuration.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
	enableGPIO2reset,boolean enable GPIO1reset)	
MifareDUOX	setConfigurationForLock Configurations(boolean lock PICCConfiguration, boolean lock DefaultKeysUpdate, boolean lock ATSUpdate,boolean lockSAKUpdate, boolean lockSecureMessaging Configuration,boolean lockCapability Data, boolean lockVCIIDOrApplication ISODFNameRenaming,boolean lockATQAUpdate, boolean lock NFCManagement, boolean lock I2CManagement,boolean lock GPIOManagement, boolean lock CryptoAPIManagement,boolean lock AuthenticationCounterAndLimitConfigu ration,boolean lockHALTAndWakeUp Configuration)	Set Configuration for Lock Configurations. Once a configuration is locked, it cannot be unlocked.
MifareDUOX	readGPIO(CommunicationType communicationType)	Returns the GPIO statuses
MifareDUOX	isoGetChallenge(final int expectedRnd Length)	The isoGetChallenge command generates a random with the given length P1 and P2 are always set to 0x00
MifareDUOX	writeData(final int fileNumber, final int offset, final byte[] data)	The writeData command allows to write data to a Standard Data File or a Standard Backup File.
MifareDUOX	writeData(final int fileNo, final int offset, final byte[] data,final Communication Type communicationSettings)	The writeData command allows to write data to a Standard Data File or a Standard Backup File.
MifareDUOX	authenticateEV2First(final int cardKey Number, final IKeyData key, final byte[] pCDcap2)	Authenticate EV2 to the PICC using single AES. After this authentication EV2 secure messaging is used. This authentication is intended to be the first in a transaction.
MifareDUOX	authenticateEV2NonFirst(final int card KeyNumber, final IKeyData key)	Non first is intended for any subsequent authentication after First authentication in a transaction
MifareDUOX	createApplication(final byte[] application ID, final EV3ApplicationKeySettings applicationSettings, final byte[] isoFile ID,final byte[] dfName)	Creates new applications on the PICC. The application is initialized according to the given settings. The application keys of the active key set are initialized with the Default Application Key. Note: PICC Application (AID 0) has to be selected in advance. Takes the extra parameters ISO file ID and directory file name to facilitate ISO 7816-4 operations.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOX	createApplication(final byte[] application ID, final EV3ApplicationKeySettings applicationSettings)	Creates new applications on the PICC. The application is initialized according to the given settings. The application keys of the active key set are initialized with the Default Application Key. Note: PICC Application (AID 0) has to be selected in advance.
MifareDUOX	credit(final int fileNumber, final int value, final CommunicationType communicationSettings)	The credit command allows to increase a value stored in a Value File. This command increases the current value stored in a file by a amount which is passes as a value parameter.Only positive values are allowed for the credit method.
MifareDUOX	credit(final int fileNumber, final int value)	The credit command allows to increase a value stored in a Value File. This command increases the current value stored in a file by a amount which is passes as a value parameter.Only positive values are allowed for the credit method.
MifareDUOX	debit(final int fileNumber, final int value)	The debit command allows decreasing a value stored in a Value File. the value parameter holds the "value" which is subtracted from the current value stored in the file. Only positive values are allowed for this method.
MifareDUOX	debit(final int fileNumber, final int value, final CommunicationType communicationSettings)	The debit command allows decreasing a value stored in a Value File. the value parameter holds the "value" which is subtracted from the current value stored in the file. Only positive values are allowed for this method.
MifareDUOX	limitedCredit(final int fileNumber, final int value)	The limitedCredit method allows a limited increase of a value stored in a Value File without having full Read and Write permission of the file.This feature can be enabled or disabled during value file creation. Note: The value for Limited Credit is limited to the sum of the Debit on this value file within the most recent transaction containing at least one Debit. After executing the limitedCredit method the new limit is set to zero (0) regardless of the amount which has been re-booked. Therefore the limited Credit method can only be used once after a Debit method is used.
MifareDUOX	limitedCredit(final int fileNumber, final int value,final CommunicationType communicationSettings)	The limitedCredit method allows a limited increase of a value stored in a Value File without having full Read and Write permission of the file.This feature can be enabled or disabled during value

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
		file creation. Note: The value for Limited Credit is limited to the sum of the Debit on this value file within the most recent transaction containing at least one Debit. After executing the limitedCredit method the new limit is set to zero (0) regardless of the amount which has been re-booked. Therefore the limited Credit method can only be used once after a Debit method is used.
MifareDUOX	deleteApplication(final int application ID, IKeyData piccDamMAcKey, final KeyType keyType)	Deletes delegated application of MifareDUOX. Authentication with KeyID.PICCDAMAuthKey or KeyID. NXPDAMAuthKey is required. Note: Application with ID 0 can't be deleted. This API is useful if you want to use integer form 3 byte application ID.
MifareDUOX	deleteApplication(final byte[] application ID, IKeyData piccDamMAcKey,final KeyType keyType)	Deletes delegated application of MifareDUOX. Authentication with KeyID.PICCDAMAuthKey or KeyID. NXPDAMAuthKey is required. Note: Application with ID 0 can't be deleted. This API is useful if you want to use integer form 3 byte application ID.
MifareDUOX	deleteApplication(final int applicationID)	Deletes an application. Depending on PICC master key setting, authentication with master key may be required Note: Application with ID 0 can't be deleted. This API is use full if you want to use integer form 3 byte application ID.
MifareDUOX	deleteApplication(final byte[] application ID)	Deletes an application. Depending on PICC master key setting, authentication with master key may be required Note: Application with ID 0 can't be deleted.
MifareDUOX	deleteFile(final int fileNumber)	Delete file method permanently deactivates a file within the file directory of the currently selected application.
MifareDUOX	format()	Format card - all applications and files will be deleted. Requires preceding authentication with PICC master key and selected application has to be 0.
MifareDUOX	formatT4T()	Formats the available free memory of the card (remaining memory after creating the NDEF File) as Type 4 Tag (T4T). Note: If the tag is already T4T formatted and T4T area size is needed to be changed, then format the tag to factory default and then use any of formatT4T() api.
MifareDUOX	getFreeMemory()	Returns the available bytes on the PICC.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOX	getAuthStatus()	current authentication status.
MifareDUOX	getCardDetails()	CardDetails object containing card information.
MifareDUOX	getCommandSet()	This API returns the current command set in use. - Native - ISO In ISO there is two modes one is ISO Wrapped mode and another is ISO Standard mode. Please Ensure you close and connect the connection before Switching from Native or ISO Wrapped mode to ISO Standard mode.
MifareDUOX	getDFName()	Returns the ISO/IEC 7816-4 DF-Names of all the active applications on PICC.
MifareDUOX	getFABIdentifier()	This API retrieves the FAB Identifier by sending getVersion and checking SW Minor version. If SW Minor Version 0x00, retrieve FAB ID from GetVersion part3 byte. If SW Minor Version 0x01, then send the command 0x60 0x01 to retreive FAB ID. Option byte 0x01 can only be set for products with SW Minor Version 0x01 onwards.
MifareDUOX	getFileIDs()	Returns all file IDs of the selected application. Authentication with master key may be required to do this operation
MifareDUOX	getISOFileIDs()	This method returns the 2 byte ISO/IEC 7816-4 File Identifiers of all active files within the currently selected application.
MifareDUOX	getReader()	Gets the underlying reader responsible for exchanging APDUs.
MifareDUOX	getValue(final int fileNumber)	The getValue command allows to read the currently stored value from Value Files.
MifareDUOX	getValue(final int fileNumber, final CommunicationType communication Settings)	The getValue command allows to read the currently stored value from Value Files.
MifareDUOX	getVersion()	This methods returns manufacturing related version data of the PICC.
MifareDUOX	isoSelectFileEFunderDF(final byte[] elementaryFileID)	This command implements the ISO/IEC 7816-4 compatible select command.
MifareDUOX	isoSelectMasterFile()	This command implements the ISO/IEC 7816-4 compatible select command and selects the master file.
MifareDUOX	isoSelectFile(final byte[] fileID, final byte selectionControl)	This command implements the ISO/IEC 7816-4 compatible select command. The registered ISO AID of MIFARE DESFire itself is "D2760000850100". When Selecting this application the

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
		MF is selected. This is done in order to ensure compliance with future micro controller cards having a MIFARE DESFire emulation inside.
MifareDUOX	isoSelectFile(final boolean is PICCMasterFile, final byte selection Control, final boolean isReturnFCI, final byte[] data)	This command implements the ISO/IEC 7816-4 compatible select command. The registered ISO AID of MIFARE DESFire itself is "D2760000850100". When Selecting this application the MF is selected. This is done in order to ensure compliance with future micro controller cards having a MIFARE DESFire emulation inside.
MifareDUOX	readData(final int fileNumber, final int offset, final int length)	The readData commands allows to read data from a Standard Data File or a Backup Data File.
MifareDUOX	readData(final int fileNumber, final int offset, final int length, final CommunicationType communication Settings, final int fileSize)	The readData commands allows to read data from a Standard Data File or a Backup Data File.
MifareDUOX	selectApplication(final int application1, final int application2)	selects application1 as primary application and application2 as secondary application
MifareDUOX	selectApplication(final byte[] application ID)	This method is used to select one specific application for further access.
MifareDUOX	selectApplication(final int application)	This method is used to select one specific application for further access. This method is useful if you want to use integer form of 3 byte application ID.
MifareDUOX	setCommandSet(CommandSet cmdSet)	This API Changes the Command set mode , Supported command set modes are - Native - ISO In ISO there is two modes one is ISO Wrapped mode and another is ISO Standard mode. Please Ensure you close and connect the connection beforeSwitching from Native or ISO Wrapped mode to ISO Standard mode.
MifareDUOX	ev2ChangeKey(final int keySetNumber, final int cardKeyNumber, final KeyType keyType, final byte[] oldKey, final byte[] newKey, final byte newKeyVersion)	Depending on the currently selected AID, this command updates a key of the PICC or of one specified application key set. NOTE: oldkey and newKey is taken as byte array instead of IKeyData.This is because changing the the Key in the card require actual key bytes. IKeyData represents the secure key object which may be in the secure environment [like HSM (Hardware secure module)] where we cant get the key contents always.
MifareDUOX	commitReaderId(final byte[] tMRI)	Commits a ReaderID for the ongoing transaction. This will allow a backend to

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
		identify the attacking merchant in case of fraud detected.
MifareDUOX	commitTransaction()	The commitTransaction command allows to validate all previous write access on Backup Data Files, Value Files and Record Files within one application.
MifareDUOX	commitAndGetTransactionMac()	This function commits the current ongoing transaction, if any operation in Transaction fails it roll backs.
MifareDUOX	abortTransaction()	The abortTransaction commands allows to invalidate all previous write access on backup Data Files, Value Files and Record Files within one application. This command is used to cancel a transaction without loosing the current authenticated status. Hence reducing the need for re-selecting and re-authentication of the application.
MifareDUOX	getApplicationIDs()	Returns the application identifiers of all active applications on a PICC. Authentication with master key may be required to run this API Note: PICC Application (AID 0) has to be selected in advance.
MifareDUOX	getFileCounters(int fileNumber)	The GetFileCounters command supports retrieving of the current values associated with the SDMReadCtr related with a StandardData file after enabling Secure Dynamic Messaging.
MifareDUOX	isoReadRecords(final byte record Number, final byte referenceControl, final int bytesToRead)	This command implements the ISO/ IEC 7816-4 compatible read records command.
MifareDUOX	isoAppendRecord(final byte record Number, final byte referenceControl, final byte[] data)	This command implements the ISO/ IEC 7816-4 compatible append record command,If a back up data file is used the MifareDUOX issues an implicit Commit transaction command. This ensures that every update of the binary file is treated as transaction.
MifareDUOX	writeRecord(final int fileNumber, final int offsetInRecord, final byte[] data)	The writeRecord command allows to write a record in a Cyclic or Linear Record File.
MifareDUOX	writeRecord(final int fileNumber, final int offsetInRecord, final byte[] data, final CommunicationType communication Settings)	The writeRecord command allows to write a record in a Cyclic or Linear Record File.
MifareDUOX	readRecords(final int fileNo, final int offsetRecords, final int noOfRecords)	The readRecords command allows to read out a set of complete records from a Cyclic or Linear Record File.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOX	readRecords(final int fileNumber, final int offsetRecords, final int numberOf Records, final int recordSize, final CommunicationType communication Settings, final int filesize)	The readRecords command allows to read out a set of complete records from a Cyclic or Linear Record File.
MifareDUOX	updateRecord(final int fileNumber, final int recordNo, final int offsetInRecord, final byte[] data)	The updateRecord command allows to update a record in a Cyclic or Linear Record File.
MifareDUOX	updateRecord(final int fileNumber, final int recordNo, final int offsetInRecord, final byte[] data, final Communication Type communicationSettings)	The updateRecord command allows to update a record in a Cyclic or Linear Record File.
MifareDUOX	clearRecordFile(final int fileNumber)	The clearRecordFile allows to reset a Cyclic or Linear Record File to the empty state. Need to call the Commit Transaction method to take effect of clearing Or AbortTransaction to invalidate the clearance.
MifareDUOX	clearRecordFile(final int file Number, final CommunicationType communicationSettings)	The clearRecordFile allows to reset a Cyclic or Linear Record File to the empty state. Need to call the Commit Transaction method to take effect of clearing Or AbortTransaction to invalidate the clearance.
MifareDUOX	getCardPlatForm()	This api will return the PLATFORM used for the cards
MifareDUOX	isoReadBinary(final int offset, final int bytesToRead)	This method implements the ISO/ IEC 7816-4 compatible read binary command. The read is done from the elementary file or directory file already selected
MifareDUOX	isoReadBinary(final byte[] fileID, final int offset, final int bytesToRead)	This method implements the ISO/ IEC 7816-4 compatible read binary command.
MifareDUOX	isoUpdateBinary(final byte[] fileID, final int offset, final byte[] data)	This command implements the ISO/ IEC 7816-4 compatible update binary command. Only 55 bytes of data is written at once from the given offset. if the Back up data file is used the MifareDUOX issues an implicit commit transaction command This ensures that every update of a binary file is treated as a transaction.
MifareDUOX	activateSecondaryApplication()	If two applications are selected, this API activates secondary application for future application operations
MifareDUOX	deActivateSecondaryApplication()	If two applications are selected, this API de-activates secondary application and activates primary application for future application operations

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOX	initializeKeySet(final byte keySet Number, final KeyType keyType)	Depending on the currently selected application, initialize the key set with specific index This command initializes the key set with KeySetNo as following: sets the type of the key set to the given KeySetType sets the key set version to 0x00 sets the content of all keys of the key set to the value and version of the corresponding key in the AKS (Active Key Set). If required due to KeySetType of different length, the corresponding key is padded with zero bytes or truncated to the targeted length. Parity bits of the source key are not adapted during this copy operation. If the PICC level has been selected, then Cmd.InitializeKeySet is rejected. If the SAI is set and no second application is selected, the command is rejected. The InitializeKeySet requires active authentication with KeyID.App ChangeKey or KeyID.AppMasterKey according to the AppKeySettings If Key SetNo is 0x00, the command is rejected as it is forbidden to initialize the AKS. A KeySetNo bigger than the number of key set in the application is also not accepted. If the targeted KeySetType is not supported by the application, the command is rejected. This happens if targeting KeyType.3TDEA, while the MaxKeySize set during application creation was only 16 bytes.
MifareDUOX	finalizeKeySet(final byte keySet Number, final byte keySetVersion)	Finalizes the application key set This command finalizes the key set targeted by KeySetNo by setting the key set version to the given KeySetVersion. The key set version value is chosen by the application owner and should be determined in accordance to the rules specified below. Once one or more key sets beside the AKS have been finalized within an application. The AKS can then be updated to point to another key set through a Cmd.RollKeySet. This action is called key set rolling. Rolling to a key set is restricted and has to fulfilled some rules specified as following. Key set version rule: The key set version of the key set targeted by RollKeySet has to be strictly bigger than the key set version of the AKS. Therefore it is also not possible to roll into key sets before they are finalized as these will always have key set version 0x00. Key set type rule: The key set targeted by Cmd.Roll

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
		KeySet has to have a key type stronger or equal than the AKS key type. If the preceding rules are fulfilled the key set rolling can take place. The AKS is updated to point to the targeted key set. The key set numbering of all key sets is changed relatively to the new AKS that receives key set number 0. The current authentication status is invalidated moving the PICC into VCState.DFNot Authenticated. Any ongoing transaction is aborted. Previous AKS and all key sets with key set version smaller than the new AKS are considered to be invalid targets for RollKeySet because they do not fulfill the key set version rule.
MifareDUOX	rollKeySet(final byte keySetNumber)	This command changes the AKS to point to the key set currently targeted with the given KeySetNo. After successful execution, the keys of the new AKS are from then on available for authentication. Any active authentication is canceled and the PICC is set to DFNotAuthenticated. Any ongoing transaction is aborted as if executing AbortTransaction. If the PICC level is selected, RollKeySet is rejected. If the SAI is set and no second application is selected, the command is rejected. Rules defined in initializeKey Set API description have to be fulfilled. Otherwise the command is rejected. The command is also rejected if the targeted key set is of type KeyType.3 TDEA, while EV1 secure messaging has been disabled for this application or at PICC level. RollKeySet needs an active authentication with the App RollKey, as configured in the AppKey SetSettings. If KeySetNo is 0x00 or bigger than the number of key set in the application, the command is not accepted.
MifareDUOX	setPICCFrameSize(final PICCFrame Size piccframesize)	Sets the desired PICC frame size to 256 or 64. Use this API to enable ISO chaining by setting PICCFrameSize to PICC_FRAME_SIZE_256. Use this API to enable Native chaining by setting PICCFrameSize to PICC_FRAME_ SIZE_64.
MifareDUOX	changeKeySettings(EV1KeySettings keySettings)	Depending on the currently selected AID, this command changes the PICCKeySettings of the PICC or the AppKeySettings of the application

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOX	changeKey(int keyNo, KeyType key Type, byte[] oldKey, byte[] newKey, byte keyVersion)	Depending on the currently selected AID, this command update a key of the PICC or of an application AKS
MifareDUOX	getDerivedNXPDAMKey(final IKeyData masterKey, final byte[] uid)	This API derives the NXP DAM key using the master key and UID.
MifareDUOX	proximityCheck(IKeyData vcProximity Key, int numOfIterations)	The API performs the 3 operations: Prepare Proximity Check: The PD is prepared to perform the Proximity Check by drawing a 7-byte random number RndR. Proximity Check: The PD answers with a prepared random number at the published response time in Cmd. PreparePC This command may be repeated up to 8 times splitting the random number for different time measurements. Verify Proximity Check: The random numbers are verified using cryptographic methods. Note: vcProximityKey can be NULL when authentication type is AuthType. AES/ authenticateEV2First or authenticate EV2NonFirst. vcProximityKey passed is not considered if in authenticated state in AuthType. AES/ authenticateEV2First or authenticateEV2NonFirst.
MifareDUOX	createDelegatedAppSymmetric Auth(final byte[] applicationID, final EV3 ApplicationKeySettings ev3Application Settings, final DelegatedApplication Settings delegatedApplicationSettings, final byte[] encryptionKey, final byte[] delegatedMac)	Creates delegated applications on the PICC with limited memory consumption. The application is initialized according to the given settings. The application keys of the active key set are initialized with the provided delegated initial key via encKey Note: PICC Application (AID 0) has to be selected in advance.
MifareDUOX	createDelegatedAppSymmetric Auth(final byte[] applicationID, final EV3 ApplicationKeySettings ev3Application Settings, final DelegatedApplication Settings delegatedApplicationSettings, final byte[] encryptionKey, final byte[] delegatedMac, final byte[] isoFileID, final byte[] dfName)	Creates delegated applications on the PICC with limited memory consumption. The application is initialized according to the given settings. The application keys of the active key set are initialized with the provided AppDAMDefault Key. Note: PICC Application (AID 0) has to be selected in advance. Takes the extra parameters ISO file ID and directory file name to facilitate ISO 7816-4 operations.
MifareDUOX	createDelegatedAppAsymmetric Auth(final byte[] applicationID, final EV3 ApplicationKeySettings ev3Application Settings, final DelegatedApplication Settings delegatedApplicationSettings, final byte[] initKey, final byte initKey Version)	Creates delegated applications on the PICC with limited memory consumption. The application is initialized according to the given settings. The application keys of the active key set are initialized with the provided AppDAMDefault Key. Note: PICC Application (AID 0) has to be selected in advance. Takes the extra parameters ISO file ID and

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
		directory file name to facilitate ISO 7816-4 operations.
MifareDUOX	createDelegatedAppAsymmetric Auth(final byte[] applicationID, final EV3 ApplicationKeySettings ev3Application Settings, final DelegatedApplication Settings delegatedApplicationSettings, final byte[] initKey, final byte initKey Version, final byte[] isoFileID, final byte[] dfName)	Creates delegated application in VCState.AuthenticatedECC
MifareDUOX	generateEncryptedDelegatedApp DefaultKey(final IKeyData encryption Key, final KeyType keyType, final byte[] defaultKey, final byte defaultKey Version)	Creates delegated applications in VCState.AuthenticatedECC
MifareDUOX	generateDelegatedAppMAC(final byte[] applicationID, final EV3Application KeySettings ev3ApplicationSettings, final DelegatedApplicationSettings delegatedApplicationSettings, final byte[] encryptionKey, final IKeyData delegatedAppMACKey, final KeyType keyType)	This helper method generates the delegated application initial encryption key.
MifareDUOX	generateDelegatedAppMAC(final byte[] applicationID, final EV3Application KeySettings ev3ApplicationSettings, final DelegatedApplicationSettings delegatedApplicationSettings, final byte[] encryptionKey, final IKeyData delegatedApplicationMACKey, final KeyType keyType, final byte[] isoFileID, final byte[] dfName)	Generates the MAC on user issued delegated application parameters. Takes the parameters ISO file ID and directory file name to facilitate ISO 7816-4 operations.
MifareDUOX	getKeyVersion(byte keyNo)	Gets the version of key with keyNo of selected application. Note: If application 0 has been selected, only key number 0 is valid.
MifareDUOX	getKeyVersionFromSpecificKeySet(byte keyNo, byte keySetNo)	Get the version of Key from specific keySet of selected application. Note: If application 0 has been selected, only key number 0 is valid.
MifareDUOX	getKeyVersionForVCProximityKey()	Gets the version of Application specific KeyID.VCProximityKey if available at application level.
MifareDUOX	getKeySetVersions()	Retrieves the all key set versions of selected application.
MifareDUOX	getDelegatedInfo(final short Delegated ApplicationSlotNo)	Gets the information about the delegated application at the given slot number.
MifareDUOX	cryptoRequestEcho(byte[] add DataBytes, CommunicationType communicationType)	Echos provided command data

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 20. Version 5.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOX	cryptoRequestECCSign_Initialize(byte algorithm, byte keyNo, byte[] inputData, CommunicationType communication Type)	Executes an ECC signature generation. Initialize
MifareDUOX	cryptoRequestECCSign_Update(byte[] inputData, CommunicationType communicationType)	Executes an ECC signature generation. Update
MifareDUOX	cryptoRequestECCSign_Finalize(byte[] inputData, CommunicationType communicationType)	Finalize get ECC Signature
MifareDUOX	cryptoRequestECCSign_OneShotWith RawData(byte algorithm, byte keyNo, byte[] inputData, CommunicationType communicationType)	Executes an ECC signature generation.
MifareDUOX	cryptoRequestECCSign_OneShot WithHash(byte algorithm, byte keyNo, byte[] inputData, CommunicationType communicationType)	Executes an ECC signature generation.
MifareDUOX	getKeySettings()	Get key settings allows to get the configuration information on the PICC and Application master key configuration settings.In addition it returns the Maximum number of keys which can be stored within the selected application.
MifareDUOX	getKeySettings(byte option)	This command retrieves the PICCKey Settings of the PICC, the AppKey Settings of the application, or meta-data of certain key types
MifareDUOX	getConfiguration()	Retrieves configuration aspects of the card or the application. Manufacturer configuration data is returned
MifareDUOX	getConfiguration(byte option)	Retrieves configuration aspects of the card or the application.

Version 4.1.0:

Table 21. Version 4.1.0 Added class/APIs

Class	API Syntax	Description
MifareDUOX	isoInternalAuthenticate(int privateKey No, byte[] rndA, byte[] optsA)	This API executes Asymmetric card-unilateral authentication.
MifareDUOX	isoGeneralAuthenticateFinal(int caRoot KeyNo, AuthMethod authMethod, byte[] certA, int certFileNo, int privateKeyNo, byte[] ePublicB, byte[] privateKeyA, byte[] caPublicKeyB, CurveID curveID)	This API executes Asymmetric mutual or reader-unilateral authentication. This is the first part initiating the authentication.
MifareDUOX	setConfigurationToEnableISOSelect FileECCBasedAuthentication(CurveID curveID)	Enable ISOSelectFile integrated ECC-based authentication. Internally uses SetConfiguration

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 21. Version 4.1.0 Added class/APIs*...continued*

Class	API Syntax	Description
		Option 0x04 and enables Bit 3 of SMConfigB byte.
MifareDUOX	setConfigurationForDUOX(byte option, byte[] data)	Configures several aspects of the card or the application
MifareDUOX	setConfigurationToEnableEVCharging CardUnilateralAuth()	Enable EV Charging card-unilatteral authentication and related commands. This option is only supported at application level. Internally uses Set Configuration Option 0x04 and enables Bit 4 of SMConfigB byte.
MifareDUOX	setConfigurationToEnableISOSelectFile FastUIDRetrieval(byte appUIDEncKey)	Enable ISOSelectFile fast UID retrieval. This option is only supported at application level. Internally uses Set Configuration Option 0x04 and enables Bit 6 of SMConfigB byte.
MifareDUOX	setConfigurationCrypto APIManagement(CommunicationType commMode, byte accessCondition)	Crypto API Management
MifareDUOX	exportKey(int keyNo, Communication Type commMode)	Exports the public key value of a KeyID. CARootKey
MifareDUOX	generateKeyPair(int keyNo, CurveID curveId, byte[] keyPolicy, byte write Access, byte[] kucLimit, Communication Type commMode)	Creates a private key entry by generating a key pair, the private key is securely stored on-card and the public key is returned using ManageKeyPair (0x46) command
MifareDUOX	importPrivateKey(int keyNo, CurveID curveId, byte[] keyPolicy, byte write Access, byte[] kucLimit, byte[] private Key, CommunicationType commMode)	Imports the provide private key using ManageKeyPair (0x46) command
MifareDUOX	updateMetaDataForKeyPair(int keyNo, CurveID curveId, byte[] keyPolicy, byte writeAccess,byte[] kucLimit, CommunicationType commMode);	Update only the metadata, that is, Key Policy and access rights, of an existing key entry using ManageKeyPair (0x46) command. This will not affect the current key value. For metadata update, also WriteAccess as configured for the targeted key entry is required. Command is rejected if the CurveID is not set to the curve associated with the current key.
MifareDUOX	getKeyPolicyBytes(boolean freeze KeyUsage, boolean unilateral Authentication,boolean mutual Authentication, boolean transaction Signature,boolean secureDynamic Messaging, boolean cryptoRequest);	Helper API to generate Key Policy bytes for ManageKeyPair API.
MifareDUOX	manageCARootKey(int keyNo, CurveID curveID, byte[] accessRights, byte writeAccess,byte readAccess, boolean enableCRL, byte crlFileNo, byte[] crl	Creates or updates a public key entry for storing a KeyID.CARootKey

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 21. Version 4.1.0 Added class/APIs*...continued*

Class	API Syntax	Description
	FileAID, byte[] publicKey,byte[] issuer, CommunicationType commMode);	
MifareDUOX	getAccessConditionsBytes(boolean AC00, boolean AC01, boolean AC02, boolean AC03,boolean AC04, boolean AC05, boolean AC06, boolean AC07,boolean AC08, boolean AC09, boolean AC0A, boolean AC0B,boolean AC0C, boolean AC0D);	Helper API to generate Key Policy bytes. An access condition can be associated with any key number (0x0 . . . 0xD), even if those do not exist as a symmetric key within the targeted application. The latter can still be obtained via an asymmetric authentication.
MifareDUOX	createFile(final int fileNumber, final MifareDUOXFile.DUOXFileSettings duoxFileSettings)	Creates a file within a Mifare DUOX application.
MifareDUOX	createFile(final int fileNumber, final byte[] isoFileID, final MifareDUOXFile. DUOXFileSettings duoxFileSettings);	Creates a file within a Mifare DUOX application. This API allows flexibility to use ISO file ID. This method also accepts the ISO file ID that is used in ISO 7816-4 operations.
MifareDUOX	getFileSettingsForMifareDUOX(int fileNo)	The getDuoxFileSettings command allows to get information on the properties of a specific DUOX file. The information provided by this command depends on the type of the file which is queried.
MifareDUOX	writeData(int fileNo, byte[] crlVersion, int offset, byte[] data, byte[] crlSignature)	The writeData command allows to write data to a Standard Data File or a Backup Data File.
MifareDUOX	writeData(int fileNo, byte[] crlVersion, int offset, byte[] data, byte[] crlSignature, CommunicationType communication Type)	The writeData command allows to write data to a Standard Data File or a Backup Data File.
MifareDUOX	doOriginalityCheck(File caFile)	Checks originality of the tag by reading ca certificate from the tag and validating against the CA file retrieved from NXP.
MifareDUOX	extractCAID(byte[] certificateBytes)	Helper API to extract CAID from certificate bytes read from the tag.
MifareDUOX	VDEWriteData(byte[] data)	The command VDE_WriteData, Writes data to FileType.StandardData with FileNo 0x01, and eventually lock the file. to lock the file use VDELock() API
MifareDUOX	VDELock()	The command VDELock, locks the File Type.StandardData with FileNo 0x01, and internally uses VDEWriteData API
MifareDUOX	VDEReadData(byte fileNum, byte[] expectedLength)	Reads data from FileType.Standard Data with FileNo 0x00 or 0x01.
MifareDUOX	changeMifareDUOXFileSettings(int fileNo, MifareDUOXFile.DUOXFile Settings duoxFileSettings)	This command changes the access parameters of an existing file. the Change only succeeds if the current

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 21. Version 4.1.0 Added class/APIs*...continued*

Class	API Syntax	Description
		access rights for "Change Access Rights" is different from never.
MifareDUOX	VDEECDSASign(byte[] data, byte[] expectedLength)	Generates and ECDSA signature over a 32-byte challenge.
MifareDUOXFile.DUOXFileSettings	toByteArray(final DUOXFileSettings filesettings)	gets the byte array for file settings.
MifareDUOXFile.DUOXFileSettings	getType()	return FileType Allowable file types
MifareDUOXFile.DUOXFileSettings	getComSettings()	return CommunicationType Allowable settings
MifareDUOXFile.DUOXFileSettings	getReadAccess()	return readAccess. Returned values are from 0x00 to 0xF
MifareDUOXFile.DUOXFileSettings	getWriteAccess()	return writeAccess. Returned values are from 0x00 to 0xF
MifareDUOXFile.DUOXFileSettings	getReadWriteAccess()	return readWriteAccess. Returned values are from 0x00 to 0xF
MifareDUOXFile.DUOXFileSettings	getChangeAccess()	return changeAccess. Returned values are from 0x00 to 0xF
MifareDUOXFile.DUOXDataFile Settings	toByteArray(final DUOXDataFile Settings fileSettings)	gets the byte array for file settings
MifareDUOXFile.DUOXDataFile Settings	getFileSize()	return file size of the data Files
MifareDUOXFile.DUOXDataFile Settings	getNumberOfAdditionalAccess Conditions()	Number of additional access conditions sets (not including the 1st and mandatory one) of the targeted file.
MifareDUOXFile.DUOXDataFile Settings	getNumberOfAdditionalAccessRights()	return numberOfAdditionalAccess Rights
MifareDUOXFile.DUOXDataFile Settings	isSDMEnabled()	Returns the boolean value Secure Messaging Value Enabled or Not
MifareDUOXFile.DUOXDataFile Settings	setSDMEnabled(boolean SDMEnabled)	Sets the value for Secure Messaging
MifareDUOXFile.DUOXDataFile Settings	isUIDMirroringEnabled()	return Returns the boolean value UID Mirroring Value Enabled or Not
MifareDUOXFile.DUOXDataFile Settings	setUIDMirroringEnabled(boolean UIDMirroringEnabled)	Sets the value for UID Mirroring
MifareDUOXFile.DUOXDataFile Settings	isSDMReadCounterEnabled()	return Returns the boolean value Read Counter Value Enabled or Not
MifareDUOXFile.DUOXDataFile Settings	setSDMReadCounterEnabled(boolean SDMReadCounterEnabled)	Sets the value for SDM Read Counter
MifareDUOXFile.DUOXDataFile Settings	isSDMReadCounterLimitEnabled()	Read Counter Limit Value Enabled or Not
MifareDUOXFile.DUOXDataFile Settings	setSDMReadCounterLimit Enabled(boolean SDMReadCounter LimitEnabled)	Sets the value for SDM Read Counter Link

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 21. Version 4.1.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOXFile.DUOXDataFile Settings	isSDMEncryptFileDataEnabled()	Encrypt Data File Value Enabled or Not
MifareDUOXFile.DUOXDataFile Settings	isGpioStatusEnabled()	is GPIO Status Enabled or Not
MifareDUOXFile.DUOXDataFile Settings	setSDMEncryptFileData Enabled(boolean SDMEncryptFileData Enabled)	Sets the value for SDM Encrypt File Data
MifareDUOXFile.DUOXDataFile Settings	setGpioStatusEnabled(boolean gpio StatusEnabled)	Sets the value for GPIO Status Enabled
MifareDUOXFile.DUOXDataFile Settings	getSdmAccessRights()	SDM Access Rights.
MifareDUOXFile.DUOXDataFile Settings	setSdmAccessRights(byte[] sdmAccess Rights)	
MifareDUOXFile.DUOXDataFile Settings	getUidOffset()	Offset of mirrored UID within the file (0x00...(FileSize - UID Length)).
MifareDUOXFile.DUOXDataFile Settings	setUidOffset(byte[] uidOffset)	
MifareDUOXFile.DUOXDataFile Settings	getSdmReadCounterOffset()	Offset of mirrored SDM read counter within the file. (0x00 to (FileSize - SDMReadCtrLength)); FFFFFFh if no SDMReadCounter mirroring
MifareDUOXFile.DUOXDataFile Settings	setSdmReadCounterOffset(byte[] sdm ReadCounterOffset)	
MifareDUOXFile.DUOXDataFile Settings	getPiccDataOffset()	Offset of mirrored PICC data within the file (0x00...(FileSize - PICC data Length)).
MifareDUOXFile.DUOXDataFile Settings	setPiccDataOffset(byte[] piccData Offset)	
MifareDUOXFile.DUOXDataFile Settings	getGpioStatusOffset()	
MifareDUOXFile.DUOXDataFile Settings	setGpioStatusOffset(byte[] gpioStatus Offset)	
MifareDUOXFile.DUOXDataFile Settings	getSdmMacInputOffset()	Offset in the file where the SDM MAC computation starts (0x00...SDMMACOffset).
MifareDUOXFile.DUOXDataFile Settings	setSdmMacInputOffset(byte[] sdmMac InputOffset)	
MifareDUOXFile.DUOXDataFile Settings	getSdmEncryptionOffset()	Offset of SDMENCFileData mirror position (SDMMACInputOffset... (SDMMACOffset-32)).
MifareDUOXFile.DUOXDataFile Settings	setSdmEncryptionOffset(byte[] sdm EncryptionOffset)	
MifareDUOXFile.DUOXDataFile Settings	getSdmEncryptionLength()	Length of the SDMENCFileData (32... (SDMMACOffset-SDMENCOffset)).

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 21. Version 4.1.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOXFile.DUOXDataFile Settings	setSdmEncryptionLength(byte[] sdm EncryptionLength)	
MifareDUOXFile.DUOXDataFile Settings	getSdmMacOffset()	Offset of SDMMAC mirror position.
MifareDUOXFile.DUOXDataFile Settings	setSdmMacOffset(byte[] sdmMacOffset)	
MifareDUOXFile.DUOXDataFile Settings	getSdmReadCounterLimit()	SDMReadCtrLimit value.
MifareDUOXFile.DUOXDataFile Settings	setSdmReadCounterLimit(byte[] sdm ReadCounterLimit)	
MifareDUOXFile.StdDataFileSettings DUOX	toByteArray(final StdDataFileSettings DUOX fileSettings)	This method generates byte array for creating standard data file based on the input file settings
MifareDUOXFile.CRLFileSettings	getFileSize()	file size of the data Files.
MifareDUOXFile.CRLFileSettings	getNumberOfAdditionalAccess Conditions()	Number of additional access conditions sets (not including the 1st and mandatory one) of the targeted file.
MifareDUOXFile.CRLFileSettings	getNumberOfAdditionalAccessRights()	Sets of access conditions to update the 2nd up to (numberOfAdditionalAccess Conditions+1)th set of the file.
MifareDUOXFile.CRLFileSettings	isCRLSignMonotonicCRLVersion Required()	
MifareDUOXFile.CRLFileSettings	setCRLSignMonotonicCRLVersion Required(boolean CRLSignMonotonic CRLVersionRequired)	
MifareDUOXFile.CRLFileSettings	isCRLVersionRequiredToIncrement By1()	
MifareDUOXFile.CRLFileSettings	setCRLVersionRequiredToIncrement By1(boolean CRLVersionRequiredTo IncrementBy1)	
MifareDUOXFile.CRLFileSettings	isCertificateRevocationDisabled()	
MifareDUOXFile.CRLFileSettings	setCertificateRevocation Disabled(boolean certificateRevocation Disabled)	
MifareDUOXFile.CRLFileSettings	getCSNSize()	
MifareDUOXFile.CRLFileSettings	setCSNSize(byte CSNSize)	
MifareDUOXFile.CRLFileSettings	getCRLSignatureKey()	
MifareDUOXFile.CRLFileSettings	setCRLSignatureKey(byte CRLSignatureKey)	
MifareDUOXFile.CRLFileSettings	toByteArray(final CRLFileSettings file Settings)	
MifareDUOXFile.ValueFileSettings	toByteArray(final MifareDUOXFile.Value FileSettings filesettings)	gets the byte array for file settings.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 21. Version 4.1.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOXFile.ValueFileSettings	getNumberOfAdditionalAccess Conditions()	Number of additional access conditions sets (not including the 1st and mandatory one) of the targeted file.
MifareDUOXFile.ValueFileSettings	getNumberOfAdditionalAccessRights()	Sets of access conditions to update the 2nd up to (numberOfAdditionalAccess Rights+1)th set of the file.
MifareDUOXFile.ValueFileSettings	isLimitedCreditValueEnabled()	true if limited credit is enabled , otherwise false.
MifareDUOXFile.ValueFileSettings	isGetFreeValueEnabled()	true if get free value is enabled , otherwise false.
MifareDUOXFile.ValueFileSettings	getUpperLimit()	upper limit of the value file.
MifareDUOXFile.ValueFileSettings	getLowerLimit()	Lower limit of the value in value file.
MifareDUOXFile.ValueFileSettings	getInitialValue()	initial value in the file.
MifareDUOXFile .RecordFileSettings	toByteArray(final MifareDUOXFile. RecordFileSettings filesettings)	gets the byte array for file settings.
MifareDUOXFile .RecordFileSettings	getCurrentNumberOfRecords()	No of actual records in record file.
MifareDUOXFile .RecordFileSettings	getMaxNumberOfRecords()	Maximum number of records can be stored in this file.
MifareDUOXFile .RecordFileSettings	getRecordSize()	Size in bytes of each record.
MifareDUOXFile .RecordFileSettings	getNumberOfAdditionalAccess Conditions()	Number of additional access conditions sets (not including the 1st and mandatory one) of the targeted file.
MifareDUOXFile .RecordFileSettings	getNumberOfAdditionalAccessRights()	Sets of access conditions to update the 2nd up to (numberOfAdditionalAccess Rights+1)th set of the file.
MifareDUOXFile .LinearRecordFile Settings	toByteArray(final DESFireFile.Record FileSettings filesettings)	gets the byte array for file settings.
MifareDUOXFile .CyclicRecordFile Settings	toByteArray(final MifareDUOXFile. RecordFileSettings filesettings)	gets the byte array for file settings.
MifareDUOXFile .DUOXTransaction MacFileSettings	getTMKeyVersion()	
MifareDUOXFile .DUOXTransaction MacFileSettings	setTMKeyVersion(byte tmKeyVersion)	
MifareDUOXFile .DUOXTransaction MacFileSettings	isTMIExclFileMapEnabled()	
MifareDUOXFile .DUOXTransaction MacFileSettings	setTMIExclFileMapEnabled(boolean TMIExclFileMapEnabled)	
MifareDUOXFile .DUOXTransaction MacFileSettings	isTransactionMacMode()	
MifareDUOXFile .DUOXTransaction MacFileSettings	setTransactionMacMode(boolean transactionMacMode)	
MifareDUOXFile .DUOXTransaction MacFileSettings	isTransactionSignatureMode()	

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 21. Version 4.1.0 Added class/APIs*...continued*

Class	API Syntax	Description
MifareDUOXFile .DUOXTransaction MacFileSettings	setTransactionSignatureMode(boolean transactionSignatureMode)	
MifareDUOXFile .DUOXTransaction MacFileSettings	getTransactionSignatureKeyNo()	
MifareDUOXFile .DUOXTransaction MacFileSettings	setTransactionSignatureKeyNo(byte transactionSignatureKeyNo)	
MifareDUOXFile .DUOXTransaction MacFileSettings	getTMIExcelFileMap()	
MifareDUOXFile .DUOXTransaction MacFileSettings	setTMIExcelFileMap(byte[] TMIExcel FileMap)	
MifareDUOXFile .DUOXTransaction MacFileSettings	getKeyType()	
MifareDUOXFile .DUOXTransaction MacFileSettings	setKeyType(KeyType keyType)	
MifareDUOXFile .DUOXTransaction MacFileSettings	isTMCLimitEnabled()	
MifareDUOXFile .DUOXTransaction MacFileSettings	setTMCLimitEnabled(boolean TMCLimit Enabled)	
MifareDUOXFile .DUOXTransaction MacFileSettings	getTmcLimit()	
MifareDUOXFile .DUOXTransaction MacFileSettings	setTmcLimit(byte[] tmcLimit)	
MifareDUOXFile .DUOXTransaction MacFileSettings	toByteArray(final MifareDUOXFile. DUOXTransactionMacFileSettings file Settings)	
MifareDUOXFileSettingsHelper	toByteArray(final MifareDUOXFile. DUOXFileSettings fileSettings)	
MifareDUOXFileSettingsHelper	generateFileSettingBitMapForMifare DUOX(byte[] fileSettingBitMap, Mifare DUOXFile.DUOXFileSettings file Settings)	
MifareDUOXFileSettingsHelper	retrieveAdditionalAccessRights Info(byte[] fileSettingsBitMap, int index)	
MifareDUOXFileSettingsHelper	parse(byte[] rawFileSetting)	
MifareDUOXFileSettingsHelper	getDuoxTransactionMacFile Settings(byte[] rawFileSetting)	

Version 4.0.0:

Table 22. Version 4.0.0 Added class/APIs

Class	API Syntax	Description
DESFireFactory	getMifareDUOX	This API Returns a Mifare DUOX Object
DESFireFactory	isCardMifareDUOX	This API Request to fetch MIFARE DUOX Status

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 22. Version 4.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
DESFireUtil	decodeASN1AndExtractPublic Key(byte[] asn1Data)	
DESFireUtil	decodeASN1AndExtractCert Signature(byte[] asn1Data)	
MifareDUOX	doOriginalityCheck()	This API allows to Check Originality of the card
MifareDUOX	retrieveSignatureB()	
IICode	inventory	This method to be used when AFI flag is not set and AFI value is not required in parameters
ICode3	getTotalMemory()	To get total memory
ICode3	getUserMemory()	To get user memory
ICode3	getConfigMemory()	To get Configuration memory
ICode3	fastReadMultipleBlocks()	To Read data from multiple blocks
ICode3	readSingleBlock()	To Read data from Single block
ICode3	writeSingleBlock()	To Write data from Single block
ICode3	readMultipleBlocks()	To Write data from multiple blocks
ICode3	readConfigurationBlock()	To Read Configuration block
ICode3	readProtectionForPageL()	To Read Protect Page L
ICode3	writeProtectionForPageL()	To Write Protect Page L
ICode3	lockPointerControlForPageL()	To Lock Pointer control for page L
ICode3	readProtectionForPageH()	To Read Protect Page H
ICode3	writeProtectionForPageH()	To Write Protect Page H
ICode3	lockPointerControlForPageH()	To Lock Pointer control for page H
ICode3	setPasswordConfigMemory()	To set Password For Configuration Memory
ICode3	isT5T()	
ICode3	disableNFCMirror()	To Disable NFC mirror
ICode3	enableUIDMirror()	To Enable UID mirror
ICode3	enableNFCCounterMirrorAnd UIDMirror()	To Enable NFC Counter Mirror and UID Mirror
ICode3	setProtectionPointer()	To Set Protection Pointer
ICode3	disableLockBlock()	To Disable Lock Block
ICode3	getMultipleBlockSecurityStatus()	To get Security Status of Multiple Blocks
ICode3	setNFCMirrorStartBlock()	To set NFC mirror Start Block
ICode3	setNFCMirrorStartByte()	To set NFC mirror Start Byte
ICode3	writePasswordConfigurationMemory()	To Write Password for Configuration Memory

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 22. Version 4.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
ICode3	writeSignature()	To write Signature
ICode3	setOriginalitySignatureLengthDefault()	To set default Originality check signature length
ICode3	enableCommandBasedCounterMode()	To enable command based counter mode
ICode3	enableNFCCounterMode()	To enable NFC Counter Mode
ICode3	writeCIDBytes()	To Write CID Bytes
ICode3	resetCounter()	To Reset Counter
ICode3	pickRandomId()	To Pick Random ID
ICode3	setPrivacyMode()	To set Privacy Mode
ICode3	checkPrivacyMode()	To Check Privacy mode
ICode3	lockPasswordConfigMemory()	To Lock Password Configuration Memory
ICode3	disableWriteConfig()	To disable Write Configuration
ICode3	lockLockPointerBytes()	To Lock LockPointer Bytes
ICode3	enableConfigPasswordProtection()	To enable Configuratioon Password Protection
ICode3	lockPageProtectionBytes()	To Lock Page Protection Bytes
ICode3	lockSignature()	To Lock Signature
ICode3	lockEAS()	To Lock EAS
ICode3	getEasStatus()	To get EAS Status
ICode3	passwordProtectAFI()	To Password Protect AFI
ICode3	passwordProtectEas()	To Password Protect EAS
ICode3	lockCIDBytes()	To Lock CID Bytes
ICode3	disableCounterPreset()	To disable counter Preset
ICode3	lockBlock()	To lock Block
ICode3	easAlarm()	To set EAS Alarm
ICode3	setNFCRetryMode()	To set NFC Retry mode
ICode3	inventoryRead()	To read Inventory
ICode3	fastInventoryRead()	To Read Inventory in faster datarate
ICode3TT	enableNFCCounterUIDTagTamperMsg Mirror()	To Enable NFCCounter,UIDTagTamper Message Mirror
ICode3TT	enableNFCCounterUIDTagTamperMsg TagTamperStatusMirror()	To Enable NFCCounter,UIDTagTamper MsgTagTamperStatusMirror
ICode3TT	enableStoreTagTamper()	To Enable Store Tag Tamper
ICode3TT	readTT()	To read TagTamper message
ICode3TT	showTTStatus()	To Show the TagTamper Message
ICode3TT	lockTTOpenMsg()	To Lock TagTamper Open Message

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 22. Version 4.0.0 Added class/APIs*...continued*

Class	API Syntax	Description
ICodeFactory	getICode3()	To get ICODE3 details
ICodeFactory	getICode3TT()	To get ICODE3TT details
ICodeFactory	isTagICode3()	To verify whether Card is ICODE3
ICodeFactory	sTagICode3TT()	To verify whether Card is ICODE3TT
ICodeUtility	isAFIFlagSet()	To verify whether AFI flag is set
ICodeUtility	validateDataRateFlagSetOnly()	To validate DataRateFlagSetOnly
ICodeUtility	validateBlockNum()	To Validate Block Number
ICodeUtility	validateGetMultipleBlockSecurityStatus Params()	To Validate GetMultipleBlock Security Status Parametres
ICodeUtility	validateByteNumber()	To Validate Byte Number
ICodeUtility	validateCounterValue()	To Validate Counter Value
ICodeUtility	validateBlockNumForMultipleRead()	To Validate Block Number For Multiple Read
ICodeUtility	inventoryCommandValidations()	To Validate Inventory command
ICodeUtility	updateBitValueAtPosition()	To update Bit Value at Position
ICodeUtility	getBitValueAtPosition()	To get Bit Value at Position
ICodeUtility	verifyCIDBytesLength()	To verify CID Bytes Length
ICodeUtility	validateOriginalitySignatureLength()	To Validate Originality Signature Length
ICodeUtility	validateOptionFlagNotSet()	To Validate whether Option Flag Not set
ICodeUtility	validateOptionFlagSet()	To Validate whether Option Flag set
ICodeUtility	validateNullValue()	To validate Null Value
ICodeUtility	validateIfCidCompareEnabled()	To Validate If CID Compare is enabled
ICodeUtility	validateTTStatusCondition()	To Validate Tag Tapmer status condition

Version 3.1.0:

Table 23. Version 3.1.0 Added class/APIs

Class	API Syntax	Description
IPlusEV1	void authenticateFirst(int, IK eyData, byte[])	This API Performs first authenticate
IPlusEV1	void authenticateNonFirst(i nt, IKey Data)	Performs non first authenticate
DESFireUtil	void checkIfApplicationSelec ted(int)	Check if application selected, throw exception if no application is selected
DESFireUtil	int updateKeyNumOrFileNu mIf SAIEnabled(int, int, int)	Update Key Number or File Number if Secondary Application is active
IDESFireEV3	String getFABIdentifier()	This API retrieves the FAB Identifier by sending getVersion and checking SW Minor version.
IPlusEV1SL3	void proximityCheck(IKeyDa ta, int)	Prepare, perform and verify Proximity Check

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 23. Version 3.1.0 Added class/APIs*...continued*

Class	API Syntax	Description
IPlusEV1SL1	byte[] readSignature()	Retrieve the ECC originality check signature
IPlusEV1SL1	byte[] readTransactionMac(i nt)	Reads the TransactionMac data from TransactionMACBlock
IPlusEV1SL1	void resetAuth()	Resets the authentication session Performs a Reset Auth command
IPlusEV1SL1	int sectorNumberToBlockNu mberFor AESKeys(byte)	Helper to convert from sector number to block number which store corresponding AES 128 keys
IPlusEV1SL1	boolean vcSupportLastISOL3 (byte[], byte[])	Check if the Virtual Card Concept is supported, communicate PCD capabilities and retrieve the UID This command is supported only in Layer 3

Version 3.0.0:

Table 24. Version 3.0.0 Added class/APIs

Class	API Syntax	Description
NTAG22xTagTamperStatus	byte[] getCounterValue2_ext()	
NTAG22xTagTamperStatus	byte getCttCfilt()	
NTAG22xTagTamperStatus	byte getCttCurrTrim()	
NTAG22xTagTamperStatus	byte[] getDifferenceMeasurement	
NTAG22xTagTamperStatus	byte getMeasDblRange()	
UltralightAES	void setNegativeAuthLimit(int)	This API allows to set a negative authentication limit value {(0 < limit < 1022)}

Version 2.0:

Table 25. Version 2.0 Added class/APIs

Class	API Syntax	Description
IDESFireEV1	void isoExternalAuthenticate(int, Key Type)	Authenticate the PCD (second part of ISO7816-4 authentication) If the previous command was not ISOGet Challenge, isoExternalAuthenticate command will be rejected
IDESFireEV1	void isoGetChallenge(KeyType, IKey Data)	Get a challenge (first part of ISO7816-4 authentication) Once the ISO/IEC7816-4 authentication has been initiated, it must be completed with ISOExternalAutheticate and ISOInternal Authenticate
IDESFireEV1	void isoInternalAuthenticate(int, Key Type)	Authenticate the PICC (third part of ISO7816-4 authentication) If the previous command was not Cmd. ISOExternalAutheticate, isoInternal Authenticate command will be rejected

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 25. Version 2.0 Added class/APIs*...continued*

Class	API Syntax	Description
IDESFireEV2	CARDPLATFORM getCardPlatForm()	This API will return the PLATFORM used for DESFIRE cards
IDESFireEV2	void isoSelect(byte[])	Selects the given IID or DF name as per ISO 7816 terminology Use this API if the AuthVCMandatory is set to false which means the response of the data is not expected and no need to call Iso ExternalAuthenticate api
IDESFireEV2	void updateRecord(int, int, int, byte[])	The updateRecord command allows to update a record in a Cyclic or Linear Record File
DESFireUtil	boolean isValidResponse(byte[], CommandSet)	Check if it is valid Response or not

Version 1.9:

Table 26. Version 1.9 Added class/APIs

Class	API Syntax	Description
IDESFireEV3C	void changeDESFireEV3CFile Settings(int, EV3CFileSettings)	This API changes the access parameters of an existing file
IDESFireEV3C	void changeToMFCStateKilled()	This methods changes the MiFare Classic Support State Unlocked to Killed
IDESFireEV3C	byte[] computeMFCLicenseMAC(byte[], byte[], IKeyData)	This methods changes the MiFare Classic Support State Unlocked to Killed
IDESFireEV3C	byte[] constructMFCLicense(EV3 CMFCMappingBlockSettings[])	This methods constructs the MFC License
IDESFireEV3C	void createFile(int, byte[], EV3CFile Settings)	Creates a file within a DESFire EV3C application
IDESFireEV3C	void createFile(int, EV3CFileSettings)	Creates a file within a DESFire EV3C application
IDESFireEV3C	void createMFCMapping(int, EV3 CMFCMappingFileSettings, EV3 CMFCMappingBlockSettings[], byte[], byte[])	This methods creates Mifare Classic Mapping in the file
IDESFireEV3C	void enableMFCLicenseMACKey(byte[], byte[])	This methods enables MFC License MAC Key
IDESFireEV3C	EV3CFileSettings getDESFireEV3CFile Settings(int)	This API gets the information on the properties of a specific DESFireEV3C file
IDESFireEV3C	byte[] getFileCounters(int)	This API retrieves the current values associated with the SDMReadCtr related with a StandardData file after enabling Secure Dynamic Messaging
IDESFireEV3C	EV3CMFCState getMifareClassicCard State()	This methods checks card for the MiFare Classic Support State Unlocked or Killed

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 26. Version 1.9 Added class/APIs*...continued*

Class	API Syntax	Description
IDESFireEV3C	void restoreTransfer(int, int, boolean, boolean)	This methods copies the value from one FileType.Value file to another
IDESFireEV3C	void restrictMFCUpdate(EV3 CMFCMappingBlockSettings[], byte[], byte[])	This methods restricts Mifare Classic Update in the Classic Blocks
IDESFireEV3C	void restrictMFCUpdate(EV3 CMFCValuePairConfiguration[], byte[], byte[])	This methods restricts Mifare Classic Update in the Classic Blocks
IPlusEV1SL3	byte[] commitReaderIDWith TMRIPrev(int, byte[])	Commits a ReaderID for the ongoing transaction
IPlusSL3	byte[] decrementTransferForTmcTmv Value(boolean, int, int, int)	Decrement followed by a transfer on a value block
IPlusSL3	byte[] incrementTransferForTmcTmv Value(boolean, int, int, int)	Increment followed by transfer on a value block
IPlusSL3	byte[] transferForTmcTmv Value(boolean, int)	Transfers the contents of the transfer buffer to the specified address
IPlusSL3	byte[] writeForTmcTmv(WriteMode, int, byte[])	Writes the data to the given block number
Crypto	IAsymmetricCryptoGram get ECDSASecp192CryptoGram()	
DESFireUtil	boolean isCommunicationTypePlainIfm IsAuthenticated(EV3CFileSettings)	Is Communication type plain authenticated
DESFireUtil	boolean isCommunicationTypePlainIfm IsNotAuthenticated(EV3CFileSettings)	Is Communication type plain not authenticated
NTagFactory	INTAG223DNA getNTAG223 DNA(CustomModules)	@param supportModules Custom Modules object holds all the user customized or library default modules for sharing across the card objects
NTagFactory	INTAG223DNAStatusDetect get NTAG223DNAStatusDetect(Custom Modules)	@param supportModules Custom Modules object holds all the user customized or library default modules for sharing across the card objects
NTagFactory	INTAG224DNA getNTAG224 DNA(CustomModules)	@param supportModules Custom Modules object holds all the user customized or library default modules for sharing across the card objects
NTagFactory	INTAG224DNAStatusDetect get NTAG224DNAStatusDetect(Custom Modules)	@param supportModules Custom Modules object holds all the user customized or library default modules for sharing across the card objects
NTagFactory	boolean isTagNTAG223DNA(Custom Modules)	helper method to check if the tag under operation is NTAG223DNA or not
NTagFactory	boolean isTagNTAG223DNAStatus Detect(CustomModules)	helper method to check if the tag under operation is NTAG223DNA Status Detect or not
NTagFactory	boolean isTagNTAG224DNA(Custom Modules)	helper method to check if the tag under operation is NTAG224DNA or not

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 26. Version 1.9 Added class/APIs*...continued*

Class	API Syntax	Description
NTagFactory	boolean isTagNTAG224DNAStatus Detect(CustomModules)	helper method to check if the tag under operation is NTAG224DNA Status Detect or not
UltralightFactory	UltralightAES getUltralightAES(Custom Modules)	@param customModules Custom Modules object holds all the user customized or library default custom Modules for sharing across the card objects
UltralightFactory	boolean isTagUltralightAES(Custom Modules)	helper method to check if the tag under operation is UltralightAES or not
IDESFireEV3	void proximityCheckEV3(IKeyData, int)	The API performs the 3 operations: 1.Prepare Proximity Check: The PD is prepared to perform the Proximity Check by drawing a 7-byte random number RndR. 2.Proximity Check: The PD answers with a prepared random number at the published response time in Cmd. PreparePC This command may be repeated up to 8 times splitting the random number for different time measurements. 3.Verify Proximity Check: The random numbers are verified using cryptographic methods.
IDESFireEV3	void updateRecord(int, int, int, byte[])	The updateRecord command allows to update a record in a Cyclic or Linear Record File
IDESFireEV3	void updateRecord(int, int, int, byte[], CommunicationType)	The updateRecord command allows to update a record in a Cyclic or Linear Record File
DESFireFactory	IDESFireEV3 getDESFireEV3(Custom Modules)	Returns a DESFire EV3 Object
DESFireFactory	boolean isCardDESFireEV3(byte[])	Fetch DESFire EV3 Status
DESFireUtil	boolean checkCommand(DESFire Command)	Check commands
DESFireUtil	boolean isCommunicationTypePlainIfm IsAuthenticated(FileSettings)	Is Communication type plain authenticated
DESFireUtil	boolean isCommunicationTypePlainIfm IsNotAuthenticated(FileSettings)	Is Communication type plain not authenticated

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Version 1.8:

Table 27. Version 1.8 Added class/APIs

Class	API Name	Description
DESFireFactory	IDESFireEV3 getDESFireEV3(CustomModules)	Returns a DESFire EV3 Object.
	boolean isCardDESFireEV3(byte[])	Fetch DESFire EV3 Status
DESFireUtil	boolean checkCommand(DESFireCommand)	Check commands
	boolean isCommunicationTypePlainIfmIsAuthenticated(FileSettings)	Is Communication type plain authenticated
	boolean isCommunicationTypePlainIfmIsNotAuthenticated(FileSettings)	Is Communication type plain not authenticated

Version 1.7:

NA

Version 1.6:

Table 28. Version 1.6 Added class/APIs

Class	API Syntax	Description
IPlusEV1SL0	void authenticateNonFirst(int, IKeyData)	Performs non first authenticate
IPlusSL0	void authenticateNonFirst(int, IKeyData)	Performs non first authenticate
IPlusSL3	void authenticateNonFirst(int, IKeyData)	Performs non first authenticate
IDESFireEV1PredictableChallenge	byte[] getAuthenticationCounter()	Return current authentication counter value
IPlusEV1	void isoSelect(byte[], IKeyData)	Selects the given IID or DF name as per ISO 7816 terminology
LibraryManager	boolean isRegistered()	Returns true if registered
LibraryManager	void registerJavaApp(String)	Registers java app using local path for the license file generated during application registration

Version 1.5:

Table 29. Version 1.5 Added class/APIs

Class	API Syntax	Description
INTag210	boolean isPwdAuthenticated()	Returns true if the password is authenticated
INTag213215216	void setMemProtectionAndPwd VerificationForReadWriteAccess(byte, boolean)	This method is used to set AUTH0 byte (page address from which password verification is required) and PROT bit (defines the memory protection, read or read and write)
INTag213215216	void setPwdProtectionForReadWrite Access(boolean)	This method is used to set PROT bit (defines the memory protection, read or read and write)

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 29. Version 1.5 Added class/APIs*...continued*

Class	API Syntax	Description
INTag213215216	void setStartPageAddressForPwd Auth(byte)	This method is used to set AUTH0 byte (page address from which password verification is required)
INTag213TagTamper	boolean isPwdAuthenticated()	return true if the password is authenticated
INTag413DNA	boolean doAsymmetricOriginality Check(byte[])	Performs the Asymmetric originality check
INTag413DNA	byte[] readData(int, int, int, CommunicationMode, int)	The readData commands allows to read data from a Standard Data File
INTag413DNA	void writeData(int, int, byte[], CommunicationMode)	The writeData command allows to write data to a Standard Data File or a Standard Backup File
IUtility	String byteToHexString(byte[], String)	Converts the byte array to Hex String

Version 1.4:

Table 30. Version 1.4 Added class/APIs

Class	API Syntax	Description
DESFireEV1PredictableChallenge	authenticateHelper (int cardkeyNo, Auth Type auth, KeyType type, IKeyData key Info)	Helper method to authenticate in predictable challenge flow
DESFireEV1PredictableChallenge	sendCommandAndGenerateRandom NumberB (byte [] cmd, AuthType authtype, KeyType keyType, IKeyData keyInfo)	Method to generate Random B based on Derived data for Predictable Challenge
DESFireEV1PredictableChallenge	generateCMACForNative (final byte[] data)	Method to generate CMA for Native auth type
DESFireEV1PredictableChallenge	generateCMACForISO (final byte [] data)	Method to generate CMA for ISO auth type
DESFireEV1PredictableChallenge	generateCMACForAES (final byte [] data)	Method to generate CMA for AES auth type
NTag413DNA	getKeyVersion (int iKeyNo)	Depending on the currently selected AID and given key number parameter, return key version of the key targeted
LibraryManager	getTapLinxVersion()	Method to retrieve the current version of TapLinx library

Version 1.3:

Table 31. Version 1.3 Added class/APIs

Class	API Syntax	Description
SecureKeyGenerator	byte[] getCMACDiversifiedKeyBytes FromKeyBytes(byte[], byte[], KeyType)	Provides the CMAC based diversified key
SecureKeyGenerator	SecureKeyGenerator get Instance(CustomModules, Provider)	To Retrieve singleton instance
SecureKeyGenerator	IKeyData getKeyFromKeyBytes(byte[], KeyType)	Gives the Secure key object from the given key bytes

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 31. Version 1.3 Added class/APIs*...continued*

Class	API Syntax	Description
IPlusSL1	boolean doOriginalityCheck(IKeyData)	For Plus SL1 we need to use SL1 card authentication key to verify chip originality
IUltraLightEV1	void setNegativePwdLimit(byte)	Sets the Limitation of negative password verification attempts
UltraLightEV1	void setNegativePwdLimit(byte)	Sets the Limitation of negative password verification attempts
IMIFAREIdentity	byte[] commitTransaction()	Commits the transaction and reads back the TMV and TMC
IMIFAREIdentity	IReader getReader()	Returns reader associated with Mifare Identity
IMIFAREIdentity	List<byte[]> readOfflineTransaction()	Read offline transaction records
IMIFAREIdentity	byte[] selectVirtualCard(byte[], IKey Data, IKeyData)	This method allows to select Virtual card base on DF name
IMIFAREIdentity	void writeOfflineTransaction(Issue TransactionParams)	Issues the offline transaction
IdentityFCIInfoForC1Type	Complete Class	Utility class for decoding FCI information
MIFAREIdentityUtility	Complete Class	Utility class for decoding FCI information
PICCFrameSize	Complete Class	Different PICC Frame size of DESFire Family cards
ICODEUtility	Complete Class	Utility class for ICODE operation
INTAG213TagTamper	Complete Class	New Tag supported
INTAG213TagTamper.MirrorType	Complete Class	Different types of mirroring supported by NTAG213 Tag Tamper
IDESFireEV2	byte[] getKeyVersionFromKeySet(byte keyNumber , byte keySetNumber)	Get the version of Key from specific keySet of selected application Note: If application 0 has been selected, only key number 0 is valid
IDESFireEV2	byte[] getAllKeySetVersion ()	Retrieves the all key set versions of selected application

Version 1.2:

NA

Version 1.1:

Table 32. Version 1.1 Added class/APIs

Class	API Syntax	Description
IDESFireEV2	byte[] isoSelectFile(final boolean is PICCMasterFile, final byte selection Control, final boolean isReturnFCI, final byte[] data);	
IDESFireEV2	SubType getSubType();	

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 32. Version 1.1 Added class/APIs*...continued*

Class	API Syntax	Description
IDESFireEV2	IKeyData getDerivedSymmetric OriginalityKey(final IKeyData master Key, final byte originalityKeyNo, final byte[] uid);	
IDESFireEV2	byte[] commitReaderId(final byte[] tMRI)	
IDESFireEV2	byte[] commitAndGetTransactionMac()	
IDESFireEV2	void changeMISMARTKey(final int cardkeyNumber, final KeyType key Type, final byte[] oldKey, final byte[] newKey, final byte newKeyVersion);	
IDESFireEV2	void changeVCKey(final int cardkey Number, final byte[] oldKey, final byte[] newKey, final byte newKeyVersion);	
DESFireFile.FileSettings	FileType getType()	
DESFireFile.FileSettings	CommunicationType getComSettings()	
DESFireFile.FileSettings	getReadAccess()	
DESFireFile.FileSettings	getWriteAccess()	
DESFireFile.FileSettings	getReadWriteAccess()	
DESFireFile.FileSettings	getChangeAccess()	
DESFireFile.StdDataFileSettings	getNumberOfAdditionalAccess Conditions	
DESFireFile.StdDataFileSettings	getNumberOfAdditionalAccessRights	
DESFireFile. ValueFileSettings	isGetFreeValueEnabled()	
DESFireFile. ValueFileSettings	getNumberOfAdditionalAccess Conditions()	
DESFireFile. ValueFileSettings	getNumberOfAdditionalAccessRights()	
DESFireFile. ValueFileSettings	isLimitedCreditValueEnabled()	
DESFireFile. ValueFileSettings	isGetFreeValueEnabled()	
DESFireFile. ValueFileSettings	getUpperLimit()	
DESFireFile. ValueFileSettings	getLowerLimit()	
DESFireFile. ValueFileSettings	getInitialValue()	
DESFireFile .LinearRecordFileSettings & DESFireFile. CyclicRecordFile Settings	getCurrentNumberOfRecords()	
DESFireFile .LinearRecordFileSettings & DESFireFile. CyclicRecordFile Settings	getMaxNumberOfRecords()	
DESFireFile .LinearRecordFileSettings & DESFireFile. CyclicRecordFile Settings	getRecordSize()	
DESFireFile .LinearRecordFileSettings & DESFireFile. CyclicRecordFile Settings	getNumberOfAdditionalAccess Conditions()	

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 32. Version 1.1 Added class/APIs *...continued*

Class	API Syntax	Description
DESFireFile .LinearRecordFileSettings & DESFireFile. CyclicRecordFile Settings	getNumberOfAdditionalAccessRights()	

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 32. Version 1.1 Added class/APIs*...continued*

Class	API Syntax	Description
EV2ApplicationKeySettings	setAppMasterKeyChangeable	
EV2PICCConfigurationSettings	public void setISODFNameForMiSmart Application(final IKeyData keyData, final KeyType keyType, final byte[] oldIsoDFName, final byte[] newIso DFName)	
DESFireFactory	public INTag413DNA getNTag413 DNA(final CustomModules custom Modules)	
DESFireFactory	public boolean isCardNTag413 DNA(final CustomModules custom ModulesObj)	

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

9 Removed classes

Version 5.0.0:

NA

Version 4.1.0:

NA

Version 4.0.0:

NA

Version 3.1.0:

NA

Version 3.0.0:

NA

Version 2.0:

NA

Version 1.9:

NA

Version 1.8:

NA

Version 1.7:

NA

Version 1.6:

1. INTAG5.java

Version 1.5:

NA

Version 1.4:

1. DESFireEV1PredictableChallenge.java.
2. IDESFireEV1PredictableChallenge.java.

Version 1.3:

NA

Version 1.2:

NA

Version 1.1:

1. InvalidArgumentException class is merged with UsageException to avoid ambiguity.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

10 Known issues

Known issues with certain Android Devices

Version 5.0.0:

Same as previous version.

Version 4.1.0:

Same as previous version.

Version 4.0.0:

Table 33. Version 4.0.0 Known issues

SL.No.	Device	Card	Issue
1.	All Google Pixel Devices	All Plus SL1 Cards	All Plus SL1 Crads are being detected as MIFare Classic
2.	Samsung Galaxy S20 FE	Plus EV1 SL0	Card is not being detected by the device
3.	Samsung Galaxy S20 FE	Plus EV1 SL3	Card is not being detected by the device
4.	All Devices running on Android 13 and above	DESFire EV3 DESFire EV3C	TransactionTimer API Failure
5.	Samsung Galaxy S20, FE Samsung A51, (All Samsung Devices)	DESFire EV3	ReadData amd WriteData API failure for 16k card

Version 3.1.0:

Table 34. Version 3.1.0 Known issues

SL.No.	Device	Card	Issue
1.	Google Pixel 3a Samsung Galaxy S20 FE Google Pixel 2 One Plus 5T Nokia 7 Plus LG Nexus 5x	Plus EV2 SL0 Plus EV2 SL1 Plus EV2 SL3	Cards are not being detected by the devices
2.	Samsung Galaxy S20 FE One Plus 5T Nokia 7 Plus LG Nexus 5x	Plus S 1K SL1 Plus S 2K SL1 Plus S 4K SL1	App is Crashing when cards are tapped to the devices and Cards are not being detected by the devices
3.	Google Pixel 3a Google Pixel 2	Plus X 2K SL1 Plus X 4K SL1	Plus X 2K SL1 and Plus X 4K SL1 are being detected as MIFare Classic
4.	Samsung Galaxy S20 FE One Plus 5T Nokia 7 Plus LG Nexus 5x	Plus X 2K SL1 Plus X 4K SL1	App is Crashing when cards are tapped to the devices and Cards are not being detected by the devices
5.	Samsung Galaxy S20 FE	Plus EV1 SL0	Card is not being detected by the device
6.	Samsung Galaxy S20 FE One Plus 5T Nokia 7 Plus LG Nexus 5x	Plus EV1 SL1	App is Crashing when cards are tapped to the devices and Cards are not being detected by the devices

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Table 34. Version 3.1.0 Known issues*...continued*

SL.No.	Device	Card	Issue
7.	Samsung Galaxy S20 FE	Plus EV1 SL3	Card is not being detected by the device
8.	Samsung Galaxy S20 FE One Plus 5T LG Nexus 5x	Plus SE SL1	App is Crashing when cards are tapped to the devices and Cards are not being detected by the devices
9.	Samsung Galaxy S20 FE Google Pixel 4a (All Android 13 devices)	DESFire EV3 DESFire EV3C	TransactionTimer API Failure
10.	Samsung Galaxy S20 FE Samsung A51 (All Samsung Devices)	DESFire EV3	ReadData amd WriteData API failure for 16k card

Version V3.0.0

Same as previous version.

Version V2.0

Table 35. Version V2.0 Known issues

SL.No.	Device	Card	Issue
1.	Google Pixel 3a Samsung Galaxy S20 FE Google Pixel 2 One Plus 5T Nokia 7 Plus LG Nexus 5x	Plus EV2 SL0 Plus EV2 SL1 Plus EV2 SL3	Cards are not being detected by the devices
2.	Samsung Galaxy S20 FE One Plus 5T Nokia 7 Plus LG Nexus 5x	Plus S 1K SL1 Plus S 2K SL1 Plus S 4K SL1	App is Crashing when cards are tapped to the devices and Cards are not being detected by the devices
3.	Google Pixel 3a Google Pixel 2	Plus X 2K SL1 Plus X 4K SL1	Plus X 2K SL1 and Plus X 4K SL1 are being detected as Mifare Classic
4.	Samsung Galaxy S20 FE One Plus 5T Nokia 7 Plus LG Nexus 5x	Plus X 2K SL1 Plus X 4K SL1	App is Crashing when cards are tapped to the devices and Cards are not being detected by the devices
5.	Samsung Galaxy S20 FE	Plus EV1 SL0	Card is not being detected by the device
6.	Samsung Galaxy S20 FE One Plus 5T Nokia 7 Plus LG Nexus 5x	Plus EV1 SL1	App is Crashing when cards are tapped to the devices and Cards are not being detected by the devices
7.	Samsung Galaxy S20 FE	Plus EV1 SL3	Card is not being detected by the device
8.	Samsung Galaxy S20 FE One Plus 5T LG Nexus 5x	Plus SE SL1	App is Crashing when cards are tapped to the devices and Cards are not being detected by the devices

Version V1.9

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Same as previous version.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Version V1.8

Table 36. Version V1.8 Known issues

SL.No.	Device	Card	Issue
1.	Model-LG Nexus 5x	Plus EV1	causes re-detection continuously
2.	Model-One Plus 1	Plus	NFC not responding properly
3.	Model-Samsung A5	Plus EV1 SL3	Plus EV1 SL3 is detected as PLUS X SL3
4.	Model- Samsung S7	Plus EV1 SL3	Plus EV1 SL3 & SL0 is detected as PLUS X SL3 & SL0 respectively
5.	All Android device that support NFC	PLUS EV2 SL1	PLUS EV2 SL1 is detected as PLUS EV1 SL1
6.	Samsung S8 Samsung S9 Samsung J7	PLUS S SL0 PLUS S SL3 PLUS X SL0 PLUS S SL3 Plus SE SL0 Plus SE SL3	Cards are not being detected by mentioned device
7.	Samsung S8 Samsung S9 Samsung J7 Samsung C7	PLUS S SL1 PLUS X SL1 Plus SE SL1	Cards are detected as MIFare Classic 1k
8.	Samsung Note 10 Lite Samsung A8 Star	Plus SE SL1	Plus SE SL1 is detected as Classic 1k

Version 1.7

Same as previous version.

Version 1.6:

Same as previous version.

Version 1.5:

Same as previous version.

Version V1.3

Table 37. Version V1.3 Known issues

SL.No.	Device	Card	Issue
1.	LG Nexus 5x	Plus EV1	causes re-detection continuously
2.	One Plus 1	Plus	NFC not responding properly
3.	Samsung A5	Plus EV1 SL3	Plus EV1 SL3 is detected as PLUS X SL3
4.	Samsung S7	Plus EV1 SL3	Plus EV1 SL3 & SL0 is detected as PLUS X SL3 & SL0 respectively

Version 1.2:

Same as previous version.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Version V1.1

Table 38. Version V1.1 Known issues

SL.No.	Device	Card	Issue
1.	OnePlus	ICODE	Not getting detected.
2.	LG Nexus 5x	Plus SL1	causes re-detection continuously
3.	Moto X Play	DESFire EV1/EV2	ISO 7816 command mode does not work
4.	Nexus 5X (Android 7.1.1) Model-OnePlus A3003 (Android 6.0.1)	Plus S Plus X	OriginalityCheck does not work
5.	Samsung Galaxy S7	DESFireEV2	Not get detected if random id is set
6.	Samsung Galaxy S7(SM-G930FD & Android 6.0.1) Samsung Galaxy S5(SM-G900H & Android 6.0.1)	MIFARE Identity	As this card is DESFire EV2 based and by default Random ID enabled and hence this also not getting detected
7.	Some devices	Ultralight NTAG	Detection takes long time
8.	All devices	Plus	Switching PLUS SL1 to SL3 is not possible because of android middleware issues
9.	All devices	UltralightNano	Total NDEF maximum size returns 46 instead of 40

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

11 Revision history

Table 39. Revision history

Document ID	Release date	Description
RN00263 v.3.0	13 November 2025	Editorial changes (typos, etc.). • Section 1 "General points": updated. • Section 2.1 "Version history": updated. • Section 3 "New features": updated. • Section 4 "Enhancements": updated. • Section 5 "Bug fixes": updated. • Section 6 "API signature changes": updated. • Section 7 "Removed/Obsolete APIs": updated. • Section 8 "Added class/APIs": updated. • Section 9 "Removed classes": updated. • Section 10 "Known issues": updated.
RN00263 v.2.0	11 July 2025	Editorial changes (typos, etc.). Document title updated to "TapLinx Android SDK". • Section 1 "General points": updated. • Section 2.1 "Version history": updated. • Section 2.2 "User guide": updated. • Section 3 "New features": updated. • Section 4 "Enhancements": updated. • Section 5 "Bug fixes": updated. • Section 6 "API signature changes": updated. • Section 7 "Removed/Obsolete APIs": updated. • Section 8 "Added class/APIs": updated. • Section 9 "Removed classes": updated. • Section 10 "Known issues": updated. • Section "Contents" removed.
RN00263 v.1.0	18 February 2025	• Initial version

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

Quick reference data — The Quick reference data is an extract of the product data given in the Limiting values and Characteristics sections of this document, and as such is not complete, exhaustive or legally binding.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Tables

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Figures

All information provided in this document is subject to legal disclaimers.

© 2025 NXP B.V. All rights reserved.

Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2025 NXP B.V. **All rights reserved.**