# CEN

# WORKSHOP

# AGREEMENT

# CWA 16008-2

August 2009

**ICS** 35.240.40

English version

## J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Part 2: Pin Keypad Device Class Interface - Programmer's Reference

This CEN Workshop Agreement has been drafted and approved by a Workshop of representatives of interested parties, the constitution of which is indicated in the foreword of this Workshop Agreement.

The formal process followed by the Workshop in the development of this Workshop Agreement has been endorsed by the National Members of CEN but neither the National Members of CEN nor the CEN Management Centre can be held accountable for the technical content of this CEN Workshop Agreement or possible conflicts with standards or legislation.

This CEN Workshop Agreement can in no way be held as being an official standard developed by CEN and its Members.

This CEN Workshop Agreement is publicly available as a reference document from the CEN Members National Standard Bodies.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

**Management Centre: Avenue Marnix 17, B-1000 Brussels**



Ref. No.:CWA 16008-2:2009 E

# Contents

# Foreword

This CWA contains the specifications that define the J/eXtensions for Financial Services (J/XFS) for the Java ™ Platform, as developed by the J/XFS Forum and endorsed by the CEN J/XFS Workshop. J/XFS provides an API for Java applications which need to access financial devices. It is hardware independent and, by using 100% pure Java, also operating system independent.

The CEN J/XFS Workshop gathers suppliers (among others the J/XFS Forum members), service providers as well as banks and other financial service companies. A list of companies participating in this Workshop and in support of this CWA is available from the CEN Secretariat , and at http://www.cen.eu/cenorm/sectors/sectors/isss/activity/jxfs_membership.asp. The specification was agreed upon by the J/XFS Workshop Meeting of 2009-05-6/9 in Brussels, and the final version was sent to CEN for publication on 2009-06-12.

The specification is continuously reviewed and commented in the CEN J/XFS Workshop. The information published in this CWA is furnished for informational purposes only. CEN makes no warranty expressed or implied, with respect to this document. Updates of the specification will be available from the CEN J/XFS Workshop public web pages pending their integration in a new version of the CWA (see http://www.cen.eu/cenorm/sectors/sectors/isss/activity/jxfs_cwas.asp).

The J/XFS specifications are now further developed in the CEN J/XFS Workshop. CEN Workshops are open to all interested parties offering to contribute. Parties interested in participating and parties wanting to submit questions and comments for the J/XFS specifications, please contact the J/XFS Workshop Secretariat hosted in CEN (jxfs-helpdesk@cen.eu).

Questions and comments can also be submitted to the members of the J/XFS Forum through the J/XFS Forum web-site http://www.jxfs.net.

This CWA is composed of the following parts:

- Part 1: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Base Architecture - Programmer's Reference
- Part 2: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Pin Keypad Device Class Interface - Programmer's Reference
- Part 3: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Magnetic Stripe & Chip Card Device Class Interface - Programmer's Reference
- Part 4: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Text Input/Output Device Class Interface - Programmer's Reference
- Part 5: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Cash Dispenser, Recycler and ATM Device Class Interface - Programmer's Reference
- Part 6: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Printer Device Class Interface - Programmer's Reference
- Part 7: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Alarm Device Class Interface - Programmer's Reference
- Part 8: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Sensors and Indicators Unit Device Class Interface - Programmer's Reference
- Part 9: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Depository Device Class Interface - Programmer's Reference
- Part 10: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Check Reader/Scanner Device Class Interface - Programmer's Reference (deprecated in favour of Part 13)
- Part 11: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Camera Device Class Interface - Programmer's Reference
- Part 12: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Vendor Dependant Mode Specification - Programmer's Reference
- Part 13: J/eXtensions for Financial Services (J/XFS) for the Java Platform – Scanner Device Class Interface - Programmer’s Reference (recommended replacement for Part 10)

## HISTORY

Main differences to CWA 14923-2:2004 are:

- Additional information to check correctness of keys with a verification code
- New feature for a secure key entry
- New supported PIN Block formats
- New feature to import an AMT Master Key Block in a PKCS#7 format

Main differences to CWA 13937-2:2000 are:

- Included the functions and features regarding the Remote Key loading.
- Added properties kuseRSAPrivateand kuseRSAPrivateSign in the *JxfsPINKeyUses* class
- Included the GENAS protocol
- A new method is created specially dedicated to import a RSA public key : *importEMVRSAPublicKey*.
- A new class *JxfsPINEMVRSAKeyToImport* is created.
- JxfsRSASignatureAlgo is replaced by JxfsPINRSAIntegrityAlgorithm.
- JxfsPINEMVCryptoModes data class is created
- *JxfsPINKeyUses* data class, added a new property : KuseRSAPublicKeyVerifiy and clarified the description of kusRSAPublicKey.
- *JxfsPINBlockData* data class : added clarification when it is used for EMV
- added a new method to delete a key from the encryption module: *deleteKey*
- *JxfsPINKeyVerificationData* added clarification in the keyVerCode property
- Added clarification in the *initialize* method
- Added an Appendix to defines the EMV requirements and clarifications
- Added EMV features:
- Removed getBMP / setBMP methods from *JxfsPINSecureMsgISO*
- Added the ZKA extension
- Added changes following the proposal on the read methods of document 2001/037
- Added *JxfsPINReadMode2* support class
- Added readData(*JxfsPINReadMode2* readMode) to *IJxfsPINKeypadControl*
- Added secureReadPin(*JxfsPINReadMode2* readMode) to *IJxfsPINKeypadControl*
- Added the eventOnStartSupported property in the *IJxfsPINKeypadControl*
- JXFS_E_CLAIMED exception removed from section 4.2
- keyEncKey property of *JxfsPINCryptoData* (section 5.29.1) changed from String to byte[]
- Added a class hierarchy diagram
- Added 2 header pages: title and history
- Added paragraph describing handling of null parameters

# 1 Scope

This document describes the Pin Keypad Device (PIN) classes based on the basic architecture of J/XFS which is similar to the JavaPOS architecture. It is event driven and asynchronous.

Three basic levels are defined in JavaPOS. For J/XFS this model is extended by a communication layer, which provides device communication that allows distribution of applications and devices within a network. So we have the following layers in J/XFS :

- Application
- Device Control and Device Manager
- Device Communication
- Device Service

Application developers program against control objects and the Device Manager which reside in the Device Control layer. This is the usual interface between applications and J/XFS devices. Device Control objects access the Device Manager to find an associated Device Service. Device Service objects provide the functionality to access the real device (i.e. like a device driver).
During application startup the Device Manager is responsible for locating the desired Device Service object and attaching this to the requesting Device Control object. Location and/or routing information for the Device Manager reside in a central repository.

To support Pin Keypad devices the basic Device Control structure is extended with various properties and methods specific to this device which are described on the following pages.

# 2 Overview

## 2.1 Description

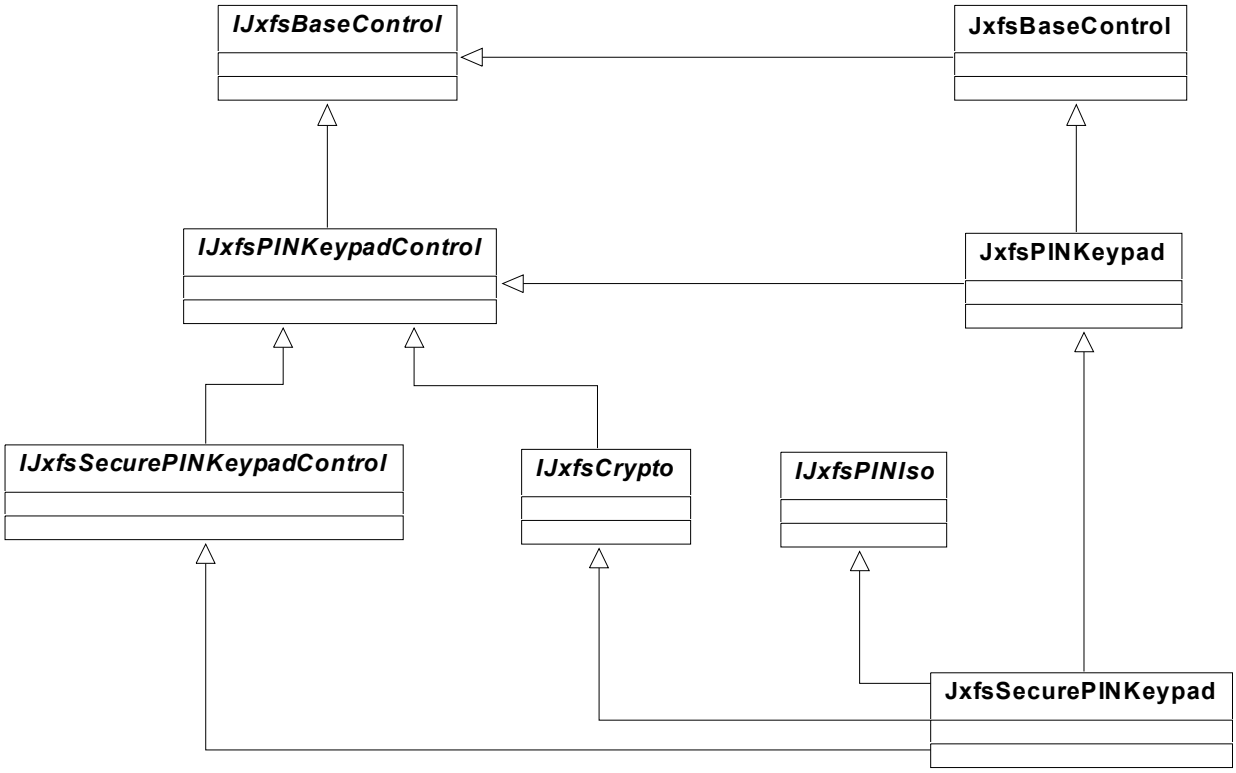
This specification covers the interfaces and classes to access personal identification keypads (PIN pads). The main functions of PIN Keypad devices supported in this specification are:

- Non secure key pad functions (like key press detection, plain PIN retrieval,...)
- Secure PIN operations (like PIN validation, data encryption with PIN as cryptography input,...)
- Cryptographic services (like data encryption/decryption, MAC generation,...)

The J/XFS PIN Keypad specification separates the PIN Keypad functions between generic non-secure keypad functions and security-related functions, that is, the ones related to cryptography.

As well as the rest of J/XFS device controls, the J/XFS PIN Keypad Device Support uses the event driven model and the same behavioral model. Therefore, the application will instantiate a J/XFS PIN Keypad Device Control Object and then use the available methods to do I/O. When an I/O method is called, the J/XFS PIN Keypad Device Service will attempt to process the requested I/O. If the request is invalid or an exception is encountered, the application will be notified by a J/XFS exception. Completion of the request will be reported by an event. Thus the application must register itself with the J/XFS PIN Keypad Device Control Object for the various types of events it wishes to handle.

## 2.2 Class Hierarchy




The *IJxfsPINIso* interface is mandatory for the JxfsSecurePINKeypad control, but is optional for the implementing device service (see Appendix A).

## 2.3 Classes and Interfaces

The following classes and interfaces are used by the J/XFS PIN Keypad Device Controls.

| Class or Interface | Name | Description | Extends / Implements |
|---|---|---|---|
| Interface | **IJxfsBaseControl** | Base interface for all the device controls. Contains methods common to all the device controls. | -- |
| Interface | **IJxfsPINKeypadControl** | Base interface for PIN controls.<br>Contains methods declarations specific to PIN device controls. | Extends:<br>**IJxfsBaseControl** |
| Interface | **IJxfsSecurePINKeypadControl** | Interface for PIN controls implementing secure PIN entry and validation.<br>Contains methods specific to device controls for the secure PIN device category. | Extends:<br>**IJxfsPINKeypadControl** |
| Interface | **IJxfsCrypto** | Interface for PIN controls implementing security and cryptographic functions. | Extends:<br>**IJxfsPINKeypadControl** |
| Class | **JxfsBaseControl** | Base class for all the device controls. Contains properties common to all the device controls. | |
| Class | **JxfsPINKeypad** | Base class for PIN controls. Contains properties specific to PIN device controls. | Implements:<br>**IJxfsPINKeypadControl** |
| Class | **JxfsSecurePINKeypad** | Class for PIN controls implementing security and cryptographic functions. | Extends:<br>**JxfsPINKeypad**<br>Implements:<br>**IJxfsSecurePINKeypadControl, IJxfsCrypto** |

## 2.4 Support Classes

| Class or Interface | Name | Description | Extends / Implements |
|---|---|---|---|
| Interface | **JxfsConst** | Interface containing the Jxfs constants that are common to several device categories | -- |
| Interface | **JxfsPINConst** | Interface containing the Jxfs constants that are common to all the PIN device controls. | -- |
| Class | **JxfsPINFKeySet** | PIN function keys selector class. Indicates for each function key if it is selected or not.<br>Properties are read only. | Extends:<br>**JxfsType** |
| Class | **JxfsPINFKeysSelection** | Subclass of JxfsPINFKeySet. It contains the same properties, but they can be set by applications. | Extends:<br>**JxfsPINFKeySet** |
| Class | **JxfsPINFDKeysSelection** | PIN function descriptor keys selector class. Indicates for each function descriptor key if it is selected or not. | Extends:<br>**JxfsType** |
| Class | **JxfsPINFDKey** | Data class that contains information about a function descriptor key (FDKey). | Extends:<br>**JxfsType** |
| Class | **JxfsPINReadMode** | Data class that defines the conditions for PIN keypad input operations. | Extends:<br>**JxfsType** |
| Class | **JxfsPINReadMode2** | Data class that defines extended conditions for PIN keypad input operations | Extends:<br>**JxfsPINReadMode** |
| Class | **JxfsPINPressedKey** | Data class that contains information about a key pressed during an input operation. | Extends:<br>**JxfsType** |
| Class | **JxfsPINReadData** | Data class that contains the information provided to the application when an input operation completes. | Extends:<br>**JxfsType** |
| Class | **JxfsPINFormats** | PIN formats selector class. Indicates for each PIN format if it is selected or not.<br>Properties are read only. | Extends:<br>**JxfsType** |
| Class | **JxfsPINValidationAlgorithms** | PIN validation algorithms selector class. Indicates for each PIN validation algorithm if it is selected or not.<br>Properties are read only. | Extends:<br>**JxfsType** |
| Class | **JxfsPINChipPresentationModes** | PIN chip presentation algorithms selector class. Indicates which presentation algorithms for chip PIN validation are supported. | Extends:<br>**JxfsType** |
| Class | **JxfsPINValidationData** | Abstract data class. Root of a hierarchy of data objects that contain data for PIN verification and used in | Extends:<br>**JxfsType** |

| | | | |
|---|---|---|---|
| | | *validationPIN()* method. | |
| Class | **JxfsPINValidationDataForDES** | Data class for PIN verification using DES algorithm. | Extends: **JxfsPINValidationData** |
| Class | **JxfsPINValidationDataForEC** | Data class for PIN verification using EUROCHEQUE specification. | Extends: **JxfsPINValidationData** |
| Class | **JxfsPINValidationDataForVISA** | Data class for PIN verification using VISA specification. | Extends: **JxfsPINValidationData** |
| Class | **JxfsPINOffsetData** | Data class for creating a PIN offset. | Extends: **JxfsPINValidationData** |
| Class | **JxfsPINBlockData** | Data class for creating a PIN block. | Extends: **JxfsPINValidationData** |
| Class | **JxfsPINChipValidationData** | Abstract data class for all PIN chip validation modes. | Extends: **JxfsType** |
| Class | **JxfsPINChipValidationDataClear** | Data class for PIN chip validation mode Clear. Used as parameter in validatePINChip() method. | Extends: **JxfsPINChipValidationData** |
| Class | **JxfsPINValidationResult** | Data class that contains the result of a PIN validation operation. | Extends: **JxfsType** |
| Class | **JxfsPINOffset** | Data class that contains computed PIN offset. | Extends: **JxfsType** |
| Class | **JxfsPINBlock** | Data class that contains computed PIN block. | Extends: **JxfsType** |
| Class | **JxfsPINChipValidationResult** | Data class that contains the result of a PINchip validation operation. | Extends: **JxfsType** |
| Class | **JxfsPINCryptoModes** | Encyption modes selector class. Indicates for each encryption mode if it is selected or not.<br>Properties are read only. | Extends: **JxfsType** |
| Class | **JxfsPINEMVCryptoModes** | Encryption modes selector class. Indicates for each encryption mode if it is selected or not.<br>Properties are read only.<br>it is only for EMV RSA keys | Extends: **JxfsType** |
| Class | **JxfsPINKeyDetail** | Data class containing information about a key from the device's key table. | Extends: **JxfsType** |
| Class | **JxfsPINEMVRSAKeyToImport** | Data class containing input data for *importEMVRSAPublicKey()* method, specially for RSA keys | Extends: **JxfsType** |
| Class | **JxfsPINKeyToImport** | Data class containing input data for *importKey()* method | Extends: **JxfsType** |
| Class | **JxfsPINInitialization** | Data class that contains result data from initialization of security module. | Extends: **JxfsType** |
| Class | **JxfsPINKeyVerificationData** | Data class that contains result data from an import key operation. | Extends: **JxfsType** |
| Class | **JxfsPINCryptoData** | Data class that contains input | Extends: |

| | | | |
|---|---|---|---|
| | | data for *encrypt*/*decrypt* operations. | **JxfsType** |
| Class | **JxfsPINMACData** | Data class that contains input data for MAC generation operation. | Extends: **JxfsPINCryptoData** |
| Class | **JxfsPINCryptoResult** | Data class that contains result data from cryptographic operations. | Extends: **JxfsType** |
| Class | **JxfsPINKeyUses** | Data class that contains information on allowed uses for a key. | Extends: **JxfsType** |
| Class | **JxfsPINIdKeyModes** | Data class that contains information on implemented uses of ID key. | Extends: **JxfsType** |
| Class | **JxfsPINEMVRSAIntegrityAlgorithm** | Data class that contains information about the type of verification to compute for the RSA key to import. | Extends: **JxfsType** |
| Class | **JxfsPINImportRSAPublicKey** | Data class that contains class contains data required as input for importRSAPublicKey*()* operation. | Extends: **JxfsType** |
| Class | **JxfsPINExportedRSAPublicKey** | data returned on *JxfsOperationCompleteEvent* of the *exportRSAPublicKey()* operation | Extends: **JxfsType** |
| Class | **JxfsPINExportRSAPublicKey** | Data class that contains information data that specifies the RSA public key to export. | Extends: **JxfsType** |
| Class | **JxfsPINExportRSADESEncipheredPublicKey** | This class is used to export the public part of RSA keys. | Extends: **JxfsType** |
| Class | **JxfsPINImportRSADESEncipheredPublicKey** | Data class that class contains data required as input for *importRSADESEncipheredPublicKey()* operation. | Extends: **JxfsType** |
| Class | **JxfsPINGenerateRSAKeyPair** | Data class that class contains data required as input for *generateRSAKeyPair()* operation. | Extends: **JxfsType** |
| Class | **JxfsPINExportId** | Data class that class contains data retrieved by the PIN device and which uniquely identifies the PIN device. | Extends: **JxfsType** |
| Class | **JxfsPINExportCertificate** | Data class that class contains data required as output for *exportCertificate()* operation. | Extends: **JxfsType** |
| Class | **JxfsPINCertificateType** | Data class that class contains the type, primary or secondary, of certificate exported from the encryptor. | Extends: **JxfsType** |
| Class | **JxfsPINCertificateKeyType** | Data class that class contains data required as input for *exportCertificate()* operation. | Extends: **JxfsType** |
| Class | **JxfsPINRSAHashAlgorithms** | This class provides properties and methods to query which type of hash algorithms is to be processed. | Extends: **JxfsType** |
| Class | **JxfsPINRSASignatureAlgo** | This class provides properties and methods to | Extends: **JxfsType** |

| | | | |
|---|---|---|---|
| | | query which type of RSA Signature algorithms is to be processed. | |
| Class | **JxfsPINRSAExponent** | This class provides properties and methods to query which exponent value of the RSA key pair to be generated. | Extends: **JxfsType** |
| Class | **JxfsPINRSAKeyVerificationData** | This class contains information about the imported RSA Public key. | Extends: **JxfsType** |
| Class | **JxfsPINRSADESKeyVerificationData** | This class contains information about the imported RSA DES enciphered public key. | Extends: **JxfsType** |
| Class | **JxfsPINRSADESLength** | This class specifies the key length that was loaded. | Extends: **JxfsType** |
| Class | **JxfsPINRSADESCheckMode** | This class specifies the mode that was used to create the check value. | Extends: **JxfsType** |
| Class | **JxfsPINRSAKeyType** | This class specifies the private signature to use. | Extends: **JxfsType** |
| Class | **JxfsPINRemoteKeyLoadModes** | This class provides properties and methods to query which remote key loading modes are supported by a secure PIN device service. | Extends: **JxfsType** |
| Class | **JxfsPINRSAAlgorithm** | This class provides properties and methods to query which RSA algorithm are supported by the secure PIN device service. | Extends: **JxfsType** |
| Class | **JxfsSHA1Data** | Data class that contains information to compute a SHA 1 digest or the result of the computation. | Extends: **JxfsType** |
| Class | **JxfsEvent** | Abstract class from which all Jxfs event classes are extended | Extends: **java.util. EventObject** |
| Class | **JxfsStatusEvent JxfsOperationCompleteEvent JxfsIntermediateEvent** | The Device Service creates *Event* event instances of this class and delivers them through the J/XFS PIN Device Control's event callbacks to the application | Extends: **JxfsEvent** |
| Class | **JxfsException** | Exception class. The J/XFS PIN Device Control creates and throws exceptions on method failure and property access failure. | Extends: **java.lang.Exception** |

# 3 Device behavior

## 3.1 Handling of null parameters

If null is passed as a method parameter, a JxfsException exception with the error Code property set to JXFS_E_PARAMETER_INVALID will be thrown, unless the handling of a null parameter is explicitly specified for a particular method.

# 4 Classes and Interfaces

All operation methods return an identificationID. If an operation cannot be processed because of an error detected before the asynchronous processing of the method begins (i.e. before the calling thread returns) a JxfsException is thrown.
After processing has taken place, a JxfsOperationCompleteEvent is generated which contains detailed information about the status of the operation, i.e., if it failed or succeeded, and eventually additional data as a result.

The Constants, Error Codes, Exceptions, Status Codes and Support Classes that are used in the methods are described in special chapters at the end of the documentation.

## 4.1 Access to properties

Please note the following when determining the meaning of a property's **Access**:

| | |
|---|---|
| **R** | The property is read only. |
| **W** | The property is write only. |
| **R/W** | The property may be read or written. |

To access these properties the applications must use the appropriated methods specified by the JavaBean specification. Note that boolean properties are read using *isProperty* method instead of *getProperty*.

### getProperty

| | |
|---|---|
| **Syntax** | **Property*** get***Property *() throws JxfsException*** |
| **Description** | Returns the requested property. |
| **Parameter** | **None** |
| **Event** | No additional events are generated. |
| **Exceptions** | Some possible JxfsException *value codes*. Common values are:<br>JXFS_E_CLOSED<br>JXFS_E_UNREGISTERED<br>JXFS_E_REMOTE |

### set*Property*

| | |
|---|---|
| **Syntax** | **void *set*Property *(value) throws JxfsException*** |
| **Description** | Sets the requested property. |
| **Parameter** | The desired property value. |
| **Event** | No additional events are generated |
| **Exceptions** | Some possible JxfsException *value codes*. Common values are:<br>JXFS_E_CLOSED<br>JXFS_E_UNREGISTERED<br>JXFS_E_REMOTE<br>JXFS_E_PARAMETER_INVALID |

## 4.2 IJxfsPINKeypadControl

### 4.2.1 Introduction

The J/XFS PIN Keypad Device Control Subclass is defined in JxfsPINKeypad and is a subclass of JxfsBaseControl. Its interface is defined in *IJxfsPINKeypadControl* interface which is a subclass of IJxfsBaseControl interface. The purpose of the J/XFS PIN Keypad Device Control object is to allow passing data and control between the application and the device support code so that the associated device can be accessed.

The JxfsPINKeypad class represents a physical PIN Keypad device with basic input keypad functions. There are no built-in security functions.

#### Summary

Although *IJxfsPINKeypadControl* is an interface, and therefore properties do not apply, properties are detailed here with the objective to provide guidance on the implementation of those classes that will implement this interface.

Therefore, the *IJxfsPINKeypadControl* consists on the following methods:

- Getters of listed properties.
- Methods listed.

**Implements :** **Extends : IJxfsBaseControl**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| supportedFDKeys | java.util.Vector | R | After successful open |
| supportedFKeys | **JxfsPINFKeySet** | R | After successful open |
| inputRawSupported | boolean | R | After successful open |
| inputCookedSupported | boolean | R | After successful open |
| beepOnPressSupported | boolean | R | After successful open |
| eventOnStartSupported | boolean | R | After successful open |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | After successful open |
| readData | identificationID | After successful open |

## Properties

### supportedFDKeys Property (R)

**Type** ***java.util.Vector***
**Initial Value** Depends on device.
**Description** This vector contains a list of all function descriptor keys (FDKeys) supported by the device.

Each vector element is a *JxfsPINFDKey* object that contains its key code and position information. See *JxfsPINFDKey* class description for more information.

If empty, then no FDKeys are supported.

### supportedFKeys Property (R)

**Type** ***JxfsPINFKeySet***
**Initial Value** Null until open.
**Description** Indicates the set of function keys supported by the device.

### inputRawSupported (R)

**Type** ***boolean***
**Initial Value** Depends on device.
**Description** Specifies if raw input mode is supported by the device, where each key pressed during an input operation will generate an intermediate event. These events will contain information about pressed keys.

| Value | Meaning |
|---|---|
| *false* | Raw input mode is not supported. |
| *true* | Raw input mode is supported. |

### inputCookedSupported (R)

**Type** ***boolean***
**Initial Value** Depends on device.
**Description** Specifies if cooked input mode is supported by the device, where no intermediate events per key pressed are generated. Data entered during an input operation is provided in the Jxfs*OperationCompleteEvent* event.

| Value | Meaning |
|---|---|
| *false* | Cooked input mode is not supported. |
| *true* | Cooked input mode is supported. |

### beepOnPressSupported (R)

**Type** ***boolean***
**Initial Value** Depends on device.
**Description** Specifies if the device has controllable capability of emitting an audible sound when a key is pressed.

| Value | Meaning |
|---|---|
| *false* | Device has no controllable beep capability. |
| *true* | Device has controllable beep capability. |

### eventOnStartSupported (R)

**Type** ***boolean***

**Initial Value** Depends on service.

**Description** Specifies if the service has the capability to send the intermediate event JXFS_I_PIN_READ_STARTED

| **Value** | **Meaning** |
|---|---|
| *false* | The service does not have this capability. |
| *true* | The service has this capability. |

### 4.2.2 Methods

**readData**

**Syntax** ***identificationID readData (JxfsPINReadMode readMode) throws JxfsException;***

**Description** This command activates the PIN Keypad to read a data entry.

Digits are read until the value of *maxLength* property of *readMode* parameter is reached (if *autoEnd* property of *readMode* is set to *true*), or a termination key is pressed. If *maxLength* is set to zero and no termination keys are specified, operation will not terminate until cancelled.

Each key pressed is notified as an intermediate event if *inputMode* property of *readMode* parameter is set to JXFS_PIN_INPUT_RAW. If *inputMode* is set to JXFS_PIN_INPUT_COOKED, then, a single *JxfsOperationCompleteEvent* event (containing input data) is issued when input operation terminates.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINReadMode | I | readMode | A data object that contains all the data required to perform a data entry (see *JxfsPINReadMode* class specification). |

**Event**

**JxfsOperationCompleteEvent**

When an input operation is completed a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINReadData* object. |

**JxfsIntermediateEvent**

Every key pressed generates an intermediate event if *inputMode* property is set to JXFS_PIN_INPUT_RAW.
*JxfsIntermediateEvent* events are sent by PIN Device Control to all registered IntermediateListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of operation. |
| *reason:* | |
| | JXFS_I_PIN_KEY_PRESSED<br>A key has been pressed. |
| *data* | A *JxfsPINPressedKey* object. |

**readData**

**Syntax** ***identificationID readData (JxfsPINReadMode2 readMode) throws JxfsException;***

**Description** This command activates the PIN Keypad to read a data entry.

Digits are read until the value of *maxLength* property of *readMode* parameter is reached (if *autoEnd* property of *readMode* is set to *true*), or a termination key is pressed. If *maxLength* is set to zero and no termination keys are specified, operation will not terminate until cancelled.

Each key pressed is notified as an intermediate event if *inputMode* property of *readMode* parameter is set to JXFS_PIN_INPUT_RAW. If *inputMode* is set to JXFS_PIN_INPUT_COOKED, then, a single *JxfsOperationCompleteEvent* event (containing input data) is issued when input operation terminates.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINReadMode2 | I | readMode | A data object that contains all the data required to perform a data entry (see *JxfsPINReadMode2* class specification). |

**Event**

**JxfsOperationCompleteEvent**

When an input operation is completed a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINReadData* object. |

**JxfsIntermediateEvent**

Every key pressed generates an intermediate event if *inputMode* property is set to JXFS_PIN_INPUT_RAW.
*JxfsIntermediateEvent* events are sent by PIN Device Control to all registered IntermediateListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of operation. |
| *reason:* | |
| | JXFS_I_PIN_KEY_PRESSED<br>A key has been pressed. |
| *data* | A *JxfsPINPressedKey* object. |

**JxfsIntermediateEvent**

If the eventOnStart property is set, the service sends this event when the operation is really started. That is the moment when the device begins accepting data entered by the user.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of operation. |
| *reason:* | |
| | JXFS_I_PIN_READ_STARTED<br>The device is ready for input operation. |
| *data* | null |

## 4.3 IJxfsSecurePINKeypadControl

### 4.3.1 Introduction

The J/XFS Secure PIN Keypad Device Control Subclass is defined in JXFSecurePINKeypad and is a subclass of JxfsPINKeypad. The Secure PIN Keypad Device Control is intended to match physical PIN Keypad devices with the following extended security capabilities:

- PIN secure read,
- PIN verification and
- Cryptographic services.

Its interface is defined in IJxfsSecurePINKeypadControl interface which is a subclass of *IJxfsPINKeypadControl* interface.

#### Summary

Although IJxfsSecurePINKeypadControl is an interface, and therefore properties do not apply, properties are detailed here with the objective to provide guidance on the implementation of those classes that will implement this interface.

Therefore, the IJxfsSecurePINKeypadControl consists on the following methods:

- Getters of listed properties.
- Methods listed.

**Implements :** **Extends :** *IJxfsPINKeypadControl*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| supportedPINFormats | **JxfsPINFormats** | R | After successful open |
| supportedValidationAlgorithms | **JxfsPINValidationAlgorithms** | R | After successful open |
| supportedChipPresentationModes | **JxfsPINChipPresentationModes** | R | After successful open |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | After successful open |
| secureReadPIN | identificationID | After successful open |
| createOffset | identificationID | After successful open |
| createPINBlock | identificationID | After successful open |
| validatePIN | identificationID | After successful open |
| createOffsetSecure | identificationID | After successful open |
| createPINBlockSecure | identificationID | After successful open |
| validatePINSecure | identificationID | After successful open |
| validatePINChip | identificationID | After successful open |

### 4.3.2 Properties

#### supportedPINFormats Property (R)

| | |
|---|---|
| **Type** | ***JxfsPINFormats*** |
| **Initial Value** | Null until open. |
| **Description** | Specifies the supported PIN formats. |

#### supportedValidationAlgorithms Property (R)

| | |
|---|---|
| **Type** | ***JxfsPINValidationAlgorithms*** |
| **Initial Value** | Null until open. |
| **Description** | Specifies the supported algorithms for PIN validation. |

#### supportedChipPresentationModes Property (R)

| | |
|---|---|
| **Type** | ***JxfsPINChipPresentationModes*** |
| **Initial Value** | Depends on device. |
| **Description** | Specifies the supported presentation algorithms for chip PIN validation. |

### 4.3.3 Methods

**secureReadPIN**

**Syntax** ***identificationID secureReadPIN (JxfsPINReadMode readMode) throws JxfsException;***

**Description** This command activates the PIN Keypad to read a PIN entry in a secure way.
Entered data is not passed to the application but retained for further cryptographic operation (like PIN validation, PIN offset generation or PIN Block generation).

Digits are read until the value of *maxLength* property of *readMode* parameter is reached (if *autoEnd* property of *readMode* is set to *true*), or a termination key is pressed. If *maxLength* is set to zero and no termination keys are specified, operation will not terminate until cancelled.

Each key pressed is notified as an intermediate event if *inputMode* property of *readMode* parameter is set to JXFS_PIN_INPUT_RAW. If *inputMode* is set to JXFS_PIN_INPUT_COOKED, then, a single *JxfsOperationCompleteEvent* event (containing input data) is issued when input operation terminates.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINReadMode | I | readMode | A data object that contains all the data required to perform a data entry (see *JxfsPINReadMode* class specification). |

**Event**

**JxfsOperationCompleteEvent**
When an input operation is completed a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINReadData* object. |

**JxfsIntermediateEvent**
Every key pressed generates an intermediate event if *inputMode* property is set to JXFS_PIN_INPUT_RAW.
*JxfsIntermediateEvent* events are sent by PIN Device Control to all registered IntermediateListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of operation. |
| *reason:* | |
| | JXFS_I_PIN_KEY_PRESSED<br>A key has been pressed. |
| *data* | A *JxfsPINPressedKey* object. |

**secureReadPIN**

**Syntax** ***identificationID secureReadPIN (JxfsPINReadMode2 readMode) throws JxfsException;***

**Description** This command activates the PIN Keypad to read a PIN entry in a secure way.

Entered data is not passed to the application but retained for further cryptographic operation (like PIN validation, PIN offset generation or PIN Block generation).

Digits are read until the value of *maxLength* property of *readMode* parameter is reached (if *autoEnd* property of *readMode* is set to *true*), or a termination key is pressed. If *maxLength* is set to zero and no termination keys are specified, operation will not terminate until cancelled.

Each key pressed is notified as an intermediate event if *inputMode* property of *readMode* parameter is set to JXFS_PIN_INPUT_RAW. If *inputMode* is set to JXFS_PIN_INPUT_COOKED, then, a single *JxfsOperationCompleteEvent* event (containing input data) is issued when input operation terminates.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINReadMode2 | I | readMode | A data object that contains all the data required to perform a data entry (see *JxfsPINReadMode2* class specification). |

**Event**

**JxfsOperationCompleteEvent**
When an input operation is completed a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINReadData* object. |

**JxfsIntermediateEvent**
Every key pressed generates an intermediate event if *inputMode* property is set to JXFS_PIN_INPUT_RAW.
*JxfsIntermediateEvent* events are sent by PIN Device Control to all registered IntermediateListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of operation. |
| *reason:* | |
| | JXFS_I_PIN_KEY_PRESSED<br>A key has been pressed. |
| *data* | A *JxfsPINPressedKey* object. |

**JxfsIntermediateEvent**
If the eventOnStart property is set, the service sends this event when the operation is really started. That is the moment when the device begins accepting data entered by the user.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_READPIN |
| *identificationID* | Identification Id of operation. |
| *reason:* | |
| | JXFS_I_PIN_READ_STARTED<br>The device is ready for input operation. |
| *data* | null |

### createOffset

**Syntax**

***identificationID createOffset (JxfsPINOffsetData offsetData) throws JxfsException;***

**Description**

This function is used to generate a PIN Offset that is used to verify PINs using the *validatePIN()* method with DES validation algorithm.

The PIN offset is computed by combining validation data with the keypad entered PIN.

This method clears the PIN.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINOffsetData | I | offsetData | A data object that contains all the data required to create the PIN offset (see *JxfsPINOffsetData* class specification). |

**Event**

**JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners. In addition a data object is returned:

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_CREATEOFFSET |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINOffset* object. It contains the computed PIN offset |

### createPINBlock

**Syntax**

***identificationID createPINBlock (JxfsPINBlockData pinBlockData) throws JxfsException;***

**Description**

This method takes the account information and a PIN entered by the user to build a formatted PIN. Encrypting this formatted PIN once or twice returns a PIN block which can be written on a magnetic card or sent to a host.

The PIN block can be calculated using one of the formats specified in the *supportedPINFormats* property.

The PIN block is computed by combining customer data with the keypad entered PIN.

This command clears the PIN.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINBlockData | I | pinBlockData | A data object that contains all the data required to create the PIN block (see *JxfsPINBlockData* class specification). |

**Event**

**JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_CREATEPINBLOCK |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINBlock* object. It contains the computed PIN block. |

### validatePIN

**Syntax** ***identificationID validatePIN (JxfsPINValidationData validationData) throws JxfsException;***

**Description** The previously entered PIN is combined with the requisite data specified by the PIN validation algorithm and locally verified for correctness.

The validationData object should specify the validation algorithm to be used for PIN validation as well as all needed data to perform the validation (*see JxfsPINValidationData* class specification)

This method clears the PIN.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINValidationData | I | validationData | Validation data object containing specific data for the actual PIN validation algorithm to be used *(see JxfsPINValidationData* class specification). |

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_VALIDATEPIN |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINValidationResult* object. It contains the results of the validation. |

### createOffsetSecure

**Syntax** ***identificationID createOffsetSecure (JxfsPINOffsetData offsetData) throws JxfsException;***

**Description** This function is used to generate a PIN Offset that is used to verify PINs using the *validatePIN()* method with DES validation algorithm.

With combined MSD-PIN devices, this function does not require that validation data be first read from the card with the MSD component and then returned to the device as a parameter. Instead, the validation data is automatically read from the card in the device.
The behavior is as follows:
1 – If card is present in reader and ejectCurrent property is *false* then go to 5.
2 – If card is present in reader and ejectCurrent property is *true* then eject the card.
3 – Arm the device to accept a magnetic stripe card.
4 – Poll card status and verify that card is seated.
5 – Perform the intended function using the offset data read from the card.
6 – Eject the card if ejectWhenComplete property is *true*.

This method clears the PIN.

| Parameter | Type | IO | Name | Meaning |
|---|---|---|---|---|
| | JxfsPINOffsetData | I | offsetData | A data object that contains all the data required to create the PIN offset (see *JxfsPINOffsetData* class specification). |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners. In addition a data object is returned:

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_CREATEOFFSET_SECURE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINOffset* object. It contains the computed PIN offset |

## createPINBlockSecure

**Syntax** ***identificationID createPINBlockSecure (JxfsPINBlockData pinBlockData) throws JxfsException;***

**Description** This method takes the account information and a PIN entered by the user to build a formatted PIN. Encrypting this formatted PIN once or twice returns a PIN block which can be written on a magnetic card or sent to a host.

The PIN block can be calculated using one of the formats specified in the *supportedPINFormats* property.

The PIN block is computed by combining customer data with the keypad entered PIN.

With combined MSD-PIN devices, this function does not require that customer data be returned to the device as a parameter. Instead, the customer data is automatically read from the card in the device.
The behavior is as follows:

1 – If card is present in reader and ejectCurrent property is *false* then go to 5.
2 – If card is present in reader and ejectCurrent property is *true* then eject the card.
3 – Arm the device to accept a magnetic stripe card.
4 – Poll card status and verify that card is seated.
5 – Perform the intended function using the customer data read from the card.
6 – Eject the card if ejectWhenComplete property is *true*.

This command clears the PIN.

| Parameter | Type | IO | Name | Meaning |
|---|---|---|---|---|
| | JxfsPINBlockData | I | pinBlockData | A data object that contains all the data required to create the PIN block (see *JxfsPINBlockData* class specification). |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_CREATEPINBLOCK_S |

| | |
|---|---|
| | ECURE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINBlock* object. It contains the computed PIN block. |

### validatePINSecure

**Syntax**

***identificationID validatePINSecure (JxfsPINValidationData validationData) throws JxfsException;***

**Description**

The previously entered PIN is combined with the requisite data specified by the DES validation algorithm and locally verified for correctness.

With combined MSD-PIN devices, this function does not require that offset and/or validation data be returned to the device as parameters. Instead, offset and/or validation data can be automatically read from the card in the device.
The behavior is as follows:
1 – If card is present in reader and ejectCurrent property is *false* then go to 5.
2 – If card is present in reader and ejectCurrent property is *true* then eject the card.
3 – Arm the device to accept a magnetic stripe card.
4 – Poll card status and verify that card is seated.
5 – Perform the intended function using the data read from the card.
6 – Eject the card if ejectWhenComplete property is *true*.

This method clears the PIN.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINValidationData | I | validationData | Validation data object containing specific data for the actual PIN validation algorithm to be used *(see JxfsPINValidationData* class specification). |

**Event**

**JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_VALIDATEPIN_SECURE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINValidationResult* object. It contains the results of the validation. |

### validatePINChip

**Syntax**

***identificationID validatePINChip (java.lang.String aCCDeviceName, JxfsPINChipValidationData validationData) throws JxfsException;***

**Description**

The previously entered PIN is combined with the requisite data specified by the chip PIN presentation algorithm and presented to the chip card device for correctness verification.

The validationData object specifies all the needed data to perform the validation (*see JxfsPINChipValidationData* class and subclasses specifications)

This method clears the PIN.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| java.lang.String | I | aCCDeviceName | The name of the Chip Card device to be used for PIN validation.<br>It is the responsibility of the application to ensure the chip card has already been inserted.<br>It is the responsibility of the J/XFS device service to instantiate a J/XFS Chip Card Control and to use it exclusively to access the chip card. If the chip card device is already claimed by someone else, a JXFS_E_CLAIMED exception is thrown.<br>The device service must release ownership of the device after using it.<br>During the validation of the PIN the application must not access the chip card; only the PinPad device service has the right to access the chip card. |
| JxfsPINChipValidationData | I | validationData | Validation data object containing specific data for the actual PIN validation algorithm to be used *(see JxfsPINChipValidationData* class specification). |

**Event**

**JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_VALIDATEPINCHIP |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINChipValidationResult* object. It contains the results of the validation. |

**JxfsIntermediateEvent**

*JxfsIntermediateEvent* events can be sent by PIN Device Control to all registered IntermediateListeners

| Field | Value |
|---|---|
| *operationID* | JXFS_O_CCD_CHIPIO |
| *identificationID* | Identification Id of operation. |
| *reason:* | JXFS_I_CCD_NO_MEDIA_PRESENT<br>The read operation request cannot progress because there is no media inserted. |

| | |
|---|---|
| | JXFS_I_CCD_MEDIA_INSERTED<br>The read operation request continues because a media has been inserted. |
| *data* | Null |

## 4.4 IJxfsCrypto

### 4.4.1 Introduction

The cryptographic services interface provides generic cryptography functions. It handles a key table and allows the user to *encrypt*, *decrypt* or calculate check codes using keys from its table. This interface is used for the sake of clarity; to separate the generic cryptographic functions from the PIN related cryptographic functions. The JxfsSecurePINKeypad class implements this interface.

#### Summary

**Implements :** - **Extends :** *IJxfsPINKeypadControl*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| supportedCryptoModes | **JxfsPINCryptoModes** | R | After successful open |
| numberOfKeys | **int** | R | After successful open |
| idKey | **JxfsPINIdKeyModes** | R | After successful open |
| secureKeyEntrySupported | **JxfsSecureKeyEntrySupportedEnum** | R | After successful open |
| secureKeyDetail | **JxfsPINSecureKeyDetail** | R | After successful open |
| secureKeyEntryState | **boolean** | R | After successful open |
| remoteKeyLoadCapabilities | **JxfsPINRemoteKeyLoadModes** | R | After successful open |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | After successful open |
| decrypt | identificationID | After successful open |
| encrypt | identificationID | After successful open |
| generateMAC | identificationID | After successful open |
| getKeyInfo | *JxfsPINKeyDetail* | After successful open |
| getKeyNameList | *java.util.Vector* | After successful open |
| importKey | identificationID | After successful open |
| initialize | identificationID | After successful open |
| importEMVRSAPublicKey | identificationID | After successful open |
| computeSHA1Digest | identificationID | After successful open |
| deleteKey | identificationID | After successful open |
| importRSAPublicKey | identificationID | After successful open |
| exportRSAPublicKey | identificationID | After successful open |
| importRSADESEnciphered PublicKey | identificationID | After successful open |
| exportRSADESEnciphered PublicKey | identificationID | After successful open |
| generateRSAKeyPair | identificationID | After successful open |
| exportPINId | identificationID | After successful open |
| importCertificate | identificationID | After successful open |
| exportCertificate | identificationID | After successful open |
| replaceCertificate | identificationID | After successful open |
| startKeyExchange | identificationID | After successful open |
| secureKeyEntry | identificationID | After successful open |
| importSecureKeyEntered | identificationID | After successful open |
| clearSecureKeyBuffer | identificationID | After successful open |
| importRSAEncipheredPKCS7Key | identificationID | After successful open |

### 4.4.2 Properties

**supportedCryptModes Property (R)**

| | |
|---|---|
| **Type** | ***JxfsPINCryptoModes*** |
| **Initial Value** | Depends on device. |
| **Description** | Specifies the supported encryption modes. |

**numberOfKeys Property (R)**

| | |
|---|---|
| **Type** | ***int*** |
| **Initial Value** | Depends on device. |
| **Description** | Specifies the number of keys that may be stored by the device. |

**idKey Property (R)**

| | |
|---|---|
| **Type** | ***JxfsPINIdKeyModes*** |
| **Initial Value** | Depends on device. |
| **Description** | Specifies whether an ID key is supported or not. |

**secureKeyEntrySupported Property (R)**

| | |
|---|---|
| **Type** | ***JxfsSecureKeyEntrySupportedEnum*** |
| **Initial Value** | Depends on device. |
| **Description** | Indicates if the pinpad is capable or not to enter manually securely a master key. |

**secureKeyDetail Property (R)**

| | |
|---|---|
| **Type** | ***JxfsPINSecureKeyDetail*** |
| **Initial Value** | Depends on device. |
| **Description** | Null if secureKeyEntrySupported property is not equal to “supported”<br>Specifies the secure key entry method used by the device. This allows an application to enable the relevant keys and inform the user how to enter the hex digits 'A' to 'F'. It reports the following information:<br>• The secure key entry mode (how to enter hex digit 'A' to 'F')<br>• The function keys and FDKs available during secure key entry<br>• The FDKs that are configured as function keys (Enter, Cancel, Clear and Backspace)<br>• The physical keyboard layout<br>The keys enter in secure key entering mode is always entered in hexadecimal.<br><br>For more details, refer to the *JxfsPINSecureKeyDetail* and *secureKeyEntry* method description. |

### secureKeyEntryState Property (R)

| | |
|---|---|
| **Type** | ***boolean*** |
| **Initial Value** | *false* |
| **Description** | This value is relevant only if secureKeyEntrySupported property is equal to "supported".<br>Specifies if the device is in secure key entry state or not. The value of this property is *true* if the device is in "secure key entry state", *false* otherwise.<br>For more details, refer to the Secure Key Entry State Diagram of this document. |

### remoteKeyLoadCapabilities Property (R)

| | |
|---|---|
| **Type** | ***JxfsPINRemoteKeyLoadModes*** |
| **Initial Value** | Depends on device. |
| **Description** | This value specifies which ways the device service supports to load a key remotely.<br>If the device does not support any of the remote key load modes it must return an object where all the modes are set to *false*. |

### 4.4.3 Methods

#### decrypt

**Syntax** ***identificationID decrypt (JxfsPINCryptoData decryptData) throws JxfsException;***

**Description** Deciphers data with the currently selected algorithm and the specified key name.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINCryptoData | I | decryptData | Contains the data and additional information required to perform a *decrypt* operation. See *JxfsPINCryptoData* specification). |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_DECRYPT |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINCryptoResult* object. It contains the results of the decryption. |

#### encrypt

**Syntax** ***identificationID encrypt (JxfsPINCryptoData encryptData) throws JxfsException;***

**Description** Encrypts data with the currently selected algorithm and the specified key name.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINCryptoData | I | encryptData | Contains the data and additional information required to perform a *encrypt* operation. See *JxfsPINCryptoData* specification). |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_ENCRYPT |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINCryptoResult* object. It contains the results of the encryption. |

#### generateMAC

**Syntax** ***identificationID generateMAC (JxfsPINMACData macData) throws JxfsException;***

**Description** Generates a MAC data with the currently selected algorithm.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|

| | | | |
|---|---|---|---|
| JxfsPINMACData | I | macData | Contains the data and additional information required to perform a *decrypt* operation. See *JxfsPINMACData* specification). |

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_GENMAC |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINCryptoResult* object. It contains the generated MAC. |

### getKeyInfo

**Syntax** ***JxfsPINKeyDetail getKeyInfo (java.lang.String keyName) throws JxfsException;***

**Description** Retrieves information about a given key
Returns a *JxfsPINKeyDetail* object with the requested info.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| String | I | keyName | Name of the key to be queried. |

**Event** No additional events are generated:

**Exceptions** Common or device dependent error code. (See section on *Error Codes*).

### getKeyNameList

**Syntax** ***java.util.Vector getKeyNameList () throws JxfsException;***

**Description** Retrieves the list of keys names used by the device.
Returns a vector of strings with the name of all keys stored in the device.

**Event** No additional events are generated.

**Exceptions** No additional exceptions are generated.

### importKey

**Syntax** ***identificationID importKey (JxfsPINKeyToImport keyToImport, boolean lastOrOnlyPart) throws JxfsException;***

**Description** Loads a key or part of a key into the encryption module. The key can be passed in clear text mode or encrypted with an accompanying “key encryption key”.

The imported key is imported into the encryption module and is used for cryptographic operations.

The key may be loaded in parts.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINKeyToImport | I | keyToImport | Contains the data required to import the key (see *JxfsPINKeyToImport* specification). |
| boolean | I | lastOrOnlyPart | If *true*, key import is finished. |

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| **Field** | **Value** |
|---|---|
| *operationID* | JXFS_O_PIN_IMPORTKEY |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINKeyVerificationData* Object. |

**Status Event**
If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to al registered listeners:

| **Field** | **Value** |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### initialize

**Syntax** ***identificationID initialize (byte[] id, byte[] key) throws JxfsException;***

**Description** Clears all loaded or imported keys from device key table.
Clears also the key imported through *secureKeyEntry* and *importSecureKeyEntered* methods.

Usually this operation is invoked by an operator task and not by the application program.

During initialization, an optional encrypted Id key can be stored in the device. The Id key and the corresponding encryption key can be passed as parameters; if not, they are generated automatically by the encryption module. The encrypted Id is returned to the application and serves, if supported (see idKey property), as authorization for the key import function.
This function also resets the HSM terminal data, except session key index and trace number.
This function resets all certificate data and authentication public/private keys (including those replaced by *generateRSAKeyPair*) back to their initial states at the time of production. Any keys installed during production, which have been permanently replaced, will not be reset. Any Verification certificates that may have been loaded must be reloaded. The Certificate state will remain the same, but the certificate must be re-imported.

**Parameter**

| **Type** | **IO** | **Name** | **Meaning** |
|---|---|---|---|
| byte[] | I | id | ID Key. This byte array is encrypted under *key* and stored into the device. An empty array if not required. |
| byte[] | I | key | Encryption key of *id*. It is also stored into the device.<br>If it's an empty array, *id* is in clear mode |

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event

will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent event listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_INITIALIZE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINInitialization* Object. |

### importEMVRSAPublicKey

**Syntax**

***identificationID importEMVRSAPublicKey (JxfsPINEMVRSAKeyToImport RSAkeyToImport,) throws JxfsException;***

**Description**

Loads a EMV RSA public key into the encryption module. The RSA key can be provided either by a Certification Authority or by the EMV application in the Chipcard.

This method is similar to the *importKey* method, but it is specifically designed to address the key formats and security features defined by EMV. Mainly the extensive use of "signed certificate" or "EMV certificate" (which is a compromise between signature and a pure certificate) to provide the public key is taken in account.

The device services is responsible for all EMV public key import validation. Once loaded, the service provider is not responsible for key/certificate expiry, this is an application responsibility.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINEMVRSAKeyToImport | I | EMVRSAkeyToImport | Contains the data required to import the key (see *JxfsPINEMVRSAKeyToImport* specification). |

**Event**

**JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_IMPORTEMVRSAKEY |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINKeyVerificationData* Object. |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| Field | Value |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### computeSHA1Digest

**Syntax**

***identificationID computeSHA1Digest (JxfsSHA1Data SHA1Data) throws JxfsException;***

**Description**

Computes a digest using a SHA-1 algorithm on a stream of data. This method can be used to verify the EMV Static Data Authentication or the Dynamic Data Authentication

| Parameter | Type | IO | Name | Meaning |
|---|---|---|---|---|
| | JxfsSHA1Data | I | SHA1Data | Contains the data and the length of data to be hashed (See ***JxfsSHA1Data*** specification). |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_SHA1_DIGEST |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsSHA1Data* object. It contains the result of SHA1 algorithm. |

### deleteKey

**Syntax** ***identificationID deleteKey ( java.lang.String keyName) throws JxfsException;***

**Description** deletes a key from the encryption module which was previously stored

| Parameter | Type | IO | Name | Meaning |
|---|---|---|---|---|
| | java.lang.String | I | keyName | Contains the name of the key to be deleted from the encryption module. |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_DELETE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | none |

### importRSAPublicKey

**Syntax** ***identificationID importRSApublicKey (JxfsPINImportRSAPublicKey RSAPublicKeyToImport, boolean lastOrOnlyPart) throws JxfsException;***

**Description** Loads a RSA public key part into the encryption module and will be used for cryptographic operations. The key can be passed in clear text mode or encrypted with an accompanying “key encryption key”.

| Parameter | Type | IO | Name | Meaning |
|---|---|---|---|---|
| | JxfsPINImportRSAPublicKey | I | RSAPubliKeyToImport | Contains the data required to import the key (see *JxfsPINImportRSAPublicKey* specification). |
| | boolean | I | lastOrOnlyPart | If *true*, key import is finished. |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_IMPORTRSAPUBLICKEY |
| *identificationID* | Identification Id of complete operation. |

| | |
|---|---|
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINRSAKeyVerificationData* Object. |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| Field | Value |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### exportRSAPublicKey

**Syntax**

***identificationID exportRSAPublicKey (JxfsPINExportRSAPublicKey RSAPublicKeyToExport) throws JxfsException;***

**Description**

This command will export the RSA Public key associated with this PIN device. The RSA public key to export is either the issuer key pair or a previously generated RSA key pair.

Other secure devices will use this key to communicate information securely with this PIN device, using RSA public key encryption.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINExportRSAPublicKey | I | RSAPublicKeyToExport | Contains the data required to export the RSA public key |

**Event**

**JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_EXPORTRSAPUBLICKEY |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINExportedRSAPublicKey* Object. |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| Field | Value |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### importRSADESEncipheredPublicKey

**Syntax**

***identificationID importRSADESEncipheredPublicKey (JxfsPINImportRSADESEncipheredPublicKey***

***RSADESEncipheredPublicKeyToImport) throws JxfsException;***

**Description** This command is used to load a Symmetric Key that is either a single or double DES length key into the encryptor. The key passed by the application is loaded in the encryption module, the (optional) signature & hash are used during validation, the key is extracted using the device's RSA Private Key, and is then stored. The loaded key will be discarded at any stage if any of the above fails.
The random number previously obtained from the *startKeyExchange* command and sent to the host is included in the signed data. This random number (when present) is verified during the load process. This command ends the Key Exchange process.
If a Signature algorithm is specified that is not supported by the PIN DS, then the message will not be decrypted and the command fails

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINImportRSADESEncipheredPublicKey | I | RSADESEncipheredPublicKeyToImport | Contains the data of the enciphered imported key. |

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_IMPORTRSADESENCIPHEREDPUBLICKEY |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINRSADESKeyVerificationData* object. |

**Status Event**
If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| Field | Value |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### exportRSADESEncipheredPublicKey

**Syntax** ***identificationID*** **exportRSADESEncipheredPublicKey** ***() throws JxfsException;***

**Description** This command will export the RSA DES enciphered Public key associated with this PIN device. Other secure devices will use this key to communicate information securely with this PIN device, using RSA public\private key encryption.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_EXPORTRSADESENCIPHERDPUBLICKEY |

| | |
|---|---|
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINExportRSADESEncipheredPublicKey* Object. |

### generateRSAKeyPair

**Syntax** ***identificationID* generateRSAKeyPair *(JxfsPINGenerateRSAKeyPair RSAKeyPairToGenerate) throws JxfsException;***

**Description** This command will generate a new RSA key pair.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINGenerateRSAKeyPair | I | RSAKeyPairToGenerate | Contains the data of the generated RSA Key pair. |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_GENERATERSAKEYPAIR |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | none. |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| Field | Value |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### exportPINId

**Syntax** ***identificationID exportPINId () throws JxfsException;***

**Description** This command is used to retrieve the Security Item that uniquely identifies the PIN device. This value may be used to uniquely identify a PIN device and therefore confer trust upon any key or data obtained from this device.

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_EXPORTPINID |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | *JxfsPINExportId* |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| Field | Value |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### importCertificate

**Syntax**

***identificationID* importCertificate *(byte [] certificateToImport) throws JxfsException;***

**Description**

This command is used to load a certificate provided by a Certification Authority (CA) to be used for remote key loading. This command can be called only once, if there are no plans for a new CA to take over the duties. If a new CA does take over the duties, then this command should be called after the **replaceCertificate** method. The type of certificate (Primary or Secondary) to be loaded will be embedded within the actual certificate structure.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| byte [] | I | certificateToImport | Contains the certificate that is to be loaded. This data should be in a binary encoded PKCS #7 format containing certificate data represented in DER encoded ASN.1 notation |

**Event**

**JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_IMPORTCERTIFICATE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | ***byte []*** SHA-1 Thumb print value. This data should be in a binary encoded PKCS #7 format containing certificate data represented in DER encoded ASN.1 notation. |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| Field | Value |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### exportCertificate

**Syntax**

***identificationID* exportCertificate *(JxfsPINCertificateKeyType certificateKeyType) throws JxfsException;***

**Description**

This command is used to read out from the encryptor the certificate

that was signed by a certification Authority (CA). This certificate is sent to the host.

| Parameter | Type | IO | Name | Meaning |
|---|---|---|---|---|
| | JxfsPINCertificateKeyType | I | certificateKeyType | Specifies which public key to be used. |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_EXPORTCERTIFICATE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | *JxfsPINExportCertificate object* |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| Field | Value |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

## replaceCertificate

**Syntax** ***identificationID* replaceCertificate *(byte [] newCertificate) throws JxfsException;***

**Description** This operation will replace either the primary or secondary Certificate Authority certificate previously loaded inside the encryptor. After this command is complete, the application should send the ***importCertificate*** and ***exportCertificate*** commands to ensure that the new HOST and the encryptor have both all the information required to perform the remote key loading process.

| Parameter | Type | IO | Name | Meaning |
|---|---|---|---|---|
| | byte [] | I | newCertificate | Contains the new certificate that is to be loaded. This data should be in a binary encoded PKCS #7 format containing certificate data represented in DER encoded ASN.1 notation. |

**Event** **JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_REPLACECERTIFICATE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | ***byte []*** SHA-1 Thumb print value. This data should be in a binary encoded PKCS #7 format containing certificate data represented in DER encoded ASN.1 notation. |

**Status Event**
If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| **Field** | **Value** |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### startKeyExchange

**Syntax** ***identificationID startKeyExchange () throws JxfsException***

**Description** This command is used to start the transfer of the host's Key Transport Key. The encryptor generates a random number that will be used to verify tke key Transport message sent by the host.
The key exchange process is ended with the command *importRSADESEncipheredPublicKey* command.

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| **Field** | **Value** |
|---|---|
| *operationID* | JXFS_O_PIN_STARTKEYEXCHANGE |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | ***byte []:*** Specifies an 8 bytes randomly generated number created by the encryptor. If the PIN device does not support random generated numbers generation an empty array is returned. |

**Status Event**
If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| **Field** | **Value** |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### secureKeyEntry

**Syntax** ***identificationID secureKeyEntry (JxfsPINSecureKeyMode secureKeyMode) throws JxfsException;***

**Description** This command allows an encryption key part to be entered directly into the pinpad without being exposed outside of the pinpad. From the point this method is called, encryption key digits (JXFS_PIN_FK_0 to JXFS_PIN_FK_9 and JXFS_PIN_FK_A to JXFS_PIN_FK_F) are **NOT** passed to the above software layer (Device Interface, Device Service, Application...) but saved internally in the keypad.
If *inputMode* property of *secureKeyMode* parameter is set to JXFS_PIN_INPUT_RAW, an intermediate event (JXFS_I_PIN_KEY_PRESSED) is sent for each active key pressed. If the key pressed is one of the encryption key digit (0-9 or A-F) the intermediate event will not contain the code of the key. It will be sent only to allow the application to perform the appropriate action. E.g.:

display a '*' to the operator. If the key pressed is not part of the encryption key digit (0-9 or A-F) the intermediate JXFS_I_PIN_KEY_PRESSED will be sent with the key code in it. The Shift key will **NOT** generate JXFS_I_PIN_KEY_PRESSED intermediate event and will not have any effect in the minimum and maximum length counter that is informed by the ***JxfsPINSecureKeyMode*** object. The secure key is entered in hexadecimal mode. The minimum and maximum length refer to the digits pressed, so, for instance, a secure key with 8 bytes (33 C0 8E D0 BC 00 7C FB) will need a maximal length equals or greater than 16. The keys that can be enabled by this command are defined by the *JxfsPINSecureKeyDetail* property. Function keys which are not associated with an encryption key digit (0-9 or A-F) may be enabled but will not contribute to the secure entry buffer (unless they are Cancel, Clear or Backspace) and will not count towards the length of the key entry. The Cancel and Clear keys will cause the encryption key buffer to be cleared. The Backspace key will cause the last encryption key digit in the encryption key buffer to be removed.
If autoEnd property of ***JxfsPINSecureKeyMode*** object is *true* the command will automatically complete when the maximal length of (maxLength property of ***JxfsPINSecureKeyMode*** object) encryption key digits have been added to the buffer. If autoEnd property is *false* then the command will not automatically complete and Enter, Cancel or any terminating key must be pressed. When maxLength hex encryption key digits have been entered then all encryption key digits keys are disabled. If the Clear or Backspace key is pressed to reduce the number of entered encryption key digits below maxLength, the same keys will be re-enabled.
Terminating keys have to be active keys to operate.
If an FDKey is associated with Enter, Cancel, Clear or Backspace then the FDKkey must be activated to operate. The Enter and Cancel FDKeys must also be marked as a terminator if they are to terminate entry. These FDKeys are reported as normal FDKeys within the intermediate event, applications must be aware of those FDKeys associated with Cancel, Clear, Backspace and Enter and handle any user interaction as required.
If inputMode is set to JXFS_PIN_INPUT_COOKED, then, a single JxfsOperationCompleteEvent event (containing data about the key entered) is issued when input operation terminates.
KCV should be checked to avoid storing an incorrect key component.
Encryption key parts entered with this command are stored through the *importSecureKeyEntered* method. Each key part can only be stored once, after which the secure key buffer will be cleared automatically.
If the *secureKeyEntry* method is called a second time before calling *importSecureKeyEntered* method the data stored in the secure buffer will be discarded.
If after the completion of *secureKeyEntry* method the application gives up of importing the secure key stored in the secure buffer **and** possible keys parts that were not definitively imported yet (only partially imported). the *clearSecureKeyBuffer* method can be called in order to clear the secure key buffer and the key parts previously partially imported.

When the *secureKeyEntry* method starts executing the device goes from the state OPEN / WORKING to the state OPEN / WORKING / SECUREKEYENTERING To check if the device is in *secure key enter* state the aplication can check the *secureKeyEntryState* property. The possible states regarding this method and the states transitions are described in the state diagram of this document.

**Security warning:**
The *secureKeyEntry* method allows secure and non-secures keys to be

pressed by the user. The secure keys *(JXFS_PIN_FK_0 to JXFS_PIN_FK_9 and JXFS_PIN_FK_A to JXFS_PIN_FK_F)* are **NOT** sent to the above software layer and these keys are the ones that will compose the key imported into the device. The non-secure keys (FDKeys: Enter, Cancel, Clear, Backspace, etc) will not be part of the secure key imported and may be used, for instance, to cancel a key entering operation. These keys are sent to the upper layers, so they are classified as non-secure key because an attacker may handle them in a malicious way. If the application believes, based in the usage context and others security measures involved in the process of a secure key entering, that the FDKeys sent to the upper layers may jeopardize security, it may not enable FDKeys entering by not setting them as active and terminate keys.

**Parameter**

| **Type** | **IO** | **Name** | **Meaning** |
|---|---|---|---|
| JxfsPINSecureKeyMode | I | secureKeyMode | A data object that contains all the data required to perform a data entry (see *JxfsPINSecureKeyMode* class specification). |

**Event**

**JxfsOperationCompleteEvent**

When an input operation is completed a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered OperationCompleteListeners

| **Field** | |
|---|---|
| *operationID* | JXFS_O_PIN_SECUREKEYENTRY |
| *identificationID* | Identification Id of operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | A *JxfsPINSecureKeyEntered* object. |

**JxfsIntermediate Event**

Every key pressed generates an intermediate event if *inputMode* property is set to JXFS_PIN_INPUT_RAW.
*JxfsIntermediateEvent* events are sent by PIN Device Control to all registered IntermediateListeners

| **Field** | **Value** |
|---|---|
| *operationID* | JXFS_O_PIN_SECUREKEYENTRY |
| *identificationID* | Identification Id of operation. |
| *reason:* | |
| | JXFS_I_PIN_KEY_PRESSED<br>A key has been pressed. |
| *data* | A *JxfsPINPressedKey* object. |

**JxfsIntermediateEvent**

If the eventOnStart property is set, the service sends this event when the operation is really started. That is the moment when the device begins accepting data entered by the user.

| **Field** | **Value** |
|---|---|
| *operationID* | JXFS_O_PIN_SECUREKEYENTRY |
| *identificationID* | Identification Id of operation. |
| *reason:* | |
| | JXFS_I_PIN_READ_STARTED<br>The device is ready for input operation. |
| *data* | null |

### importSecureKeyEntered

**Syntax**

***identificationID importSecureKeyEntered (JxfsPINSecureKeyToImport keyToImport, boolean lastOrOnlyPart, JxfsVerificationTypeEnum verificationType, byte[] verificationData) throws JxfsException;***

**Description**

Loads a key or part of a key into the encryption module. The key to be imported was securely entered by the operator after the execution of the *secureKeyEntry* method.

The imported key is imported into the encryption module and is used for cryptographic operations.

The key may be loaded in parts.

If the *lastOrOnlyPart* parameter is *false* it means that the key being imported is only a key part and it will have at least one more part. The keys imported in this way are defined in this document as "partially" imported keys and may be cleared by the *clearSecureKeyBuffer* method. If the *lastOrOnlyPart* parameter is *true* the key is a role key or this is the last part of the key. At this moment the key entered and all its possible parts are "definitively" imported. It means that the key imported cannot be deleted by the *clearSecureKeyBuffer* method. After the key has been "definitively" imported it can be removed only by the *initialize, deleteKey* and *enterSecureKey* methods.

The *verificationType* parameter informs which type of verification will be performed by the Device Service to calculate the verification type data.
The verification data will be calculated by the Device Service and compared with the verification data informed by the application.
If the Device Service does not support the verification type informed, the error JXFS_E_PIN_VERIFICATION_TYPE_NOT_SUPPORTED will be returned. After this error the Device Service will be in the same state, so calling this method again with a verification type supported can be performed.
If the secure key being entering is being entered in parts, the verification data is related only with the part being entered (*lastOrOnlyPart* set to *false*). When the last part is entered (*lastOrOnlyPart* set to *true*) the verification data is calculated from the entire key (all the parts).
If the verification data informed by the application does not match the verification data computed by the Device Service the key was not correctly imported and JXFS_E_PIN_KEY_VERIFICATION_DATA_DOESNOT_MATCH will be returned.
If key verification checking is not supported by the Device Service or is not requested by the application the value of the parameter *verificationType* must be *none* and the *verificationData byte array* must be empty.

The JXFS_RC_SUCCESSFUL return code means that the key or part of key was correctly imported.

After the last part of the key is entered successfully the device leaves the *secure key entering* state.

**Parameter**

| Type | IO | Name | Meaning |
|---|---|---|---|
| JxfsPINKeyToImport | I | keyToImport | Contains the data required to import the key (see *JxfsPINKeyToImport* specification). |
| boolean | I | lastOrOnlyPart | If *false* the key is |

| | | | |
|---|---|---|---|
| | | | "partially" imported. If *true*, key import is finished and the keys with all its possible key parts are "definitively" imported. |
| JxfsVerificationTypeEnum | I | verificationType | The verification type that the *JxfsPINKeyVerificationData* object content will have. |
| byte[] | I | verificationData | The verification data (kcv) that the Device Service will check against verification data of the key or key part being imported. If the verification data informed in this parameters doesn't match the verification data of the key being imported a JXFS_E_PIN_KEY_VERIFICATION_DATA_DOESNOT_MATCH error will be returned. |

**Event**

**JxfsOperationCompleteEvent**

When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| **Field** | **Value** |
|---|---|
| *operationID* | JXFS_O_PIN_IMPORTKEY |
| *identificationID* | Identification Id of complete operation. |
| *Result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | None. |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to al registered listeners:

| **Field** | **Value** |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

### clearSecureKeyBuffer

**Syntax**

***identificationID clearSecureKeyBuffer() throws JxfsException;***

**Description**

Clears the secure key buffer of the device **and** the keys parts that were partially imported by *importSecureKeyEntered* method (*lastOrOnlyPart* parameter of *importSecureKeyEntered* method set to *false*). The secure key buffer contains a key or part of a key that was entered by the *secureKeyEntry* method.

After the key is definitively imported by the *importSecureKeyEntered* (*lastOrOnlyPart* parameter set to *true*) it **cannot** be cleared by this

method any more.

Note:
If the key is not going to be imported (e.g.: because the KCV is not correct) but a second try is going to be performed, calling this method is not necessary because the second call of the *secureKeyEntry* will discard the previous key stored in the secure buffer.
If the key is imported (*importSecureKeyEntered* method), calling this method is also not necessary because after the key is imported the secure key buffer will be automatically cleared.
This method will be useful when the application gives up of importing the previous key entered by *secureKeyEntry* method and possible keys parts that were not definitively imported yet (only partially imported).

After the successful execution of this method (JXFS_RC_SUCCESSFUL) the device leaves the *secure key entering* state.

To get a better understanding of the usage of this method and the secure key entering state see the sequence and state diagrams of this document.

| **Parameter** | **Type** | **IO** | **Name** | **Meaning** |
|---|---|---|---|---|

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| **Field** | **Value** |
|---|---|
| *operationID* | JXFS_O_PIN_CLEARSECUREKEYBUFFER |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *Data* | null |

### importRSAEncipheredPKCS7Key

**Syntax** ***identificationID importRSAEncipheredPKCS7Key (JxfsPINImportRSAEncipheredPKCS7Key RSAEncipheredPKCS7KeyToImport) throws JxfsException;***

**Description** This command is used to load a symmetric key that is either a 16 or 32 byte DES key into the encryption module. It is verified by the host public key and decrypted by using the device RSA private key and is then stored. The loaded key will be discarded at any stage if any of the above steps failed.
The random number previously obtained from the *startKeyExchange* command and sent to the host is included in the signed data. This random number (when present) is verified during the load process. This command ends the Key Exchange process.

| **Parameter** | **Type** | **IO** | **Name** | **Meaning** |
|---|---|---|---|---|
| | JxfsPINImportRSAEncipheredPKCS7Key | I | RSAEncipheredPKCS7KeyToImport | Contains the data of the enciphered key which has to be imported. |

**Event** **JxfsOperationCompleteEvent**
When the operation completes a *JxfsOperationCompleteEvent* event will be sent by J/XFS PINKeypad Device Control to all registered JxfsOperationCompleteEvent listeners.

| **Field** | **Value** |
|---|---|
| *operationID* | JXFS_O_PIN_IMPORTRSAENCIPHE |

| | |
|---|---|
| | REDPKCS7KEY |
| *identificationID* | Identification Id of complete operation. |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | *JxfsPINExportRSASignedPKCS7KeyConfirmation.* In case of an error the content of all properties is irrelevant. |

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| **Field** | **Value** |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |

# 5 Support Classes

## 5.1 JxfsPINFKeySet

This class provides properties and methods to query which function keys are supported or are active.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| fk0 | boolean | R | |
| fk1 | boolean | R | |
| fk2 | boolean | R | |
| fk3 | boolean | R | |
| fk4 | boolean | R | |
| fk5 | boolean | R | |
| fk6 | boolean | R | |
| fk7 | boolean | R | |
| fk8 | boolean | R | |
| fk9 | boolean | R | |
| fkEnter | boolean | R | |
| fkCancel | boolean | R | |
| fkClear | boolean | R | |
| fkBackspace | boolean | R | |
| fkHelp | boolean | R | |
| fkDecPoint | boolean | R | |
| fk00 | boolean | R | |
| fk000 | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| allFKeys | boolean | |
| noFKeys | boolean | |
| JxfsPINFKeySet | (constructor of the class) | |

### 5.1.1 Properties

#### fk0 .. fk000 Properties (R)

**Type** ***boolean***
**Initial Value** *false*
**Description** Indicates if related function key is selected.
Note: fk00 and fk000 (hundred's and thousand's keys) are treated as sequences of two and three fk0, respectively.

| Value | Meaning |
|---|---|
| *false* | Function key is not selected. |
| *true* | Function key is selected. |

### 5.1.2 Methods

#### allFKeys Method

**Syntax** ***boolean allFKeys ()***
**Description** Returns *true* if all properties are set to *true*.

### noFKeys Method

**Syntax** ***boolean noFKeys ()***
**Description** Returns *true* if all properties are set to *false*.

### JxfsPINFKeySet Constructor

**Syntax** ***JxfsPINFKeySet (boolean fk0, boolean fk1, ... , boolean fk000)***
**Description** Constructor of the class.

## 5.2 JxfsPINFKeysSelection

This class provides properties and methods to query and select which function keys are active.

### Summary

**Implements :** **Extends :** **JxfsPINFKeySet**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| **No additional properties.** | | | |

| Method | Return | May use after |
|---|---|---|
| set*Property* | void | |
| setAllFKeys | void | |
| setNoFKeys | void | |
| JxfsPINFKeysSelection | (constructor of the class) | |

### 5.2.1 Properties

No additional properties to those inherited from base class *JxfsPINFKeySet*.

### 5.2.2 Methods

#### setAllFKeys Method

**Syntax** ***void setAllFKeys ()***
**Description** Sets all properties to *true*.

#### setNoFKeys Method

**Syntax** ***void setNoFKeys ()***
**Description** Sets all properties to *false*.

#### JxfsPINFKeysSelection Constructor

**Syntax** ***JxfsPINFKeysSelection (boolean fk0, ... , boolean fk000)***
**Description** Constructor of the class.

## 5.3 JxfsPINFDKeysSelection

This class provides properties and methods to query and select which function descriptor keys (FDKeys) are active.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| fdk01 | boolean | R/W | |
| fdk02 | boolean | R/W | |
| fdk03 | boolean | R/W | |
| fdk04 | boolean | R/W | |
| fdk05 | boolean | R/W | |
| fdk06 | boolean | R/W | |
| fdk07 | boolean | R/W | |
| fdk08 | boolean | R/W | |
| fdk09 | boolean | R/W | |
| fdk10 | boolean | R/W | |
| fdk11 | boolean | R/W | |
| fdk12 | boolean | R/W | |
| fdk13 | boolean | R/W | |
| fdk14 | boolean | R/W | |
| fdk15 | boolean | R/W | |
| fdk16 | boolean | R/W | |
| fdk17 | boolean | R/W | |
| fdk18 | boolean | R/W | |
| fdk19 | boolean | R/W | |
| fdk20 | boolean | R/W | |
| fdk21 | boolean | R/W | |
| fdk22 | boolean | R/W | |
| fdk23 | boolean | R/W | |
| fdk24 | boolean | R/W | |
| fdk25 | boolean | R/W | |
| fdk26 | boolean | R/W | |
| fdk27 | boolean | R/W | |
| fdk28 | boolean | R/W | |
| fdk29 | boolean | R/W | |
| fdk30 | boolean | R/W | |
| fdk31 | boolean | R/W | |
| fdk32 | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| set*Property* | void | |
| allFDKeys | boolean | |
| noFDKeys | boolean | |
| setAllFDKeys | void | |
| setNoFDKeys | void | |
| JxfsPINFDKeysSelection | (constructor of the class) | |

### 5.3.1 Properties

#### fdk01 .. fdk32 Properties (R/W)

**Type** ***boolean***
**Initial Value** *false*
**Description** Indicates if related function descriptor key is selected.

| Value | Meaning |
|---|---|
| *false* | Function descriptor key is not selected. |
| *true* | Function descriptor key is selected. |

### 5.3.2 Methods

#### allFDKeys Method

**Syntax** ***boolean allFDKeys ()***
**Description** Returns *true* if all properties are set to *true*.

#### noFDKeys Method

**Syntax** ***boolean noFDKeys ()***
**Description** Returns *true* if all properties are set to *false*.

#### setAllFDKeys Method

**Syntax** ***void setAllFDKeys ()***
**Description** Sets all properties to *true*.

#### setNoFDKeys Method

**Syntax** ***void setNoFDKeys ()***
**Description** Sets all properties to *false*.

#### JxfsPINFDKeysSelection Constructor

**Syntax** ***JxfsPINFDKeysSelection (boolean fdk01, ... , boolean fdk32)***
**Description** Constructor of the class.

## 5.4 JxfsPINFDKey

**Summary**

The JxfsPINFDKey class contains information about a function descriptor key (FDKey).

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyCode | int | R | |
| relativeX | int | R | |
| relativeY | int | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINFDKey | (constructor of the class) | |

### 5.4.1 Properties

**keyCode Property (R)**

**Type** ***int***

**Description** Specifies the code used for the function descriptor key FDKey.
Its value is one of the following:

**Value**
JXFS_PIN_FK_FDK01
JXFS_PIN_FK_FDK02
JXFS_PIN_FK_FDK03
JXFS_PIN_FK_FDK04
JXFS_PIN_FK_FDK05
JXFS_PIN_FK_FDK06
JXFS_PIN_FK_FDK07
JXFS_PIN_FK_FDK08
JXFS_PIN_FK_FDK09
JXFS_PIN_FK_FDK10
JXFS_PIN_FK_FDK11
JXFS_PIN_FK_FDK12
JXFS_PIN_FK_FDK13
JXFS_PIN_FK_FDK14
JXFS_PIN_FK_FDK15
JXFS_PIN_FK_FDK16
JXFS_PIN_FK_FDK17
JXFS_PIN_FK_FDK18
JXFS_PIN_FK_FDK19
JXFS_PIN_FK_FDK20
JXFS_PIN_FK_FDK21
JXFS_PIN_FK_FDK22
JXFS_PIN_FK_FDK23
JXFS_PIN_FK_FDK24
JXFS_PIN_FK_FDK25
JXFS_PIN_FK_FDK26
JXFS_PIN_FK_FDK27
JXFS_PIN_FK_FDK28
JXFS_PIN_FK_FDK29
JXFS_PIN_FK_FDK30
JXFS_PIN_FK_FDK31
JXFS_PIN_FK_FDK32

**relativeX Property (R)**

**Type** ***int***

**Description** Specifies the FDKey position relative to the left hand side of the screen expressed as a percentage of the width of the screen. For this, the

FDKey position is defined by the point which results from perpendicular projection of the key center onto the edge of the screen.

#### relativeY Property (R)

**Type** ***Int***

**Description** Specifies the FDKey position relative to the top of the screen expressed as a percentage of the height of the screen. For this, the FDKey position is defined by the point which results from perpendicular projection of the key center onto the edge of the screen.

### 5.4.2 Methods

#### JxfsPINFDKey Constructor

**Syntax** ***JxfsPINFDKey (int keyCode, int relativeX, int relativeY)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Some parameter is out of range. |

## 5.5 JxfsPINReadMode

This class specifies the conditions for PIN keypad data entry when using *readData()* and *secureReadPIN()* methods.

### Summary

**Implements :** **Extends :** *JxfsType*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| activeFDKeys | **JxfsPINFDKeysSelection** | R/W | |
| activeFKeys | **JxfsPINFKeysSelection** | R/W | |
| terminateFDKeys | **JxfsPINFDKeysSelection** | R/W | |
| terminateFKeys | **JxfsPINFKeysSelection** | R/W | |
| autoEnd | boolean | R/W | |
| beepOnPress | boolean | R/W | |
| inputMode | int | R/W | |
| maxLength | int | R/W | |
| minLength | int | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINReadMode | (constructor of the class) | |

### 5.5.1 Properties

#### activeFDKeys Property (R/W)

**Type** ***JxfsPINFDKeysSelection***
**Initial Value** Null until open.
**Description** Indicates the set of function descriptor keys (FDKeys) enabled for subsequent input operations.

#### activeFKeys Property (R/W)

**Type** ***JxfsPINFKeysSelection***
**Initial Value** Null until open.
**Description** Indicates the set of function keys enabled for subsequent input operations.

#### terminateFDKeys Property (R/W)

**Type** ***JxfsPINFDKeysSelection***
**Initial Value** Null until open.
**Description** Specifies the set of function descriptor keys (FDKeys) that, if pressed during an input operation, will terminate a data entry.
It must be a subset of the set defined by *activeFDKeys.*

#### terminateFKeys Property (R/W)

**Type** ***JxfsPINFKeysSelection***
**Initial Value** Null until open.
**Description** Specifies the set of function keys that, if pressed during an input operation, will terminate a data entry.
It must be a subset of the set defined by *activeFKeys.*

### autoEnd Property (R/W)

**Type** ***boolean***
**Initial Value** *false*
**Description** Indicates the criteria used to terminate subsequent input operations.

If *maxLength* is set to 0, this property is ignored and input is only terminated by a termination key (see *terminateFKeys* and *terminateFDKeys* properties).

| Value | Meaning |
|---|---|
| *true* | PIN entry terminates when the maximun number of digits are entered (*maxLength* property). |
| *false* | PIN entry terminates when a termination key (*terminateFKey*s and *terminateFDKeys* properties) has been pressed. In this case, when *maxLength* is reached, numeric keys are disabled by the device service. |

### beepOnPress Property (R/W)

**Type** ***boolean***
**Initial Value** *false*
**Description** Specifies if the device must generate an audible sound at every key press or not.

| Value | Meaning |
|---|---|
| *false* | The device must not beep. |
| *true* | The device must beep. |

### inputMode Property (R/W)

**Type** ***int***
**Initial Value** JXFS_PIN_INPUT_COOKED
**Description** Specifies the input mode to be used in subsequent input operations.

| Value | Meaning |
|---|---|
| JXFS_PIN_INPUT_RAW | Each key pressed during an input operation will generate an intermediate event. These events will contain information about pressed keys. |
| JXFS_PIN_INPUT_COOKED | No intermediate events per key pressed are generated. Data entered during an input operation is provided in the *JxfsOperationCompleteEvent* event. |

### maxLength Property (R/W)

**Type** ***int***
**Initial Value** 8
**Description** Specifies the maximum number of digits which can be entered in an input operation.

If autoEnd is set to *true*, the input operation ends when this maximun number of digits has been entered.

If it is set to zero, the input operation does not end until a termination key is pressed (see *terminateKeys* and *terminateFDKeys* properties). If no termination keys are specified, the input operation will not terminate

until a *cancel()* operation is issued.

#### minLength Property (R/W)

| | |
|---|---|
| **Type** | ***int*** |
| **Initial Value** | 1 |
| **Description** | Specifies the minimum number of digits which must be entered for a valid input operation.<br><br>A value of JXFS_PIN_NO_MINUMUM_LENGTH (zero) indicates no minimum PIN length verification. |

### 5.5.2 Methods

#### JxfsPINReadMode Constructor

| | |
|---|---|
| **Syntax** | ***JxfsPINReadMode (JxfsPINFDKeysSelection activeFDKeys, JxfsPINFKeysSelection activeFKeys, JxfsPINFDKeysSelection terminateFDKeys, JxfsPINFKeysSelection terminateFKeys, boolean autoEnd, boolean beepOnPress, int inputMode, int maxLength, int minLength)*** |
| **Description** | Constructor of the class. |
| **Exceptions** | Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes. |

| **Value** | **Meaning** |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*activeFDKeys* is null.<br>*activeFKeys* is null.<br>*terminateFDKeys* is null.<br>*terminateFKeys* is null.<br>*inputMode* is not one of the listed values.<br>*maxLength* is less than *minLength*.<br>*minLength* is negative. |

## 5.6 JxfsPINReadMode2

This class specifies extended conditions for PIN keypad data entry when using the readData*()* and secureReadPIN*()* methods.

### Summary

**Implements :** **Extends :** ***JxfsPINReadMode***

| Property | Type | Access | Initialized after |
|---|---|---|---|
| eventOnStart | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINReadMode2 | (constructor of the class) | |

### 5.6.1 Properties

#### eventOnStart Property (R/W)

**Type** ***boolean***

**Initial Value** *false*.

**Description** Specifies if the service must send an intermediate event when the device is ready to accept user entered data.

| Value | Meaning |
|---|---|
| *false* | The service must not send the event |
| *true* | The service must send the event |

### 5.6.2 Methods

#### JxfsPINReadMode2 Constructor

**Syntax** ***JxfsPINReadMode2 (JxfsPINFDKeysSelection activeFDKeys, JxfsPINFKeysSelection activeFKeys, JxfsPINFDKeysSelection terminateFDKeys, JxfsPINFKeysSelection terminateFKeys, boolean autoEnd, boolean beepOnPress, int inputMode, int maxLength, int minLength, boolean eventOnStart)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*activeFDKeys* is null.<br>*activeFKeys* is null.<br>*terminateFDKeys* is null.<br>*terminateFKeys* is null.<br>*inputMode* is not one of the listed values.<br>*maxLength* is less than *minLength.*<br>*minLength* is negative. |

## 5.7 JxfsPINPressedKey

This class contains the data associated to a pressed key during an input operation.

**Summary**

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyCode | int | R | |
| keyType | int | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINPressedKey | (constructor of the class) | |

### 5.7.1 Properties

**keyCode Property (R)**

**Type** ***Int***
**Description** Code of key.

| Value | Meaning |
|---|---|
| JXFS_PIN_FK_NONE | If result of a *secureReadPIN()* operation and the key is a numeric function key. This value may be used to output substitution signs on a display. |
| JXFS_PIN_FK_FDK01 | |
| JXFS_PIN_FK_FDK02 | |
| JXFS_PIN_FK_FDK03 | |
| JXFS_PIN_FK_FDK04 | |
| JXFS_PIN_FK_FDK05 | |
| JXFS_PIN_FK_FDK06 | |
| JXFS_PIN_FK_FDK07 | |
| JXFS_PIN_FK_FDK08 | |
| JXFS_PIN_FK_FDK09 | |
| JXFS_PIN_FK_FDK10 | |
| JXFS_PIN_FK_FDK11 | |
| JXFS_PIN_FK_FDK12 | |
| JXFS_PIN_FK_FDK13 | |
| JXFS_PIN_FK_FDK14 | |
| JXFS_PIN_FK_FDK15 | |
| JXFS_PIN_FK_FDK16 | |
| JXFS_PIN_FK_FDK17 | |
| JXFS_PIN_FK_FDK18 | |
| JXFS_PIN_FK_FDK19 | |
| JXFS_PIN_FK_FDK20 | |
| JXFS_PIN_FK_FDK21 | |
| JXFS_PIN_FK_FDK22 | |
| JXFS_PIN_FK_FDK23 | |
| JXFS_PIN_FK_FDK24 | |
| JXFS_PIN_FK_FDK25 | |
| JXFS_PIN_FK_FDK26 | |
| JXFS_PIN_FK_FDK27 | |
| JXFS_PIN_FK_FDK28 | |
| JXFS_PIN_FK_FDK29 | |
| JXFS_PIN_FK_FDK30 | |
| JXFS_PIN_FK_FDK31 | |
| JXFS_PIN_FK_FDK32 | |

JXFS_PIN_FK_0
JXFS_PIN_FK_1
JXFS_PIN_FK_2
JXFS_PIN_FK_3
JXFS_PIN_FK_4
JXFS_PIN_FK_5
JXFS_PIN_FK_6
JXFS_PIN_FK_7
JXFS_PIN_FK_8
JXFS_PIN_FK_9
JXFS_PIN_FK_ENTER
JXFS_PIN_FK_CANCEL
JXFS_PIN_FK_CLEAR
JXFS_PIN_FK_BACKSPACE
JXFS_PIN_FK_HELP
JXFS_PIN_FK_DECPOINT
JXFS_PIN_FK_00
JXFS_PIN_FK_000

#### keyType Property (R)

**Type** ***int***

**Description** Type of key pressed

It can be one of the following values:

| Value | Meaning |
|---|---|
| JXFS_PIN_KP_FUNCTION | Function key. |
| JXFS_PIN_KP_FDKEY | Function descriptor key (FDKey). |

### 5.7.2 Methods

#### JxfsPINPressedKey Constructor

**Syntax** ***JxfsPINPressedKey (int keyCode, int keyType)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*keyCode* is not one of the listed values.<br>*keyType* is not one of the listed values. |

## 5.8 JxfsPINReadData

This class contains the data returned by a *JxfsOperationCompleteEvent* event for *readData()* and *secureReadPIN()* operations.

**Summary**

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| endReason | int | R | |
| pinLength | int | R | |
| pressedKeys | java.util.Vector | R | |
| readData | java.lang.String | R | |
| terminationKey | int | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINReadData | (constructor of the class) | |

### 5.8.1 Properties

**endReason Property (R)**

**Type** ***int***

**Description** Indicates the input operation termination reason.

It can be one of the following values:

| Value | Meaning |
|---|---|
| JXFS_PIN_COMP_AUTO | Input operation terminated because *maxLength* was reached. |
| JXFS_PIN_COMP_FK | A termination key was pressed. |
| JXFS_PIN_COMP_FDKEY | A termination FDKey was pressed |

**pinLength Property (R)**

**Type** ***int***

**Description** If *inputMode* property is set to JXFS_PIN_INPUT_RAW, it contains the count of keys pressed.
If *inputMode* property is set to JXFS_PIN_INPUT_COOKED, it contains the count of digits entered.

**pressedKeys Property (R)**

**Type** ***java.util.Vector***

**Description** Vector of *JxfsPINPressedKey* objects. It represents the list of all the keys pressed during the input operation.

If *inputMode* Property was set to JXFS_PIN_INPUT_RAW this property is optional and can be set to null. *Keys will also be received in* `JXFS_I_PIN_KEY_PRESSED` *Intermediate Events*

**readData Property (R)**

**Type** ***java.lang.String***

**Description** Cooked data entered in input operation.

| Value | Meaning |
|---|---|

| | |
|---|---|
| null | • if result of a *secureReadPIN()* operation.<br>• if result of a *readData()* operation and *inputMode* Property was set to JXFS_PIN_INPUT_RAW. |
| Non formatted string representation of numeric value entered.<br>Function keys are omitted. | • if result of a *readData ()* operation and *inputMode* Property was set to JXFS_PIN_INPUT_COOKED. |

#### terminationKey Property (R)

**Type** ***int***

**Description** Code of termination function key or FDKey if end reason was JXFS_PIN_COMP_FK or JXFS_PIN_COMP_FDKEY.

If termination reason was JXFS_PIN_COMP_AUTO, it is set to JXFS_PIN_FK_NONE.

### 5.8.2 Methods

#### JxfsPINReadData Constructor

**Syntax** ***JxfsPINReadData (int endReason, int pinLength, java.util.Vector pressedKeys, java.lang.String readData, int terminationKey)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*endReason* is not one of the listed values.<br>*pinLength* is negative.<br>*pressedKeys* is null and inputMode is JXFS_PIN_INPUT_COOKED.<br>*readData* is null and inputMode is JXFS_PIN_INPUT_COOKED.<br>*terminationKey* has an invalid value. |

## 5.9 JxfsPINFormats

This class provides properties and methods to query which PIN formats are supported by a PIN device service.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| fmt3624 | boolean | R | |
| fmtANSI | boolean | R | |
| fmtISO0 | boolean | R | |
| fmtISO1 | boolean | R | |
| fmtEC12 | boolean | R | |
| fmtEC13 | boolean | R | |
| fmtEC13_Rand | boolean | R | |
| fmtVISA | boolean | R | |
| fmtDiebold | boolean | R | |
| fmtDieboldC0 | boolean | R | |
| fmtEMV | boolean | R | |
| fmtISO3 | boolean | R | |
| fmtVISA3 | boolean | R | |
| fmtItAP | boolean | R | |
| fmtBanksys | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINFormats | (constructor of the class) | |

### 5.9.1 Properties

#### fmt3624 Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN left justified, filled with padding characters, PIN length 4-16 digits.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

#### fmtANSI Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, minimum 12 digits without check number).

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

#### fmtISO0 Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character

0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, no minimum length specified, missing digits are filled with 0x00).

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtISO1 Property (R)

**Type** ***boolean***

**Initial Value** Depends on device

**Description** Indicates if the device supports the format: PIN is preceded by 0x01 and the length of the PIN (0x04 to 0x0C), padding characters are taken from a transaction field (10 digits).

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtEC12 Property (R)

**Type** ***boolean***

**Initial Value** Depends on device

**Description** Indicates if the device supports the format: similar to fmt3624, PIN only 4 digits.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtEC13 Property (R)

**Type** ***boolean***

**Initial Value** Depends on device

**Description** Indicates if the device supports the format: PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from X’0’ through X’F’.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtEC13_Rand Property (R)

**Type** ***boolean***

**Initial Value** Depends on device

**Description** Indicates if the device supports the format: PIN is preceded by the length (digit), PIN length 4-6 digits, padded with random data.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtVISA Property (R)

**Type** ***boolean***

**Initial Value** Depends on device

**Description** Indicates if the device supports the format: PIN is preceded by the length (digit), PIN length 4-6 digits. If the PIN length is less than six digits the PIN is filled with X’0’ to the length of six, the padding character can range from X ' 0 ' through X ' 9 ' (This format is also referred to as VISA2).

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtDiebold (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtDieboldC0 (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN with the length of 4 to 12 digits, each one with a value of X'0' to X'9', is preceded by the one-digit coordination number with a value from X'0' to X'F', padded with the padding character with a value from X'0' to X'F' and may be not encrypted, single encrypted or double encrypted.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtEMV

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the EMV PIN format: PIN is preceded by 0x02 and the length of the PIN (0x04 to 0x0C), padded with the padding character 0x0F to the right. PIN is formatted up to 248 bytes according to EMV specification V 4.0and finally. PIN is encrypted with a RSA key.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtISO3 Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN is preceded by 0x03 and the length of the PIN (0x04 to 0x0C), padding characters sequentially or randomly chosen, XORed with digits from PAN.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### fmtVISA3 Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN with the length of 4 to 12 digits, each one with a value of X'0' to X'9', is followed by a delimiter with the value of X'F' and then padded by the padding character with a value between X'0' to X'F'.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

#### fmtItAP Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN is encrypted and formatted according to the Italian format AP specifications.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

#### fmtBanksys Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the format: PIN is encrypted and formatted according to the Banksys Pin Block specifications.

| Value | Meaning |
|---|---|
| *false* | Format is not supported. |
| *true* | Format is supported. |

### 5.9.2 Methods

#### JxfsPINFormats Constructor deprecated

**Syntax** ***JxfsPINFormats (boolean fmt3624, boolean fmtANSI, boolean fmtSO0, boolean fmtSO1, boolean fmtEC12, boolean fmtEC13, boolean fmtEC13_Rand, boolean fmtVISA, boolean fmtDiebold, boolean fmtDieboldC0)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

#### JxfsPINFormats Constructor

**Syntax** ***JxfsPINFormats (boolean fmt3624, boolean fmtANSI, boolean fmtSO0, boolean fmtSO1, boolean fmtEC12, boolean fmtEC13, boolean fmtEC13_Rand, boolean fmtVISA, boolean fmtDiebold, boolean fmtDieboldC0, boolean fmtEMV)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

#### JxfsPINFormats Constructor

**Syntax** ***JxfsPINFormats (boolean fmt3624, boolean fmtANSI, boolean fmtSO0, boolean fmtSO1, boolean fmtEC12, boolean fmtEC13, boolean fmtEC13_Rand, boolean fmtVISA, boolean fmtDiebold, boolean fmtDieboldC0, boolean fmtEMV, boolean fmtISO3, boolean fmtVISA3, boolean fmtItAP, boolean fmtBanksys )***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on

JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

## 5.10 JxfsPINValidationAlgorithms

This class provides properties and methods to query which algorithms for PIN validation are supported by a PIN device service.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| valDES | boolean | R | |
| valEC | boolean | R | |
| valVISA | boolean | R | |
| valDESOffset | boolean | R | |
| valEMVRSA | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINValidationAlgorithms | (constructor of the class) | |

### 5.10.1 Properties

#### valDES Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports DES algorithm for PIN validation.

| Value | Meaning |
|---|---|
| *false* | Algorithm is not supported. |
| *true* | Algorithm is supported. |

#### valEC Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports EUROCHEQUE algorithm for PIN validation.

| Value | Meaning |
|---|---|
| *false* | Algorithm is not supported. |
| *true* | Algorithm is supported. |

#### valVISA Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports VISA algorithm for PIN validation.

| Value | Meaning |
|---|---|
| *false* | Algorithm is not supported. |
| *true* | Algorithm is supported. |

#### valDESOffset Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports DES offset generation algorithm.

| Value | Meaning |
|---|---|
| *false* | Offset generation is not supported. |
| *true* | Offset generation is supported. |

#### valEMVRSA Property (R)

| | |
|---|---|
| **Type** | ***boolean*** |
| **Initial Value** | Depends on device |
| **Description** | Indicates if the device supports EMV RSA algorithm for PIN generation |

| **Value** | **Meaning** |
|---|---|
| *false* | EMV RSA algorithm is not supported. |
| *true* | EMV RSA algorithm is supported. |

### 5.10.2 Methods

#### JxfsPINValidationAlgorithms Constructor

| | |
|---|---|
| **Syntax** | ***JxfsPINValidationAlgorithms (boolean valDES, boolean valEC, boolean valVISA, boolean valDESOffset)*** |
| **Description** | Constructor of the class. |

#### JxfsPINValidationAlgorithms Constructor

| | |
|---|---|
| **Syntax** | ***JxfsPINValidationAlgorithms (boolean valDES, boolean valEC, boolean valVISA, boolean valDESOffset, boolean valEMVRSA)*** |
| **Description** | Constructor of the class. |

## 5.11 JxfsPINChipPresentationModes

This class provides properties and methods to query which presentation algorithms for PIN chip validation are supported by a PIN device service.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| presentClear | boolean | R | |
| presentEMVRSAEnciphered | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINChipPresentationModes | (constructor of the class) | |

### 5.11.1 Properties

#### presentClear Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports presentation of a clear text PIN to a chip card.

| Value | Meaning |
|---|---|
| *false* | Presentation algorithm is not supported. |
| *true* | Presentation algorithm is supported. |

#### PresentEMVRSAEnciphered Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports presentation of a EMV RSA enciphered PIN Block to the chip card

| Value | Meaning |
|---|---|
| *false* | EMV RSA enciphered presentation algoritm is not supported. |
| *true* | EMV RSA enciphered presentation algoritm is supported. |

### 5.11.2 Methods

#### JxfsPINChipPresentationModes Constructor

**Syntax** ***JxfsPINChipPresentationModes (boolean presentClear)***
**Description** Constructor of the class.

#### JxfsPINChipPresentationModes Constructor

**Syntax** ***JxfsPINChipPresentationModes (boolean presentClear, boolean PresentEMVRSAEnciphered)***

## 5.12 JxfsPINValidationData

Abstract class.

The J/XFS PIN Validation Data is the root of a hierarchy of data objects that contain data for PIN verification and used in *validatePIN(),createOffset(), createPINBlock(), validatePINSecure(),createOffsetSecure(), createPINBlockSecure()* methods of JxfsSecurePINKeypad Device Control class.

### Summary

**Implements :** **Extends : JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| validationAlgorithm | int | R | |
| keyName | java.lang.String | R/W | |
| keyEncrKey | byte[] | R/W | |
| validationTrackNumber | int | R/W | |
| validationLength | int | R/W | |
| validationIndex | int | R/W | |
| offsetTrackNumber | int | R/W | |
| offsetLength | int | R/W | |
| offsetIndex | int | R/W | |
| ejectCurrent | boolean | R/W | |
| ejectWhenComplete | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |

### 5.12.1 Properties

#### validationAlgorithm Property (R)

**Type** ***int***

**Description** Validation algorithm for which this object is intended to be used. Set by the constructor of each of the specific subclasses of *JxfsPINValidationData* to one of the following values:

| Value | Meaning |
|---|---|
| JXFS_PIN_VAL_DES | DES PIN validation. |
| JXFS_PIN_VAL_EC | EUROCHEQUE PIN validation. |
| JXFS_PIN_VAL_VISA | VISA PIN validation. |

#### keyName Property (R/W)

**Type** ***java.lang.String***

**Description** Name of the key to be used by the algorithms.
If *keyEncrKey* property is other than **null**, then this key is used to decrypt the keyEncrKey encrypted key and its result is used instead.
If *keyEncrKey* property is **null**, then this key is directly used.

For *JxfsPINBlockData* subclass, it specifies the name of the key used to encrypt the formatted PIN for the first time, or **null** if no encryption is required..

#### keyEncrKey Property (R/W)

**Type** ***byte[]***

**Description** Optional encrypted (under *keyName*) key to be used for PIN validation.

For *JxfsPINBlockData* subclass, it specifies the name of the key used to format the once encrypted formatted PIN, or **null** if no second encryption is required.

#### validationTrackNumber Property (R/W)

**Type** ***int***
**Description** Track where validation data is located.
Optional property.

#### validationLength Property (R/W)

**Type** ***int***
**Description** Length of validation data.
Optional property.

#### validationIndex Property (R/W)

**Type** ***int***
**Description** Location of validation data from index zero.
Optional property.

#### offsetTrackNumber Property (R/W)

**Type** ***int***
**Description** Track where offset data is located.
Optional property.

#### offsetLength Property (R/W)

**Type** ***int***
**Description** Length of offset data.
Optional property.

#### offsetIndex Property (R/W)

**Type** ***int***
**Description** Location of offset data from index zero.
Optional property.

#### ejectCurrent Property (R/W)

**Type** ***boolean***
**Description** Set *true* to eject any card currently in reader.
Optional property.

#### ejectWhenComplete Property (R/W)

**Type** ***boolean***
**Description** Set *true* to eject card on completion.
Optional property.

### 5.12.2 Exceptions

Exception JXFS_E_PARAMETER_INVALID is thrown by the setter methods in the following cases:

- The value for an int property is negative.

## 5.13 JxfsPINValidationDataForDES

Class that contains data required for DES PIN validation.

### Summary

**Implements :** **Extends :** *JxfsPINValidationData*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| decimalTable | byte[] | R/W | |
| maxPIN | int | R/W | |
| noLeadingZero | boolean | R/W | |
| offset | byte[] | R/W | |
| offsetUsed | boolean | R/W | |
| paddingChar | byte | R/W | |
| validationData | byte [] | R/W | |
| validationDigits | int | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINValidationDataFor DES | (constructor of the class) | |

### 5.13.1 Properties

#### decimalTable Property (R/W)

**Type** ***byte[]***
**Description** ASCII decimalization table (16 character string containing ‘0’ to ‘9’). Used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

#### maxPIN Property (R/W)

**Type** ***int***
**Description** Maximum number of PIN digits to be used for validation.

#### noLeadingZero Property (R/W)

**Type** ***boolean***
**Description** If set to *true* and the first digit of result of the modulo 10 addition is a X’0’, it is replaced with X’1’ before performing the verification against the entered PIN. If set to *false*, a leading zero is allowed in entered PINs.

#### offset Property (R/W)

**Type** ***byte []***
**Description** Offset for the PIN block.
If this property is set to **null**, the offset is to be read from the card in the device.
Optional property.

#### offsetUsed Property (R/W)

**Type** ***boolean***
**Description** Specifies if offset is used for PIN validation.

#### paddingChar Property (R/W)

**Type** *byte*
**Description** Specifies the padding character for validation data.

#### validationData Property (R/W)

**Type** *byte []*
**Description** Validation data.
If this property is set to **null**, the validation data is to be read from the card in the device.

#### validationDigits Property (R/W)

**Type** *int*
**Description** Number of Validation digits to be used for validation.

### 5.13.2 Methods

#### JxfsPINValidationDataForDES Constructor

**Syntax** ***JxfsPINValidationDataForDES ( java.lang.String keyName, byte[] keyEncrKey, byte[] decimalTable, int maxPIN, boolean noLeadingZero, byte[] offset, boolean offsetUsed, byte paddingChar, byte[] validationData, int validationDigits)***

***JxfsPINValidationDataForDES ( java.lang.String keyName, byte[] keyEncrKey, int validationTrackNumber, int validationLength, int validationIndex, int offsetTrackNumber, int offsetLength, int offsetIndex, boolean ejectCurrent, ejectWhenComplete, byte[] decimalTable, int maxPIN, boolean noLeadingZero, byte paddingChar, byte[] validationData, int validationDigits)***
**Description** Constructors of the class.

### 5.13.3 Exceptions

Exception JXFS_E_PARAMETER_INVALID is thrown by the setter methods in the following cases:

- The value for an int property is negative.
- The value for decimal Table is null.

## 5.14 JxfsPINValidationDataForEC

Class that contains data required for EUROCHEQUE PIN validation.

### Summary

**Implements :** **Extends :** *JxfsPINValidationData*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| decimalTable | byte[] | R/W | |
| eurochequeData | byte[] | R/W | |
| firstEncDigits | int | R/W | |
| firstEncOffset | int | R/W | |
| PINVV | byte [] | R/W | |
| PINVVDigits | int | R/W | |
| PINVVOffset | int | R/W | |

| Method | Return | May use after |
|---|---|---|
| Get*Property* | *Property* | |
| Set*Property* | *void* | |
| JxfsPINValidationDataFor EC | (constructor of the class) | |

### 5.14.1 Properties

#### decimalTable Property (R/W)

**Type** ***byte[]***
**Description** ASCII decimalization table (16 character string containing ‘0’ to ‘9’). Used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

#### eurochequeData Property (R/W)

**Type** ***byte[]***
**Description** Track 3 Eurocheque data.

#### firstEncDigits Property (R/W)

**Type** ***int***
**Description** Number of digits to extract after first encryption.

#### firstEncOffset Property (R/W)

**Type** ***int***
**Description** Offset of digits to extract after first encryption.

#### PINVV Property (R/W)

**Type** ***byte []***
**Description** PIN Validation Value from track data.

#### PINVVDigits Property (R/W)

**Type** ***int***
**Description** Number of validation digits to extract for PVV.

#### PINVVOffset Property (R/W)

| | |
|---|---|
| **Type** | ***int*** |
| **Description** | Offset of digits to extract for PVV. |

### 5.14.2 Methods

#### JxfsPINValidationDataForEC Constructor

| | |
|---|---|
| **Syntax** | ***JxfsPINValidationDataForEC ( java.lang.String keyName, byte[] keyEncrKey, byte[] decimalTable, byte[] eurochequeData, int firstEncDigits, int firstEncOffset, byte[] PINVV, int PINVVDigits, int PINVVOffset)*** |
| **Description** | Constructor of the class. |

### 5.14.3 Exceptions

Exception JXFS_E_PARAMETER_INVALID is thrown by the setter methods in the following cases:

- The value for an int property is negative.
- The value for decimalTable, eurochequeData or PINVV is null.

## 5.15 JxfsPINValidationDataForVISA

Class that contains data required for VISA PIN validation.

### Summary

**Implements :** **Extends :** *JxfsPINValidationData*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| PAN | byte[] | R/W | |
| PINVV | byte[] | R/W | |
| PINVVDigits | int | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINValidationDataFor VISA | (constructor of the class) | |

### 5.15.1 Properties

#### PAN Property (R/W)

**Type** ***byte[]***
**Description** Primary Account Number from track data.

#### PINVV Property (R/W)

**Type** ***byte[]***
**Description** PIN Validation Value from track data.

#### PINVVDigits Property (R/W)

**Type** ***int***
**Description** Number of digits of PVV.

### 5.15.2 Methods

#### JxfsPINValidationDataForVISA Constructor

**Syntax** ***JxfsPINValidationDataForVISA ( java.lang.String keyName, byte[] keyEncrKey, byte[] PAN, byte[] PINVV, int PINVVDigits)***
**Description** Constructor of the class.

### 5.15.3 Exceptions

Exception JXFS_E_PARAMETER_INVALID is thrown by the setter methods in the following cases:

- The value for PINVVDigits is negative or zero.
- The value for PAN or PINVV is null.

## 5.16 JxfsPINOffsetData

Data class for data required for *createOffset*() method of JxfsSecurePINKeypad.

### Summary

**Implements :** **Extends :** *JxfsPINValidationData*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| decimalTable | byte[] | R/W | |
| maxPIN | int | R/W | |
| paddingChar | byte | R/W | |
| validationData | byte[] | R/W | |
| validationDigits | int | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINOffsetData | (constructor of the class) | |

### 5.16.1 Properties

#### decimalTable Property (R/W)

**Type** ***byte[]***
**Description** ASCII decimalization table (16 position byte array containing ‘0’ to ‘9’ characters). Used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

#### maxPIN Property (R/W)

**Type** ***int***
**Description** Maximum number of PIN digits to be used for validation.

#### paddingChar Property (R/W)

**Type** ***byte***
**Description** Specifies the padding character for validation data.

#### validationData Property (R/W)

**Type** ***byte[]***
**Description** Validation data.
If this property is set to **null**, the validation data is to be read from the card in the device.

#### validationDigits Property (R/W)

**Type** ***int***
**Description** Number of Validation digits to be used for validation.

### 5.16.2 Methods

#### JxfsPINOffsetData Constructor

**Syntax** ***JxfsPINOffsetData (java.lang.String keyName, byte[] keyEncrKey, byte[] decimalTable, int maxPIN, byte paddingChar, byte[] validationData, int validationDigits)***

| | |
|---|---|
| | ***JxfsPINOffsetData ( java.lang.String keyName, byte[] keyEncrKey, int validationTrackNumber, int validationLength, int validationIndex, boolean ejectCurrent, ejectWhenComplete, byte[] decimalTable, int maxPIN, byte paddingChar, int validationDigits)*** |
| **Description** | Constructor of the class. |

### 5.16.3 Exceptions

Exception JXFS_E_PARAMETER_INVALID is thrown by the setter methods in the following cases:

- The value for maxPIN or validationDigits is negative or zero.
- The value for decimalTable is null.

## 5.17 JxfsPINBlockData

Data class for data required for *createPINBlock()* method of JxfsSecurePINKeypad.

### Summary

**Implements :** **Extends :** *JxfsPINValidationData*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| customerData | byte[] | R/W | |
| paddingChar | byte | R/W | |
| pinBlockFormat | int | R/W | |
| XORData | byte[] | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINBlockData | (constructor of the class) | |

## 5.17.1 Properties

### customerData Property (R/W)

**Type** ***byte[]***

**Description** Used for ANSI, ISO-0 and ISO-1 algorithm to build the formatted PIN. For ANSI and ISO-0 the PAN (Primary Account Number) is used, for ISO-1 a ten digit transaction field is required. If not used a **null** is required.
Used for DIEBOLD with coordination number, as a two digit coordination number.
If this property is set to **null**, the validation data is to be read from the card in the device.
Used for EMV, with the unpredictable number (8 bytes) obtained from the chip card. This number is formatted unpacked. For example if the unpredictable number is "0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF", it is passed as follows "0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45 0x46"
If this property is set to **null**, the validation data is to be read from the card in the device.

### paddingChar Property (R/W)

**Type** ***byte***

**Description** Specifies the padding character.

### pinBlockFormat Property (R/W)

**Type** ***int***

**Description** Specifies the format of the PIN block.
Possible values are:

| Value | Meaning |
|---|---|
| JXFS_PIN_FMT_3624 | PIN left justified, filled with padding characters, PIN length 4-16 digits. The Padding Character is a Hexadecimal Digit in the range 0x00 to 0x0F. |

| | |
|---|---|
| JXFS_PIN_FMT_ANSI | PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, minimum 12 digits without check number) |
| JXFS_PIN_FMT_ISO0 | PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, no minimum length specified, missing digits are filled with 0x00) |
| JXFS_PIN_FMT_ISO1 | PIN is preceded by 0x01 and the length of the PIN (0x04 to 0x0C), padding characters are taken from a transaction field (10 digits). |
| JXFS_PIN_FMT_EC12 | (similar to WFS_PIN_FORM3624), PIN only 4 digits |
| JXFS_PIN_FMT_EC13 | PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from X’0’ through X’F’. |
| JXFS_PIN_FMT_EC13RAND | Format EC13, random padding. |
| JXFS_PIN_FMT_VISA | Indicates if the device supports the format: PIN is preceded by the length (digit), PIN length 4-6 digits. If the PIN length is less than six digits the PIN is filled with X’0’ to the length of six, the padding character can range from X ' 0 ' through X ' 9 ' (This format is also referred to as VISA2). |
| JXFS_PIN_FMT_DIEBOLD | PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted. |
| JXFS_PIN_FMT_DIEBOLDC0 | PIN with the length of 4 to 12 digits, each one with a value of X’0’ to X’9’, is preceded by the one-digit coordination number with a value from X’0’ to X’F’, padded with the padding character with a value from X’0’ to X’F’ and may be not encrypted, single encrypted or double encrypted. |
| JXFS_PIN_FMT_EMV | The PIN block is constructed as follows: PIN is preceded by 0x02 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, formatted up to 248 bytes of other data as defined within the EMV 4.0 specifications and finally encrypted with an RSA key. |
| JXFS_PIN_FMT_ISO3 | PIN is preceded by 0x03 and the length of the PIN (0x04 to 0x0C), padding characters sequentially or randomly chosen, XORed with digits from PAN. |

| | |
|---|---|
| JXFS_PIN_FMT_VISA3 | PIN with the length of 4 to 12 digits, each one with a value of X'0' to X'9', is followed by a delimiter with the value of X'F' and then padded by the padding character with a value between X'0' to X'F'. |
| JXFS_PIN_FMT_BANKSYS | PIN is encrypted and formatted according to the Banksys Pin Block specifications. |
| JXFS_PIN_FMT_ITAP | PIN is encrypted and formatted according to the Italian AP Pin Block specifications. |

#### XORData Property (R/W)

**Type** ***byte[]***

**Description** If the formatted PIN is encrypted twice to build the resulting PIN block, this data can be used to modify the result of the first encryption by an XOR-operation.

### 5.17.2 Methods

#### JxfsPINBlockData Constructor

**Syntax** ***JxfsPINBlockData ( java.lang.String keyName, byte[] keyEncrKey, byte[] customerData, byte paddingChar, int pinBlockFormat, byte[] XORData)***

***JxfsPINBlockData ( java.lang.String keyName, byte[] keyEncrKey, int validationTrackNumber, int validationLength, int validationIndex,, boolean ejectCurrent, ejectWhenComplete, byte paddingChar, int pinBlockFormat, byte[] XORData)***

**Description** Constructor of the class.
If KeyName specifies a RSA Key, RSA encryption will be performed.

### 5.17.3 Exceptions

Exception JXFS_E_PARAMETER_INVALID is thrown by the setter methods in the following cases:

- The value for pinBlockFormat is out of range.
- The value for XORData is null.

## 5.18 JxfsPINChipValidationData

Abstract class.

The J/XFS PIN Chip Validation Data is the root of a hierarchy of data objects that contain data for PIN chip verification and used in *validationPINChip()* method of JxfsSecurePINKeypad Device Control class.

### Summary

**Implements :** **Extends : JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| presentationMode | int | R/W | |
| chipProtocol | int | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |

### 5.18.1 Properties

#### presentationMode Property (R/W)

**Type** ***int***

**Description** Presentation mode for which this object is intended to be used.

Set by the constructor of each of the specific subclasses of *JxfsPINChipValidationData*.

Possible values are:

| Value | Meaning |
|---|---|
| JXFS_PIN_PRES_CLEAR | Clear text presentation of PIN to chip card device. |

#### chipProtocol Property (R/W)

**Type** ***int***

**Description** Protocol to be used with chip.
Possible values are:

| Value | Meaning |
|---|---|
| 0 .. 15 | Protocols T=0 .. T=15. |

### 5.18.2 Exceptions

Exception JXFS_E_PARAMETER_INVALID is thrown by the setter methods in the following cases:

- The value for presentationMode or chipProtocol is out of range.

## 5.19 JxfsPINChipValidationDataClear

Class that contains data required for Clear chip PIN validation.

### Summary

**Implements :**

**Extends :**
**JxfsPINChipValidationData**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| chipData | byte[] | R/W | |
| insertPosition | int | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINChipValidationDataClear | (constructor of the class) | |

### 5.19.1 Properties

#### chipData Property (R/W)

**Type** ***byte[]***
**Description** Data to be sent to the chip.

#### insertPosition Property (R/W)

**Type** ***int***
**Description** Contains the bit position where to insert the PIN in the *chipData* buffer (0 means is bit 0 of first byte, and so on).

### 5.19.2 Methods

#### JxfsPINChipValidationDataClear Constructor

**Syntax** ***JxfsPINChipValidationDataClear (int chipProtocol, byte[] chipData, int insertPosition)***
**Description** Constructor of the class.

### 5.19.3 Exceptions

Exception JXFS_E_PARAMETER_INVALID is thrown by the setter methods in the following cases:

- The value for insertPosition is negative.
- The value for chipData is null.

## 5.20 JxfsPINValidationResult

This class contains the result of a PIN validation operation.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| validationResult | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINValidationResult | (constructor of the class) | |

### 5.20.1 Properties

#### validationResult Property (R)

**Type** ***boolean***
**Description** *true* if PIN was validated, otherwise *false*.

### 5.20.2 Methods

#### JxfsPINValidationResult Constructor

**Syntax** ***JxfsPINValidationResult (boolean validationResult)***
**Description** Constructor of the class.

## 5.21 JxfsPINOffset

This class contains a PIN offset.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| offsetValue | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINOffset | (constructor of the class) | |

### 5.21.1 Properties

#### offsetValue Property (R)

**Type** ***byte[]***
**Description** A PIN Offset

### 5.21.2 Methods

#### JxfsPINOffset Constructor

**Syntax** ***JxfsPINOffset (byte[] offsetValue)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | *offsetValue* is null. |

## 5.22 JxfsPINBlock

This class contains a PIN block.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| PINBlockValue | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINBlock | (constructor of the class) | |

### 5.22.1 Properties

#### PINBlockValue Property (R)

**Type** ***byte[]***
**Description** A PIN Block.

### 5.22.2 Methods

#### JxfsPINBlock Constructor

**Syntax** ***JxfsPINBlock (byte[] PINBlockValue)***
**Description** Constructor of the class.

## 5.23 JxfsPINChipValidationResult

This class contains the result of a PIN chip validation operation.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| validationResult | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINChipValidationResult | (constructor of the class) | |

### 5.23.1 Properties

#### validationResult Property (R)

**Type** ***byte[]***
**Description** Data returned from chip.

### 5.23.2 Methods

#### JxfsPINChipValidationResult Constructor

**Syntax** ***JxfsPINChipValidationResult (byte[] validationResult)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | *validationResult* is null. |

## 5.24 JxfsPINCryptoModes

This class provides properties and methods to query which encryption modes are supported by a secure PIN device service.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| cryptDESECB | boolean | R | |
| cryptDESCBC | boolean | R | |
| cryptDESCFB | boolean | R | |
| cryptDESMAC | boolean | R | |
| cryptRSA | boolean | R | |
| cryptECMA | boolean | R | |
| cryptTRIDESECB | boolean | R | |
| cryptTRIDESCBC | boolean | R | |
| cryptTRIDESCFB | boolean | R | |
| cryptTRIDESMAC | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINCryptoModes | (constructor of the class) | |

### 5.24.1 Properties

#### cryptDESECB Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports Electronic Code Book encryption.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### cryptDESCBC Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports Cipher Block Chaining encryption.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### cryptDESCFB Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports Cipher Feed Back encryption.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

### cryptDESMAC Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports MAC calculation using CBC.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

### cryptRSA Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports RSA encryption.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

### cryptECMA Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports ECMA encryption.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

### cryptTRIDESECB Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports Triple DES with Electronic Code Book.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

### cryptTRIDESCBC Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports Triple DES with Cypher Block Chaining.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

### cryptTRIDESCFB Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports Triple DES with Cipher Feed Back.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### cryptTRIDESMAC Property (R)

**Type** ***boolean***

**Initial Value** Depends on device

**Description** The triple DES MACing algorithm as specified by ISO (ISO/IEC 9797-1: 1999) will result in only the last block of the MAC being encrypted with the full triple DES key. Specifically using block length n=64, Padding Method 1 ( when bPadding=0 ), MAC Algorithm 3, and MAC length m where 32<=m<=64.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

### 5.24.2 Methods

#### JxfsPINCryptoModes Constructor

**Syntax** ***JxfsPINCryptoModes (boolean cryptDESECB, boolean cryptDESCBC, boolean cryptDESCFB, boolean cryptDESMAC, boolean cryptRSA, boolean cryptECMA, boolean cryptTRIDESECB, boolean cryptTRIDESCBC, boolean cryptTRIDESCFB, boolean cryptTRIDESMAC)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

## 5.25 JxfsPINEMVCryptoModes

This class provides properties and methods to query which encryption modes are supported by a secure PIN device service for importing a RSA public key for EMV.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| EMVPlainTextCA | boolean | R | |
| EMVChecksumCA | boolean | R | |
| EMVEPICA | boolean | R | |
| EMVIssuer | boolean | R | |
| EMVICC | boolean | R | |
| EMVICCPIN | boolean | R | |
| EMVPKCSV1_5CA | boolean | R | |
| Hash_SHA1 | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINEMVCryptoModes | (constructor of the class) | |

### 5.25.1 Properties

#### EMVPlainTextCA Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the import of a Certification Authority plain text CA public key with no verification.

| Value | Meaning |
|---|---|
| *false* | Plain text mode not supported. |
| *true* | Plain text mode is supported. |

#### EMVChecksumCA Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the import of a Certification Authority plain text public key using the EMV 2000 verification algorithm.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### EMVEPICA Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the import of a Certification Authority public key using the self sign scheme defined in the EUROPAY International, EPI CA module Technical- Interface specification Version 1.4

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### EMVIssuer Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the import of an issuer public key as defined in the EMV 2000 book II.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### EMVICC Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the import of an ICC public key as defined in EMV specifications book II.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### EMVICCPIN Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the import of an ICC PIN public key as defined in EMV specifications book II.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### EMVPKCSV1_5CA Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the import of a certification Authority public key verified using a signature generated with a private key for which the public key is already loaded.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

#### Hash_SHA1 Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports SHA1 digest algorithm.

| Value | Meaning |
|---|---|
| *false* | Encryption mode is not supported. |
| *true* | Encryption mode is supported. |

### 5.25.2 Methods

#### JxfsPINEMVCryptoModes Constructor

**Syntax** ***JxfsPINEMVCryptoModes (boolean EMVPlainTextCA, boolean EMVChecksumCA, boolean EMVEPICA, boolean EMVIssuer,***

***boolean EMVICC, boolean EMVICCPIN, boolean EMVPKCSV1_5CA, boolean Hash_SHA1)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| **Value** | **Meaning** |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

## 5.26 JxfsPINKeyDetail

The J/XFS PIN Key Detail data class contains relevant information for an application about a key in the device key table.

### Summary

**Implements :** **Extends : JxfsType**

| Property | Type | Access |
|---|---|---|
| keyLoaded | boolean | R |
| keyName | java.lang.String | R |
| keyReload | boolean | R |
| keyUse | JxfsPINKeyUses | R |
| keyVerificationCode | byte [] | R |

| Constructor #1 | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINKeyDetail | keyLoaded | boolean |
| | keyName | java.lang.String |
| | keyReload | boolean |
| | keyUse | JxfsPINKeyUses |

| Constructor #2 | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINKeyDetail | keyLoaded | boolean |
| | keyName | java.lang.String |
| | keyReload | boolean |
| | keyUse | JxfsPINKeyUses |
| | keyVerificationCode | byte[] |

| Method | Return |
|---|---|
| get*Property* | *Property* |
| is*Property* | *Property* |

### 5.26.1 Properties

#### keyLoaded Property (R)

**Type** ***boolean***

**Description** Indicates whether the key has been loaded/imported.

| Value | Meaning |
|---|---|
| *true* | Key has been loaded/imported and is ready to be used. |
| *false* | Key is not operationally ready. |

#### keyName Property (R)

**Type** ***jave.lang.String***

**Description** Name of the key.

#### keyReload Property (R)

**Type** ***boolean***

**Description** Indicates whether the key can be loaded/imported just once.

| Value | Meaning |
|---|---|
| *true* | Key can be reloaded/re-imported. |
| *false* | Key can only be loaded/imported once. |

### keyUse Property (R)

**Type** ***JxfsPINKeyUses***
**Description** Type of access for which the key is intended to be used.

### keyVerificationCode Property (R)

**Type** ***byte[]***
**Remarks** Specifies the key verification code of the imported key.
Returns a zero length byte[] if key verification code generation is not supported.

## 5.26.2 Methods

### JxfsPINKeyDetail Constructor

**Syntax** ***JxfsPINKeyDetail (boolean keyLoaded, java.lang.String keyName, boolean keyReload, JxfsKeyUses keyUse)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*keyName* is null.<br>*keyUse* is null. |

### JxfsPINKeyDetail Constructor

**Syntax** ***JxfsPINKeyDetail (boolean keyLoaded, java.lang.String keyName, boolean keyReload, JxfsKeyUses keyUse, byte[] keyVerificationCode)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*keyName* is null.<br>*keyUse* is null.<br>keyVerificationCode is null. |

## 5.27 JxfsPINKeyToImport

The J/XFS PIN Key to Import data class contains data required as input for *importKey()* operation.

### Summary

**Implements :** **Extends : JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| Key | java.lang.String | R/W | |
| keyEncKey | java.lang.String | R/W | |
| keyReload | boolean | R/W | |
| keyUse | JxfsPINKeyUses | R/W | |
| keyValue | byte[] | R/W | |
| idKey | byte[] | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINKeyToImport | (constructor of the class) | |

### 5.27.1 Properties

#### key Property (R/W)

**Type** ***java.lang.String***
**Description** Name of the key being loaded.

#### keyEncKey Property (R/W)

**Type** ***java.lang.String***
**Description** Name of the key encrypting key that was used to encrypt the *keyValue* property data.
If this property is set to null, the key specified in *keyValue* is directely stored in the device‘s key table.

#### keyReload Property (R/W)

**Type** ***boolean***
**Description** Indicates whether the key can be loaded only once.

| Value | Meaning |
|---|---|
| *true* | Key can be loaded/imported may times. |
| *false* | Key can only be loaded/imported once. |

#### keyUse Property (R/W)

**Type** ***JxfsPINKeyUses***
**Description** Type of access for which the key is intended to be used.

#### keyValue Property (R/W)

**Type** ***byte[]***
**Description** Key value.

#### idKey Property (R/W)

**Type** ***byte[]***
**Description** Specifies the key owner identification or null.

## 5.27.2 Methods

### JxfsPINKeyToImport Constructor

**Syntax** ***JxfsPINKeyToImport ( java.lang.String key, java.lang.String keyEncKey, boolean keyReload, JxfsKeyUses keyUse, byte[] keyValue, byte[] idKey)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*key* is null.<br>*keyUse* is null.<br>*keyValue* is null.<br>*idKey* is null. |

## 5.28 JxfsPINEMVRSAKeyToImport

The JxfsPINEMVRSAKeyToImportdata class contains data required as input for *importEMVRSAKey()* operation. This class is similar to *JxfsPINKeyToImport* but it is specifically designed to address the key formats and security features defined by EMV.

This class is used to import the EMV RSA public keys. The RSA keys to import are provided either by the Certification Authority (VISA or MASTERCARD EUROPE), or by the EMV application in the chip card (ISSUER KEY and ICC KEY).

### Summary

**Implements :** **Extends : JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyName | java.lang.String | R/W | |
| keyUse | JxfsPINKeyUses | R/W | |
| idKey | byte[] | R/W | |
| EMVRSAIntegrityAlgorithm | JxfsPINEMVRSAIntegrityAlgorithm | R/W | |
| EMVRSAIntegrityData | byte [] | R/W | |
| signatureKeyName | java.lang.String | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINEMVRSAKeyToImport | (constructor of the class) | |

### 5.28.1 Properties

#### keyName Property (R/W)

**Type** ***java.lang.String***
**Description** Name of the key being loaded.

#### keyUse Property (R/W)

**Type** ***JxfsPINKeyUses***
**Description** Type of access for which the key is intended to be used. Only the properties *kuseRSAPublicEncrypt* and *kuseRSAPublicVerify* are supported

#### idKey Property (R/W)

**Type** ***byte[]***
**Description** Specifies the key owner identification or null.

#### EMVRSAIntegrityAlgorithm Property (R/W)

**Type** ***jxfsPINEMVRSAIntegrityAlgorithm***
**Description** Specifies the algorithm used to verify the integrity of the Certification Authority RSA public key. See JxfsPINEMVRSAIntegrity data class for more detailed information

#### EMVRSAIntegrityData Property (R/W)

**Type** ***byte[]***
**Description** Contains all the necessary data to complete the import RSA public key according to the EMVRSAIntegrityAlgorithm property.
The content of this parameter in dependant of the EMVRSAIntegrityAlgorithm parameter:
**plaintext_CA**: EMVRSAIntegrityData contains a DER encoded

PKCS#1 public key. No verification is possible. signatureKeyName is ignored.
**checksum_CA** : EMVRSAIntegrityData contains table 23 data, as specified in EMV 2000 Book 2.The plain text key is verified as defined within EMV2000 Book 2.signatureKeyName is ignored.
**EPI_CA** : EMVRSAIntegrityData contains the concatenation of tables 4 and 13, as specified in " Europay International, EPI CA Module Technical – Interface specification Version 1.4. These tables are also described in the EMV Clarifications Appendix
signatureKeyName is ignored.
**issuer** : EMVRSAIntegrityData contains the EMV public key certificate. It consists of the concatenation of :

- the key exponent length (1 byte),
- the key exponent value (variable length – EMV Tag value : '9F32'),
- the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value : '90') ,
- the remainder length (1 byte).
- The remainder value (variable length – EMV Tag value : '92'),
- the PAN length (1 byte)
- and the PAN value (variable length – EMV Tag value : '5A').

The device services will compare the leftmost three-eight digits of the PAN to the Issuer Identification Number retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book2 – Security & Key Management Version 4.0, Table 4.
signatureKeyName defines the previously loaded key used to verify the signature
**ICC** : EMVRSAIntegrityData contains the EMV public key certificate. It consists of the concatenation of :

- the key exponent length (1 byte),
- the key exponent value (variable length– EMV Tag value : '9F47'),
- the EMV certificate length (1 byte),
- the EMV certificate value (variable length – EMV Tag value :'9F46'),
- the remainder length (1 byte),
- the remainder value (variable length – EMV Tag value : '9F48'),
- the SDA length (1 byte), the SDA value (variable length),
- the PAN length (1 byte)
- and the PAN value (variable length – EMV Tag value : '5A'),

The Device Services will compare the PAN to the PAN retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book2 – Security & Key Management Version 4.0, Table 9.
signatureKeyName defines the previously loaded key used to verify the signature
**ICC_PIN** : EMVRSAIntegrityData contains the EMV public key certificate. It consists of the concatenation of :

- the key exponent length (1 byte),
- the key exponent value (variable length – EMV Tag value : '9F2E'),
- the EMV certificate length (1 byte),
- the EMV certificate value (variable length – EMV Tag value :'9F2D'),
- the remainder length (1 byte),
- the remainder value (variable length – EMV Tag value : '9F2F'),
- the SDA length (1 byte),
- the SDA value (variable length),
- the PAN length (1 byte)
- and the PAN value (variable length – EMV Tag value : '5A').

The Device services will compare the PAN to the PAN retrieved from the certificate. For more explanations, the reader can refer to EMVco, Book2 – Security & Key Management Version 4.0, Table 9.

signatureKeyName defines the previously loaded key used to verify the signature
**PKCSV1_5_CA** : EMVRSAIntegrityData contains the CA public key signed with the previously loaded public key specified in signatureKeyName. lpxImportData consists of the concatenation of EMV 2000 Book II Table 23) + 8 byte random number + Signature. The 8 byte random number is not used for validation; it is used to ensure the signature is unique. The Signature consists of all the bytes in the EMVRSAIntegrityData buffer after table 23 and the 8 byte random number

#### signatureKeyName Property (R/W)

**Type** ***java.lang.String***

**Description** Specifies the name of an asymmetric key, previously stored, which will be used to compute the certificate defined in JxfsPINEMVRSAIntegrity .

### 5.28.2 Methods

#### JxfsPINEMVKeyToImport Constructor

**Syntax** ***JxfsEMVPINKeyToImport ( java.lang.String keyName, JxfsPINKeyUses keyUse, byte[] idKey, JxfsPINEMVRSAIntegrityAlgorithm EMVRSAIntegrityAlgorithm, byte [] EMVRSAIntegrityData, java.lang.String signatureKeyName)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*keyName* is null or not set correctly.<br>*keyUse* is null or not set correctly.<br>*idKey* is null.<br>*EMVRSAIntegrityAlgorithm* is null or not set correctly<br>*signatureKeyName* is null or not set correctly |

## 5.29 JxfsPINInitialization

This class contains the result of a security module's initialization operation.

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| idKey | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINInitialization | (constructor of the class) | |

### 5.29.1 Properties

#### idKey Property (R)

**Type** ***byte[]***
**Description** Value of the ID key encrypted by the ID encryption key. Can be used as authorization for *importKey*() method.

Null if not supported by the device.

### 5.29.2 Methods

#### JxfsPINInitialization Constructor

**Syntax** ***JxfsPINInitialization (byte[] idKey)***
**Description** Constructor of the class.

## 5.30 JxfsPINKeyVerificationData

This class contains data returned after the completion of a *importKey()* operation..

**Summary**

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyVerCode | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINKeyVerificationData | (constructor of the class) | |

### 5.30.1 Properties

**keyVerCode Property (R)**

**Type** ***byte[]***
**Description** Key verification code data that can be used for verification of the loaded key.
For the *importEMVRSAPublicKey* , if applied , it contains the expiry date of the certificate in the following format YYYY-MM
Null if this function is not supported by the device.

### 5.30.2 Methods

**JxfsPINKeyVerificationData Constructor**

**Syntax** ***JxfsPINKeyVerificationData (byte[] keyVerCode)***
**Description** Constructor of the class.

## 5.31 JxfsPINCryptoData

**Summary**

The J/XFS PIN Cryptographic data class contains data required for encryption/decryption methods.

**Implements :** **Extends : JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| cryptoMode | int | R/W | |
| data | byte[] | R/W | |
| key | java.lang.String | R/W | |
| keyEncKey | byte[] | R/W | |
| paddingChar | byte | R/W | |
| startValue | byte[] | R/W | |
| startValueKey | java.lang.String | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINCryptoData | (constructor of the class) | |

### 5.31.1 Properties

#### cryptoMode Property (R/W)

**Type** ***Int***

**Description** Indicates the algorithm to be used.

| Value | Meaning |
|---|---|
| JXFS_PIN_CRYPT_MODE_DESECB | Electronic Code Book |
| JXFS_PIN_CRYPT_MODE_DESCBC | Cipher Block Chaining |
| JXFS_PIN_CRYPT_MODE_DESMAC | MAC calculation using CBC |
| JXFS_PIN_CRYPT_MODE_DESCFB | Cipher Feed Back |
| JXFS_PIN_CRYPT_MODE_RSA | RSA Encryption |
| JXFS_PIN_CRYPT_MODE_ECMA | ECMA Encryption |
| JXFS_PIN_CRYPT_MODE_TRIDESECB | Triple DES with Electronic Code Book |
| JXFS_PIN_CRYPT_MODE_TRIDESCBC | Triple DES with Cipher Block Chaining |
| JXFS_PIN_CRYPT_MODE_TRIDESCFB | Triple DES with Cipher Feed Back |
| JXFS_PIN_CRYPT_MODE_TRIDESMAC | *The triple DES MACing algorithm as specified by ISO (ISO/IEC 9797-1: 1999) will result in only the last block of the MAC being encrypted with the full triple DES key. Specifically using block length n=64, Padding Method 1 ( when bPadding=0 ), MAC Algorithm 3, and MAC length m where 32<=m<=64.* |

### data Property (R/W)

**Type** ***byte[]***
**Description** Data to be encrypted, decrypted or MACed.

### key Property (R/W)

**Type** ***java.lang.String***
**Description** Name of the key to be used in cryptographic operation.

### keyEncKey Property (R/W)

**Type** ***byte[]***
**Description** Encrypted key, under the key contained in *key* property, to be used in cryptographic operation.
If null, key contained in *key* property is used.

### paddingChar Property (R/W)

**Type** ***byte***
**Description** Specifies the padding character used.

### startValue Property (R/W)

**Type** ***byte[]***
**Description** DES and Triple DES initialization vector for the CBC, CFB and MAC.
If null, s*tartValueKey* property is used as the Initialization Vector.
If both are null the default is 16 hexadecimal digits 0x00.

### startValueKey Property (R/W)

**Type** ***java.lang.String***
**Description** Name of the stored key used to decrypt the s*tartValue* property to obtain the Initialization Vector.
If null, *startValue* is used as the initialization vector.

## 5.31.2 Methods

### JxfsPINCryptoData Constructor

**Syntax** ***JxfsPINCryptoData (int cryptoMode, byte[] data, java.lang.String key, byte[] keyEncKey, byte paddingChar, byte[] startValue, java.lang.String startValueKey)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*cryptoMode* is out of range.<br>*data* is null.<br>*key* is null or not set correctly<br>*keyEncKey* is null or not set correctly<br>*paddingChar* is null<br>*startValue* is null or not set correctly<br>*startValueKey* is null or not set correctly. |

## 5.32 JxfsPINMACData

The J/XFS PIN Cryptographic MAC data class contains data required for MAC generation operation.

It is a subclass of *JxfsPINCryptoData*.

### Summary

**Implements :** **Extends : JxfsPINCryptoData**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| compression | boolean | R/W | |
| compressionChar | byte | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| is*Property* | *Property* | |
| JxfsPINMACData | (constructor of the class) | |

### 5.32.1 Properties

#### compression Property (R/W)

**Type** ***boolean***
**Description** Specifies whether data is to be compressed (blanks removed) before building the MAC.

#### compressionChar Property (R/W)

**Type** ***byte***
**Description** If compression is ***true***, it specifies the representation of the blank character in the actual code table.

### 5.32.2 Methods

#### JxfsPINMACData Constructor

**Syntax** ***JxfsPINMACData (int cryptoMode, byte[] data, java.lang.String key, java.lang.String keyEncKey, byte paddingChar, byte[] startValue, java.lang.String startValueKey, boolean compression, byte compressionChar)***
**Description** Constructor of the class.

## 5.33 JxfsPINCryptoResult

The J/XFS PIN Cryptographic result data class contains data returned by cryptographic operations (*encrypt*, *decrypt* and *generateMAC*).

### Summary

**Implements :** **Extends : JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| cryptoResult | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINCryptoResult | (constructor of the class) | |

### 5.33.1 Properties

#### cryptoResult Property (R/W)

**Type** ***byte[]***
**Description** Data returned by a cryptographic operation.

### 5.33.2 Methods

#### JxfsPINCryptoResult Constructor

**Syntax** ***JxfsPINCryptoResult (byte[] cryptoResult)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | *cryptoResult* is null. |

## 5.34 JxfsPINKeyUses

**Summary**

This class provides properties and methods to query which type of access a key is intended for.

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| kuseEncDec | boolean | R/W | |
| kusePin | boolean | R/W | |
| kuseMac | boolean | R/W | |
| kuseKek | boolean | R/W | |
| kuseVek | boolean | R/W | |
| kuseMaster | boolean | R/W | |
| kuseRSAPublicEncrypt | boolean | R/W | |
| kuseRSAPublicVerify | boolean | R/W | |
| kuseRSAPrivateSign | boolean | R/W | |
| kuseRSAPrivate | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINKeyUses | (constructor of the class) | |

### 5.34.1 Properties

#### kuseEncDec Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used for encryption and decryption.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kusePin Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used for PIN functions.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kuseMac Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used for MAC generation.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kuseKek Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used as key encryption key.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kuseVek Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used as CBC Start Value encryption key.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kuseMaster Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used as Master encryption key.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kuseRSAPublicEncrypt Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used as Public key for RSA encryption or for EMV PIN Block creation.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kuseRSAPublicVerify Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used as a public key for RSA verification

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kuseRSAPrivate Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used as a private key for RSA encryption

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### kuseRSAPrivateSign Property (R/W)

**Type** ***boolean***

**Description** Indicates if the key may be used as a private key for RSA signature generation RSA encryption

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

### 5.34.2 Methods

#### JxfsPINKeyUses Constructor

**Syntax** ***JxfsPINKeyUses (boolean kuseEncDec, boolean kusePin, boolean kuseMac, boolean kuseKek, boolean kuseVek, boolean kuseMaster)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

#### JxfsPINKeyUses Constructor

**Syntax** ***JxfsPINKeyUses (boolean kuseEncDec, boolean kusePin, boolean kuseMac, boolean kuseKek, boolean kuseVek, boolean kuseMaster,***

***boolean kuseRSAPublicEncrypt, boolean kuseRSAPublicVerify, boolean kuseRSAPrivate, boolean kuseRSAPrivateSign)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| **Value** | **Meaning** |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

## 5.35 JxfsPINIdKeyModes

**Summary**

This class provides properties and methods to query which type of uses of ID keys are implemented.

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| idKeyInitialize | boolean | R | |
| idKeyImport | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINIdKeyModes | (constructor of the class) | |

### 5.35.1 Properties

#### idKeyInitialize Property (R)

**Type** ***boolean***

**Description** ID key is supported in the *initialize* method.

| Value | Meaning |
|---|---|
| *false* | Feature is not supported. |
| *true* | Feature is supported. |

#### idKeyImport Property (R)

**Type** ***boolean***

**Description** ID key is supported in the *importKey* method.

| Value | Meaning |
|---|---|
| *false* | Feature is not supported. |
| *true* | Feature is supported. |

### 5.35.2 Methods

#### JxfsPINIdKeyModes Constructor

**Syntax** ***JxfsPINIdKeyModes (boolean idKeyInitialize, boolean idKeyImport)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

## 5.36 JxfsPINEMVRSAIntegrityAlgorithm

This class provides properties and methods to query which type of verification is to be used for the verification of an RSA public key when it is loaded in the encryption module. This class is used with *JxfsPINEMVRSAKeyToImport* class.

Only one property can be set to *true*.

**Summary**

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| plaintext_CA | boolean | R/W | |
| checksum_CA | boolean | R/W | |
| EPI_CA | boolean | R/W | |
| issuer | boolean | R/W | |
| ICC | boolean | R/W | |
| ICC_PIN | boolean | R/W | |
| PKCSV1_5_CA | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| set*Property* | *Property* | |
| get*Property* | *Property* | |
| JxfsPINEMVRSAIntegrity Algorithm | (constructor of the class) | |

### 5.36.1 Properties

#### plaintext_CA Property (R/W)

**Type** ***boolean***

**Description** No integrity verification will be applied. It is used for keys provided by a Certification Authority with no verification.
The two parts of the key (modulus and exponent) are passed in clear mode as a DER encoded PKCS#1 public key. The key is loaded directly in the encryption module. This method is used by VISA

| Value | Meaning |
|---|---|
| *false* | This verification method is not applied |
| *true* | This verification method is applied |

#### checksum_CA Property (R/W)

**Type** ***boolean***

**Description** A plaintext CA public key is imported using the EMV 2000 Book II verification algorithm and it is verified before being loaded in the encryption module.
the checksum value is computed on the contents of all parts of the certification Authority public key (See EMV 2000, book 2 , p 71, table 13)

| Value | Meaning |
|---|---|
| *false* | This verification method is not applied |
| *true* | This verification method is applied |

#### EPI_CA Property (R/W)

**Type** ***boolean***

**Description** Use of EPI CA (MASTERCARD EUROPE) Key self signed integrity verification method. This key is provided in self signed format.

| Value | Meaning |
|---|---|
| *false* | This verification method is not applied |
| *true* | This verification method is applied |

#### issuer Property (R/W)

**Type** ***boolean***

**Description** An Issuer public key is imported as defined in EMV 2000 Book

| Value | Meaning |
|---|---|
| *false* | This verification method is not applied |
| *true* | This verification method is applied |

#### ICC Property (R/W)

**Type** ***boolean***

**Description** An ICC public key is imported as defined in EMV 2000 Book II

| Value | Meaning |
|---|---|
| *false* | This verification method is not applied |
| *true* | This verification method is applied |

#### ICC_PIN Property (R/W)

**Type** ***boolean***

**Description** An ICC PIN public key is imported as defined in EMV 2000 Book II

| Value | Meaning |
|---|---|
| *false* | This verification method is not applied |
| *true* | This verification method is applied |

#### PKCSV1_5_CA Property (R/W)

**Type** ***boolean***

**Description** A Certification Authority CA public key is imported and verified using a signature generated with a private key for which the public key is already loaded

| Value | Meaning |
|---|---|
| *false* | This verification method is not applied |
| *true* | This verification method is applied |

### 5.36.2 Methods

#### JxfsPINRSAIntegrityAlgorithm Constructor

**Syntax** ***JxfsPIN RSAIntegrityAlgorithm (boolean plainText_CA, boolean checksum_CA ,boolean EPI_CA, boolean issuer, boolean ICC, boolean ICC_PIN, boolean PKCSV1_5_CA )***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false* or more then one parameter is set to *true*. |

## 5.37 JxfsSHA1Data

Data class containing data
-As an input for computing a digest using a SHA_1 algorithm.
-As an output for the result of "SHA-1" algorithm

### Summary

**Implements :** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| length | int | R | |
| SHA1Data | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| *getProperty* | *Property* | |
| JxfsSHA1Data | (constructor of the class) | |

## 5.37.1 Properties

### length Property (R)

**Type** ***int***
**Description** This property is deprecated and will be always ignored.

### SHA1Data Property (R)

**Type** ***byte[]***
**Description** Data to be hashed if it is an input parameter
Digest data (result)

## 5.37.2 Methods

### JxfsSHA1Data Constructor

**Syntax** ***JxfsSHA1Data (int length, byte [] SHA1Data )***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | SHA1Data array is empty or null. |

## 5.38 JxfsPINImportRSAPublicKey

The JxfsPINImportRSAPublicKey data class contains data required as input for *importRSAPublicKey()* operation. This class is similar to *JxfsPINKeyToImport* but it is specifically designed to address the RSA public key formats.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| key | java.lang.String | RW | |
| keyUse | JxfsPINKeyUses | RW | |
| keyValue | byte[] | RW | |
| signatureKey | java.lang.String | RW | |
| RSASignatureAlgorithm | JxfsPINRSASignatureAlgo | RW | |
| signature | byte[] | RW | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINImportRSAPublicKey | (Constructor of the class) | |

### 5.38.1 Properties

#### key Property (R/W)

**Type** ***java.lang.String***
**Description** Name of the key being loaded.

#### keyUse Property (R/W)

**Type** ***JxfsPINKeyUses***
**Description** Type of access for which the key is intended to be used.
The valid properties for this kind of key are kuseRSAPublicEncrypt and kuseRSAPublicVerify

#### KeyValue Property (R/W)

**Type** ***byte []***
**Description** Specifies the value of the public RSA key to be loaded.
It is a PKCS #1 formatted RSA public key represented in DER encoded ASN.1.

#### signatureKey Property (R/W)

**Type** ***java.lang.String***
**Description** Specifies the name of an asymmetric key, previously stored in the encryptor, which will be used to verify the signature passed in the signature property.

#### RSASignatureAlgorithm Property (R/W)

**Type** ***JxfsPINRSASignatureAlgo***
**Description** Specifies the algorithm used to generate the signature specified in signature property.

#### signature Property (R/W)

**Type** ***byte []***
**Description** Contains the signature associate with the key being imported. The Signature is used to validate the key has been received from a trusted sender. Contains NULL when no key validation is required.

## 5.38.2 Methods

### JxfsPINImportRSAPublicKey Constructor

**Syntax**

***JxfsPINImportRSAPublicKey ( java.lang.String key,***
***JxfsPINKeyUses keyUse,***
***byte[] keyValue,***
***java.lang.String signatureKey,***
***JxfsPINRSASignatureAlgo RSASignatureAlgorithm,***
***byte [] signature)***

**Description**

Constructor of the class.

**Exceptions**

Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*key* is null or not set correctly.<br>*keyUse* is null or not set correctely.<br>*keyValue* is null or not set correctly.<br>signaturekey is null or not set correctly.<br>*RSASignatureAlgorithm* is null or not set correctly.<br>*signature* is null or not set correctly. |

## 5.39 JxfsPINExportRSAPublicKey

The JxfsPINExportRSAPublicKey data class contains information data that specifies the RSA public key to export

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| key | java.lang.String | R | Name of the key to export |
| keyType | JxfsPINRSAKeyType | R | Type of the key |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINExportRSAPublic Key | (Constructor of the class) | |

### 5.39.1 Properties

#### key Property (R)

**Type** ***java.lang.String***
**Description** Specifies the name of the public key to be exported. This can either be the name of a key-pair generated through *generateRSAKeyPair* operation or the name of one of the default key-pairs installed during manufacture the exported RSA public key part.

#### keyType Property (R)

**Type** ***JxfsPINRSAKeyType***
**Description** Specifies the PIN device RSA Key to export.

### 5.39.2 Methods

#### JxfsPINExportRSAPublicKey Constructor

**Syntax** ***JxfsPINExportRSAPublicKey (java.lang.String key, JxfsPINRSAKeyType keyType )***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*key* is null. or not set correctly<br>*keyType* is null or not set correctly |

## 5.40 JxfsPINExportedRSAPublicKey

The JxfsPINExportedRSAPublicKey data class contains data returned on *JxfsOperationCompleteEvent* of the *exportRSAPublicKey()* operation.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyValue | byte[] | R | |
| RSASignatureAlgorithm | JxfsPINRSASignatureAlgo | R | |
| signature | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINExportedRSAPublicKey | (Constructor of the class) | |

### 5.40.1 Properties

#### KeyValue Property (R)

**Type** ***byte []***
**Description** Contains the exported RSA public key part.

#### RSASignatureAlgorithm Property (R)

**Type** ***JxfsPINRSASignatureAlgo***
**Description** Specifies the algorithm used to generate the signature specified in signature property.

#### signature Property (R)

**Type** ***byte []***
**Description** Contains the signature of the RSA public key exported.

### 5.40.2 Methods

#### JxfsPINExportedRSAPublicKey Constructor

**Syntax** ***JxfsPINExportedRSAPublicKey (byte[] keyValue, JxfsPINRSAHashAlgorithms hashAlgorithm, JxfsPINRSASignatureAlgo RSASignatureAlgorithm, JxfsPINRSAKeyType signaturekey, byte [] signature ) throws JxfsException.***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>- keyValue is a null reference.<br>- hashAlgorithm is a null reference.<br>- RSASignatureAlgorithm is a null reference.<br>- signaturekey is a null reference.<br>- signature is a null reference. |

## 5.41 JxfsPINImportRSADESEncipheredPublicKey

The JxfsPINImportRSADESEncipheredPublicKey data class contains data required as input for *importRSADESEncipheredPublicKey()* operation. This class is similar to *JxfsPINKeyToImport* but it is specifically designed to address the RSA enciphered public key formats.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| key | java.lang.String | R/W | |
| keyUse | JxfsPINKeyUses | R/W | |
| keyValue | byte[] | R/W | |
| encipherAlgorithm | JxfsPINRSASignatureAlgo | R/W | |
| HashAlgorithm | JxfsPINRSAHashAlgorithms | R/W | |
| HashData | byte [] | R/W | |
| signatureKey | java.lang.String | R/W | |
| RSASignatureAlgorithm | JxfsPINRSASignatureAlgo | R/W | |
| signature | byte[] | R/W | |
| keyVerification | byte[] | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINImportRSADESEncipheredPublicKey | (Constructor of the class) | |

### 5.41.1 Properties

#### key Property (R/W)

**Type** ***java.lang.String***
**Description** Name of the key being loaded.

#### keyUse Property (R/W)

**Type** ***JxfsPINKeyUses***
**Description** Type of access for which the key is intended to be used.

#### KeyValue Property (R/W)

**Type** ***byte []***
**Description** Specifies the value of the public RSA key to be loaded. It contains the concatenation of the random number (when present) and enciphered key

#### encipherAlgorithm Property (R/W)

**Type** ***JxfsPINRSASignatureAlgo***
**Description** Specifies the RSA algorithm that is used, along with the private key of the PIN, to decipher the imported key.

#### hashAlgorithm Property (R/W)

**Type** ***JxfsPINRSAHashAlgorithms***
**Description** Specifies the algorithm used to generate the Hash value for the key

#### hashData Property (R/W)

**Type** ***byte []***

**Description** The Hash data is used to verify the key is still valid after it has been stored by the PIN. The PIN runs the stored key through the algorithm described by hashAlgorithm. The result should be identical to the value contained within hashData .

#### signatureKey Property (R/W)

**Type** ***java.lang.String***
**Description** Specifies the name of an asymmetric key, previously stored, which will be used to verify the signature passed in the signature property.

#### RSASignatureAlgorithm Property (R/W)

**Type** ***JxfsPINRSASignatureAlgo***
**Description** Specifies the algorithm used to generate the signature specified in signature property.

#### signature Property (R/W)

**Type** ***byte []***
**Description** Contains the signature associate with the key being imported. The signature is used to validate the key has been received from a trusted sender. Contains an empty array when no key validation is required, and when the key is transmitted by a PKCS#7 message.

#### keyVerification Property (R/W)

**Type** ***byte []***
**Description** Contains the key verification code data that can be used for verification of the loaded key, *null* if device does not have that capability

### 5.41.2 Methods

#### JxfsPINImportRSADESEncipheredPublicKey Constructor

**Syntax** ***JxfsPINImportRSADESEncipheredPublicKey ( java.lang.String key, JxfsPINKeyUses keyUse, byte[] keyValue, JxfsPINRSASignatureAlgo encipherAlgorithm, JxfsPINRSAHashAlgorithms hashAlgorithm, byte [] hashData, java.lang.String signatureKey, JxfsPINRSASignatureAlgo RSASignatureAlgorithm, byte [] signature, byte [] keyVerification )***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|

| | |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*key* is null or not set correctly.<br>*keyUse* is null or not set correctly.<br>*keyValue* is null or not set correctly<br>*encipherAlgorithm* is null or not set correctly.<br>*hashAlgorithm* is null or not set correctly.<br>*hashData* is null.<br>*signatureKey* is null or not set correctly.<br>*RSASignatureAlgorithm* is null or not set correctly.<br>*signature* is null or not set correctly.<br>*keyVerification* is null or not set correctly. |

## 5.42 JxfsPINExportRSADESEncipheredPublicKey

The JxfsPINExportRSADESEncipheredPublicKey data class contains data returned on *JxfsOperationCompleteEvent* of the *exportRSADESEncipheredPublicKey()* operation. This class is used to export the public part of RSA keys.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyValue | byte[] | R | |
| EncipherAlgorithm | JxfsPINRSASignatureAlgo | R | |
| hashAlgorithm | JxfsPINRSAHashAlgorithms | R | |
| hashData | byte [] | R | |
| RSASignatureAlgorithm | JxfsPINRSASignatureAlgo | R | |
| signatureKey | JxfsPINRSAKeyType | R | |
| signature | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *Property* | |
| JxfsPINExportRSADESEncipheredPublicKey | (Constructor of the class) | |

### 5.42.1 Properties

#### keyValue Property (R)

**Type** ***byte []***
**Description** Contains the exported RSA public key part.

#### EncipherAlgorithm Property (R)

**Type** ***JxfsPINRSASignatureAlgo***
**Description** Specifies the enciphering algorithm used for the creation of the signature for the exported RSA public key.

#### hashAlgorithm Property (R)

**Type** ***JxfsPINRSAHashAlgorithms***
**Description** Specifies the hash algorithm used for the creation of the signature for the exported RSA public key.

#### hashData Property (R)

**Type** ***byte[]***
**Description** The Hash data is used to verify if the key is still valid after it has been stored in the PIN. The PIN runs the stored key through the algorithm described by hashAlgorithm. The result should be identical to the value contained within hashData.

#### RSASignatureAlgorithm Property (R)

**Type** ***JxfsPINRSASignatureAlgo***
**Description** Specifies the algorithm used to generate the signature specified in signature property.

#### signatureKey Property (R)

**Type** ***JxfsPINRSAKeyType***

**Description** Specifies the private key used to generate the signature returned in the signature property

### signature Property (R)

**Type** ***byte []***
**Description** Contains the signature of the RSA public Key exported.

## 5.42.2 Methods

### JxfsPINExportRSADESEncipheredPublicKey Constructor

**Syntax** ***JxfsPINExportRSADESEncipheredPublicKey (byte[] keyValue, JxfsPINRSASignatureAlgo encipherAlgorithm, JxfsPINRSAHashAlgorithms hashAlgorithm, byte[] hashData, JxfsPINRSASignatureAlgo RSASignatureAlgorithm, JxfsPINRSAKeyType signatureKey , byte [] signature )***
**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*keyValue* is null or not set correctly<br>*encipherAlgorithm* is null or not set correctly.<br>*hashAlgorithm* is null or not set correctly.<br>*hashData* is null.<br>*signatureAlgo* is null or not set correctly.<br>*signatureAlgorithm* is null or not set correctly.<br>*SignatureKey* is null or not set correctly.<br>*signature* is null or not set correctly. |

## 5.43 JxfsPINGenerateRSAKeyPair

The JxfsPINGenerateRSAKeyPair data class contains data required as input for *generateRSAKeyPair()* operation.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| key | java.lang.String | R | |
| keyUse | JxfsPINKeyUses | R | |
| modulusLength | int | R | |
| exponentValue | JxfsPINRSAExponent | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINGenerateRSAKeyPair | (Constructor of the class) | |

## 5.43.1 Properties

### key Property (R)

**Type** ***java.lang.String***
**Description** Name of the key pair o be generated.

### keyUse Property (R)

**Type** ***JxfsPINKeyUses***
**Description** Type of access for which the key is intended to be used.

### modulusLength Property (R)

**Type** ***int***
**Description** Specifies the length of the modulus of the generated RSA key Pair

### exponentValue Property (R)

**Type** ***JxfsPINRSAExponent***
**Description** Specifies the value of the exponent of the generated RSA key pair.

## 5.43.2 Methods

### JxfsPINGenerateRSAKeyPair Constructor

**Syntax** ***JxfsPINGenerateRSAKeyPair (java.lang.String key, JxfsPINKeyUses keyUse, int modulusLength, JxfsPINRSAExponent exponentValue)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*key* is null or not set correctly.<br>*keyUse* is null or not set correctly.<br>*keyValue* is null or not set correctly<br>*modulusLength* is equal to 0.<br>*exponentValue* null or not set correctly. |

## 5.44 JxfsPINExportId

The JxfsPINExportId data class contains data retrieved by the PIN device and which uniquely identifies the PIN device (e.g.: serial number). This data are as output for *exportPINId()* operation.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| PINId | byte[] | R/W | |
| hashAlgorithm | JxfsPINRSAHashAlgorithms | R/W | |
| RSASignatureAlgorithm | JxfsPINRSASignatureAlgo | R./W | |
| signature | byte[] | R/W | |

| Method | Return | May use after |
|---|---|---|
| Get*Property* | *Property* | |
| Set*Property* | *void* | |
| JxfsPINExportId | (Constructor of the class) | |

### 5.44.1 Properties

#### PINId Property (R/W)

**Type** ***byte []***
**Description** Specifies the item that is unique to the PIN device. This data is vendor dependant item

#### hashAlgorithm Property (R/W)

**Type** ***JxfsPINRSAHashAlgorithms***
**Description** Specifies the hash algorithm used during the creation of the signature of the security item. to decipher the imported key.

#### RSASignatureAlgorithm Property (R/W)

**Type** ***JxfsPINRSASignatureAlgo***
**Description** Specifies the algorithm used to generate the signature specified in signature property.

#### signature Property (R/W)

**Type** ***byte []***
**Description** Contains the signature associated with the PINid. The Signature is used to validate the PINid.

### 5.44.2 Methods

#### JxfsPINExportId Constructor

**Syntax** ***JxfsPINExportId ( byte [] PINId, JxfsPINRSAHashAlgorithms hashalgorithm, JxfsPINRSASignatureAlgo RSASignatureAlgorithm, byte [] signature)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

## 5.45 JxfsPINExportCertificate

The JxfsPINExportCertificate data class contains data required as output for *exportCertificate()* operation. It specifies the certificate type to export and the certificate itself.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| certificateType | JxfsPINCertificateType | R/W | |
| certificate | byte[] | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| JxfsPINExportCertificate | (Constructor of the class) | |

### 5.45.1 Properties

#### certificateType Property (R/W)

**Type** ***JxfsPINCertificateType.***
**Description** Specifies that a primary certificate is to be returned.

#### certificate Property (R/W)

**Type** ***byte[].***
**Description** Contains the certificate that is to be loaded. This data should be in a binary encoded PKCS #7 format containing certificate data represented in DER encoded ASN.1 notation

### 5.45.2 Methods

#### JxfsPINExportCertificate Constructor

**Syntax** ***JxfsPINExportCertificate (JxfsPINCertificateType certificateType, byte [] certificate)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

## 5.46 JxfsPINCertificateType

The JxfsPINCertificateType data class contains the type, primary or secondary, of certificate exported from the encryptor.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| primary | boolean | R | |
| secondary | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *boolean* | |
| JxfsPINCertificateType | (Constructor of the class) | |

### 5.46.1 Properties

#### primary Property (R)

**Type** ***boolean.***
**Description** Specifies that a primary certificate is to be returned.

#### secondary Property (R)

**Type** ***boolean.***
**Description** Specifies that a secondary certificate is to be returned.

### 5.46.2 Methods

#### JxfsPINCertificateType Constructor

**Syntax** ***JxfsPINCertificateType (boolean primary, boolean secondary)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Both parameters are set to *true* or to *false*. |

## 5.47 JxfsPINCertificateKeyType

The JxfsPINCertificateKeyType data class contains data required as input for *exportCertificate()* operation. It specifies the public key to use

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| encryptionKey | boolean | R/W | |
| verificationKey | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINCertificateKeyTyp e | (Constructor of the class) | |

### 5.47.1 Properties

#### encryptionKey Property (R/W)

**Type** ***boolean***
**Description** Specifies the encryption key is to be returned

#### verificationKey Property (R/W)

**Type** ***boolean***
**Description** Specifies the verification key is to be returned.

### 5.47.2 Methods

#### JxfsPINCertificateKeyType Constructor

**Syntax** ***JxfsPINCertificateKeyType (boolean encryptionKey, boolean verificationKey)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Both parameters are set to *true* or to *false*. |

## 5.48 JxfsPINRSAHashAlgorithms

This class provides properties and methods to query which type of hash algorithms is to be processed

### Summary

**Implements:** **Extends : JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| Hash_NO | boolean | R/W | |
| Hash_SHA1 | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINRSAHashAlgorithms | (Constructor of the class) | |

### 5.48.1 Properties

#### Hash_NO Property (R/W)

**Type** ***boolean***

**Description** Indicates that no hash algorithm is specified. No hash verification will be applied.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### Hash_SHA1 Property (R/W)

**Type** ***boolean***

**Description** Indicates that SHA1 algorithm is supported

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

### 5.48.2 Methods

#### JxfsPINRSAHashAlgorithms Constructor

**Syntax** ***JxfsPINRSAHashAlgorithms (boolean hash_NO, boolean hash_SHA1)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

## 5.49 JxfsPINRSASignatureAlgo

This class provides properties and methods to query which type of RSA Signature algorithms is to be processed

### Summary

**Implements:** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| RSASignature_NO | boolean | R/W | |
| RSASignature_PKCS1_V1_5 | boolean | R/W | |
| RSASignature_PSS | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINRSASignatureAlgo | (Constructor of the class) | |

### 5.49.1 Properties

#### RSASignature_NO Property (R/W)

**Type** ***boolean***

**Description** Indicates that no RSA signature algorithm is specified. No signature verification

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### RSASignature_PKCS1_V1_5 Property (R/W)

**Type** ***boolean***

**Description** Indicates that the RSASSA-PKCS1-V.5 algorithm is used.

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### RSASignature_PSS Property (R/W)

**Type** ***boolean***

**Description** Indicates that the RSASSA-PSS algorithm is used

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

### 5.49.2 Methods

#### JxfsPINRSASignatureAlgo Constructor

**Syntax** ***JxfsPINRSASignatureAlgo (boolean RSASignature_NO, boolean RSASignature_PKCS1_V1_5, boolean RSASignature_PSS);***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false* or more than one signature is set to *true* |

## 5.50 JxfsPINRSAExponent

This class provides properties and methods to query which exponent value of the RSA key pair to be generated

### Summary

**Implements:** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| PIN_Default | boolean | R/W | |
| PIN_Exponent_1 | boolean | R/W | |
| PIN_Exponent_4 | boolean | R/W | |
| PIN_Exponent_16 | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINRSAExponent | (Constructor of the class) | |

### 5.50.1 Properties

#### PIN_Default Property (R/W)

**Type** ***boolean***
**Description** Indicates that the device will decide of the exponent length

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### PIN_Exponent_1 Property (R/W)

**Type** ***boolean***
**Description** Indicates that the exponent length is $2^1 + 1$ (3)

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### PIN_Exponent_4 Property (R/W)

**Type** ***boolean***
**Description** Indicates that the exponent length is $2^4 + 1$ (17)

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

#### PIN_Exponent_16 Property (R/W)

**Type** ***boolean***
**Description** Indicates that the exponent length is $2^{16} + 1$ (65537)

| Value | Meaning |
|---|---|
| *false* | This use is not supported. |
| *true* | This use is supported. |

### 5.50.2 Methods

#### JxfsPINRSAExponent Constructor

**Syntax** ***JxfsPINRSAExponent (boolean PIN_Default, boolean PIN_Exponent_1, boolan PIN_Exponent_4, boolean PIN_Exponent_16);***
**Description** Constructor of the class.

**Exceptions**

Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false* or more than one parameter is set to *true*. |

## 5.51 JxfsPINRSAKeyVerificationData

The JxfsPINRSAKeyVerificationData data class contains information about the imported RSA Public key.

This data class is returned on *JxfsOperationCompleteEvent* of the *importRSAPublicKey()* operation.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| hashAlgorithm | JxfsPINRSAHashAlgorithms | R | |
| hashData | byte [] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *void* | |
| JxfsPINRSAKeyVerificationData | (Constructor of the class) | |

### 5.51.1 Properties

#### hashAlgorithm Property (R)

**Type** ***JxfsPINRSAHashAlgorithms***
**Description** Specifies the hash algorithm used to verify and import the RSA public key.

#### hashData Property (R)

**Type** ***byte[]***
**Description** Contains the Hash data value computed when verifying and importing the key.

### 5.51.2 Methods

#### JxfsPINRSAKeyVerificationData Constructor

**Syntax** ***JxfsPINRSAKeyVerificationData (JxfsPINRSAHashAlgorithms hashAlgorithm, byte [] hashData )***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*hashAlgoithm* is null or not set correctly<br>*hashData* is null. |

## 5.52 JxfsPINRSADESKeyVerificationData

The JxfsPINRSADESkeyVerificationData data class contains information about the imported RSA DES enciphered public key.
This data class is returned on *JxfsOperationCompleteEvent* of the *importRSADESEncipherdPublicKey ()* operation.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyLength | JxfsPINRSADESLength | R | |
| checkMode | JxfsPINRSADESCheck Mode | R | |
| checkValue | byte [] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *property* | |
| JxfsPINRSADESkeyVerifi cationData | (Constructor of the class) | |

### 5.52.1 Properties

#### keyLength Property (R)

**Type** ***JxfsPINRSADESLength***
**Description** Specifies the length of the key loaded (simple or double).

#### checkMode Property (R)

**Type** ***JxfsPINRSADESCheckMode***
**Description** Specifies the mode that was use to create the check value.

#### checkValue Property (R)

**Type** ***byte []***
**Description** Specifies the verification data that can be used for verification of the loaded key.

### 5.52.2 Methods

#### JxfsPINRSADESKeyVerificationData Constructor

**Syntax** ***JxfsPINRSADESKeyVerificationData (JxfsPINRSADESLength keyLength, JxfsPINRSADESCheckMode checkMode, byte [] checkValue)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

## 5.53 JxfsPINRSADESLength

The JxfsPINRSADESLength data specifies the key length that was loaded

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| single | boolean | R/W | |
| double | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINRSADESLength | (Constructor of the class) | |

### 5.53.1 Properties

#### single Property (R/W)

**Type** ***boolean***
**Description** Specifies the length of the key loaded is simple

#### double Property (R/W)

**Type** ***boolean***
**Description** Specifies the length of the key loaded is double

### 5.53.2 Methods

#### JxfsPINRSADESLength Constructor

**Syntax** ***JxfsPINRSADESLength (boolean single, boolean double)***
**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Both parameters are set to *true* or to *false*. |

## 5.54 JxfsPINRSADESCheckMode

The JxfsPINRSADESCheckMode specifies the mode that was used to create the check value.

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| noCheck | boolean | R/W | |
| selfCheck | boolean | R/W | |
| zeroCheck | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINRSADESCheckMode | (Constructor of the class) | |

### 5.54.1 Properties

#### noCheck Property (R/W)

**Type** ***boolean***
**Description** Specifies the no check value provided

#### selfCheck Property (R/W)

**Type** ***boolean***
**Description** Specifies that the key check value is created by an encryption of the key with itself

#### zeroCheck Property (R/W)

**Type** ***boolean***
**Description** Specifies that the key check value is created by an encryption of the key with a zero value

### 5.54.2 Methods

#### JxfsPINRSADESCheckMode Constructor

**Syntax** ***JxfsPINRSADESCheckMode (boolean noCheck, boolean selfCheck, boolean zeroCheck )***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are set to *false* or more than one parameter is set to *true*. |

## 5.55 JxfsPINRSAKeyType

The JxfsPINRSAKeyType data class specifies the private signature to use

### Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| signatureIssuer | boolean | R/W | |
| signatureDevice | boolean | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINRSAKeyType | (Constructor of the class) | |

### 5.55.1 Properties

#### signatureIssuer Property (R/W)

**Type** ***boolean***

**Description** Specifies that the issuer RSA private/public key pair is to be used or has been used to sign the exported key.
These issuer public/private key pairs are installed during manufacture process typically is a secure way.

#### signatureDevice Property (R/W)

**Type** ***boolean***

**Description** Specifies that the devices unique private/public key pair is to be used or has been used to sign the exported key.
The public/private key pairs are created by the device with the command *generateRSAKeyPair*.

### 5.55.2 Methods

#### JxfsPINRSAKeyType Constructor

**Syntax** ***JxfsPINRSAKeyType (boolean signatureIssuer , boolean signatureDevice )***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Both parameters are set to *false* or to *true*. |

## 5.56 JxfsPINRemoteKeyLoadModes

This class provides properties and methods to query which remote key loading modes are supported by a secure PIN device service.

### Summary

**Implements:** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyLoadCertificate | boolean | R | |
| keyLoadSignature | boolean | R | |
| GenerateRSAkeyPair | boolean | R | |
| keyCheckRSA | boolean | R | |
| keyLoadCertificatePKCS7 | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINRemoteKeyLoadModes | (Constructor of the class) | |

### 5.56.1 Properties

#### keyLoadCertificate Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports key loading using Three-party authentication through Certificates.

| Value | Meaning |
|---|---|
| *false* | Key loading mode is not supported. |
| *true* | Key loading mode is supported. |

#### keyLoadSignature Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports key loading using Two-party authentication through Signatures.

| Value | Meaning |
|---|---|
| *false* | Key loading mode is not supported. |
| *true* | Key loading mode is supported. |

#### GenerateRSAkeyPair Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the RSA Key pair generation.

| Value | Meaning |
|---|---|
| *false* | The device does not support generation of RSA key pairs |
| *true* | The device does support generation of key pairs. |

#### keyCheckRSA Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports thumbprint calculation for key loading using Two-party authentication through Signatures.

| Value | Meaning |
|---|---|

| | |
|---|---|
| *false* | SHA1 digest calculation is not supported. |
| *true* | SHA1 digest calculation is supported. |

#### keyLoadCertificatePKCS7 Property (R)

**Type** ***boolean***

**Initial Value** Depends on device

**Description** Indicates if the device supports key loading using Three-party authentication through Certificates, where the Master Key Block is transmitted in a PKCS#7 Message represented in a DER encoded ASN.1 notation.

| Value | Meaning |
|---|---|
| *false* | Key loading mode is not supported. |
| *true* | Key loading mode is supported. |

### 5.56.2 Methods

#### JxfsPINRemoteKeyLoadModes Constructor

**Syntax** ***JxfsPINRemoteKeyLoadModes (boolean keyLoadCertificate, boolean keyLoadSignature, boolean generateRSAkeyPair, boolean keyCheckRSA, boolean keyLoadCertificatePKCS7)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

## 5.57 JxfsPINRSAAlgorithm

This class provides properties and methods to query which RSA algorithm are supported by the secure PIN device service.

### Summary

**Implements:** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| cryptRSA_OAEP | boolean | R | |
| cryptRSA_PKCS_V1_5 | boolean | R | |
| signatureRSA_OAEP | boolean | R | |
| signatureRSA_PKCS_V1_5 | boolean | R | |

| Method | Return | May use after |
|---|---|---|
| is*Property* | *Property* | |
| JxfsPINRSAAlgorithm | (Constructor of the class) | |

### 5.57.1 Properties

#### cryptRSA_OAEP Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the RSA encryption / decryption using the OAEP scheme (Optimal Asymmetric Eencryption Padding).

| Value | Meaning |
|---|---|
| *false* | RSA OAEP encryption / decryption are not supported. |
| *true* | RSA OAEP encryption / decryption are supported. |

#### cryptRSA_PKCS_V1_5 Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the RSA encryption / decryption using the PKCS V1.5 scheme (Public-Key Cryptography Standards ).

| Value | Meaning |
|---|---|
| *false* | RSA PKCS encryption / decryption are not supported. |
| *true* | RSA PKCS encryption / decryption are supported. |

#### signatureRSA_OAEP Property (R)

**Type** ***boolean***
**Initial Value** Depends on device
**Description** Indicates if the device supports the creation of a RSA signature using the OAEP scheme (Optimal Asymmetric Encryption Padding).

| Value | Meaning |
|---|---|
| *false* | RSA OAEP signature creation is not supported. |
| *true* | RSA OAEP signature creation is supported. |

#### signatureRSA_ PKCS_V1_5 Property (R)

**Type** ***boolean***
**Initial Value** Depends on device

**Description** Indicates if the device supports the creation of a RSA signature using the PKCS V1.5 scheme (Optimal Asymmetric Encryption Padding).

| Value | Meaning |
|---|---|
| *false* | RSA PKCS signature creation is not supported. |
| *true* | RSA PKCS signature creation is supported. |

## 5.57.2 Methods

### JxfsPINRSAAlgorithm Constructor

**Syntax** ***JxfsPINRSAAlgorithm (boolean cryptRSA_OAEP, boolean cryptRSA_PKCS_V1_5, boolean signatureRSA_OAEP, boolean signatureRSA_ PKCS_V1_5)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | All the parameters are *false*. |

## 5.58 JxfsPINSecureKeyMode

This class provides properties and methods to check the integrity of the key entered by the *secureKeyEntry* method.

### Summary

**Implements:** **Extends: JxfsPINReadMode2**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| *verificationType* | JxfsVerificationTypeEnum | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| **JxfsPINSecureKeyMode** | (Constructor of the class) | |

## 5.58.1 Properties

### verificationType Property (R)

**Type** ***JxfsVerificationTypeEnum***

**Initial Value** Depends on device

**Description** Specifies the type of verification to be done on the entered key

## 5.58.2 Methods

### JxfsPINSecureKeyMode Constructor

**Syntax** ***JxfsPINSecureKeyMode (JxfsPINFDKeysSelection activeFDKeys, JxfsPINFKeysSelection activeFKeys, JxfsPINFDKeysSelection terminateFDKeys, JxfsPINFKeysSelection terminateFKeys, boolean autoEnd, boolean beepOnPress, int inputMode, int maxLength, int***

***minLength, boolean eventOnStart, JxfsVerificationTypeEnum verificationType)***

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*activeFDKeys* is null.<br>*activeFKeys* is null.<br>*terminateFDKeys* is null.<br>*terminateFKeys* is null.<br>*inputMode* is not one of the listed values.<br>*maxLength* is less than *minLength*.<br>*minLength* is negative.<br>verificationType is null |

## 5.59 JxfsPINSecureKeyEntered

This class provides properties and methods to check the integrity of the key entered by the *secureKeyEntry* method.

### Summary

**Implements:** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| endReason | int | R | |
| keyLength | int | R | |
| kcv | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| **JxfsPINSecureKeyEntered** | (Constructor of the class) | |

## 5.59.1 Properties

### endReason Property (R)

**Type** ***int***

**Initial Value** Depends on device

**Description** Indicates the input operation termination reason.

| Value | Meaning |
|---|---|
| JXFS_PIN_COMP_AUTO | Input operation terminated because *maxLength* was reached. |
| JXFS_PIN_COMP_FK | A termination key was pressed |
| JXFS_PIN_COMP_FDKEY | A termination FDKey was pressed |

### keyLength Property (R)

| | |
|---|---|
| **Type** | ***int*** |
| **Initial Value** | 0 |
| **Description** | Indicates the number of keys entered. |

### kcv Property (R)

| | |
|---|---|
| **Type** | ***byte[]*** |
| **Initial Value** | An empty array. |
| **Description** | Contains the key check value data that can be used for verification of the entered key. This parameter is **empty array** and keyLength is zero if the device does not have this capability, or the key entry was not fully entered, e.g. the entry was terminated by Enter before the required number of digits was entered. |

## 5.59.2 Methods

### JxfsPINSecureKeyEntered Constructor

| | |
|---|---|
| **Syntax** | ***JxfsPINSecureKeyEntered (int endReason, int keyLength, byte[] kcv)*** |
| **Description** | Constructor of the class. |
| **Exceptions** | Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes. |

| **Value** | **Meaning** |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*endReason* has some value different from the supported values described in the *endReason* property section.<br>*keyLength* is negative.<br>kcv is assigned to ***null***. |

## 5.60 JxfsPINSecureKeyDetail

This class provides properties and methods to inform the secure key entry method used by the device. This allows an application to enable the relevant keys and inform the user how to enter the hex digits 'A' to 'F', e.g by displaying an image indicating which key pad locations correspond to the 16 hex digits and/or shift key.
It reports the following information:

- The secure key entry mode (uses a shift key to access the hex digit 'A' to 'F' or each hex digit has a specific key assigned to it).
- The function keys and FDKeys available during secure key entry.
- The FDKeys that are configured as function keys (Enter, Cancel, Clear and Backspace).
- The physical keyboard layout.

The keys that are active during the secure key entry command are vendor specific but must be sufficient to enter a secure encryption key. On some systems a unique key is assigned to each encryption key digit. On some systems encryption key digits are entered by pressing a shift key and then a numeric digit, e.g. to enter 'A' the shift key (JXFS_PIN_FK_SHIFT) is pressed followed by the zero key (JXFS_PIN_FK_0). On these systems JXFS_PIN_FK_SHIFT is not returned to the application in an intermediate event. The exact behavior of the shift key is vendor dependent, some devices will require the shift to be used before every key and some may require the shift key to enter and exit shift mode.
There are many different styles of pinpads in operation. Most have a regular shape with all keys having the same size and are laid out in a regular matrix. However, some devices have a

layout with keys of different sizes and different numbers of keys on some rows and columns. This command returns information that allows an application to provide user instructions and an image of the keyboard layout to assist with key entry.

### Summary

**Implements:** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| keyEntryMode | JxfsKeyEntryModeEnum | R | |
| funcKeyDetail | List | R | |
| clearFDK | int | R | |
| cancelFDK | int | R | |
| backspaceFDK | int | R | |
| enterFDK | int | R | |
| columns | int | R | |
| rows | int | R | |
| hexKeys | java.util.List | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| **JxfsPINSecureKeyDetail** | (Constructor of the class) | |

## 5.60.1 Properties

### keyEntryMode Property (R)

**Type** ***JxfsKeyEntryModeEnum***
**Initial Value** Depends on device
**Description** Specifies the method to be used to enter the encryption key digits (including 'A' to 'F') during secure key entry. The value can be one of the following.

### funcKeyDetail Property (R)

**Type** ***java.util.List***
**Initial Value** Depends on device
**Description** A list of *JxfsPINFDKey* objects that contains information about the Function Keys and FDKeys supported by the device while in secure key entry mode. It describes the function keys that represent the hex digits and shift key, but also reports any other keys that can be enabled while in secure key entry mode.
The double zero, triple zero and decimal point function keys are not valid during secure key entry so they are never reported.
On a pinpad where the physical Enter, Clear, Cancel and Backspace keys are used for hex digits (e.g *JxfsKeyEntryModeEnum*.regUnique mode), the logical function keys JXFS_PIN_FK_ENTER, JXFS_PIN_FK_CLEAR, JXFS_PIN_FK_CANCEL and JXFS_PIN_FK_BACKSPACE will not be reported by this command (unless there is another physical key offering this functionality).
In addition to the existing definition for function keys, the following definitions replace function keys that have hexadecimal values:
JXFS_PIN_FK_A (hex digit A)
JXFS_PIN_FK_B (hex digit B)
JXFS_PIN_FK_C (hex digit C)

JXFS_PIN_FK_D (hex digit D)
JXFS_PIN_FK_E (hex digit E)
JXFS_PIN_FK_F (hex digit F)
JXFS_PIN_FK_SHIFT (Shift key used during hex entry)

### clearFDK Property (R)

| | |
|---|---|
| **Type** | ***int[]*** |
| **Initial Value** | Depends on device |
| **Description** | The FDKey value(s) reporting the FDKey associated with Clear. If this field is an empty array then clear through an FDKeys is not supported, otherwise the value(s) (one or more) reports which FDKey is associated with Clear. |

### cancelFDK Property (R)

| | |
|---|---|
| **Type** | ***int[]*** |
| **Initial Value** | Depends on device |
| **Description** | The FDKey value(s) reporting the FDKey associated with Cancel. If this field is an empty array then Cancel through an FDKey is not supported, otherwise the value(s) (one or more) reports which FDKey is associated with Cancel. |

### backspaceFDK Property (R)

| | |
|---|---|
| **Type** | ***int[]*** |
| **Initial Value** | Depends on device |
| **Description** | The FDKey value(s) reporting any FDKey associated with Backspace. If this field is an empty array then Backspace through an FDKey is not supported, otherwise the value(s) (one or more) reports which FDKey is associated with backspace. |

### enterFDK Property (R)

| | |
|---|---|
| **Type** | ***int[]*** |
| **Initial Value** | Depends on device |
| **Description** | The FDKey value(s) reporting the FDKey associated with Enter. If this field is an empty array then Enter through an FDKey is not supported, otherwise the value(s) (one or more) reports which FDKey is associated with Enter. |

### columns Property (R)

| | |
|---|---|
| **Type** | ***int*** |
| **Initial Value** | Depends on device |
| **Description** | Specifies the maximum number of columns on the pinpad (the columns are defined by the x coordinate values within the *hexKeys* object). When the *keyEntryMode* parameter represents an irregular shaped keyboard the *rows* and *columns* parameters define the ratio of the width to height, i.e.square if the parameters are the same or rectangular if *columns* is larger than *rows*, etc. |

### rows Property (R)

**Type** ***int***
**Initial Value** Depends on device
**Description** Specifies the maximum number of rows on the pinpad (the rows are defined by the y coordinate values within the *hexKeys* structure below). When the *keyEntryMode* parameter represents an irregular shaped keyboard the *rows* and *columns* parameters define the ratio of the width to height, ie square if the parameters are the same or rectangular if *columns* is larger than *rows*, etc.

#### hexKeys Property (R)

**Type** ***List***
**Initial Value** Depends on device
**Description** A List that contains key layout objects (*JxfsPINHexKey*) describing the physical keys on the pinpad, it does not include FDKeys.
This list represents the pinpad keys ordered left to right and top to bottom

### 5.60.2 Methods

#### JxfsPINSecureKeyDetail Constructor

**Syntax** ***JxfsPINSecureKeyMode (int keyEntryMode, List funcKeyDetail, int[] clearFDK, int[] cancelFDK, int[] backspaceFDK, int[] enterFDK, int columns, int rows, List hexKeys)***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | At least one of the constructor parameters is invalid as described:<br>*keyEntryMode* is assigned with a value different from the possible ones described in *keyEntryMode* property section..<br>*funcKeyDetail* is assigned with *null*.<br>*columns* is assigned with a negative number.<br>*rows* is assigned with a negative number.<br>*hexKeys* is assigned with *null*.<br>*clearFDK, cancelFDK, backspaceFDK* and *enterFDK* assigned with *null*. |

## 5.61 JxfsPINHexKey

This class describes the position, size and function key associated with a key when enter secure key is activated by secureEnterKey method.

### Summary

**Implements:** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| xPos | int | R | |
| yPos | int | R | |
| xSize | int | R | |
| ySize | int | R | |
| fk | int | R | |
| shiftFK | int | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| **JxfsPINHexKey** | (Constructor of the class) | |

## 5.61.1 Methods

### JxfsPINHexKey Constructor

**Syntax** ***JxfsPINHexKey* (int xPos, int yPos, int xSize, int ySize, int fk, int shiftFK)**

**Description** Constructor of the class.

**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | At least one of the constructor parameters is invalid as described: *xPos* and *yPos* are less than zero or greater than 999. *xSize* and *ySize* is less than 1 or greater than 1000. fk is neither a valid FK code nor JXFS_PIN_FK_UNUSED. shiftFK is neither a valid FK code nor JXFS_PIN_FK_UNUSED. |

## 5.62 JxfsPINSecureKeyToImport

### Summary

**Implements:** **Extends :** **JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| key | java.lang.String | R/W | |
| keyReload | boolean | R/W | |
| keyUse | JxfsPINKeyUses | R/W | |
| idKey | byte[] | R/W | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *Property* | |
| **JxfsPINSecureKeyToImport** | (Constructor of the class) | |

### 5.62.1 Properties

#### key Property (R/W)

| | |
|---|---|
| **Type** | **java.lang.String** |
| **Initial Value** | Depends on device. |
| **Description** | Name of the key being loaded. |

#### keyReload Property (R/W)

| | |
|---|---|
| **Type** | **boolean** |
| **Initial Value** | Depends on device. |
| **Description** | Indicates whether the key can be loaded only once.<br><br>Possible values:<br>*true* - Key can be loaded/imported many times.<br>*false* - Key can only be loaded/imported once. |

#### keyUse Property (R/W)

| | |
|---|---|
| **Type** | ***JxfsPINKeyUses*** |
| **Initial Value** | Depends on device. |
| **Description** | Type of access for which the key is intended to be used. |

#### idKey Property (R/W)

| | |
|---|---|
| **Type** | **byte[]** |
| **Initial Value** | Depends on device |
| **Description** | Specifies the key owner identification or null. |

### 5.62.2 Methods

#### JxfsPINSecureKeyToImport Constructor

| | |
|---|---|
| **Syntax** | **JxfsSecurePINKeyToImport (java.lang.*String key, boolean keyReload, JxfsKeyUses keyUse, byte[] idKey)*** |
| **Description** | Constructor of the class. |
| **Exceptions** | Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes. |

| **Value** | **Meaning** |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*key* is null.<br>*keyUse* is null.<br>*idKey* is null. |

## 5.63 JxfsPINImportRSAEncipheredPKCS7Key

The *JxfsPINImportRSAEncipheredPKCS7Key* data class contains data required as input for *importRSAEncipheredPKCS7Key* operation. It is specifically designed to import a Terminal

Master Key via RKL based certificates and using a PKCS#7 message. The PKCS#7 message is transmitted in the keyValue Property. The key Property is used as usual.

As it is only allowed to import a key transport key a special key use Property is not necessary.

### 5.63.1 Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| key | java.lang.String | R | |
| keyValue | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINImportRSAEnciphEredPKCS7Key | (Constructor of the class) | |

### 5.63.2 Properties

#### key Property (R)

**Type** ***java.lang.String***
**Description** Name of the key being loaded. If this String is empty, the name of the key is encrypted in the PKCS#7 message (keyValue).

#### keyValue Property (R)

**Type** ***byte []***
**Description** Specifies a binary encoded PKCS #7, represented in DER encoded ASN.1 notation. It has an outer Signed-data content type with the SignerInfo encryptedDigest field containing the Host's signature. The random numbers are included as authenticatedAttributes within the SignerInfo. The inner content is a enveloped-data content type which contains the EPP identifier as an issuerAndSerialNumber sequence and it contains the encrypted key.

### 5.63.3 Methods

#### JxfsPINImportRSAEncipheredPKCS7Key Constructor

**Syntax** ***JxfsPINImportRSAEncipheredPKCS7Key ( java.lang.String key, byte[] keyValue) throws JxfsException***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>*key* is null.<br>*keyValue* is null, empty or not a byte array. |

## 5.64 JxfsPINExportRSASignedPKCS7KeyConfirmation

### 5.64.1 Summary

**Implements:** **Extends: JxfsType**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| key | java.lang.String | R | |
| keyLength | JxfsPINRSADESLength | R | |
| keyConfirmation | byte[] | R | |

| Method | Return | May use after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | *void* | |
| JxfsPINexportRSASigned PKCS7KeyConfirmation | (Constructor of the class) | |

### 5.64.2 Properties

#### keyProperty (R)

**Type** ***java.lang.String***
**Description** Name of the key that was loaded.

#### keyLength Property (R)

**Type** ***JxfsPINRSADESLength***
**Description** Specifies the length of the key loaded (16 or 32 byte).

#### keyConfirmation Property (R)

**Type** ***byte []***
**Description** Specifies a binary encoded PKCS #7, represented in DER encoded ASN.1 notation. The message is a confirmation to the method *importRSAEncipheredPKCS7Key*. It has an outer Signed-data content type with the SignerInfo encryptedDigest field containing the ATM's signature. The random numbers are included as authenticatedAttributes within the SignerInfo. The inner content is a data content type which contains the HOST identifier as a Host serial number.

### 5.64.3 Methods

#### JxfsPINExportRSASignedPKCS7KeyConfirmation Constructor

**Syntax** ***JxfsPINExportRSASignedPKCS7KeyConfirmation (String keyProperty, JxfsPINRSADESLength keyLength, byte[] keyConfirmation) throws JxfsException;***
**Description** Constructor of the class.
**Exceptions** Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes.

| Value | Meaning |
|---|---|
| JXFS_E_PARAMETER_INVALID | Any of the following conditions is met:<br>- *keyLength* is null or not set correctly<br>- *keyValue* is null |

# 6 Enum Classes

All enumerations are defined in terms of a class. The following describes all the enumerated classes.

## 6.1 JxfsVerificationTypeEnum

This enumerated data type represents the possible types of key verification.

| Extends | Implements |
|---|---|
| JxfsEnum | |

| Field | Description |
|---|---|
| none | No verification required. |
| kcvSelf | The key check value is created by an encryption of the key with itself. |
| kcvZero | The key check value is created by an encryption of a zero value with the key. |
| md5 | The key check value is created by calculating the hash code of the key using the MD5 hash algorithm. The MD5 hash code returned will always be a buffer with 16 bytes length. |
| sha1 | The key check value is created by calculating the hash code of the key using the SHA1 hash algorithm. The SHA1 hash code returned will always be a buffer with 20 bytes length. |

## 6.2 JxfsKeyEntryModeEnum

This enumerated data type represents the possible methods to be used to enter the encryption key digits (including 'A' to 'F') during secure key entry.

| Extends | Implements |
|---|---|
| JxfsEnum | |

| Field | Description |
|---|---|
| notSupp | Secure key entry is not supported, all other parameters are undefined. |
| regShift | Secure key hex digits 'A' – 'F' are accessed through the shift key. Digits 'A' – 'F' are accessed through the shift key followed by one of the other function keys. The keys associated with 'A' to 'F' are defined within the *hexKeys* parameter. The keyboard has a regular shaped key layout where all rows have the same number of keys and all columns have the same number of keys, e.g. 5x4. The h*exKeys* parameter must contain one entry for each key on the pinpad (i.e. the product of *rows* by *columns*). |
| irregShift | Secure key hex digits 'A' – 'F' are accessed through the shift key. Digits 'A' – 'F' are accessed through the shift key followed by one of the other function keys. The keys associated with 'A' to 'F' are defined within the *hexKeys* parameter. The keyboard has an irregular shaped key layout, e.g there are more or less keys on one row or column than on the others. The *hexKeys* parameter must contain one entry for each key on the pinpad. |
| regUnique | Secure key hex digits are accessed through specific keys assigned to each hex digit. The keyboard has a regular shaped key layout where all rows have the same number of keys and all columns have the same number of keys, e.g. 5x4. The *hexKeys* parameter must contain one entry for each key on the pinpad (i.e. the product of *rows* by *columns*). |

| | |
|---|---|
| irregUnique | Secure key hex digits are accessed through specific keys assigned to each hex digit. The keyboard has an irregular shaped key layout, e.g. there are more or less keys on one row or column than on the others. The *hexKeys* must contain one entry for each key on the pinpad. |

## 6.3 JxfsSecureKeyEntrySupportedEnum

This enumerated data type indicates if the pinpad is supporting or not entering securely a master key.

| Extends | Implements |
|---|---|
| JxfsEnum | |

| Field | Description |
|---|---|
| notSupported | The pinpad is NOT supporting manually secure entry of a master key. |
| supported | The pinpad is supporting manually secure entry of a master key |
| unknown | The capability of the pinpad for supporting manually secure entry of a master key is unknown. |

## 6.4 JxfsPINStatusSelectorEnum

This enumeration class is used for the base getStatus(java.util.List) method.

| Extends | Implements |
|---|---|
| JxfsStatusSelectorEnum | |

| Field | Returned Type | Description |
|---|---|---|
| status | JxfsStatus | General status of the device. |
| keyList | HashMap | List of loaded keys names including detailed information. A map of name (String) and key information (JxfsPINKeyDetail) that it is a combination of the information you get by getKeyNameList() and getKeyInfo(). |
| secureKeyEntryState | Boolean | Specifies if the device is in secure key entry state or not |

# 7 Codes

## 7.1 Error Codes

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_E_PIN_READ_FAILURE | Read error. | 5098 |
| JXFS_E_PIN_KEYINVALID | At least one of the specified active function keys or FDKeys is invalid. | 5099 |
| JXFS_E_PIN_NOACTIVEKEYS | No active function key or FDKey specified. | 5100 |
| JXFS_E_PIN_KEYNOTSUPPORTED | At least one of the specified active function keys or FDKeys (*activeFKeys* or *activeFDKeys* properties of *readMode* parameter) is not supported by the device service. | 5101 |
| JXFS_E_PIN_MINIMUMLENGTH | The *minLength* property is invalid or greater than the *maxLength* property. | 5102 |
| JXFS_E_PIN_NO_PIN | PIN has not been entered or has been cleared. | 5103 |
| JXFS_E_PIN_NOT_ALLOWED | PIN entered by the user is not allowed. | 5104 |
| JXFS_E_PIN_KEY_NOT_FOUND | The specified key was not found. | 5105 |
| JXFS_E_PIN_KEY_NO_VALUE | The specified key is not loaded. | 5106 |
| JXFS_E_PIN_USE_VIOLATION | The specified use is not supported by this key. | 5107 |
| JXFS_E_PIN_ACCESS_DENIED | The encryption module is either not initialized or not ready for any vendor specific reason. | 5108 |
| JXFS_E_PIN_NOTSUPPORTEDCAP | The requested function is not supported. | 5109 |
| JXFS_E_PIN_FORMAT_NOTSUPPOR TED | The specified PIN block format is not supported. | 5110 |
| JXFS_E_PIN_LENGTH_ERROR | The length of the start value specified is not supported. | 5111 |
| JXFS_E_PIN_CRYPTNOTSUPPORTED | The encryption or decryption method is not supported. | 5112 |
| JXFS_E_PIN_DUPLICATE_KEY | A key exists with the specified name and cannot be overwritten. | 5113 |
| JXFS_E_PIN_NOTERMINATEKEYS | There are no terminate keys specified and *autoEnd* property of *JxfsPINSecureKeyMode* object is *false* | 5143 |
| JXFS_E_PIN_VERIFICATION_TYPE_ NOT_SUPPORTED | The verification type requested is not supported by the Device Service. | 5144 |
| JXFS_E_PIN_KEY_VERIFICATION_D ATA_DOESNOT_MATCH | The key verification data calculated by the Device Service or hardware in the importing process does not match the one informed by the application. | 5145 |

| | | |
|---|---|---|
| JXFS_E_PIN_ERR_SIGNATURE | The imported key failed its signature verification. It is not stored in the PIN device. | 5130 |
| JXFS_E_PIN_ERR_HASH | The imported key failed its hash verification. It is not stored in the PIN device. | 5131 |
| JXFS_E_PIN_INVALID_FORMAT | The format of the message is invalid. | 5148 |
| JXFS_E_PIN_EMV_VERIFY_FAILED | The verification of the key failed and the key is discarded | 5149 |
| JXFS_E_PIN_NO_RSA_KEY_PAIR | The encryption module does not have a RSA private key. | 5154 |
| JXFS_E_PIN_HSMSTATEINVALID | The HSM is not in a correct state to handle this message | 5502 |

## 7.2 Status Codes

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_S_PIN_KEY | A new key has been loaded/imported into the device's key table. | 5097 |

## 7.3 Operation Codes

The following codes identify the operation that generated a JxfsOperationCompleteEvent or JxfsIntermediateEvent:

| Value | Method | Numerical Value |
|---|---|---|
| JXFS_O_PIN_READPIN | *readData, secureReadPIN* | 5083 |
| JXFS_O_PIN_CREATEOFFSET | *createOffset* | 5084 |
| JXFS_O_PIN_CREATEPINBLOCK | *createPINBlock* | 5085 |
| JXFS_O_PIN_VALIDATEPIN | *validatePIN* | 5086 |
| JXFS_O_PIN_CREATEOFFSET_SECURE | *createOffsetSecure* | 5087 |
| JXFS_O_PIN_CREATEPINBLOCK_SECURE | *createPINBlockSecure* | 5088 |
| JXFS_O_PIN_VALIDATEPIN_SECURE | *validatePINSecure* | 5089 |
| JXFS_O_PIN_VALIDATEPINCHIP | *validatePINChip* | 5115 |
| JXFS_O_PIN_DECRYPT | *decrypt* | 5091 |
| JXFS_O_PIN_ENCRYPT | *encrypt* | 5092 |
| JXFS_O_PIN_GENMAC | *generateMAC* | 5093 |
| JXFS_O_PIN_IMPORTKEY | *importKey, importSecureKeyEntered* | 5094 |
| JXFS_O_PIN_INITIALIZE | *initialize* | 5095 |
| JXFS_O_PIN_IMPORTEMVRSAKEY | *importEMVRSAPublicKey* | 5117 |
| JXFS_O_PIN_SHA1_DIGEST | *computeSHA1Digest* | 5118 |
| JXFS_O_PIN_DELETE | *deleteKey* | 5119 |
| JXFS_O_PIN_IMPORTRSAPUBLICKEY | *importRSAPublicKey* | 5120 |
| JXFS_O_PIN_EXPORTRSAPUBLICKEY | *exportRSAPublicKey* | 5121 |
| JXFS_O_PIN_IMPORTRSADESENCIPHEREDPUBLICKEY | *importRSADESEncipheredPublicKey* | 5122 |
| JXFS_O_PIN_EXPORTRSADESENCIP | *exportRSADESEncipheredPubl* | 5123 |

| | | |
|---|---|---|
| HEREDPUBLICKEY | *icKey* | |
| JXFS_O_PIN_GENERATERSAKEYPAIR | *generateRSAKeyPair* | 5124 |
| JXFS_O_PIN_EXPORTPINID | *exportPINId* | 5125 |
| JXFS_O_PIN_IMPORTCERTIFICATE | *importCertificate* | 5126 |
| JXFS_O_PIN_EXPORTCERTIFICATE | *exportCertificate* | 5127 |
| JXFS_O_PIN_REPLACECERTIFICATE | *replaceCertificate* | 5128 |
| JXFS_O_PIN_STARTKEYEXCHANGE | startKeyExchange | 5129 |
| JXFS_O_PIN_SECUREKEYENTRY | *secureKeyEntry* | 5132 |
| JXFS_O_PIN_CLEARSECUREKEYBUFFER | *clearSecureKeyBuffer* | 5133 |
| JXFS_O_PIN_IMPORTRSAENCIPHEREDPKCS7KEY | *importRSAEncipheredPKCS7Key* | 5160 |

The following codes identify the reason for a JxfsIntermediateEvent:

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_I_PIN_KEY_PRESSED | A key has been pressed. | 5096 |
| JXFS_I_PIN_READ_STARTED | The PIN input operation has started. | 5116 |

## 7.4 Constants

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_PIN_FK_FDK01 | Function descriptor key code. | 5001 |
| JXFS_PIN_FK_FDK02 | Function descriptor key code. | 5002 |
| JXFS_PIN_FK_FDK03 | Function descriptor key code. | 5003 |
| JXFS_PIN_FK_FDK04 | Function descriptor key code. | 5004 |
| JXFS_PIN_FK_FDK05 | Function descriptor key code. | 5005 |
| JXFS_PIN_FK_FDK06 | Function descriptor key code. | 5006 |
| JXFS_PIN_FK_FDK07 | Function descriptor key code. | 5007 |
| JXFS_PIN_FK_FDK08 | Function descriptor key code. | 5008 |
| JXFS_PIN_FK_FDK09 | Function descriptor key code. | 5009 |
| JXFS_PIN_FK_FDK10 | Function descriptor key code. | 5010 |
| JXFS_PIN_FK_FDK11 | Function descriptor key code. | 5011 |
| JXFS_PIN_FK_FDK12 | Function descriptor key code. | 5012 |
| JXFS_PIN_FK_FDK13 | Function descriptor key code. | 5013 |
| JXFS_PIN_FK_FDK14 | Function descriptor key code. | 5014 |
| JXFS_PIN_FK_FDK15 | Function descriptor key code. | 5015 |
| JXFS_PIN_FK_FDK16 | Function descriptor key code. | 5016 |
| JXFS_PIN_FK_FDK17 | Function descriptor key code. | 5017 |
| JXFS_PIN_FK_FDK18 | Function descriptor key code. | 5018 |
| JXFS_PIN_FK_FDK19 | Function descriptor key code. | 5019 |
| JXFS_PIN_FK_FDK20 | Function descriptor key code. | 5020 |
| JXFS_PIN_FK_FDK21 | Function descriptor key code. | 5021 |
| JXFS_PIN_FK_FDK22 | Function descriptor key code. | 5022 |
| JXFS_PIN_FK_FDK23 | Function descriptor key code. | 5023 |
| JXFS_PIN_FK_FDK24 | Function descriptor key code. | 5024 |
| JXFS_PIN_FK_FDK25 | Function descriptor key code. | 5025 |
| JXFS_PIN_FK_FDK26 | Function descriptor key code. | 5026 |
| JXFS_PIN_FK_FDK27 | Function descriptor key code. | 5027 |
| JXFS_PIN_FK_FDK28 | Function descriptor key code. | 5028 |
| JXFS_PIN_FK_FDK29 | Function descriptor key code. | 5029 |
| JXFS_PIN_FK_FDK30 | Function descriptor key code. | 5030 |
| JXFS_PIN_FK_FDK31 | Function descriptor key code. | 5031 |

| | | |
|---|---|---|
| JXFS_PIN_FK_FDK32 | Function descriptor key code. | 5032 |
| JXFS_PIN_FK_0 | Function key code. | 0 |
| JXFS_PIN_FK_1 | Function key code. | 1 |
| JXFS_PIN_FK_2 | Function key code. | 2 |
| JXFS_PIN_FK_3 | Function key code. | 3 |
| JXFS_PIN_FK_4 | Function key code. | 4 |
| JXFS_PIN_FK_5 | Function key code. | 5 |
| JXFS_PIN_FK_6 | Function key code. | 6 |
| JXFS_PIN_FK_7 | Function key code. | 7 |
| JXFS_PIN_FK_8 | Function key code. | 8 |
| JXFS_PIN_FK_9 | Function key code. | 9 |
| JXFS_PIN_FK_ENTER | Function key code. | 5043 |
| JXFS_PIN_FK_CANCEL | Function key code. | 5044 |
| JXFS_PIN_FK_CLEAR | Function key code. | 5045 |
| JXFS_PIN_FK_BACKSPACE | Function key code. | 5046 |
| JXFS_PIN_FK_HELP | Function key code. | 5047 |
| JXFS_PIN_FK_DECPOINT | Function key code. | 5048 |
| JXFS_PIN_FK_00 | Function key code. | 5049 |
| JXFS_PIN_FK_000 | Function key code. | 5050 |
| JXFS_PIN_FK_SHIFT | Function key code. | 5134 |
| JXFS_PIN_FK_UNUSED | Function key code. | 5135 |
| JXFS_PIN_FK_A | Function key code. | 5136 |
| JXFS_PIN_FK_B | Function key code. | 5137 |
| JXFS_PIN_FK_C | Function key code. | 5138 |
| JXFS_PIN_FK_D | Function key code. | 5139 |
| JXFS_PIN_FK_E | Function key code. | 5140 |
| JXFS_PIN_FK_F | Function key code. | 5141 |

| **Value** | **Meaning** | **Numerical Value** |
|---|---|---|
| JXFS_PIN_FK_NONE | Result of a *secureReadPIN()* operation when key is not a function key. | 5051 |
| JXFS_PIN_KP_FUNCTION | Key is a Function key. | 5052 |
| JXFS_PIN_KP_FDKEY | Key is a Function descriptor key (FDKey). | 5053 |
| JXFS_PIN_INPUT_RAW | Each key pressed during an input operation will generate an intermediate event. These events will contain information about pressed keys. | 5054 |
| JXFS_PIN_INPUT_COOKED | No intermediate events per key pressed are generated. Data entered during an input operation is provided in a *JxfsOperationCompleteEvent* event. | 5055 |
| JXFS_PIN_COMP_AUTO | Input operation terminated because *maxLength* was reached. | 5056 |
| JXFS_PIN_COMP_FK | A termination key was pressed. | 5057 |
| JXFS_PIN_COMP_FDKEY | A termination FDKey was pressed | 5058 |

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_PIN_VAL_DES | DES PIN validation. | 5059 |
| JXFS_PIN_VAL_EC | EUROCHEQUE PIN validation. | 5060 |
| JXFS_PIN_VAL_VISA | VISA PIN validation. | 5061 |
| JXFS_PIN_PRES_CLEAR | Clear text presentation of PIN to chip card device. | 5062 |

PIN block formats:

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_PIN_FMT_3624 | 3624. | 5063 |
| JXFS_PIN_FMT_ANSI | ANSI. | 5064 |
| JXFS_PIN_FMT_ISO0 | ISO0. | 5065 |
| JXFS_PIN_FMT_ISO1 | ISO1. | 5066 |
| JXFS_PIN_FMT_EC12 | EC12. | 5067 |
| JXFS_PIN_FMT_EC13 | EC13. | 5068 |
| JXFS_PIN_FMT_EC13RAND | EC13, random padding. | 5069 |
| JXFS_PIN_FMT_VISA | VISA. | 5070 |
| JXFS_PIN_FMT_DIEBOLD | DIEBOLD. | 5071 |
| JXFS_PIN_FMT_DIEBOLDC0 | DIEBOLD C0. | 5072 |
| JXFS_PIN_FMT_EMV | EMV PIN Format | 5114 |
| JXFS_PIN_FMT_ISO3 | ISO3. | 5150 |
| JXFS_PIN_FMT_VISA3 | VISA 3. | 5151 |
| JXFS_PIN_FMT_ITAP | Italien AP. | 5152 |
| JXFS_PIN_FMT_BANKSYS | Banksys. | 5153 |

Encryption/decryption algorithms:

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_PIN_CRYPT_MODE_DESECB | Electronic Code Book | 5073 |
| JXFS_PIN_CRYPT_MODE_DESCBC | Cipher Block Chaining | 5074 |
| JXFS_PIN_CRYPT_MODE_DESMAC | MAC calculation using CBC | 5075 |
| JXFS_PIN_CRYPT_MODE_DESCFB | Cipher Feed Back | 5076 |
| JXFS_PIN_CRYPT_MODE_RSA | RSA Encryption | 5077 |
| JXFS_PIN_CRYPT_MODE_ECMA | ECMA Encryption | 5078 |
| JXFS_PIN_CRYPT_MODE_TRIDESECB | Triple DES with Electronic Code Book | 5079 |
| JXFS_PIN_CRYPT_MODE_TRIDESCBC | Triple DES with Cipher Block Chaining | 5080 |
| JXFS_PIN_CRYPT_MODE_TRIDESCFB | Triple DES with Cipher Feed Back | 5081 |
| JXFS_PIN_CRYPT_MODE_TRIDESMAC | Triple DES MAC calculation using CBC | 5082 |

# 8 Appendix A: ZKA Extensions for the Pin Keypad Device Class Interface

This chapter describes a proposal for an extension of the Pin Keypad device class interface to cover the functionality needed to implement the parts of the ZKA 3.0 specification that are necessary for self-service automates.

For the access of the ZKA functionality the appropriate interfaces, classes and events for handling the ISO messages and the HSM data are defined in addition to the current Pin Keypad device class interface specification.

**Important Notes:**

· This revision of this specification does not define key management procedures; key management is vendor-specific.
· Key space management is customer-specific, and is therefore handled by vendor-specific mechanisms.
· Only numeric PIN pads are handled in this specification.

Multiple HSMs will be handled by multiple device services. If more than one HSM is implemented in one hardware device, these “logical” HSMs may be presented by one instance of a device service. In this case it is a complex device service that offers its services via different “logical” devices.

This specification supports the Hardware Security Module (HSM), which is necessary for the German ZKA Electronic Purse transactions. Furthermore the HSM stores terminal specific data.
This data will be compared against the message data fields (Sent and Received ISO8583 messages) prior to HSM-MAC generation/verification. HSM-MACs are generated/verified only if the message fields match the data stored.

Keys used for cryptographic HSM functions are stored separate from other keys. This must be considered when importing keys.

This version of PinPad complies to the current ZKA specification 3.0. It supports loading and unloading against card account for both card types (Type 0 and Type 1) of the ZKA electronic purse. It also covers the necessary functionality for ‘Loading against other legal tender’.

Key values are passed to the API as binary hexadecimal values, for example:
0123456789ABCDEF = 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF

The implementation for the *IJxfsPINIso* interface is optional for the device service. If a device service is not intended to be used in Germany it does not need to implement any of the functionality described in this document. If the device service offers the functionality to support ZKA 3.0 it has to implement the IJxfsPINIso interface.

## 8.1 Class and Interface Summary

The following classes and interfaces are used by the J/XFS Zka extensions:

| Class or Inter-face | Name | Description | Extends / Implements |
|---|---|---|---|
| Inter-face | **IJxfsPINIso** | Interface for ISO messages, etc. | -- |
| Inter-face | **IJxfsPINIsoConst** | Interface containing the JXFS constants that are common to the ISO messages interface.. | -- |
| Class | **JxfsPINSupportedProtocols** | Capabilities for the *IJxfsPINIso* interface. | Extends: JxfsType |

| Class or Inter-face | Name | Description | Extends / Implements |
|---|---|---|---|
| Class | **JxfsPINSecureMsg** | Representation of a secure raw data message | Extends: JxfsType |
| Class | **JxfsPINSecureMsgRawData** | Representation of a secure message | Extends: JxfsPINSecureMsg |
| Class | **JxfsPINSecureMsgChipZka** | Representation of a secure smart card ZKA message | Extends: JxfsPINSecureMsg |
| Class | **JxfsPINSecureMsgPbm** | Representation of a secure PBM message | Extends: JxfsPINSecureMsg |
| Class | **JxfsPINSecureMsgHsmLdi** | Representation of a secure LDI message | Extends: JxfsPINSecureMsg |
| Class | **JxfsPINSecureMsgGenAs** | Representation of a secure Gen AS message | Extends: JxfsPINSecureMsg |
| Class | **JxfsPINSecureMsgISO** | Representation of aa ISO message | Extends: JxfsPINSecureMsg |
| Class | **JxfsPINSecureMsgISOPs** | Representation of a secure Ps message | Extends: JxfsPINSecureMsgISO |
| Class | **JxfsPINSecureMsgISOAs** | Representation of a secure As message | Extends: JxfsPINSecureMsgISO |
| Class | **JxfsPINSecureMsgISOLz** | Representation of a secure Lz message | Extends: JxfsPINSecureMsgISO |
| Class | **JxfsPINProtocolSelection** | This class specifies a certain protocol. | Extends: JxfsType |
| Class | **JxfsPINJournalData** | Object to represent journal data. | Extends: JxfsType |
| Class | **JxfsPINTData** | Object to represent data to be set in the HSM. | Extends: JxfsType |
| Class | **JxfsPINIsoSupportedModes** | Object to specify the supported charge modes. | Extends: JxfsType |

## 8.2 Messages

All message classes are derived from the global abstract *JxfsPINSecureMsg* class. The access to the binary message data is done via the synchronized getMessageData() and setMessageData() methods. If the message data will be modified, the whole message has to be set.

## 8.3 Classes and Interfaces

### 8.3.1 IJxfsPINIso

This is the main interface for accessing ZKA specific functionalities.

**Summary**

| Property | Type | Access | Initialized after |
|---|---|---|---|
| supportedProtocols | **JxfsPINSupportedProtocols** | R | successfull open() |
| supportedJournalingProtocols | **JxfsPINSupportedProtocols** | R | successfull open() |
| hsmVendor | **java.lang.String** | R | successfull open() |
| chargingMode | **JxfsPINIsoSupportedModes** | R | successfull open() |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| secureMsgSend | identificationID | |
| secureMsgReceive | identificationID | |
| getJournalData | identificationID | |
| getHsmTData | identificationID | |
| setHsmTData | identificationID | |
| hsmInit | identificationID | |

**Properties**

**supportedProtocols (R)**

**Type** ***JxfsPINSupportedProtocols***
**Initial Value** *none*
**Description** Definition of the supported protocols by the device service (see *JxfsPINSupportedProtocols)*

**supportedJournalingProtocols (R)**

**Type** ***JxfsPINSupportedProtocols***
**Initial Value** *none*
**Description** Definition for which protocols the device service provides journal data (see *JxfsPINSupportedProtocols)*

**hsmVendor (R)**

**Type** ***java.lang.String***
**Initial Value** *none*
**Description** String identifying the vendor of the HSM module. Examples for this string are "KRONE", "ASCOM", "IBM" or "NCR".

**chargingMode (R)**

**Type** ***JxfsPINIsoSupportedModes***
**Initial Value** *none*
**Description** Specification of the charging modes that are supported by the HSM.

**Methods**

**secureMsgSend**

**Syntax** ***identificationID secureMsgSend(JxfsPINSecureMsg message) throws JxfsException;***
**Description** This command handles all messages that should be sent through a

secure messaging to a authorization system, German "Ladezentrale", personalisation system or the chip. The encryption module adds the security relevant fields to the message and returns the modified message in the appropriate OC event. All messages must be presented to the encryptor via this command even if they do not contain security fields in order to keep track of the transaction status in the internal state machine.

**Parameter**

| Type | Name | Meaning |
|---|---|---|
| *JxfsPINSecureMsg* | message | Specifies the message.<br>The following protocols are supported:<br>JXFS_PIN_PROTISOAS<br>JXFS_PIN_PROTISOLZ<br>JXFS_PIN_PROTISOPS<br>JXFS_PIN_PROTCHIPZKA<br>JXFS_PIN_PROTRAWDATA<br>JXFS_PIN_PROTPBM<br>JXFS_PIN_PROTHSMLDI<br>JXFS_PIN_PROTGENAS |

**Exceptions**
No additional exceptions generated.

**Events**

**JxfsOperationCompleteEvent**
When the operation completes a JxfsOperationCompleteEvent will be sent by J/XFS PIN Device Control to all registered OperationCompleteListeners with the following data:

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_SEND_MSG |
| *identificationID* | The corresponding ID |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | *JxfsPINSecureMsg* object containing the modified message that can now be send to an authorization system, German "Ladezentrale", personalization system or the chip. If the secure message object could not be generated, the data reference is null. |

## secureMsgReceive

**Syntax**

***identificationID secureMsgReceive(JxfsPINSecureMsg message) throws JxfsException;***

**Description**

This command handles all messages that are received through a secure messaging from a authorization system, German "Ladezentrale", personalisation system or the chip. The encryption checks the security relevant fields. All messages must be presented to the encryptor via this command even if they do not contain security fields in order to keep track of the transaction status in the internal state machine.

**Parameter**

| Type | Name | Meaning |
|---|---|---|
| *JxfsPINSecureMsg* | message | Specifies the message<br>The following protocols are supported:<br>JXFS_PIN_PROTISOAS<br>JXFS_PIN_PROTISOLZ<br>JXFS_PIN_PROTISOPS<br>JXFS_PIN_PROTCHIPZKA<br>JXFS_PIN_PROTRAWDATA<br>JXFS_PIN_PROTPBM<br>JXFS_PIN_PROTGENAS |

**Exceptions**
No additional exceptions generated.

**Events**

**JxfsOperationCompleteEvent**
When the operation completes a JxfsOperationCompleteEvent will be

sent by J/XFS PIN Device Control to all registered OperationCompleteListeners with the following data:

| **Field** | **Value** |
|---|---|
| *operationID* | JXFS_O_PIN_RECEIVE_MSG |
| *identificationID* | The corresponding ID |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | none. |

### getJournalData

**Syntax** ***identificationID getJournalData(JxfsPINProtocolSelection protocol) throws JxfsException;***

**Description** This command is used to get journal data from the encryptor module. It retrieves cryptographically secured information about the result of the last transaction that was done with the indicated protocol. When the device service supports journaling (see supportedJournalingProtocols) then it is impossible to do any *secureMsgSend*/*secureMsgReceive* method calls with this protocol, unless the journal data is retrieved. It is possible – especially after restarting a system – to get the same journal data again.
Calling this method is obligatory between transactions and after failures as it can be used by the device service to initialize its internal state machine.

**Parameter**

| Type | Name | Meaning |
|---|---|---|
| *JxfsPINProtocolSelection* | protocol | Specifies the protocol. Only the ISOAS, ISOLZ, ISOPS or PBM protocols are supported for this method. |

**Exceptions** No additional exceptions generated.

**Events**

**JxfsOperationCompleteEvent**
When the operation completes a JxfsOperationCompleteEvent will be sent by J/XFS PIN Device Control to all registered OperationCompleteListeners with the following data:

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_GET_JOURNAL |
| *identificationID* | The corresponding ID |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | *JxfsPINJournalData* object, if the operation succeeded. null, if the data could not be retrieved. |

### getHsmTData

**Syntax** ***identificationID getHsmTData() throws JxfsException;***

**Description** This function allows to get the current HSM terminal data except keys, trace number and session key index. The data is provided as a series of "tag/length/value" items in the tData class.

**Exceptions** No additional exceptions generated.

**Events**

**JxfsOperationCompleteEvent**
When the operation completes a JxfsOperationCompleteEvent will be sent by J/XFS PIN Device Control to all registered OperationCompleteListeners with the following data:

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_HSM_GET_TDATA |
| *identificationID* | The corresponding ID |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | If the operation was successful the data is the terminal data as an instance of the *JxfsPINTData* class. This value is null, if the data could not be retrieved. |

### setHsmTData

**Syntax** ***identificationID setHsmTData(JxfsPINTData tData) throws JxfsException;***

**Description** This function allows to set the HSM terminal data except keys, trace number and session key index. The data must be provided as a series of "tag/length/value" items in the Data class.

| Parameter | Type | Name | Meaning |
|---|---|---|---|
| | *JxfsPINTData* | tData | Specifies the values to set |

**Exceptions** No additional exceptions generated.

**Events**

**JxfsOperationCompleteEvent**
When the operation completes a JxfsOperationCompleteEvent will be sent by J/XFS PIN Device Control to all registered OperationCompleteListeners with the following data:

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_HSM_SET_TDATA |
| *identificationID* | The corresponding ID |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | none |

### hsmInit

**Syntax** ***identificationID hsmInit(JxfsPINHsmInitData hsmInitData) throws JxfsException;***

**Description** This command is used to set an HSM out of order. At the same time the online time can be set to control when the online dialog will be started to initialize the HSM again.

| Parameter | Type | Name | Meaning |
|---|---|---|---|
| | *JxfsPINHsmInitData* | hsmInitData | Specifies the data for the initialization. |

**Exceptions** No additional exceptions generated.

**Events**

**JxfsOperationCompleteEvent**
When the operation completes a JxfsOperationCompleteEvent will be sent by J/XFS PIN Device Control to all registered OperationCompleteListeners with the following data:

| Field | Value |
|---|---|
| *operationID* | JXFS_O_PIN_HSM_INIT |
| *identificationID* | The corresponding ID |
| *result* | Common or device dependent error code. (See section on *Error Codes*). |
| *data* | none. |

## 8.3.2 JxfsPINSecureMsg

This class defines a secure message. As every specific message has its own class type, this class is abstract.

### Summary

**Implements :** *Serializable, Clonable* **Extends :** *JxfsType*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| messageData | byte[] | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsg | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

### Properties

#### messageData (R)

| | |
|---|---|
| **Type** | ***byte[]*** |
| **Initial Value** | none |
| **Description** | Message data. null is permitted in the special case that during the message receive no response was received from the communication partner during a specified time period. This exception is necessary to set the internal state machine to the correct state. |

## 8.3.3 JxfsPINSecureMsgRawData

This class defines a secure message with raw data contents that may be used by a vendor for specific purpose.

### Summary

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsg*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsgRawData | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

## 8.3.4 JxfsPINSecureMsgChipZka

This class defines a secure message for chip card data.

### Summary

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsg*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsgChipZka | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

## 8.3.5 JxfsPINSecureMsgPbm

This class defines a secure message for embedded PBM protocol data.

### Summary

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsg*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsgPbm | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

### 8.3.6 JxfsPINSecureMsgHsmLdi

This class defines a secure message that contains LDI Information.

#### Summary

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsg*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsgHsmLdi | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

### 8.3.7 JxfsPINSecureMsgGenAs

This class defines a secure message that contains PAC/MAC information for non-ISO8583 message formats.

#### Summary

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsg*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsgGenAs | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

### 8.3.8 JxfsPINSecureMsgISO

This abstract class defines the base for all ISO 8583 secure messages.

**Summary**

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsg*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| none | none | none |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

### 8.3.9 JxfsPINSecureMsgISOAs

This class defines a secure message that contains an ISO 8583 secure message for the authorization system.

**Summary**

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsgISO*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsgISOAs | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

### 8.3.10 JxfsPINSecureMsgISOLz

This class defines a secure message that contains an ISO 8583 secure message for the german "Ladezentrale".

**Summary**

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsgISO*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsgISOLz | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

### 8.3.11 JxfsPINSecureMsgISOPs

This class defines a secure message that contains an ISO 8583 secure message for the personalization system

#### Summary

**Implements :** *Serializable, Clonable* **Extends :** *JxfsPINSecureMsgISO*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| none | none | | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSecureMsgISOPs | messageData | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| set*Property* | | |

| Event | May occur after |
|---|---|
| none | |

### 8.3.12 JxfsPINSupportedProtocols

This class is used to specify the capabilites (supported protocol types).

#### Summary

**Implements :** *Serializable* **Extends :** *JxfsType*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| protocolIsoAs | boolean | R | |
| protocolIsoLz | boolean | R | |
| protocolIsoPs | boolean | R | |
| protocolChipZka | boolean | R | |
| protocolRawData | boolean | R | |
| protocolPbm | boolean | R | |
| protocolHsmLdi | boolean | R | |
| protocolGenAs | boolean | R | |

| Constructors | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINSupportedProtocols | protocolIsoAs | boolean |
| | protocolIsoLz | boolean |
| | protocolIsoPs | boolean |
| | protocolChipZka | boolean |
| | protocolRawData | boolean |
| | protocolPbm | boolean |
| | protocolHsmLdi | boolean |
| JxfsPINSupportedProtocols | protocolIsoAs | boolean |
| | protocolIsoLz | boolean |
| | protocolIsoPs | boolean |
| | protocolChipZka | boolean |
| | protocolRawData | boolean |
| | protocolPbm | boolean |
| | protocolHsmLdi | boolean |
| | protocolGenAs | boolean |

| Method | Return | May be used after |
|---|---|---|
| isProtocolIsoAs | boolean | |
| isProtocolIsoLz | boolean | |

| | | |
|---|---|---|
| isProtocolIsoPs | boolean | |
| isProtocolChipZka | boolean | |
| isProtocolRawData | boolean | |
| isProtocolPbm | boolean | |
| isProtocolHsmLdi | boolean | |
| isProtocolGenAs | boolean | |

| Event | May occur after |
|---|---|
| none | |

### Properties

#### protocolIsoAs (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if the ISO 8583 protocol functionality for the authorization system is supported.

#### protocolIsoLz (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if the ISO 8583 protocol functionality for the german "Ladezentrale" is supported.

#### protocolIsoPs (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if the ISO 8583 protocol functionality for the personalisation system is supported.

#### protocolChipZka (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if the ZKA chipcard protocol functionality is supported

#### protocolRawData (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if the raw data protocol functionality is supported.

#### protocolPbm (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if the PBM protocol functionality is supported.

#### protocolHsmLdi (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if the Hsm LDI protocol functionality is supported.

#### protocolGenAs (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if the Gen AS protocol functionality is supported.

### Methods

#### isProtocolIsoAs

**Syntax** ***boolean isProtocolAs();***
**Description** This method returns *true*, if the protocolIsoAs property is set to *true*.

#### isProtocolIsoLz

**Syntax** ***boolean isProtocolLz();***
**Description** This method returns *true*, if the protocolIsoLz property is set to *true*.

#### isProtocolIsoPs

**Syntax** ***boolean isProtocolPs();***
**Description** This method returns *true*, if the protocolIsoPs property is set to *true*.

#### isProtocolChipZka

**Syntax** ***boolean isProtocolChipZka();***
**Description** This method returns *true*, if the protocolChipZka property is set to *true*.

#### isProtocolRawData

**Syntax** ***boolean isProtocolRawData();***
**Description** This method returns *true*, if the protocolRawData property is set to *true*.

#### isProtocolPbm

**Syntax** ***boolean isProtocolPbm();***
**Description** This method returns *true*, if the protocolPbm property is set to *true*.

#### isProtocolHsmLdi

**Syntax** ***boolean isProtocolHsmLdi();***
**Description** This method returns *true*, if the protocolHsmLdi property is set to *true*.

#### isProtocolGenAs

**Syntax** ***boolean isProtocolGenAs();***
**Description** This method returns *true*, if the protocolGenAs property is set to *true*.

## 8.3.13 JxfsPINProtocolSelection

This class is used to select a certain protocol.

### Summary

**Implements :** *Serializable* **Extends :** *JxfsType*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| protocol | int | R | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINProtocolSelection | protocol | int |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |

| Event | May occur after |
|---|---|
| none | |

### Properties

#### protocol (R)

**Type** ***int***
**Initial Value** none
**Description** Specifies the selected protocol to be used.

| Value | Meaning |
|---|---|
| JXFS_PIN_PROTISOAS | ISO 8583 protocol for the authorization system. |
| JXFS_PIN_PROTISOLZ | ISO 8583 protocol for the german "Ladezentrale". |
| JXFS_PIN_PROTISOPS | ISO 8583 protocol for the personalization system |
| JXFS_PIN_PROTCHIPZKA | ZKA chip protocol |
| JXFS_PIN_PROTRAWDATA | Raw data protocol |
| JXFS_PIN_PROTPBM | PBM protocol |
| JXFS_PIN_PROTHSMLDI | HSM LDI protocol |
| JXFS_PIN_PROTGENAS | Gen AS protocol |

### Constructor

#### JxfsPINProtocolSelection

**Syntax** ***JxfsPINProtocolSelection(int protocol) throws JxfsException;***
**Description** .
**Parameter**

| Type | Name | Meaning |
|---|---|---|
| *int* | protocol | Specifies the protocol to be used |

**Exceptions**

| | |
|---|---|
| JXFS_E_PIN_PROTINVALID | Unknown value for the protocol. |

## 8.3.14 JxfsPINTData

This class defines tag/length/value items with no separator to be set/get in the HSM. The methods to access the data are synchronized.

### Summary

**Implements :** *Serializable* **Extends :** *JxfsType*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| data | byte[] | R | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINTData | data | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |
| getTag | *byte[]* | |
| setTag | *Property* | |

| Event | May occur after |
|---|---|
| none | |

### Properties

#### data (R)

**Type** ***byte[]***
**Initial Value** None

**Description** Specifies a set of tag/length/value items where each item consists of
- one byte tag (see list of tags below)
- one byte specifiying the length of the following data as an unsigned binary number
- n bytes of data

| tag (hexdec) | Format | Length | Meaning |
|---|---|---|---|
| C2 | BCD | 4 | Terminal ID ISO BMP 41 |
| C3 | BCD | 4 | Blank Code ISO BMP 42 (rightmost 4 bytes) |
| C4 | BCD | 9 | Account data for terminal account ISO BMP 60 (loading against other card) |
| C5 | BCD | 9 | Account data for fee account ISO BMP 60 ("Laden vom Kartenkonto") |
| C6 | EBCDIC | 40 | Terminal Location ISO BMP 43 |
| C7 | ASCII | 3 | Terminal Currency |
| C8 | BCD | 7 | Online date and time (YYYYMMDDHHMMSS) ISO_BMP 61 |
| C9 | BCD | 4 | Minimum load fee in units of 1/100 of terminal currency, checked against leftmost 4 bytes of ISO BMP 42 |
| CA | BCD | 4 | Maximum load fee in units of 1/100 of terminal currency, checked against leftmost 4 bytes of ISO BMP 42 |
| CB | BIN | 3 | Logical HSM binary coded serial number (starts with 1; 0 means that there are no logical HSMs) |
| CC | EBCDIC | 16 | ZKA ID (is filled during the preinitialisation of the HSM) |
| CD | BIN | 1 | HSM status (1 = irreversibly out of order 2 = out of order, K_UR is not loaded 3 = not pre-initialized, K_UR is loaded 4 = pre-initialized, K_INIT is loaded 5 = initialized/personalized, K_PERS is loaded) |

## Constructor

### JxfsPINTData

**Syntax** ***JxfsPINTData(byte data[]) throws JxfsException;***

**Description** If the data is either null or is not in a valid format, the JXFS_E_PIN_INVALID_TAG exception is thrown.

| Parameter | Type | Name | Meaning |
|---|---|---|---|
| | *byte[]* | data | Specifies the Tag data. |

### Methods

#### getTag

**Syntax** ***synchronized byte[] getTag(byte tag) throws JxfsException;***

**Description** This method returns the contents of the specified tag.

**Parameter**

| Type | Name | Meaning |
|---|---|---|
| *byte* | tag | Specifies the tag. |

**Exceptions**

| Value | Meaning |
|---|---|
| JXFS_E_PIN_INVALID_TAG | The specified tag does not exist in the data. |

#### setTag

**Syntax** ***synchronized void setTag(byte tag, byte value[]) throws JxfsException;***

**Parameter**

| Type | Name | Meaning |
|---|---|---|
| *byte* | tag | Specifies the tag. |
| *byte[]* | value | Specifies the value of the tag to be set. |

**Description** This method sets the appropriate tag in the message. Any same tag will be overwritten.

**Exceptions**

| Value | Meaning |
|---|---|
| JXFS_E_PIN_INVALID_TAG | The specified tag is invalid like in the case of zero or more than 255 bytes of data. |

## 8.3.15 JxfsPINJournalData

This class defines journal data from the HSM.

### Summary

**Implements :** *Serializable* **Extends :** *JxfsType*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| data | byte[] | R | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINJournalData | data | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |

| Event | May occur after |
|---|---|
| none | |

### Properties

#### data (R)

**Type** ***byte[]***

**Initial Value** none

**Description** Journal Data.

### Constructor

#### JxfsPINJournalData

**Syntax** ***JxfsPINJournalData(byte data[]) throws JxfsException;***

**Description** If the data is null, the JXFS_E_PARAMETER_INVALID exception is thrown.

| **Parameter** | **Type** | **Name** | **Meaning** |
|---|---|---|---|
| | *byte[]* | data | Specifies the journal data. |

### 8.3.16 JxfsPINIsoSupportedModes

This class is used to specify the supported charging modes.

#### Summary

**Implements :** *Serializable* **Extends :** *JxfsType*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| chargeAccount | boolean | R | |
| chargeCreditCard | boolean | R | |
| chargeECcard | boolean | R | |
| chargeCash | boolean | R | |
| chargeInternationalECcard | boolean | R | |
| dischargeECcard | boolean | R | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINIsoSupportedModes | chargeAccount | boolean |
| | chargeCreditCard | boolean |
| | chargeECcard | boolean |
| | chargeCash | boolean |
| | chargeInternationalECcard | boolean |
| | dischargeECcard | boolean |

| Method | Return | May be used after |
|---|---|---|
| isChargeAccount | boolean | |
| isChargeCreditCard | boolean | |
| isChargeECcard | boolean | |
| isChargeCash | boolean | |
| isChargeInternationalECcard | boolean | |
| isDischargeECcard | boolean | |

| Event | May occur after |
|---|---|
| none | |

#### Properties

##### chargeAccount (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if charging against an account is supported.

##### chargeCreditCard (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if charging against a credit card is supported.

##### chargeECcard (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if charging against an EC-card is supported.

##### chargeCash (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if charging against cash is supported.

#### chargeInternationalECcard (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if charging against an international EC-card is supported.

#### dischargeECcard (R)

**Type** ***boolean***
**Initial Value** none
**Description** Specifies if discharging against an account of a EC-card is supported.

### Methods

#### isChargeAccount

**Syntax** ***boolean isChargeAccount();***
**Description** This method returns *true*, if the chargeAccount property is set to *true*.

#### isChargeCreditCard

**Syntax** ***boolean isChargeCreditCard();***
**Description** This method returns *true*, if the chargeCreditCard property is set to *true*.

#### isChargeECcard

**Syntax** ***boolean isChargeECcard();***
**Description** This method returns *true*, if the chargeECcard property is set to *true*.

#### isChargeCash

**Syntax** ***boolean isChargeCash();***
**Description** This method returns *true*, if the chargeCash property is set to *true*.

#### isChargeInternationalECcard

**Syntax** ***boolean isChargeInternationalECcard();***
**Description** This method returns *true*, if the chargeInternationalECcard property is set to *true*.

#### isDischargeECcard

**Syntax** ***boolean isDishargeECcard();***
**Description** This method returns *true*, if the dischargeECcard property is set to *true*.

## 8.3.17 JxfsPINHsmInitData

This class defines the necessary data for setting an HSM out of order.

### Summary

**Implements :** *Serializable* **Extends :** *JxfsType*

| Property | Type | Access | Initialized after |
|---|---|---|---|
| initMode | int | R | |
| onlineTime | byte[] | R | |

| Constructor | Parameter | Parameter-Type |
|---|---|---|
| JxfsPINHsmInitData | initMode | int |
| | onlineTime | byte[] |

| Method | Return | May be used after |
|---|---|---|
| get*Property* | *Property* | |

| Event | May occur after |
|---|---|
| none | |

## Properties

### initMode (R)

**Type** *int*
**Initial Value** none
**Description** Specifies the initialization mode as one of the following values:

| Value | Meaning |
|---|---|
| JXFS_PIN_INITTEMP | Initialize the HSM temporarily (K_UR remains loaded) |
| JXFS_PIN_INITDEFINITE | Initialize the HSM definitely (K_UR is deleted) |
| JXFS_PIN_INITIRREVERSIBLE | Initialize the HSM irreversibly (can only be restored by the vendor) |

### onlineTime (R)

**Type** *byte[]*
**Initial Value** none
**Description** Specifies the Online date and time in the format YYYYMMDDHHMMSS like in ISO BMP 61 as BCD packed characters. This parameter is ignored when the init mode equals JXFS_PIN_INITDEFINITE or JXFS_PIN_INITIRREVERSIBLE. If this parameter is null, the length of the array is zero or the value is 0x00 0x00 0x00 0x00 0x00 0x00 0x00 the online time will be set to a value in the past.

## 8.4 Codes

### 8.4.1 Error Codes

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_E_PIN_PROTINVALID | The specified protocol is invalid | 5501 |
| JXFS_E_PIN_MACINVALID | The MAC of the message is not correct | 5503 |
| JXFS_E_PIN_ACCESSDENIED | The encryption module is either not initialized or not ready for any vendor specific reason. | 5504 |
| JXFS_E_PIN_FORMATINVALID | The format of the message is invalid. | 5505 |
| JXFS_E_PIN_CONTENTINVALID | The contents of one of the security relevant fields are invalid. | 5506 |
| JXFS_E_PIN_MODENOTSUPPORTED | Initialization mode not supported. | 5507 |
| JXFS_E_PIN_INVALID_TAG | The value of the tag data is invalid. | 5509 |

### 8.4.2 Status Events

| Value | Meaning | Numerical Value |
|---|---|---|
| JXFS_S_PIN_OPT_REQUIRED | This status event indicates that the online data/time stored in a HSM has been reached.<br>As there are no more details available, the details property of this status event is null.<br><br>This event may be triggered by the clock reaching a previously stored online time or by the online time being set to a time that lies in the past.<br>The online time may be set by the *setHsmTData* method or by a *secureMsgReceive* method that contains a message from a host system containing a new online date/time.<br><br>The event does not mean that any keys or other data in the HSM is out of date now. It just indicates that the terminal should communicate with a "Personalisierungsstelle" as soon as possible using the methods *secureMsgSend* / *secureMsgReceive* and the ISOPS protocol. | 5522 |
| JXFS_S_PIN_HSM_TDATA_CHANGED | This event indicates that one of the values of the terminal data has changed (these are the data that can be set using *setHsmTData*). I.e. this event will be sent especially when the online time or the HSM status is changed because of a | 5523 |

| | | |
|---|---|---|
| | *hsmInit* command or an OPT online dialog (*secureMsgSend* / *secureMsgReceive* with JXFS_PIN_PROTPS).<br><br>The data is a *JxfsPINTData* object. | |

### 8.4.3 Operation Codes

| Constant | Numerical Value |
|---|---|
| JXFS_O_PIN_SEND_MSG | 5524 |
| JXFS_O_PIN_RECEIVE_MSG | 5525 |
| JXFS_O_PIN_GET_JOURNAL | 5526 |
| JXFS_O_PIN_GET_TDATA | 5527 |
| JXFS_O_PIN_SET_TDATA | 5528 |
| JXFS_O_PIN_HSM_INIT | 5529 |

### 8.4.4 Constants

#### ZKA Protocols

| Constant | Numerical Value |
|---|---|
| JXFS_PIN_PROTISOAS | 5511 |
| JXFS_PIN_PROTISOLZ | 5512 |
| JXFS_PIN_PROTISOPS | 5514 |
| JXFS_PIN_PROTCHIPZKA | 5515 |
| JXFS_PIN_PROTHSMLDI | 5516 |
| JXFS_PIN_PROTBMP | 5517 |
| JXFS_PIN_PROTRAWDATA | 5518 |

#### ZKA Initialization Modes

| Constant | Numerical Value |
|---|---|
| JXFS_PIN_INITTEMP | 5519 |
| JXFS_PIN_INITDEFINITE | 5520 |
| JXFS_PIN_INITIRREVERSIBLE | 5521 |

## 8.5 German ZKA GeldKarte

### 8.5.1 Source of ZKA information

The PIN device is able to handle the German "GeldKarte", which is an electronic purse specified by the ZKA (Zentraler Kreditausschuß).

For anyone attempting to write an application that handles these chipcards, it is essential to read and understand the specifications published by

**Bank-Verlag, Köln**
Postfach 30 01 91
D-50771 Köln

Phone: +49 221 5490-0

Fax: +49 221 5490-120

### 8.5.2 How to use the secureMsg methods

This is to describe how an application should use the *secureMsgSend* and secureMessageReceive commands for transactions involving chipcards with a German ZKA GeldKarte chip.

- Applications must call *secureMsgSend* for every command they send to the chip or to a host system, including those commands that do not actually require secure messaging. This enables the device service to remember security-relevant data that may be needed or checked later in the transaction.
- Applications must pass a complete message as input to *secureMsgSend*, with all fields - including those that will be filled by the device service - being present in the correct length. All fields that are not filled by the device service must be filled with the ultimate values in order to enable MACing by the device service.
- Every command *secureMsgSend* that an application issues must be followed by exactly one command secureMessageReceive that informs the device service about the response from the chip or host. If no response is received (timeout or communication failure) the application must issue a secureMessageReceive command with no data (message == null) to inform the device service about this fact.
- If a system is restarted after a *secureMsgSend* was issued to the device service but before the secureMessageReceive was issued, the restart has the same effect as a secureMessageReceive command with message ==null.
- Between a *secureMsgSend* and the corresponding secureMessageReceive no *secureMsgSend* with the same protocol value must be issued. Other executional commands of the PIN device – including *secureMsgSend* / Receive with different protocol – may be used.

### 8.5.3 Protocol JXFS_PIN_PROTISOAS

This protocol handles ISO8583 messages between an ATM and an authorization system (AS).

Only messages in the new ISO format, with new PAC/MAC-format using session keys and Triple-DES are supported

Authorization messages may be used to dispense the amount authorized in cash or to load the amount into an electronic purse (GeldKarte).

For loading a GeldKarte the only type of authorization supported is a transaction originating from track 3 of a German ec-card (message types 0200/0210 for authorization and 0400/0410 for reversal)

For dispensing cash, transactions originating from international cards (message types 0100/0110 and 0400/0410) are supported as well.
The following Bitmaps are overridden by the device service, if present in the given input message:

- BMP52 PAC
- BMP57 Verschlüsselungsparameter(Schlüsselgeneration, Schlüsselversion, RNDMES and RNDPAC)

- BMP61 PAC new (in case of PIN change for MagneticStripeDevice only, according to BDB specification); the Onlinezeitpunkt will always be set by the application, because the application knows it anyway.
- BMP64 MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

The following bitmap positions are checked by the device service and have to be filled by the application:

- Nachrichtentyp
- BMP3 Abwicklungskennzeichen (only for GeldKarte, not for cash)
- BMP4 Transaktionsbetrag (only for GeldKarte, not for cash)
- BMP41 Terminal-ID
- BMP42 Betreiber-BLZ

For a documentation of authorization messages see:

Regelwerk für das deutsche ec-Geldautomaten-System
Date: 09/2003;
Errata: 05. October 2004

Bank-Verlag, Köln
Autorisierungszentrale GA/POS der privaten Banken
Spezifikation für GA-Betreiber
Version 3.21
14. March 2001

Bank-Verlag, Köln
Spezifikationen für den Wechsel der PIN im online-Prozeß
Version 1.1
02. October 1997

dvg Hannover
Schnittstellenbeschreibung für Autorisierungsanfragen bei nationalen GA-Verfügungen unter Verwendung der Spur 3
Version 2.5
Stand: 15.03.2000

dvg Hannover
Schnittstellenbeschreibung für Autorisierungsanfragen bei internationalen Verfügungen unter Verwendung der Spur 2
Version 2.6
Stand: 30.03.2000

### 8.5.4 Protocol JXFS_PIN_PROTISOLZ

This protocol handles ISO8583 messages between a „Ladeterminal" and a „Ladezentrale" (LZ).

Only messages in the new ISO format, with new MAC-format using session keys and Triple-DES are supported.

Both types of GeldKarte chip (type 0 = DEM, type 1 = EUR) are supported.

The following bitmap positions are filled by the device service:

- BMP11: Trace-Nummer
- BMP57: Verschlüsselungsparameter (only the challenge value $RND_{MES}$)
- BMP64: MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

The following bitmap positions are checked by the device service and have to be filled by the application:

- Nachrichtentyp

- BMP3: Abwicklungskennzeichen
- BMP4: Transaktionsbetrag
- BMP12: Uhrzeit
- BMP13: Datum
- BMP25: Konditionscode
- BMP41: Terminal-ID
- BMP42: Betreiber-BLZ (caution: "Ladeentgelt" also in BMP42 is not set by the EPP)
- BMP61: Online-Zeitpunkt
- BMP62: Chipdaten

The following bitmap positions are only checked if they are available:

- BMP43: Standort
- BMP60: Kontodaten Ladeterminal

For a documentation of the Ladezentrale interface see:
ZKA / Bank-Verlag, Köln
Schnittstellenspezifikation für die ec-Karte mit Chip
Geldkarte Ladeterminals
Version 3.0
2. 4. 1998

### 8.5.5 Protocol JXFS_PIN_PROTISOPS

This protocol handles ISO8583 messages between a terminal and a "Personalisierungsstelle" (PS). These messages are about OPT.

The device service creates the whole message with *secureMsgSend*, including message type and bitmap.

For a documentation of the Personalisierungsstelle interface see:
ZKA / Bank-Verlag, Köln
Schnittstellenspezifikation für die ec-Karte mit Chip
Online-Personalisierung von Terminal-HSMs
Version 3.0
2. 4. 1998

### 8.5.6 Protocol JXFS_PIN_PROTCHIPZKA

This protocol is intended to handle messages between the application and a GeldKarte.

Both types of GeldKarte are supported.

Both types of load transactions ("Laden vom Kartenkonto" and "Laden gegen andere Zahlungsmittel") are supported.

See the chapter "Command Sequence" below for the actions that device service s take for the various chip card commands.

Only the command APDUs to and the response APDUs from the chip must be passed to the device service, the ATR (answer to reset) data from the chip is not passed to the device service.

For a documentation of the chip commands used to load a GeldKarte see:
ZKA / Bank-Verlag, Köln
Schnittstellenspezifikation für die ec-Karte mit Chip
Ladeterminals
Version 3.0
2. 4. 1998

ZKA / Bank-Verlag, Köln
Schnittstellenspezifikation für die ZKA-Chipkarte
Online-Vor-Initialisierung und Online-Anzeige
einer Außerbetriebnahme von Terminal-HSMs

Version 1.0
04.08.2000

### 8.5.7 Protocol JXFS_PIN_PROTRAWDATA

This protocol is intended for vendor-specific purposes. Generally the use of this protocol is not recommended and should be restricted to issues that are impossible to handle otherwise.

For example a HSM that requires vendor-specific, cryptographically secured data formats for importing keys or terminal data may use this protocol.

Applicaton programmers should be aware that the use of this command may prevent their applications from running on different hardware.

### 8.5.8 Protocol JXFS_PIN_PROTPBM

This protocol handles host messages between a terminal and a host system, as specified by IBM's PBM protocol.
For a documentation of this protocol see:
IBM 473x / Personal Banking Machines / Programmer's Reference
Volume 1 - 4 / GA19-5510 - GA19-5513

Some additions are defined to the PBM protocol in order to satisfy the German ZKA 3.0 PAC/MAC standard. See:
Diebold's and IBM's Specification for support of Online Preinitialization and Personalization of Terminal HSMs (OPT) and support for the PAC/MAC standards for th 473x Protocol.
Diebold USA, Revision 1.8, revised on Jan-03-2001

The commands *secureMsgSend* and secureMessageReceive handle the PAC and MAC in the VARDATA 'K' subfield of transactions records and responses. The MAC in the traditional MACODE field is not affected.
In order to enable the service provider to understand the messages, the application must provide the messages according to the following rules:

- All alphanumeric fields must be coded in EBCDIC
- Pre-Edit (padding and blank compression) must not be done by the application. The service provider will check the MACMODE field and do what has to be done.
- In order to enable the service provider to find the vardata subfield 'K', it must be included in the message by the application, with the indicator 'K' and its length set.
- Because CARDDATA (track 2) and T3DATA (track 3) fields always take part in the MAC computation for a transaction record, these fields must be included in the message, even if they already have been sent to the host in a previous transaction record and the CI-Option SHORTREC prevents them from being sent again.

### 8.5.9 Protocol JXFS_PIN_PROTHSMLDI

With this protocol an application can request information about the personalized OPT groups.

The information returned consists of personalisation record like in BMP62 of an OPT response but without MAC.

Data format:

XX XX VV group ID and versions number
XX number of LDIs within the group (binary coded)
...
first LDI of the group
...
last LDI of the group
XX XX VV group ID and versions number
...
etc. for several groups

Each LDI consists of

| | |
|---|---|
| NN | Number of the LDI |
| 00 | Alg. code |
| LL | Length of the following data |
| XX...XX | data of the LDI |

The device service must at least return the standard LDI, but can return more LDIs.

### 8.5.10 Protocol JXFS_PIN_PROTGENAS

This protocol allows one to create a PAC (encrypted Pin-Block) and to create and verify a MAC for a proprietary message. As the device service doesn't know the message format, it cannot complete the message by adding security relevant fields like random values, PAC and MAC, like it does for the protocol JXFS_PIN_PROTISOAS. Only the application is able to place these fields into the proper locations. Using this protocol, an application can generate the PAC and the random values in separate steps, add them to the proprietary send-message, and finally let the device service generate the MAC. The generated MAC can then be added to the send-message as well.
For a received message, the application extracts the MAC and the associated random value and passes them along with the entire message data to the device service for MAC verification.
PAC generation supports Pin-Block ISO-Format 0 and 1.

Command description:
The first byte of the field *messageData* of the JxfsPINSecureMsgPacMac object contains a subcommand, which is used to qualify the type of operation. The remaining bytes of the message data are dependent on the value of the subcommand.

The following sub-commands are defined:

- GeneratePAC (Code 0x01)
  Returns the encrypted Pin-Block together with generation and version values of the Master Key and the PAC random value.

- GetMACRandom (Code 0x02)
  Returns the generation and version values of the Master Key and the MAC random value.

- GenerateMAC (Code 0x03)
  Returns the generated MAC for the message data passed in. Note, that the MAC is generated for exactly the data that is presented (contents and sequence). Data that should not go into the MAC calculation must not be passed in.

- VerifyMAC (Code 0x04)
  Generates a MAC for the data passed in and compares it with the provided MAC value. MAC random value, key generation and key version must be passed in separately.

**PROTGENAS Error Codes**

The error code JXFS_E_PIN_FORMATINVALID is returned when:

- the subcommand in Byte 0 of msgData for Command *secureMsgSend* with protocol JXFS_PIN_PROTGENAS is not 01, 02 or 03.
- the subcommand in Byte 0 of msgData for Command *secureMsgReceive* with protocol JXFS_PIN_PROTGENAS is not 04.
- the subcommand in Byte 0 of msgData for Command *secureMsgReceive* with protocol JXFS_PIN_PROTGENAS is 01 and Byte 1 is not 00 and not 01 (Pin-Block format is not ISO-0 and ISO-1).
- the individual command data length for a subcommand is less than specified.

The error code JXFS_E_PIN_HSMSTATEINVALID is returned when:

- the subcommand in Byte 0 of msgData for Command *secureMsgSend* with protocol JXFS_PIN_PROTGENAS is 03 (Generate MAC) without a preceding GetMACRandom (*secureMsgSend* with subcommand 02).

The error code JXFS_E_PIN_MACINVALID is returned when

- the subcommand in Byte 0 of msgData for Command *secureMsgReceive* with protocol JXFS_PIN_PROTGENAS is 04 (Verify MAC) and the MACs didn't match.

The error code JXFS_E_PIN_KEYNOTFOUND is returned when
- the subcommand in Byte 0 of msgData for Command *secureMsgSend* with protocol JXFS_PIN_PROTGENAS is 01 (Generate PAC) and the device service doesn't find a master key.
- the subcommand in Byte 0 of msgData for Command *secureMsgSend* with protocol JXFS_PIN_PROTGENAS is 02 (Get MAC Random) and the device service doesn't find a master key.
- the subcommand in Byte 0 of msgData for Command *secureMsgReceive* with protocol JXFS_PIN_PROTGENAS is 04 (Verify MAC) and the device service doesn't find a key for the provided key generation and key version values.

The error code JXFS_E_PIN_NOPIN is returned when
- the subcommand in Byte 0 of msgData for Command *secureMsgSend* with protocol JXFS_PIN_PROTGENAS is 01 (Generate PAC) and no PIN or insufficient PIN-digits have been entered.

### 8.5.11 Command Sequence

The following list shows the sequence of actions an application has to take for the various GeldKarte Transactions. Please note that this is a summary and is just intended to clarify the purpose of the chipcard-related methods of the JxfsPINIso interface. In no way it can replace the ZKA specifications mentioned above.

| Method | protocol | message data | action of device service |
|---|---|---|---|
| **Preparation for Load/Unload** | | | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>SELECT FILE DF_BÖRSE | |
| *secureMsgReceive* | CHIPZKA | Response APDU | recognize type of chip |
| *secureMsgSend* | CHIPZKA | Command APDU<br>READ RECORD EF_ID | |
| *secureMsgReceive* | CHIPZKA | record EF_ID | store EF_ID |
| *secureMsgSend* | CHIPZKA | Command APDU<br>READ RECORD EF_LLOG | |
| *secureMsgReceive* | CHIPZKA | record EF_LLOG | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>READ_RECORD EF_BÖRSE | |
| *secureMsgReceive* | CHIPZKA | record EF_BÖRSE | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>READ_RECORD<br>EF_BETRAG | |
| *secureMsgReceive* | CHIPZKA | record EF_BETRAG | |
| **Load against other ec-Card** | | | |
| *secureMsgSend* | CHIPZKA | **for type 0 chips only**<br>Command APDU<br>READ RECORD EF_KEYD | |
| *secureMsgReceive* | CHIPZKA | record EF_KEYD | |
| *secureMsgSend* | CHIPZKA | **for type 1 chips only**<br>Command APDU<br>GET KEYINFO | |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>GET CHALLENGE | |
| *secureMsgReceive* | CHIPZKA | Random number RND1 from Chip | store RND1 |

| Method | protocol | message data | action of device service |
|---|---|---|---|
| *secureMsgSend* | CHIPZKA | Command APDU<br>LADEN EINLEITEN<br>with Secure Msg. | fill<br>-Terminal ID<br>-Traceno.<br>-RND2<br>-MAC |
| *secureMsgReceive* | CHIPZKA | Response APDU | store response APDU for later check of ISOLZ message, BMP 62 |
| *secureMsgSend* | ISOAZ | ISO8583 message 0200<br>Authorization Request | fill<br>- Traceno. (BMP 11)<br>- PAC (BMP 52)<br>- $RND_{MES}$ + $RND_{PAC}$ (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields |
| *secureMsgReceive* | ISOAZ | ISO8583 message 0210<br>Authorization Response | check MAC and other security relevant fields |
| *secureMsgSend* | ISOLZ | ISO8583 message 0200<br>Ladeanfrage | fill<br>- Traceno. (BMP 11)<br>- $RND_{MES}$ (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields. |
| *secureMsgReceive* | ISOLZ | ISO8583 message 0210<br>Ladeantwort | check MAC and other security relevant fields, store BMP62 for later use in LADEN command. |
| *secureMsgSend* | CHIPZKA | Command APDU<br>GET CHALLENGE | |
| *secureMsgReceive* | CHIPZKA | Random number RND3 from chip | store RND3 |
| *secureMsgSend* | CHIPZKA | Command APDU<br>LADEN with Secure Msg. | provide complete command from BMP62 of ISOLZ response , compute command MAC |
| *secureMsgReceive* | CHIPZKA | Response APDU | check response MAC |
| GET_JOURNAL | ISOLZ | Vendor specific | |
| GET_JOURNAL | ISOAZ | Vendor specific | |
| **Reversal of a Load against other ec-Card** | | | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>SELECT FILE DF_BÖRSE | |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>GET CHALLENGE | |
| *secureMsgReceive* | CHIPZKA | Random number RND5 from chip | store RND5 |
| *secureMsgSend* | CHIPZKA | Command APDU<br>LADEN EINLEITEN<br>with Secure Msg. | fill<br>-Terminal ID<br>-Traceno.<br>-RND6<br>-Keyno. $KGK_{LT}$<br>-MAC |
| *secureMsgReceive* | CHIPZKA | Response APDU | store response APDU for later check of ISOLZ message, BMP 62 |
| *secureMsgSend* | ISOAZ | ISO8583 message 0400<br>Storno | fill<br>- Traceno. (BMP 11)<br>- PAC (BMP 52)<br>- $RND_{MES}$ + $RND_{PAC}$ (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields |
| *secureMsgReceive* | ISOAZ | ISO8583 message 0410<br>Storno Response | check MAC and other security relevant fields. |
| *secureMsgSend* | ISOLZ | ISO8583 message 0400<br>Storno | fill<br>- Traceno. (BMP 11)<br>- $RND_{MES}$ (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields. |

| Method | protocol | message data | action of device service |
|---|---|---|---|
| *secureMsgReceive* | ISOLZ | ISO8583 message 0410<br>Storno Response | check MAC and other security relevant fields, store BMP62 for later use in LADEN command. |
| *secureMsgSend* | CHIPZKA | Command APDU<br>GET CHALLENGE | |
| *secureMsgReceive* | CHIPZKA | Random number RND7 from chip | store RND7 |
| *secureMsgSend* | CHIPZKA | Command APDU<br>LADEN with Secure Msg. | provide complete command from BMP62 of ISOLZ response , compute command MAC |
| *secureMsgReceive* | CHIPZKA | Response APDU | check response MAC |
| GET_JOURNAL | ISOLZ | Vendor specific | |
| GET_JOURNAL | ISOAZ | Vendor specific | |
| **PIN Verification Type 0** | | | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>GET CHALLENGE | |
| *secureMsgReceive* | CHIPZKA | Random number RND0 from chip | store RND0 |
| *secureMsgSend* | CHIPZKA | Command APDU<br>EXTERNAL AUTHENTICATE | fill<br>-Keyno. KINFO<br>-ENCRND |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>PUT DATA | fill RND1 |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>READ RECORD<br>EF_INFO<br>with Secure Messaging | |
| *secureMsgReceive* | CHIPZKA | record EF_INFO | check MAC |
| *secureMsgSend* | CHIPZKA | Command APDU<br>GET CHALLENGE | |
| *secureMsgReceive* | CHIPZKA | Random number RND2 from chip | store RND2 |
| *secureMsgSend* | CHIPZKA | Command APDU<br>VERIFY | provide complete command APDU |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| **PIN Verification Type 1** | | | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>GET KEYINFO | |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>GET CHALLENGE | |
| *secureMsgReceive* | CHIPZKA | Random number RND0 from chip | store RND0 |
| *secureMsgSend* | CHIPZKA | Command APDU<br>MUTUAL AUTHENTICATE | fill ENC0 |
| *secureMsgReceive* | CHIPZKA | Response APDU | check ENC1 |
| *secureMsgSend* | CHIPZKA | Command APDU<br>VERIFY | provide complete command APDU |
| *secureMsgReceive* | CHIPZKA | Response APDU | check MAC |
| **„Laden vom Kartenkonto“ (both types)** | | | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>LADEN EINLEITEN | fill<br>-Terminal ID<br>-Trace No. |
| *secureMsgReceive* | CHIPZKA | Response APDU | |

| Method | protocol | message data | action of device service |
|---|---|---|---|
| *secureMsgSend* | ISOLZ | ISO8583 message 0200<br>Ladeanfrage | fill<br>- Traceno. (BMP 11)<br>- RNDMES (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields. |
| *secureMsgReceive* | ISOLZ | ISO8583 message 0210<br>Ladeantwort | check MAC and other security relevant fields. |
| *secureMsgSend* | CHIPZKA | Command APDU<br>LADEN | |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| GET_JOURNAL | ISOLZ | Vendor specific | |
| **Reversal of a „Laden vom Kartenkonto“** | | | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>SELECT FILE DF_BÖRSE | |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | CHIPZKA | Command APDU<br>LADEN EINLEITEN | fill<br>-Terminal ID<br>-Traceno. |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | ISOLZ | ISO8583 message 0400<br>Storno | fill<br>- Traceno. (BMP 11)<br>- RNDMES (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields. |
| *secureMsgReceive* | ISOLZ | ISO8583 message 0410<br>Storno Response | check MAC and other security relevant fields |
| *secureMsgSend* | CHIPZKA | Command APDU<br>LADEN | |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| GET_JOURNAL | ISOLZ | Vendor specific | |
| **Unload** | | | |
| *secureMsgSend* | CHIPZKA | ENTLADEN EINLEITEN | fill<br>-Terminal ID<br>-Trace No. |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | ISOLZ | ISO8583 message<br>Entladeanfrage 0200 | fill<br>- Traceno. (BMP 11)<br>- RNDMES (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields. |
| *secureMsgReceive* | ISOLZ | ISO8583 message<br>Entladeantwort 0210 | check MAC and other security relevant fields |
| *secureMsgSend* | CHIPZKA | ENTLADEN | |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | CHIPZKA | ENTLADEN EINLEITEN | fill<br>-Terminal ID<br>-Trace No. |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| *secureMsgSend* | ISOLZ | ISO8583 message<br>Entladequittung 0202 | fill<br>- Traceno. (BMP 11)<br>- RNDMES (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields. |

| Method | protocol | message data | action of device service |
|---|---|---|---|
| *secureMsgReceive* | ISOLZ | ISO8583 message<br>Entladebestätigung 0212 | check MAC and other security relevant fields |
| *secureMsgSend* | CHIPZKA | Command APDU<br>ENTLADEN | |
| *secureMsgReceive* | CHIPZKA | Response APDU | |
| GET_JOURNAL | ISOLZ | Vendor specific | |
| **Repeated Messages (Stornowiederholung / Entladequittungswiederholung)** | | | |
| *secureMsgSend* | ISOLZ | ISO8583 message<br>Stronowiederholung 0401 or<br>Entladebestätigungswiederholung 0203 | fill<br>- Traceno. (BMP 11)<br>- RNDMES (BMP 57)<br>- MAC (BMP 64)<br>check other security relevant fields. |
| *secureMsgReceive* | ISOLZ | ISO8583 message<br>Stornoantwort 410 or<br>Entladequittung 0212 | check MAC and other security relevant fields |
| GET_JOURNAL | ISOLZ | Vendor specific | |

### 8.5.12 Command Sequence MAC/PAC non-ISO 8583

The following list shows sample sequence information of actions for handling non-ISO 8583 related MAC/PAC handling. Please note that this is a summary and is just intended to clarify the purpose of the chipcard-related methods of the JxfsPINIso interface. In no way it can replace the appropriate specifications.

| Method | protocol | message data | action of device service |
|---|---|---|---|
| *secureMsgSend* | GENAS | Byte 0: 0x01<br>(Generate PAC)<br>Byte 1: format (0 or 1)<br>Byte 2-9: ANF (Primary Account Number, if length is less than 12 digits, value must be left padded with binary 0,<br>only applicable for format | Generates a session key for PAC generation and finally the PAC itself.<br>Determine generation and version values of Master-Key and return them along with the random value.<br>OC data:<br>Byte 0: key generation<br>Byte 1: key version<br>Byte 2-17: PAC random<br>Byte 18-25: PAC value<br>(all values are binary values) |
| *secureMsgReceive* | GENAS | Byte 0: 0x02<br>(Get MAC Random) | Generates a session key for MAC generation (see next step below).<br>Determine generation and version values of Master-Key and return them along with the random value.<br>OC data:<br>Byte 0: key generation<br>Byte 1: key version<br>Byte 2-17: MAC random<br>(all values are binary values) |
| *secureMsgSend* | GENAS | Byte 0: 0x03<br>(Generate MAC)<br>Byte 1-n: Message to be mac'ed<br>(all values are binary values) | Generates MAC over bytes 1-n of the inbound message using the session key created in the previous step.<br>OC data:<br>Byte 0-7: generated MAC(binary value). |
| *secureMsgReceive* | GENAS | Byte 0: 0x04<br>(Verify MAC)<br>Byte 1: key generation<br>Byte 2: key version<br>Byte 3-18: MAC random<br>Byte 19-26: MAC<br>Byte 27-n: Message to be verified<br>(all values are binary values)<br>Note: If no message has been received, this function must be called by omitting Bytes 1-n | Generates a session key using the Master key identified by key generation and version by using the random value passed in.<br>Generates a MAC for the message data passed in and compare the resulting MAC with the MAC passed in. |

# 9 Appendix B: EMV Clarifications

EMV support by this specification consists in the ability of :

- importing Certification Authority and Chip Card rsa public keys,
- creating the PIN Blocks for offline PIN verification
- verifying static and dynamic authenticaton data

## 9.1 EMV Support

The PIN service is able to manage the EMV chip card regarding the card authentication and the RSA local PIN verification. Two steps are mandatory in order to reach these two functions

- The loading of the keys which come from the Certification Authorities or from the Chipcard itself
- AND the EMV PIN block format management

The Device Services is responsible for all key validation during the import process. The application is responsible for management of the key lifetime and expiry after the key is successfully imported

## 9.2 Key loading

The final goal of an application is to retrieve the keys located on a Chip card to perform the operations of authentication or local PIN check (RSA encrypted). These keys are provided by the card using EMV certificates and can be retrieved using a public key provided by a Certification Authority. The application should first load the keys issued by the Certification Authority. At transaction time the application will use these keys to load the keys that the application has retrieved from the chip card.

## 9.3 Certification Authority keys

These keys are provided in the following formats :

- Plain text
- Plain Text with EMV 2000 Verification Data
- EPI CA (or self signed) format as specified in the Europay International, EPI CA Module Technical – Interface specification Version 1.4
- PKCSV1_5 encrypted (as used by GIECB in France).

## 9.4 EPI CA format

The following table corresponds to table 4 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 and identifies the Europay Public Key (self-certified) and the associated data:

| **Field name** | **Length** | **Description** | **Format** |
|---|---|---|---|
| ID of Certificate Subject | 5 | RID for Europay | Binary |
| Europay public key Index | 1 | Europay public key Index | Binary |
| Subject public key Algorithm Indicator | 1 | Algorithm to be used with the Europay public key Index, set to 0x01 | Binary |
| Subject public key Length | 1 | Length of the Europay public key Modulus (equal to *Nca)* | Binary |
| Subject public key Exponent Length | 1 | Length of the Europay public key Exponent | Binary |
| Subject public key Exponent | 1 | | Binary |
| Leftmost Digits of Subject public key | *Nca*-37 | *Nca*-37 most significant bytes of the Europay public key Modulus | Binary |
| Subject public key Remainder | *37* | 37 least significant bytes of the Europay public key Modulus | Binary |
| Subject public key Exponent | *1* | Exponent for Europay public key | Binary |
| Subject public key Certificate | *Nca* | Output of signature algorithm | Binary |

Table 1

The following table corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 and identifies the Europay Public Key Hash code and associated data:

| Field name | Length | Description | Format |
|---|---|---|---|
| ID of Certificate Subject | 5 | RID for Europay | Binary |
| Europay public key Index | 1 | Europay public key Index | Binary |
| Subject public key Algorithm Indicator | 1 | Algorithm to be used with the Europay public key Index, set to 0x01 | Binary |
| Certification Authority public key Check Sum | 20 | Hash-code for Europay public key | Binary |

Table 2

Table 2 corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4

**Chip card keys**

These keys are provided as EMV certificates which come from the chip card in a multiple layer structure (issuer key first, then the ICC keys). Two kinds of algorithm are used with these certificates in order to retrieve the keys : One for the issuer key and the other for the ICC keys (ICC public key and ICC PIN encipherment key). The associated data with these algorithms – The PAN (Primary Account Number) and the SDA(Static Data to be Authenticated) - come also from the chip card

## 9.5 PIN block management

The PIN block management is done through the *createPINBlock* method. A new format JXFS_PIN_FMT_EMV has been added to indicate to the PIN service that the PIN block must follow the requirements of the EMVco, Book2 – Security & Key management Version 4.0 document The parameter *customerData* is used in this case to transfer to the PIN service the challenge number coming from the chip card. The final encryption must be done using a RSA public key. Please note that the application is responsible to send the PIN block to the chip card inside the right APDU

## 9.6 SHA-1 Digest

The SHA-1 Digest is a hash algorithm used by EMV in validating ICC static and dynamic data item. The SHA-1 Digest is supported through the *computeSHA1Digest* command. The application will pass the data to be hashed to the Device Service. Once the encryptor completes the SHA-1 hash code, the Device Service will return the 20-byte hash value back to the application.

# 10 Appendix C: Remote Key Loading Clarifications

## 10.1 Background Information

Most cryptographic functions used within Financial Industry transactions will continue to be based on symmetric key technology using either DES or triple DES. It is essential within symmetric key cryptography for the keys to be kept secret.

All the key exchanges between the host and the financial terminal are based on the initial encryption key (master key). This key was loaded at the installation of the self-service terminal (SST) and, usually, was the same during all the lifetime of the SST. The replacement of the master key needed human intervention and heavy safety rules in order to keep the master key secret and this key could not be downloaded because the current specifications do not provide all features required for supporting it in a branch or self-service environment for Remote Key Loading.

**The new cryptographic rules in several countries mandate to replace the master key regularly.**

This proposal provides mechanisms for the exchange of these symmetric keys in a secure automated way where the end points can be sure the data communicated is from a trusted source.

Remote Key loading allows an initial encryption symmetric key (master key) to be downloaded from a host using the Public Key Infrastructure (PKI) for encryption and verification of the master key.

The public key infrastructure (**PKI**) is based on asymmetric keys. An asymmetric key is composed of two parts:

- The **public key** part. It can be distributed to trusted persons.
- The **private key** part. It is kept secret by the one who generate it

This document is a proposal for 2 different mechanisms for remote key loading.

- **Remote key loading using signatures:** It is 2-parties Authentication scheme, where the host and SST authenticate each other directly. The key sent by the host is enciphered with RSA cryptographic method. The SST will decipher and verify the validity of the key before loading it into the security module.
- **Remote key loading using certificates:** It is a 3-parties Authentication scheme, where a third party, the certification authority, is ultimately responsible for the authentication / trust relationship.

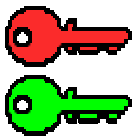
## 10.2 Appendix C1: REMOTE KEY LOADING USING SIGNATURES

### 10.2.1 What is a digital signature?

A **digital signature** is a digital code that can be attached to an electronically transmitted message that uniquely identifies the sender. Like a written signature, the purpose of a digital signature is to guarantee that the individual sending the message really is who he or she claims to be.

### 10.2.2 How it works?

Digital signatures rely on a public key infrastructure (PKI). The PKI model involves an entity, such as a Host, having a pair of encryption keys – one private, one public. These keys work in consort to encrypt, decrypt and authenticate data. One way authentication occurs is through the application of a digital signature.



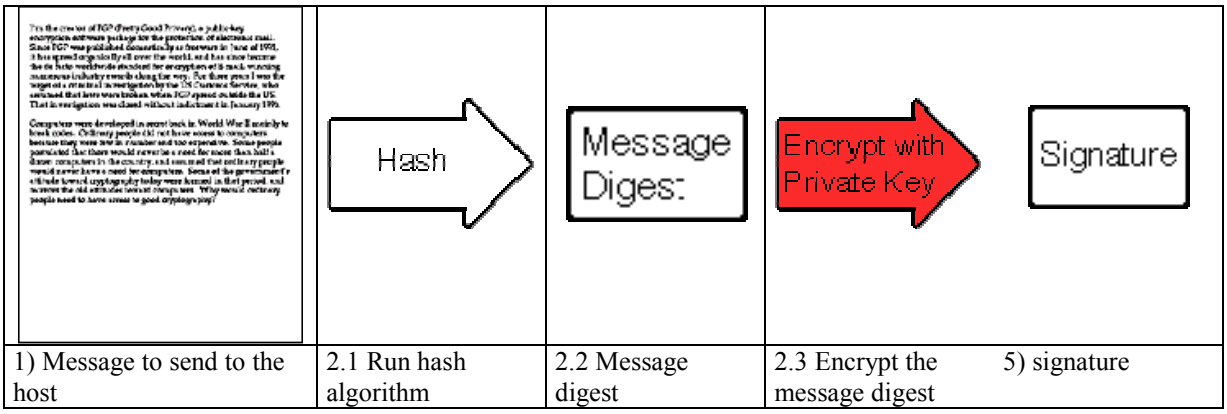
Private key: This is kept secret by the host



Public key: This key is distributed to trusted SST

For example:

- 1) The Host creates some data that it would like to digitally sign;
- 2) Host runs the data through a hashing algorithm to produce a hash or digest of the data. The digest is unique to every block of data – a digital fingerprint of the data, much smaller and therefore more economical to encrypt than the data itself;
- Encrypt the Digest the Host's private key. **This is the digital signature.**

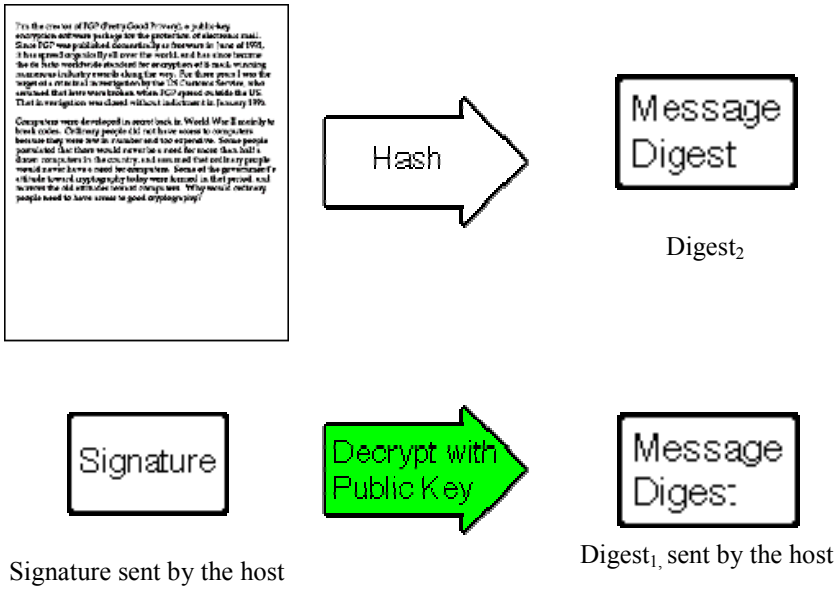



The Host then sends the following to the SST:

- data block;
- digital signature
- Host's public key

To validate the signature, the SST performs the following:

- SST runs data through the standard hashing algorithm – the same one used by the Host – to produce a digest of the data received. *Consider this digest*$_2$;
- SST uses the Host's public key to decrypt the digital signature. The digital signature was produced using the Host's private key to encrypt the data digest; therefore, when decrypted with the Host's public key it produces the same digest. *Consider this digest*$_1$. Incidentally, no other public key in the world would work to decrypt digest$_1$ – only the public key corresponding to the signing private key.
- SST compares digest$_1$ with digest$_2$




If digest$_1$ matches digest$_2$ exactly, the SST has confirmed the following:

- Data was not tampered with in transit. Changing a single bit in the data sent from the Host to the SST would cause digest$_2$ to be different than digest$_1$. Every data block has a unique digest; therefore, the SST detects an altered data block.
- Public key used to decrypt the digital signature corresponds to the private key used to create it. No other public key could possibly work to decrypt the digital signature, so the SST was not handed someone else's public key.

This gives an overview of Digital Signatures can be used in **Data Authentication**, in particular, to validate and securely install encryption keys.

### 10.2.3 Key Exchange using Digital Signatures

This section describes Key Exchange using Digital signatures.

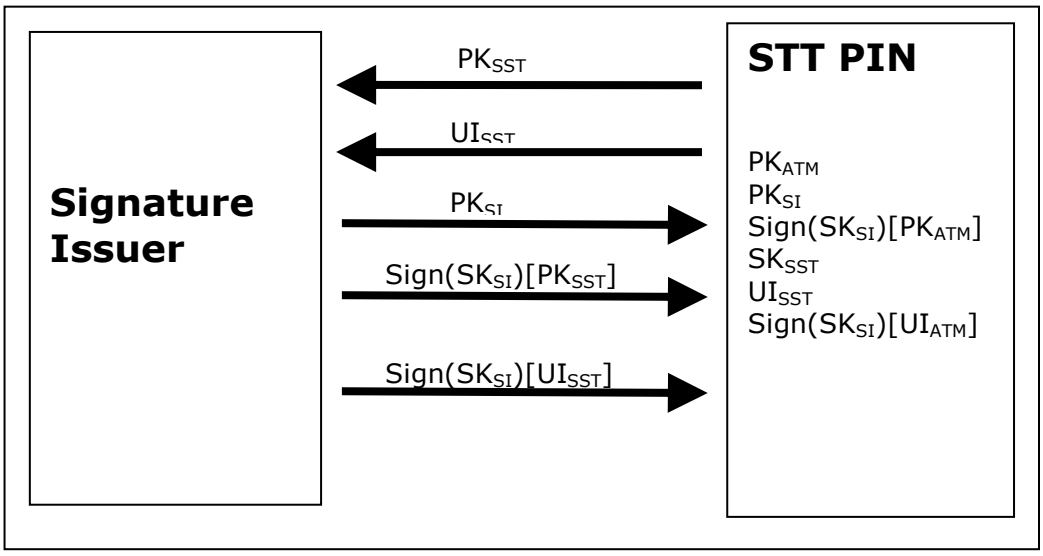
#### Initialization phase

At production time, a RSA key pair, which is unique for each device, is loaded into the PinPad.

Then a trusted third party, the Signature Issuer, is used to generate the signatures for the Public keys of each end point, ensuring their validity.

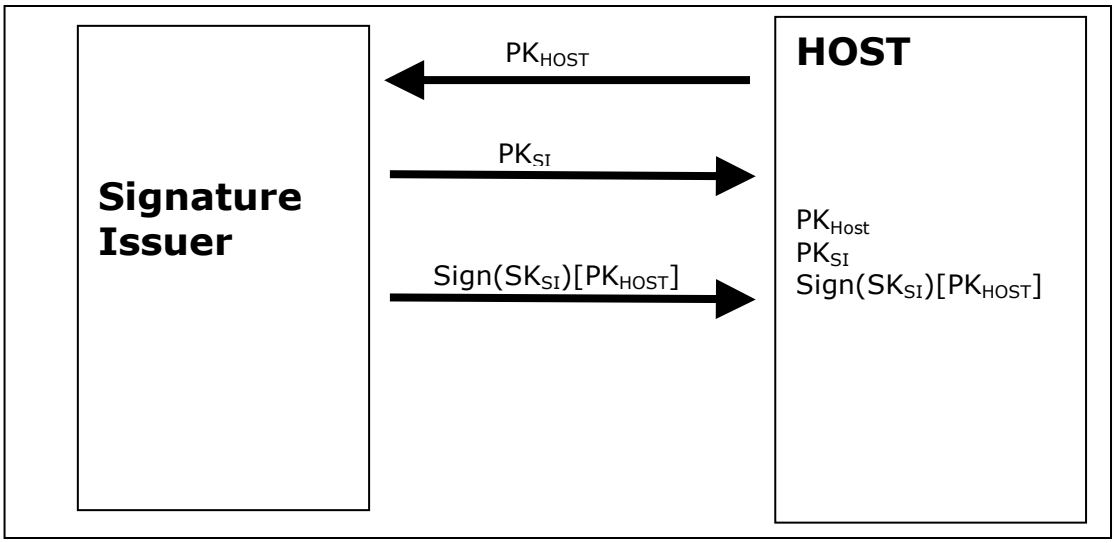
The initialization of the device is done in a secure environment; typically this is done during manufacture time or at the installation time in the customer premises.

**Initialization Phase – Signature Issuer & SST PIN**



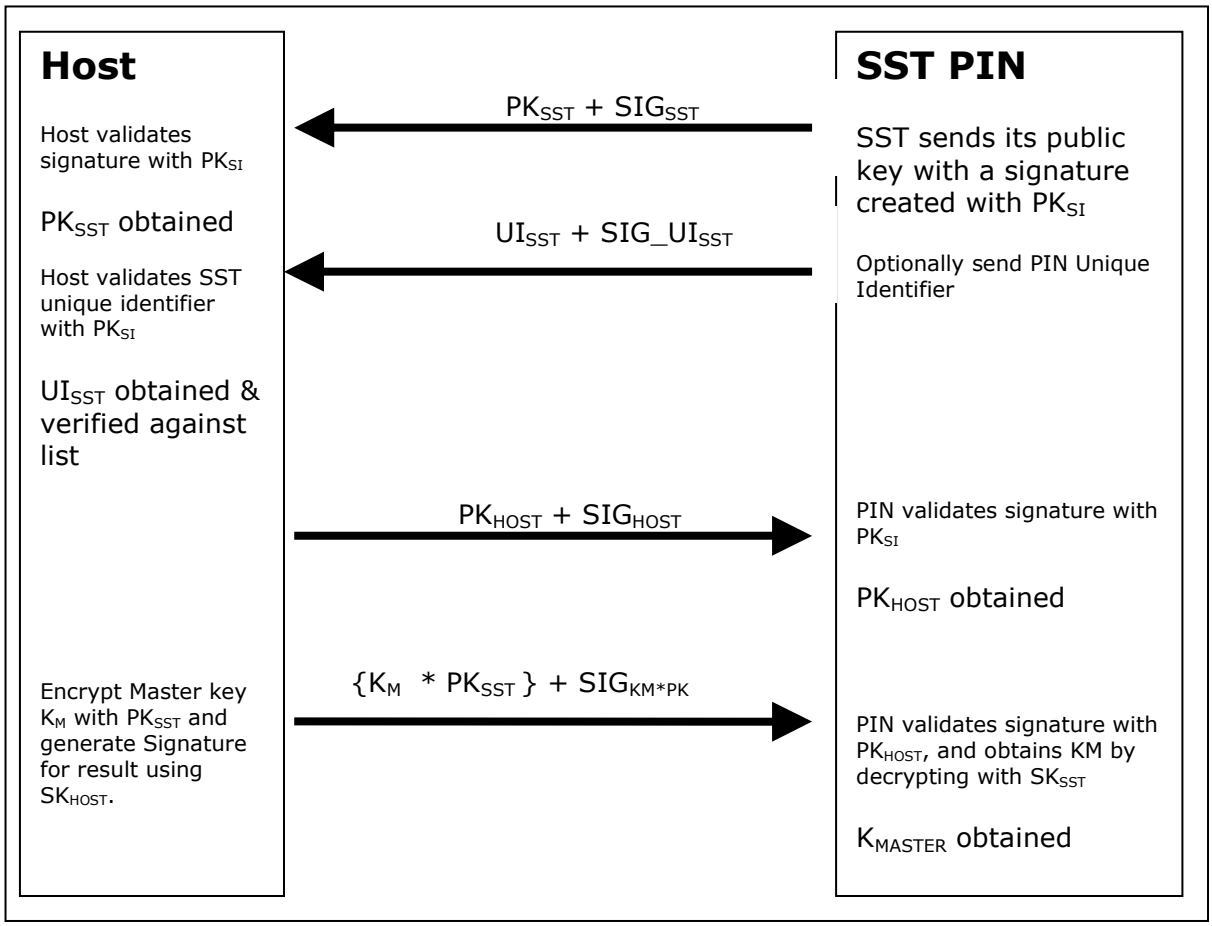

| | |
|---|---|
| $PK_{SST}$ | Public key of the Self Service Terminal. This key is either the one loaded in the PinPad in production or installation process or generated by the PinPad by the command *generateRSAKeyPair*. |
| $UI_{SST}$ | Unique Identifier of the Self Service Terminal (Optional) |
| $PK_{SI}$ | Public key of the Signature Issuer |
| $Sign(SK_{SI})[PK_{SST}]$ | Signature of the Public key of the Self Service enciphered with the signature issuer private key. |
| $Sign(SK_{SI})[UI_{SST}]$ | Signature of the unique identifier of the Self Service enciphered with the signature issuer private key. |

**Initialization Phase – Signature Issuer & HOST**



| | |
|---|---|
| $PK_{HOST}$ | Public key of the Host |
| $PK_{SI}$ | Public key of the Signature Issuer |
| $Sign(SK_{SI})[PK_{HOST}]$ | Signature of the Public key of the Host enciphered with the signature issuer private key. |

**Key Exchange – Host & SST PIN**




Step 1

**The SST sends its Public Key to the Host** in a secure structure: The SST PIN sends its SST Public Key with its associated Signature created by the Issuer 's Public Key. When the Host receives this information it will use the Signature Issuer's Public Key to validate the signature and obtain the SST Public Key.

Step 2 (Optional)

1. **The Host verifies that the key** it has just received is from a valid sender. It does this by obtaining the PIN device unique identifier. The SST PIN sends its Unique Identifier with its associated Signature created by the Issuer 's Private Key. When the Host receives this information it will use the Signature Issuer's Public Key to validate the signature and retrieve the PIN Unique Identifier. It can then check this against the list it received from the Signature Issuer. In a private SST network, it should not have any possibility of impersonation, i.e. that another device takes the role of an SST and fools the Host. The unique identifier prevents from any impersonation.

Step 3

**The Host sends its public key to the SST**: The Host sends its Public Key and associated Signature. The SST PIN verifies the signature using $PK_{SI}$ and stores the key if it is valid.

Step 4

**The SST PIN receives its Master Key from the Host**: The Host encrypts the Master Key ($K_M$) with $PK_{SST}$. A signature for this is then created using $SK_{HOST}$. The SST PIN will then validate the signature using $PK_{HOST}$ and then obtain the master key by decrypting using $SK_{SST}$.

## 10.3 Appendix C 2: REMOTE KEY LOADING USING CERTIFICATES

### 10.3.1 What is a digital certificate?

**Digital certificates** are electronic files containing the user's public key and specific identifying information about the user. They are tamper-proof and cannot be forged. Much as a passport office does in issuing a passport, a Certification Authority certifies that the individual granted the digital certificate is who he or she claims to be.

Digital certificates do two things:

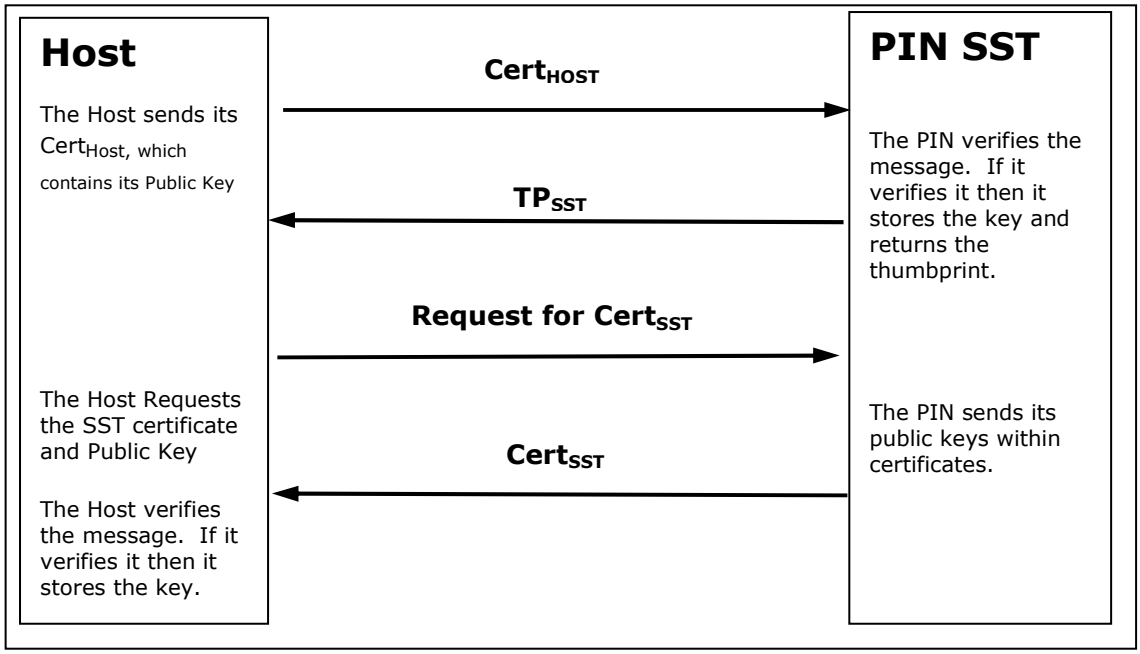
- They authenticate that their holders - people, web sites, and even network resources such as routers - are truly who or what they claim to be.
- They protect data exchanged online from theft or tampering.

### 10.3.2 What is a certification Authority?

A **Certification Authority** is a main component of a PKI. It is a trusted third party responsible for issuing digital certificates and managing them throughout their lifetime.
Certificate authorities (CA) are the digital world's equivalent of passport offices. They issue digital certificates and validate the holder's identity and authority. CA embed an individual's or an organization's public key along with other identifying information into each digital certificate and then cryptographically "sign" it as a tamper-proof seal, verifying the integrity of the data within it and validating its use.

### 10.3.3 Certificate Exchange and authentication

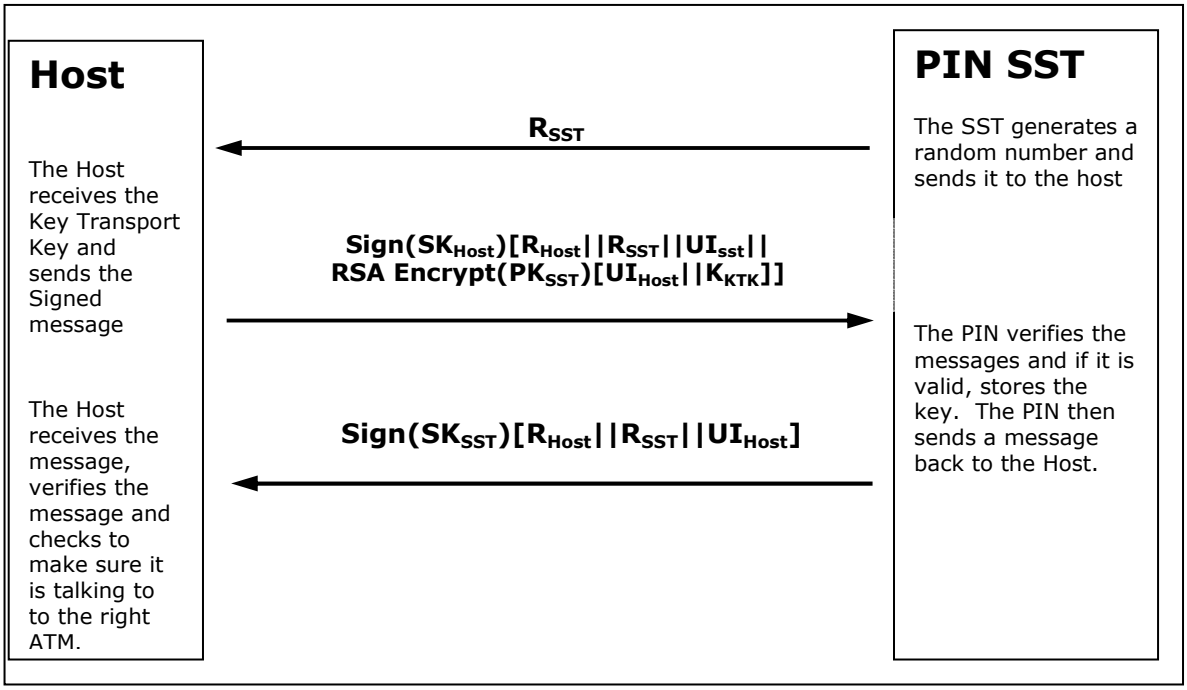
At this step the host and the SST exchange their certificates. The certificate contains the Public key of the sender.




| | |
|---|---|
| **$Cert_{HOST}$** | Host certificate provided by a Certification Authority |
| **$TP_{SST}$** | SHA-1 Thumb print value returned by the *importCertificate* command. |
| **$Cert_{SSt}$** | SST certificate provided by a Certification Authority |

Once the exchange of certificates is done the remote key loading can start.

### 10.3.4 Key Exchange – Host & SST PIN




| | |
|---|---|
| $R_{SST}$ | Random number generated by the SST encryptor |
| $SK_{Host}$ | Secret key of the host |
| $R_{Host}$ | Random number generated by the Host |
| $UI_{sst}$ | SST Unique identifier |
| $PK_{SST}$ | SST private key |
| $UI_{HOST}$ | HostUnique identifier |
| $K_{KTK}$ | Keys Transport Key |

Step 1

**The SST generates a random number and sends it to the host:** The random number is unique for each key exchange process.

Step 2

**The host constructs a key block data and sends it to the SST.**

1) The host has obtained a Key Transport Key and wants to transfer it to the encryptor.
2) The host constructs a key block containing an identifier of the host, $UI_{HOST}$, the key, $K_{KTK}$, and enciphers the block, using the SST public key.
   RSA Encryption $(PK_{SST})[UI_{HOST}] \| K_{KTK}]$

3) The host generates random data and builds the outer message containing the random number of the host, $R_{HOST}$, the random number of the SST, $R_{SST}$, and the SST unique Identifier, $UI_{SST}$,

4) The host signs the whole block, containing sub steps 2 and 3 results, using its private key and sends the message to the SST.

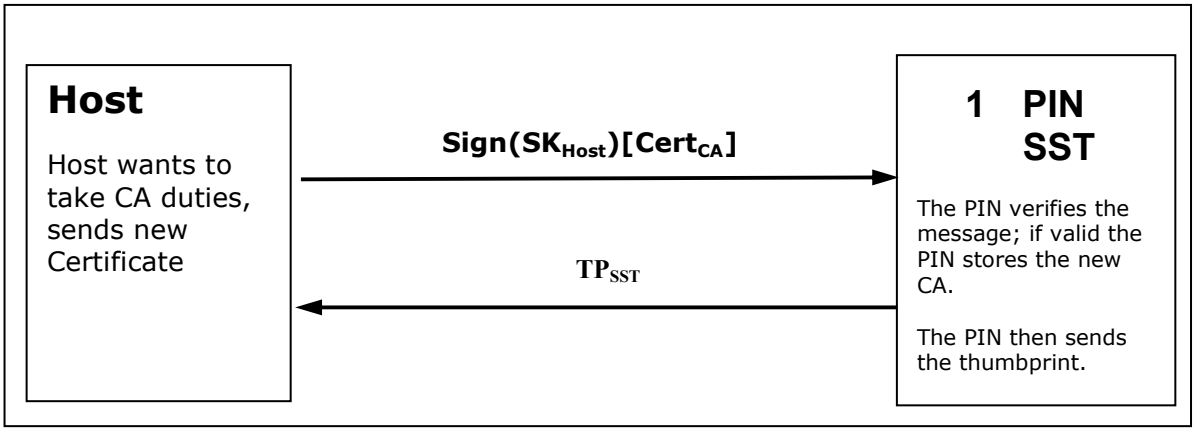
Step 3

**The SST validates the key. The SST constructs a message that contains the host random number**, $R_{HOST}$, the SST random number, $R_{SST}$, and the host identifier, $UI_{HOST}$ , signed by the private signature of the SST and sends the message to the host.

### 10.3.5 Replace certificate

The replacement of a CA certificate requires that there is already a CA certificate loaded. The implementation of this command is optional.

This is done by entity that would like to take over the job of being the CA. The new CA requests a Certificate from the previous Certificate Authority. The host must over-sign the message to take over the role of the CA to ensure that the HOST accepts the new Certificate Authority.




Step 1

**The HOST sends the message to the SST.**

Step 2

**The SST uses the host public key to verify the host signature.** The SST uses the previous CA public key to verify the signature on the new Certificate sent by the host. If valid, the SST stores the new CA certificate and uses the new CA public key, as it's new CA verification key.

Step 3

The SST sends to the host the thumb print value returned by the *replaceCertificate* command

#### Primary and Secondary certificates

Primary and Secondary Certificates for both the public verification key and public encipherment key are pre-loaded into the encryptor. Primary Certificates will be used until told otherwise by the host via the ***loadCertificate*** or ***replaceCertificate*** commands. The reason why the host would want to change states is because the HOST thinks that the Primary Certificates have been compromised.

After the host tells the encryptor to shift to the secondary certificate state, only Secondary Certificates can be used. The encryptor will no longer be able to go back to the Primary State and any attempts from the host to get or load a Primary Certificate will return an error.

# 11 Appendix D: Usage of Verification Codes

## 11.1 IJxfsCrypto.importKey( ) and IJxfsCrypto.importEMVRSAPublicKey( )

- Both *IJxfsCrypto.importKey( )* and *IJxfsCrypto.importEMVRSAPublicKey( )* return a *JxfsPINKeyVerificationData* object.

**JxfsOperationCompleteEvent.data:-**

| JxfsPINKeyVerificationData |
|---|
| keyVerCode : byte[ ] |
| JxfsPINKeyVerificationData(keyVerCode : byte[ ])<br>getKeyVerCode() : byte[ ] |

**keyVerCode Description:-**
Key verification code data that can be used for verification of the loaded key.
For the *importEMVRSAPublicKey* , if applied , it contains the expiry date of the certificate in the following format YYYY-MM
Null if this function is not supported by the device.

## 11.2 IJxfsCrypto.importRSAPublicKey( )

- *IJxfsCrypto.importRSAPublicKey( )* returns a *JxfsPINRSAKeyVerificationData* object.

**JxfsOperationCompleteEvent.data:-**

| JxfsPINRSAKeyVerificationData |
|---|
| hashAlgorithm : JxfsPINRSAHashAlgorithm<br>hashData : byte[ ] |
| JxfsPINRSAKeyVerificationData(hashAlgorithm : JxfsPINRSAHashAlgorithm, hashData : byte[ ])<br>getHashAlgorithm() : JxfsPINRSAHashAlgorithm<br>getHashData() : byte[ ] |

**hashData Description:-**
Contains the Hash data value computed when verifying and importing the key.

## 11.3 IJxfsCrypto.importRSADESEncipheredPublicKey( )

- *IJxfsCrypto.importRSADESEncipheredPublicKey( )* returns a *JxfsPINRSADESKeyVerificationData* object.

**JxfsOperationCompleteEvent.data:-**

| JxfsPINRSADESkeyVerificationData |
|---|
| keyLength : JxfsPINRSADESLength<br>checkMode : JxfsPINRSADESCheckMode<br>checkValue : byte[ ] |
| JxfsPINRSADESkeyVerificationData(keyLength : JxfsPINRSADESLength, checkMode : JxfsPINRSADESCheckMode, checkValue : byte[ ])<br>getKeyLength() : JxfsPINRSADESLength<br>getCheckMode() : JxfsPINRSADESCheckMode<br>getCheckValue() : byte[ ] |

**checkValue Description:-**
Specifies the verification data that can be used for verification of the loaded key.

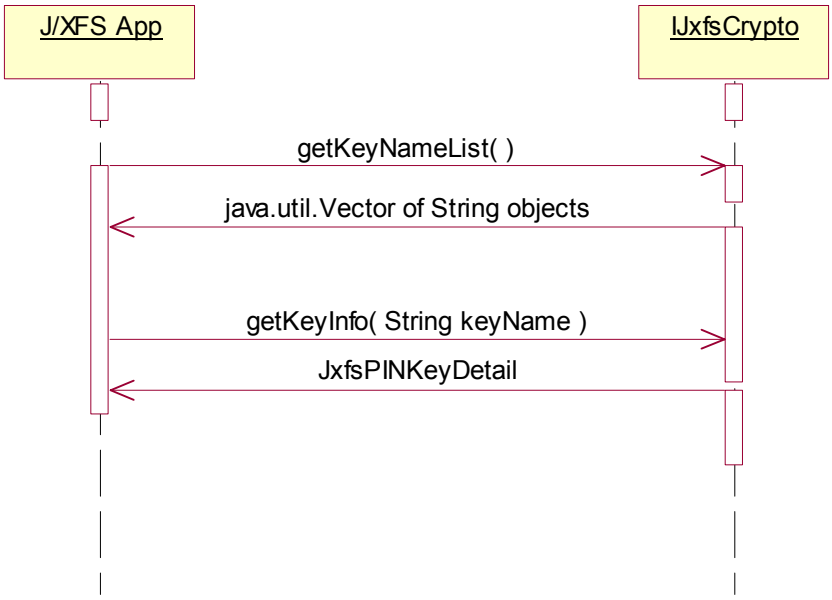
All import key methods (i.e. *IJxfsCrypto. importKey( )*, *importEMVRSAPublicKey( )*, *importRSAPublicKey( )* and *importRSADESEncipheredPublicKey( )* ) generate the JXFS_S_PIN_KEY status event:-

**Status Event**

If the completion of this operation results in an updated key in device's table key, then the J/XFS PIN Keypad device control will fire a JxfsStatusEvent to all registered listeners:

| **Field** | **Value** |
|---|---|
| status | JXFS_S_PIN_KEY<br>A new key has been loaded/imported into the device's key table. |
| details | A *JxfsPINKeyDetail* object containing information about the added key. |

## 11.4 Get Key Information Sequence

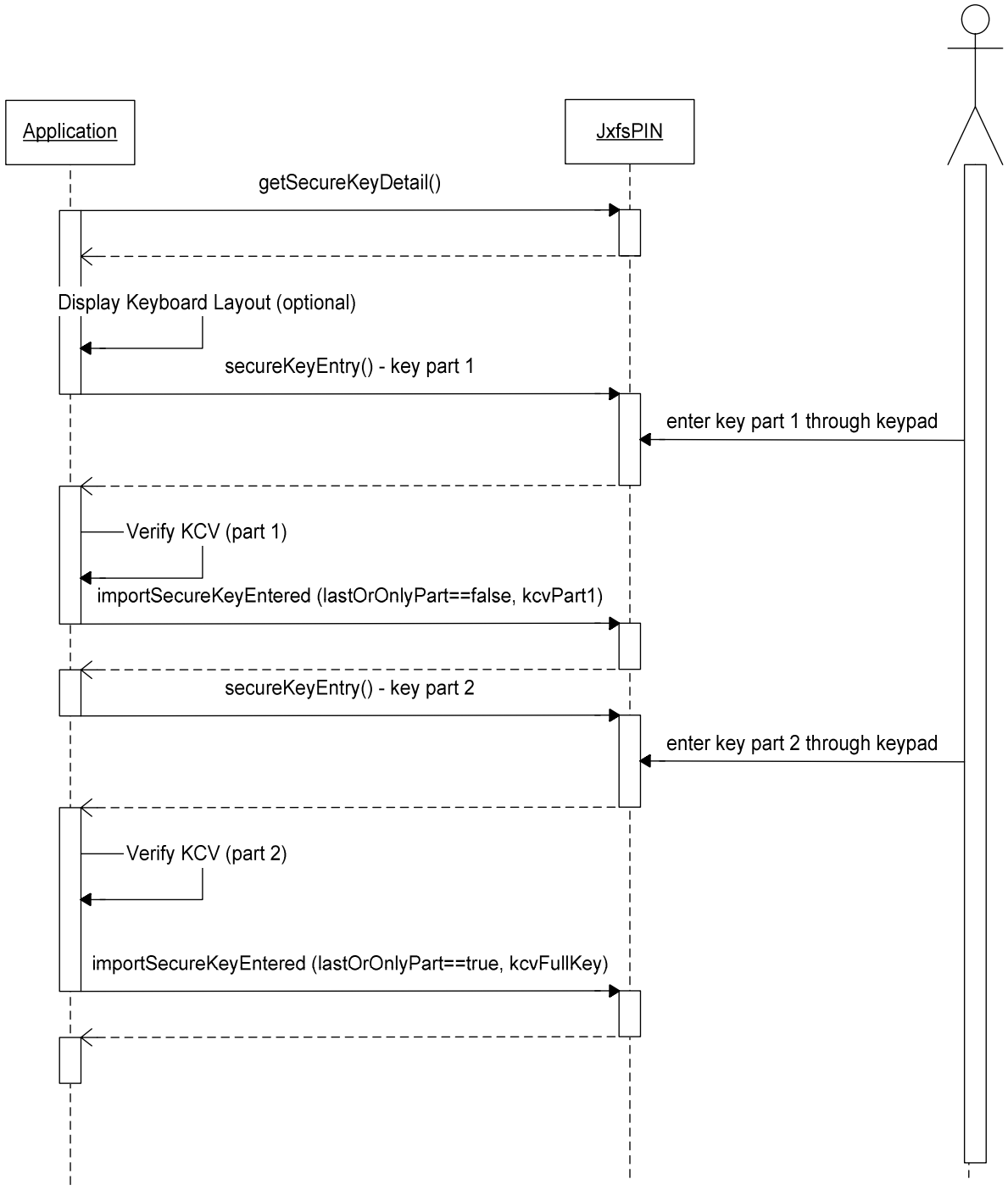



***JxfsPINKeyDetail getKeyInfo (java.lang.String keyName) throws JxfsException;***

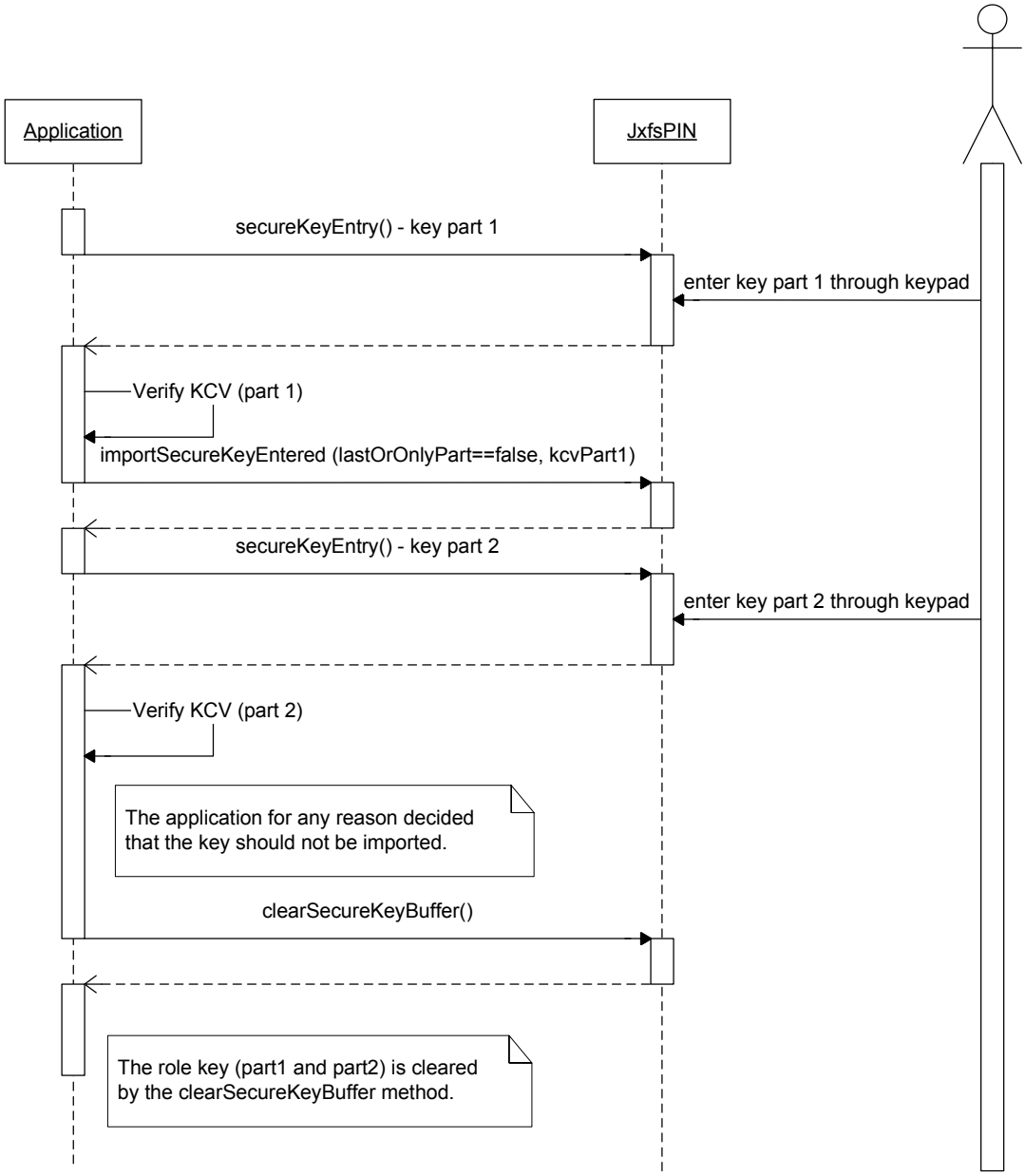
# 12 Appendix E: Secure Key Entry Handling

## 12.1 Sequence Diagrams

Typical scenario of non-encrypted key entering through keypad. The *secureKeyEntry* method is used. In this example, the key is entered by the operator in 2 steps (parts).



Typical scenario of non-encrypted key entering through keypad with the action aborted by the application through *clearSecureKeyBuffer* method.

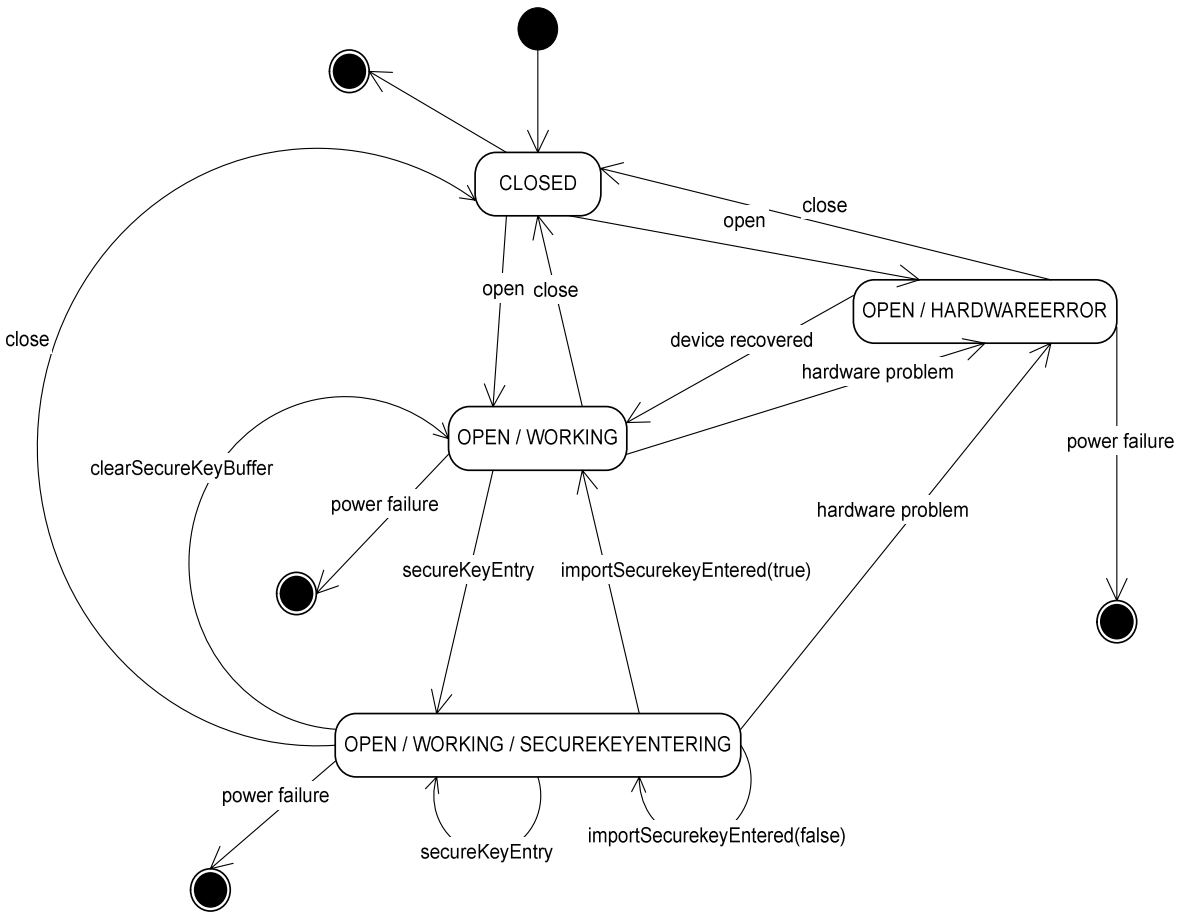


## 12.2 Secure Key Entry State Diagram

Statechart diagram that represents the status transition regarding the secure key enter process.
To enter a key in the secure key entry mode the application calls *secureKeyEntry* method. In the moment that the operation complete event of this method arrives with the code JXFS_RC_SUCCESSFUL the device service will be in *secure key entry* state. In this state the only methods that can be performed are *close, getStatus, getCapabilities, secureKeyEntry, importSecureKeyEntered* and *clearSecureKeyBuffer*. If another method was performed it will return JXFS_E_INVALID_SEQUENCE as the error code of the operation complete event.
Once the device is in *secure enter key* state it will leave this state in the following cases:

1. The secure key enter process has finished successfully after the application has called *importSecureKeyEntered* method with the parameter *lastOrOnlyPart* set to *true*.This is the **only** case where a key is correctly imported by the secure key enter process.
2. The secure key enter process was aborted by the application through *clearSecureKeyBuffer* method. Any key part that was previously imported by the *importSecureKeyEntered* method (*lastOrOnlyPart* set to *false)* is cleared from the device buffer.
3. The device was closed by either the Device Control that initiated the secure key entry or from the last open Device Control.; No key is imported. It has the same functionality of calling *clearSecureKeyBuffer*. For robustness the device service must provide this functionality, but is highly recommended that the applications that want to stop the secure key enter process call the *clearSecureKeyBuffer* method before closing.
4. A hardware error occurred. No key or key part is imported. The device goes to hardware error state.
5. Power failure. No key is imported.

Any time the device leaves *secure key entering* state, but when the key is successfully imported (case 1), the key is **NOT** imported and any key part that has been partially imported is cleared from the device buffer (cases 2,3,4 and 5). If the key was not successfully imported the process from importing it again must start from scratch.


CLOSED
open
close
OPEN / HARDWAREERROR
open
close
close
device recovered
hardware problem
OPEN / WORKING
power failure
clearSecureKeyBuffer
power failure
hardware problem
secureKeyEntry
importSecurekeyEntered(true)
OPEN / WORKING / SECUREKEYENTERING
power failure
secureKeyEntry
importSecurekeyEntered(false)

## 12.3 Keyboard Layout

The following sections describe what is returned by the *JxfsPINSecureKeyDetail* object to describe the physical keyboard layout. These descriptions are purely examples to help understand the usage of the parameters they do not indicate a specific layout per Key Entry Mode. In the following section all references to the properties of the *JxfsPINSecureKeyDetail* object. When k*eyEntryMode* represents a regular shaped pin pad (*JxfsKeyEntryModeEnum*.regUnique or *JxfsKeyEntryModeEnum*.regShift) then *hexKeys* array must contain one entry for each physical key on the pinpad (i.e. the product of *rows* by *columns*). On a regular shaped pinpad the application can choose to ignore the position and size data and just use the *rows* and *columns* parameters to define the layout. However, a device service must return the position and size data for each key.
When *keyEntryMode* is *JxfsKeyEntryModeEnum*.regUnique then the values in the array report which physical keys are associated with the function keys 0-9, A-F and any other function keys that can be enabled as defined in the f*uncKeyDetail* property. Any positions on the pinpad that are not used must be defined as a JXFS_PIN_FK_UNUSED in the *fk* and *shiftFK* properties.

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | Clear (A) |
| 4 | 5 | 6 | Cancel (B) |
| 7 | 8 | 9 | Enter (C) |
| (D) | 0 | (E) | (F) |

In the above example, where all keys are the same size and the hex digits are located as shown the *hexKeys* will contain the entries in the array as defined in the following table.

| Index | xPos | yPos | xSize | ySize | fk | shiftFK |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 250 | 250 | FK_1 | FK_UNUSED |
| 1 | 250 | 0 | 250 | 250 | FK_2 | FK_UNUSED |
| 2 | 500 | 0 | 250 | 250 | FK_3 | FK_UNUSED |
| 3 | 750 | 0 | 250 | 250 | FK_A | FK_UNUSED |
| 4 | 0 | 250 | 250 | 250 | FK_4 | FK_UNUSED |
| 5 | 250 | 250 | 250 | 250 | FK_5 | FK_UNUSED |
| 6 | 500 | 250 | 250 | 250 | FK_6 | FK_UNUSED |
| 7 | 750 | 250 | 250 | 250 | FK_B | FK_UNUSED |
| 8 | 0 | 500 | 250 | 250 | FK_7 | FK_UNUSED |
| 9 | 250 | 500 | 250 | 250 | FK_8 | FK_UNUSED |
| 10 | 500 | 500 | 250 | 250 | FK_9 | FK_UNUSED |
| 11 | 750 | 500 | 250 | 250 | FK_C | FK_UNUSED |
| 12 | 0 | 750 | 250 | 250 | FK_D | FK_UNUSED |
| 13 | 250 | 750 | 250 | 250 | FK_0 | FK_UNUSED |
| 14 | 500 | 750 | 250 | 250 | FK_E | FK_UNUSED |
| 15 | 750 | 750 | 250 | 250 | FK_F | FK_UNUSED |

When *keyEntryMode* is *JxfsKeyEntryModeEnum*.regShift then the values in the array report which physical keys are associated with the function keys 0-9, A-F and the shift key as defined in the f*uncKeyDetail* property. Other function keys as defined by the f*uncKeyDetail* property that can be enabled must also be reported. Any positions on the pinpad that are not used must be defined as a JXFS_PIN_FK_UNUSED in the *fk* and *shiftFK* property. Digits 0 to 9 are accessed through the numeric keys as usual. Digits A - F are accessed by using the shift key in combination with another function key, e.g. shift-0(zero) is hex digit A.

| | | | |
|---|---|---|---|
| 1 (B) | 2 (C) | 3 (D) | Clear |
| 4 (E) | 5 (F) | 6 | Cancel |
| 7 | 8 | 9 | Enter |
| SHIFT | 0 (A) | | |

In the above example, where all keys are the same size and the hex digits 'A' to 'F' are accessed through shift '0' to '5', then the *hexKeys* will contain the entries in the array as defined in the following table.

| Index | xPos | yPos | xSize | ySize | fk | shiftFK |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 250 | 250 | FK_1 | FK_B |
| 1 | 250 | 0 | 250 | 250 | FK_2 | FK_C |
| 2 | 500 | 0 | 250 | 250 | FK_3 | FK_D |
| 3 | 750 | 0 | 250 | 250 | FK_CLEAR | FK_UNUSED |
| 4 | 0 | 250 | 250 | 250 | FK_4 | FK_E |
| 5 | 250 | 250 | 250 | 250 | FK_5 | FK_F |
| 6 | 500 | 250 | 250 | 250 | FK_6 | FK_UNUSED |
| 7 | 750 | 250 | 250 | 250 | FK_CANCEL | FK_UNUSED |
| 8 | 0 | 500 | 250 | 250 | FK_7 | FK_UNUSED |
| 9 | 250 | 500 | 250 | 250 | FK_8 | FK_UNUSED |
| 10 | 500 | 500 | 250 | 250 | FK_9 | FK_UNUSED |
| 11 | 750 | 500 | 250 | 250 | FK_ENTER | FK_UNUSED |
| 12 | 0 | 750 | 250 | 250 | FK_SHIFT | FK_UNUSED |
| 13 | 250 | 750 | 250 | 250 | FK_0 | FK_A |
| 14 | 500 | 750 | 250 | 250 | FK_UNUSED | FK_UNUSED |
| 15 | 750 | 750 | 250 | 250 | FK_UNUSED | FK_UNUSED |

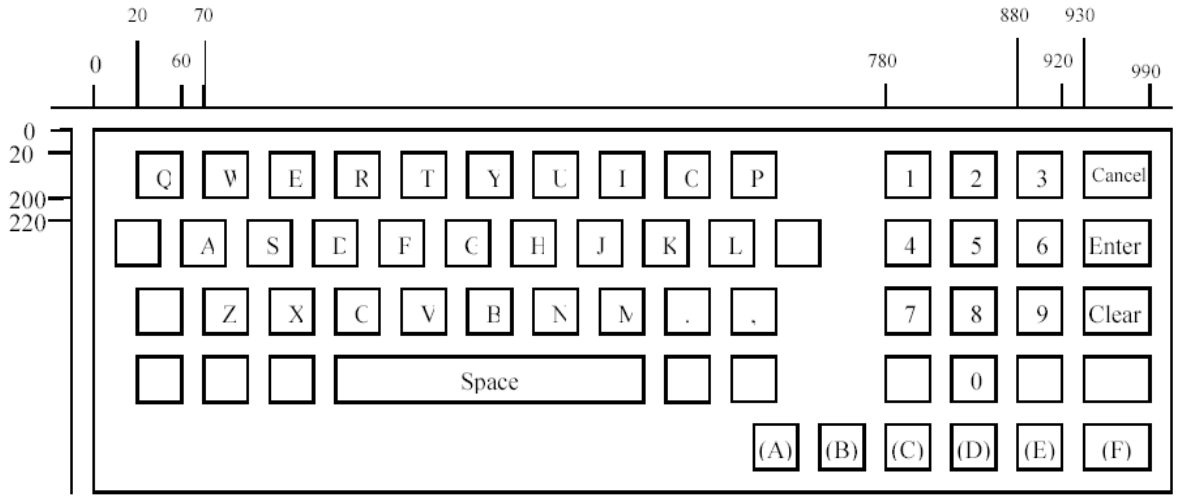
When *keyEntryMode* represents an irregular shaped pin pad the *rows* and *columns* parameters define the ratio of the width to height, i.e. square if the parameters are the same or rectangular if *columns* is larger than *rows*, etc. A service provider must return the position and size data for each key reported.
When *keyEntryMode* is *JxfsKeyEntryModeEnum*.irregShift then the values in the array must be the function keys codes for 0-9 and the shift key as defined in the f*uncKeyDetail* property. Other function keys as defined by the f*uncKeyDetail* property that can be enabled must also be reported. Any positions on the pinpad that are not used must be defined as a JXFS_PIN_FK_UNUSED in the *fk* and *shiftFK* property. Digits 0 to 9 are accessed through the numeric keys as usual. Digits A - F are accessed by using the shift key in combination with another function key,e.g. shift-0(zero) is hex digit A.

| | | | |
|---|---|---|---|
| 1 (B) | 2 (C) | 3 (D) | Clear |
| 4 (E) | 5 (F) | 6 | Cancel |
| 7 | 8 | 9 | Enter |
| | 0 (A) | | |
| SHIFT | | | |

In the above example, where the hex digits 'A' to 'F' are accessed through shift '0' to '5', *columns* will be 4, *rows* will be 5 and the *hexKeys* will contain the entries in the array as defined in the following table.

| Index | xPos | yPos | xSize | ySize | fk | shiftFK |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 250 | 200 | FK_1 | FK_B |
| 1 | 250 | 0 | 250 | 200 | FK_2 | FK_C |
| 2 | 500 | 0 | 250 | 200 | FK_3 | FK_D |
| 3 | 750 | 0 | 250 | 200 | FK_CLEAR | FK_UNUSED |
| 4 | 0 | 200 | 250 | 200 | FK_4 | FK_E |
| 5 | 250 | 200 | 250 | 200 | FK_5 | FK_F |
| 6 | 500 | 200 | 250 | 200 | FK_6 | FK_UNUSED |
| 7 | 750 | 200 | 250 | 200 | FK_CANCEL | FK_UNUSED |
| 8 | 0 | 400 | 250 | 200 | FK_7 | FK_UNUSED |
| 9 | 250 | 400 | 250 | 200 | FK_8 | FK_UNUSED |
| 10 | 500 | 400 | 250 | 200 | FK_9 | FK_UNUSED |
| 11 | 750 | 400 | 250 | 200 | FK_ENTER | FK_UNUSED |
| 12 | 0 | 600 | 250 | 200 | FK_UNUSED | FK_UNUSED |
| 13 | 250 | 600 | 250 | 200 | FK_0 | FK_A |
| 14 | 500 | 600 | 250 | 200 | FK_UNUSED | FK_UNUSED |
| 15 | 750 | 600 | 250 | 200 | FK_UNUSED | FK_UNUSED |
| 16 | 0 | 800 | 1000 | 200 | FK_SHIFT | FK_UNUSED |

When *keyEntryMode* is *JxfsKeyEntryModeEnum*.regUnique then the values in the array report which physical keys are associated with the function keys 0-9, A-F and any other function keys that can be enabled as defined in the f*uncKeyDetail* property. The *rows* and *columns* parameters define the ratio of the width to height, ie square if the parameters are the same or rectangular if c*olumns* is larger than *rows*, etc. A device service must return the position and size data for each key.




In the above example, where an alphanumeric keyboard supports secure key entry and the hex digits are located as shown, the *hexKeys* will contain the entries in the array as defined in the following table. All the hex digits and function keys that can be enabled must be included in the array; in addition any keys that would help an application display an image of the keyboard can be included. In this example only the pinpad digits (the keys on the right) and the unique hex digits are reported. Note that the position data in this example may not be 100% accurate as the diagram is not to scale.

| Index | xPos | yPos | xSize | ySize | Fk | ulShiftFK |
|---|---|---|---|---|---|---|
| 0 | 780 | 18 | 40 | 180 | FK_1 | FK_UNUSED |
| 1 | 830 | 18 | 40 | 180 | FK_2 | FK_UNUSED |
| 2 | 880 | 18 | 40 | 180 | FK_3 | FK_UNUSED |
| 3 | 930 | 18 | 60 | 180 | FK_CANCEL | FK_UNUSED |
| 4 | 780 | 216 | 40 | 180 | FK_4 | FK_UNUSED |
| 5 | 830 | 216 | 40 | 180 | FK_5 | FK_UNUSED |
| 6 | 880 | 216 | 40 | 180 | FK_6 | FK_UNUSED |
| 7 | 930 | 216 | 60 | 180 | FK_ENTER | FK_UNUSED |
| 8 | 780 | 414 | 40 | 180 | FK_7 | FK_UNUSED |
| 9 | 830 | 414 | 40 | 180 | FK_8 | FK_UNUSED |
| 10 | 880 | 414 | 40 | 180 | FK_9 | FK_UNUSED |
| 11 | 930 | 414 | 60 | 180 | FK_CLEAR | FK_UNUSED |
| 12 | 780 | 612 | 40 | 180 | FK_UNUSED | FK_UNUSED |
| 13 | 830 | 612 | 40 | 180 | FK_0 | FK_UNUSED |
| 14 | 880 | 612 | 40 | 180 | FK_UNUSED | FK_UNUSED |
| 15 | 930 | 612 | 60 | 180 | FK_UNUSED | FK_UNUSED |
| 16 | 680 | 810 | 40 | 180 | FK_A | FK_UNUSED |
| 17 | 730 | 810 | 40 | 180 | FK_B | FK_UNUSED |
| 18 | 780 | 810 | 40 | 180 | FK_C | FK_UNUSED |
| 19 | 830 | 810 | 40 | 180 | FK_D _ | FK_UNUSED |
| 20 | 880 | 810 | 40 | 180 | FK_E _ | FK_UNUSED |
| 21 | 930 | 810 | 60 | 180 | FK_F FK | FK_UNUSED |