CEN

WORKSHOP

AGREEMENT

CWA 16008-3

August 2009

ICS 35.240.40

English version

J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Part 3: Magnetic Stripe & Chip Card 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 16008-3:2009 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 - ensors 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 - Release 2009 - 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)

Note: Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. The Java Trademark Guidelines are currently available on the web at http://www.sun.com All other trademarks are trademarks of their respective owners.

This CEN Workshop Agreement is publicly available as a reference document from the National Members of CEN : AENOR, AFNOR, ASRO, BDS, BSI, CSNI, CYS, DIN, DS, ELOT, EVS, IBN, IPQ, IST, LVS, LST, MSA, MSZT, NEN, NSAI, ON, PKN, SEE, SIS, SIST, SFS, SN, SNV, SUTN and UNI.

Comments or suggestions from the users of the CEN Workshop Agreement are welcome and should be addressed to the CEN Management Centre.

History

The main differences to the previous CWA 14923-3:2004 are:

- New reader types in the deviceType property of chip card
- Added the cardStatus property and the JxfsCCDCardStatus class
- Added new methods for activation / deactivation / warm reset
- Updated high coercitivity support
- Added permanent chip card
- Added EMV clarification paragraphs
- Clarifications/Amendments about the behaviour of the chipIO method in case of errors added.
- testResult and cim86info modified to clarify results of a security check with MM modules.
- Added Appendix with Card Reader Fraud Behaviour information
- added new manipulationStatus properties and associated resources to provide Hardware Manipulation information.
- New JxfsMSDStatusSelectorEnum enumeration introduced to allow use of new getStatus method defined in base architecture documentation.
- open job handling clarified at base architecture level so specific chapter in this document is removed.
- specific declaration of result codes used by each job has been removed, and now result refers to common section at the end of the document.

The main differences to the previous CWA 13937-3:2000 are:

- Modified readata method description
- Modified ejectCard method, status event added
- Modified retainCard method, status event added
- Corrected some typing errors
- Added missing clarification on the writeData method
- Removed the JXFS_E_CLAIMED exception
- Removed “media taken” as a code for an intermediate event, at section 6.3
- Added JXFS_S_MEDIA_STATUS events at the ejectCard and reatinCard methods of the motorized card
- interface.
- Added class hierarchy diagram
- Modified the Description of the readData method of the IJxfsMagStripeControl interface, relating to the
- magnetic pre-head detection.
- Added paragraph describing handling of null parameters
- Changed from lowercase “j” to uppercase “J” in all interface names starting with “IJxfs…”

1 Scope

This document describes the Magnetic Stripe Device (MSD) as well as Chip Card Device (CCD) 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 Magnetic Stripe devices and Chip Card 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 document describes the J/XFS support classes for both Magnetic Stripe devices (MSD) as well as Chip Card devices (CCD).

As well as the rest of J/XFS device controls, J/XFS Magnetic Stripe and J/XFS Chip Card devices use the event driven model and the same behavioral model. Therefore, in the case of a Magnetic Stripe device, the application will instantiate a J/XFS Magnetic Stripe Device Control Object and then use the available methods to do I/O. When an I/O method is called, the J/XFS Magnetic Stripe 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 Magnetic Stripe Device Control Object for the various types of events it wishes to handle.

The same model applies to all J/XFS device controls and, in particular, to the Chip Card Device control.

2.1.1 Magnetic Stripe Device

The J/XFS Magnetic Stripe Reader/Encoder Device Support allows for the operation of devices with magnetric stripe read/write capabilities. Following are typical devices with such a capability:

- motor driven card reader/writer
- pull through card reader/writer
- dip card reader/writer

The following tracks and the corresponding international standards are taken into account in this document:

Track 1	ISO 7811
Track 2	ISO 7811
Track 3	ISO 7811 / ISO 4909

In addition to the pure reading of the tracks mentioned above, security boxes can be used via this service to check the data of writable tracks for manipulation. These boxes (such as CIM or MM) are sensor-equipped devices that are able to check some other information on the card and compare it with the track data.

Leds handling will be defined based on initialization configuration so no reference to them is made in this document.

Handling of *watermark* is also considered.

2.1.2 Chip Card Device

The J/XFS Chip Card Device Support allows for the operation of devices with chip access capabilities. Following are typical devices with such a capability:

- Motor driven chip card devices.
- Dip chip card devices.
- Permanent chip card devices.

The following chips and the corresponding international standards are taken into account in this document:

- Chip (contacted) ISO 7816

2.1.3 EMV Level 1 & Level 2

EMVCo has defined a set of specifications that terminals have to implement to in order to support EMV processing. These specifications are public and available on the EMVCo website www.emvco.com. Any update is communicated on the website, either through new releases or bulletins.

The specifications contain requirements for hardware -card reader/IFM (Interface Module)- and software (EMV application Kernel). The specifications apply to different terminals and solutions.

The definition of:

"Interface module (IFM): a virtual or abstract device that contains the necessary hardware and software to power the ICC and to support communication between the terminal and the ICC up to the transport layer. The three main functional components are the mechanical, electrical and logical ICC interfaces."
From **EMVCo Type Approval Terminal Level 1 Administrative Process Version 4.0 February 26th, 2003**

"EMV application kernel: a software module, core, or library, forming part of an overall terminal application architecture, developed for exclusive support of the EMV debit/credit functions and application requirements."
From: **EMVCo Type Approval Terminal Level 2 Administrative Process Version 1.0 April 24, 2001**

EMVCo has also described the terminal architecture that isolates on one side the IFM and necessary software/platform/firmware and on the other side the software application which runs on a terminal.

This distinction is made calling the first Level 1 and the latter Level 2. To prove this compliancy a vendor must submit its solution for type approval.

The Level 1 type approval process tests compliance with electromechanical characteristics, logical interface, and transmission protocol requirements defined in part I of the EMV Specifications.

Level 2 type approval tests compliance with debit/credit application requirements defined in the remainder of the EMV Specifications.

More in detail the levels and the corresponding sections in the different EMV Specifications.

For EMV '96:

- Level 1 is based on part I of the EMV '96 Integrated Circuit Card Specification for Payment Systems.
- Level 2 is based on parts II, III, and IV, of the EMV '96 Integrated Circuit Card Specification for Payment Systems as well as the Application and the Terminal documentation.

For EMV 2000:

- Level 1 is based on EMV 2000 Integrated Circuit Card Specification for Payment Systems (book I-part I). Level 2 is based on of the EMV 2000 Integrated Circuit Card Specification for Payment Systems (book I-part II and books II, II and IV) "

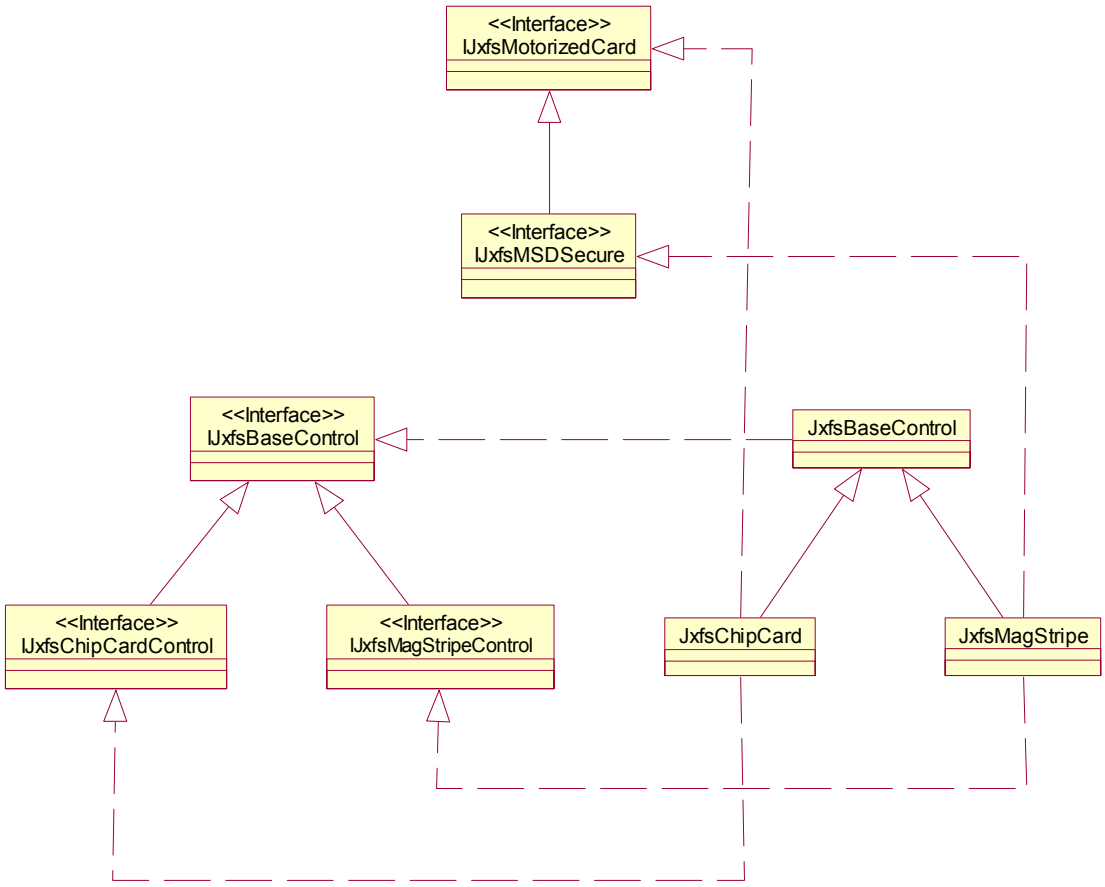
2.1.4 EMV Responsibilities within the MSD Device Service

For a better understanding, it is important to know:

- EMV Level 2 interaction is handled above the J/XFS API
- EMV Level 1 interaction is handled below the J/XFS API.

Please see above the section “EMV Level 1 & Level 2” for a description of these terms.

2.2 Class Hierarchy



2.3 Classes and Interfaces

The following classes and interfaces are used by the J/XFS MSD and CCD Device Controls. In order to support the definition of the different properties of the different devices (see Introduction), the Device Controls are defined in a class hierarchy.

Class or Interface	Name	Description	Extends or Implements
Interface	**IJxfsBaseControl**	Base interface for all the device controls. Contains methods common to all the device controls.	
Interface	**IJxfsMagStripeControl**	Base interface for MSD controls. Contains method declarations specific to MSD controls.	Extends: **IJxfsBaseControl**
Interface	**IJxfsMagStripeService**	Base interface for MSD services. Contains the methods specific to the device services for the MSD device category.	Extends: **IJxfsBaseService**
Interface	**IJxfsChipCardControl**	Base interface for CCD controls. Contains method declarations specific to CCD controls.	Extends: **IJxfsBaseControl**
Interface	**IJxfsChipCardService**	Base interface for CCD services. Contains the methods specific to the device services for the CCD device category.	Extends: **IJxfsBaseService**
Interface	**IJxfsMotorizedCard**	Interface for motorized card devices. Contains method declarations specific to motorized card devices.	
Interface	**IJxfsMotorizedCardService**	This interface should be implemented by MSD or CCD device services that provide access to a motorized device.	
Interface	**IJxfsMSDSecure**	Interface for motorized card devices with secure module. Contains method declarations specific to card devices with secure module.	Extends: **IJxfsMotorizedCard**
Interface	**IJxfsMSDSecureService**	This interface should be implemented by device services that provide access to devices with a secure module.	
Class	**JxfsBaseControl**	Base class for all the device controls. Contains properties common to all the deviceb controls.	
Class	**JxfsMagStripe**	Base class for MSD controls. Contains properties specific to MSD device controls.	Implements: **IJxfsMagStripeControl IJxfsMSDSecure**
Class	**JxfsChipCard**	Base class for CCD controls. Contains properties specific to CCD device controls.	Implements: **IJxfsChipCardControl IJxfsMotorizedCard**

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	**JxfsMSDConst**	Interface containing the Jxfs constants that are common to all the MSD device controls.	--
Interface	**JxfsCCDConst**	Interface containing the Jxfs constants that are common to all the CCD device controls.	--
Interface	**JxfsMotorizedCardConst**	Interface containing the Jxfs constants for motorized card devices.	--
Class	**JxfsMSDTracks**	MSD Track selector class. Indicates for each track if it's selected or not. Properties are read only.	Extends: **JxfsType**
Class	**JxfsMSDTrackSelection**	Subclass of MSD Track selector class. It contains the same properties but they can be set by applications.	Extends: **JxfsMSDTracks**
Class	**JxfsMSDReadData**	Data class that contains data returned in Operation Complete events for MSD *readData()* operation.	Extends: **JxfsType**
Class	**JxfsCCDData**	Data class that contains data returned in Operation Complete events for CCD input/output operations.	Extends: **JxfsType**
Class	**JxfsCCDCardStatus**	Data class that contains information on the state of a present chip card.	Extends: **JxfsType**
Class	**JxfsMSDWmData**	Data class that contains data returned in Operation Complete events for MSD *readWMtrack()* operation.	Extends: **JxfsType**
Class	**JxfsMSDSecureMode**	Data class that provides required properties for *readData()* operation in secure mode.	Extends: **JxfsType**
Class	**JxfsMSDReadDataSecure**	Data class that contains data returned in Operation Complete events for MSD *readData()* in secure mode.	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 instances of this classes and delivers them through the J/XFS MSD Device Control event callbacks to the application	Extend: **JxfsEvent**
Class	**JxfsException**	Exception class. The J/XFS MSD 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 errorCode 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.

get*Property*

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*: JXFS_E_CLOSED JXFS_E_UNREGISTERED 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*: JXFS_E_CLOSED JXFS_E_UNREGISTERED JXFS_E_REMOTE JXFS_E_PARAMETER_INVALID

4.2 Exceptions

All the methods described for the specified interfaces can throw at least some of the following exceptions:

Value	**Meaning**
JXFS_E_CLOSED	The Device Control has not been opened.
JXFS_E_UNREGISTERED	The device is not registered at the JxfsDeviceManager.
JXFS_E_REMOTE	A network error ocurred.
JXFS_E_PARAMETER_INVALID	A parameter is invalid.
JXFS_E_NOT_SUPPORTED	The function is not supported.

Only if a method can throw additional exceptions this is explicitly mentioned.

4.3 IJxfsMagStripeControl

4.3.1 Introduction

The J/XFS MSD Device Control Subclass is defined in JxfsMagStripe and is a subclass of JxfsBaseControl. Its interface is defined in IJxfsMagStripeControl interface which is a subclass of IJxfsBaseControl interface. The purpose of the J/XFS MSD 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.

Summary

Although IJxfsMagStripeControl 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 IJxfsMagStripeControl consists on the following methods:

- Getters of listed properties.
- Methods listed.

Property	Type	Access	Initialized after
deviceType	**int**	R	After service instantiation
mediaStatus	**JxfsMediaStatus**	R	After successful open
supportedReadTracks	**JxfsMSDTracks**	R	After successful open
supportedWriteTracks	**JxfsMSDTracks**	R	After successful open
supportedWriteHiCoTracks	**JxfsMSDTracks**	R	After successful open
writeMode	**int**	R	After successful open
manipulationStatus	**JxfsManipulationStatusEnum**	R	After successful open

Method	Return	May be used after
get*Property*	*Property*	After successful open
readData	identificationID	After successful open
writeData	identificationID	After successful open
writeData	identificationID	After successful open

4.3.2 Properties

deviceType Property (R)

Type ***int***
Initial Value Depends on device type.
Description Identifies a type of MSD device. Depending on the device type it will be a combination of the following flags:

Value	Meaning
JXFS_MSD_TYPE_SWIPE	Swipe/pull through magnetic stripe reader/encoder.
JXFS_MSD_TYPE_DIP	Dip magnetic card reader/encoder.
JXFS_MSD_TYPE_MOTOR	Motorized card reader.

mediaStatus Property (R)

Type ***JxfsMediaStatus***
Initial Value A JxfsMediaStatus (see related section in Base Architecture document).
Description Specifies the state of the media.
Event If the value of this property changes, the Device Service will send all registered status listeners a *JxfsStatusEvent* with the following values:

Field	Value
status	JXFS_S_MSD_MEDIA_STATUS *mediaStatus* has changed.
details	A ***JxfsMediaStatus*** object.

supportedReadTracks Property (R)

Type	***JxfsMSDTracks***
Initial Value	null until open.
Description	Indicates which tracks can be physically read by the device.

supportedWriteTracks Property (R)

Type	***JxfsMSDTracks***
Initial Value	null until open.
Description	Indicates which tracks can be physically written by the device.

supportedWriteHiCoTracks Property (R)

Type	***JxfsMSDTracks***
Initial Value	null until open.
Description	Indicates which tracks can be physically written in HiCo (High Coercitive) mode.

writeMode Property (R)

Type	***int***
Initial Value	Depends on device service implementation.
Description	Indicates the write capabilities of the device. It can be a combination of the following flags:

Field	Value
JXFS_MSD_NOT_SUPPORTED	The device does not support writing to magnetic stripes.
JXFS_MSD_HICO	The device supports writing to high coercivity magnetic stripes.
JXFS_MSD_LOCO	The device supports writing to low coercivity magnetic stripes.
JXFS_MSD_AUTO	The device service is capable of determining whether it should write to high or low coercivity magnetic stripes.

manipulationStatus Property (R)

Type	***JxfsManipulationStatusEnum***
Initial Value	Default *JxfsManipulationStatusEnum* object.
Description	Specifies the state of any present anti fraud feature.
Event	If the value of this property changes, the Device Service will send all registered status listeners a JxfsStatusEvent with the following values:

Field	Value
status	JXFS_S_MSD_MANIPULATION_STATUS *manipulationStatus* has changed.
details	A *JxfsManipulationStatusEnum* object.

4.3.3 Methods

readData

Syntax

identificationID readData (JxfsMSDTrackSelection tracksToRead) throws JxfsException;

Description

This method launches a read operation to obtain the data contained in the tracks specified by the *tracksToRead* parameter.

If media is present, the read operation is performed immediately. Otherwise, the device waits until it is present or the operation is cancelled.
After a successful completion of this input operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the results.
Many motorized card readers on the market have an option called magnetic pre-head detection. If this option is active, then only cards with a magnetized stripe may enter the device, so in this case a card is never entered the wrong way. In the case that the device does not have this option or the option has to be deactivated because the device shall also accept smart cards without magnetic stripe, then current devices cannot distinguish between the cases of a card entered in the wrong way and a card with read errors on all stripes. Therefore in both cases JXFS_E_MSD_READFAILURE should be returned.

Parameter

Type	Name	Meaning
JxfsMSDTrackSelection	tracksToRead	Tracks to be read.

Event

JxfsOperationCompleteEvent
When a *readData()* operation is completed an *JxfsOperationCompleteEvent* event will be sent by MSD Device Control to all registered operation complete listeners. It will contain the data read.

Field	Value
operationID	JXFS_O_MSD_READDATA
identificationID	Identification ID of complete operation.
result	Common or device dependent error code. (See section on Error codes).
data	A **JxfsMSDReadData** object.

JxfsIntermediateEvent
JxfsIntermediateEvent can be sent by MSD Device Control to all registered intermediate listeners

Field	Value
operationID	JXFS_O_MSD_READDATA
identificationID	Identification ID of operation.
reason	JXFS_I_MSD_NO_MEDIA_PRESENT The read operation request cannot progress because there is no media inserted. JXFS_I_MSD_MEDIA_INSERTED The read operation request continues because a media has been inserted.
data	null

Exceptions

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

Value	Meaning
JXFS_E_MSD_NOTSUPPORTEDTRACK	At least one track specified in *tracksToRead* parameter is not supported by the device.

JXFS_E_MSD_NOTRACKS	No tracks specified in *tracksToRead* parameter.

writeData

Syntax

identificationID writeData (java.util.Vector wdata, boolean newCard) throws JxfsException;

Description

This method initiates a write operation of the data contained in w*data*.

If media is present, the write operation is performed immediately. Otherwise, the device waits until it is present or the operation is cancelled. If the parameter *newCard* contains *true*, the card must be inserted *after* the operation is started.

Each vector element of w*data* is a byte [] with the data to be written in each track. Vector element 0 contains data for track 1, vector element 1 contains data for track 2, and so on.

The track data should have no hardware control characters or BCC included (like SS, SE or BCC). The data for ISO track #1 (6 bits per character) is transformed in the range of 0x20 to 0x5F and the data for the ISO tracks #2 and #3 (4 bits per character) are transformed in the range from 0x30 to 0x3F.

If the card is removed from the device during the write operation, the JXFS_E_MSD_NOMEDIA error code should be returned.
The use of the newCard parameter is deprecated. It is recommended that it is not used, i.e., that *false* is specified.

If no data has to be written for a given track, the corresponding vector element has to contain null.

If the device supports automatic detection of high/low coercivity tracks, it will write using the correct mode. If the device does not support this capability, it will write using the low coercivity mode.

After a successful completion of this output operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the results.

Parameter

Type	Name	Meaning
java.util.Vector	wdata	Data to be written. Each vector element contains a byte [ ] of raw data per track. A null vector element is assumed no data to be written for its associated track.
boolean	newCard	If false, it specifies that the operation may proceed when a card is already present.

Event

JxfsOperationCompleteEvent
When a *writeData ()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by MSD Device Control to all registered operation complete listeners.

Field	Value
operationID	JXFS_O_MSD_WRITEDATA
identificationID	Identification Id of complete operation.
result	Common or device dependent error code. (See section on Error codes).
data	A **JxfsMSDTracks** object.

JxfsIntermediateEvent

JxfsIntermediateEvent can be sent by MSD Device Control to all registered intermediate listeners

Field	Value
operationID	JXFS_O_MSD_WRITEDATA
identificationID	Identification Id of operation.
reason	JXFS_I_MSD_NO_MEDIA_PRESENT The write operation request cannot progress because there is no media inserted. JXFS_I_MSD_MEDIA_INSERTED The write operation request continues because a media has been inserted.
data	null

Exceptions

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

Value	Meaning
JXFS_E_MSD_NOTSUPPORTEDTRACK	At least one of the specified tracks is not supported by the device.
JXFS_E_MSD_NOTRACKS	No track data has been specified.

writeData

Syntax

identificationID writeData (java.util.Vector wdata, int writeMode) throws JxfsException;

Description

This method initiates a write operation of the data contained in w*data*.

If media is present, the write operation is performed immediately. Otherwise, the device waits until it is present or the operation is cancelled.

Each vector element of w*data* is a byte [] with the data to be written in each track. Vector element 0 contains data for track 1, vector element 1 contains data for track 2, and so on.

The track data should have no hardware control characters or BCC included (like SS, SE or BCC). The data for ISO track #1 (6 bits per character) is transformed in the range of 0x20 to 0x5F and the data for the ISO tracks #2 and #3 (4 bits per character) are transformed in the range from 0x30 to 0x3F.

If the card is removed from the device during the write operation, the JXFS_E_MSD_NOMEDIA error code should be returned.

If no data has to be written for a given track, the corresponding vector element has to contain null.

The writeMode parameter defines how to write the track data. It defines whether the tracks are high or low coercivity tracks.

After a successful completion of this output operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the results.

Parameter	Type	Name	Meaning
	java.util.Vector	wdata	Data to be written. Each vector element contains a byte [ ] of raw data per track. A null vector element is assumed no data to be written for its associated track.
	int	writeMode	Defines how to write the track data. It may be one of the following: JXFS_MSD_LOCO for writing a track in low coercitivity (LoCo) mode. JXFS_MSD_HICO for writing a track in high coercitivity (HiCo) mode. JXFS_MSD_AUTO where the device service determines whether to write in high or low coercivity mode.

Event

JxfsOperationCompleteEvent

When a *writeData ()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by MSD Device Control to all registered operation complete listeners.

Field	Value
operationID	JXFS_O_MSD_WRITEDATA
identificationID	Identification Id of complete operation.
result	Common or device dependent error code. (See section on Error codes).
data	A **JxfsMSDTracks** object.

JxfsIntermediateEvent

JxfsIntermediateEvent can be sent by MSD Device Control to all registered intermediate listeners

Field	Value
operationID	JXFS_O_MSD_WRITEDATA
identificationID	Identification Id of operation.
reason	JXFS_I_MSD_NO_MEDIA_PRESENT The write operation request cannot progress because there is no media inserted. JXFS_I_MSD_MEDIA_INSERTED The write operation request continues because a media has been inserted.
data	null

Exceptions

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

Value	Meaning
JXFS_E_MSD_NOTSUPPORTEDTRACK	At least one of the specified tracks is not supported by the device.
JXFS_E_MSD_NOTRACKS	No track data has been specified.

4.4 IJxfsChipCardControl

4.4.1 Introduction

The J/XFS Chip Card Device Control Subclass is defined in JxfsChipCard and is a subclass of JxfsDeviceControl. Its interface is defined in IJxfsCCDControl interface which is a subclass of IJxfsBaseControl interface. The purpose of the J/XFS CCD 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.

This class represents a physical device (or part of it) that has chip card access capabilities (send/receive of commands and data).

Summary

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

Therefore, the IJxfsChipCardControl consists on the following methods:

- Getters of listed properties.
- Methods listed.

Property	Type	Access	Initialized after
deviceType	**int**	R	After service instantiation
mediaStatus	**JxfsMediaStatus**	R	After successful open
cardStatus	**JxfsCCDCardStatus**	R	After successful open
manipulationStatus	**JxfsManipulationStatusEnum**	R	After successful open

Method	Return	May be used after
get*Property*	*Property*	After successful open
isCcdT	*boolean*	After successful open
chipInit	*identificationID*	After successful open
chipIO	*identificationID*	After successful open
isActivateCardSupported	*boolean*	After successful open
isDeactivateCardSupported	*boolean*	After successful open
isWarmResetCardSupported	*boolean*	After successful open
activateCard	*identificationID*	After successful open
deactivateCard	*identificationID*	After successful open
warmResetCard	*identificationID*	After successful open

4.4.2 Properties

deviceType Property (R)

Type ***int***

Initial Value Depends on device type.

Description Identifies a type of Chip Card device. Depending on the device type it will be a combination of the following flags.
The device type specifies the way a user has interaction with the device and thus the necessary actions by an application controlling this user interaction.

Value	Meaning
JXFS_CCD_TYPE_SWIPE	Swipe/pull through chip card device.
JXFS_CCD_TYPE_DIP	Dip chip card device.
JXFS_CCD_TYPE_MOTOR	Motorized chip card device.

JXFS_CCD_TYPE_CONTACTLESS	Contactless chip card device.
JXFS_CCD_TYPE_DIP_LATCHED	Dip chip card device with a latch to fix the card.
JXFS_CCD_TYPE_PERMANENT	Permanent chip card.

Devices of type JXFS_CCD_TYPE_DIP_LATCHED are JXFS_CCD_TYPE_DIP readers with an additional lock to fix the card. In order to release the card (unlock the latch), the *ejectCard*() method of the IJxfsMotorizedCard interface must be called.

JXFS_CCD_TYPE_PERMANENT chip cards are continuously contacted (but not always activated) cards that cannot be moved by the user. These cards are often in the SIM form factor and can be found as additional slots on some modern CCD devices. For this device type all available methods of the IJxfsMotorizedCard interface will fail with a JXFS_E_NOTSUPPORTED exception.

mediaStatus Property (R)

Type ***JxfsMediaStatus***
Initial Value A JxfsMediaStatus (see related section in Base Architecture document).
Description Specifies the state of the media.
Event If the value of this property changes, the Device Service will send all registered status listeners a *JxfsStatusEvent* with one of the following values:

Field	Value
status	JXFS_S_CCD_MEDIA_STATUS *mediaStatus* has changed.
details	A ***JxfsMediaStatus*** object.

cardStatus Property (R)

Type ***JxfsCCDCardStatus***
Initial Value Depends on device service
Description Specifies the current state of the chip card.
Event If the value of this property changes, the Device Service will send all registered status listeners a *JxfsStatusEvent* with one of the following values:

Field	Value
status	JXFS_S_CCD_CARD_STATUS *cardStatus* has changed.
details	A ***JxfsCCDCardStatus o***bject.

manipulationStatus Property (R)

Type ***JxfsManipulationStatusEnum***
Initial Value Default *JxfsManipulationStatusEnum* object.
Description Specifies the state of any present anti fraud feature.
Event If the value of this property changes, the Device Service will send all registered status listeners a JxfsStatusEvent with the following values:

Field	Value
status	JXFS_S_CCD_MANIPULATION_STATUS *manipulationStatus* has changed.
details	A *JxfsManipulationStatusEnum* object.

4.4.3 Methods

isCcdT

Syntax ***boolean isCcdT (int noOfProtocol) throws JxfsException;***

Description This method is used to obtain information on which protocols are supported by the device.
Returns *true* if protocol Tnn, where nn is the value of the parameter, is supported, *false* otherwise.

Parameter

Type	Name	Meaning
int	noOfProtocol	Number of protocol being queried, from 0 to 15 for protocols T0 to T15.

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

chipInit

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

Description Performs a chip card initialization and reads the answer to reset (ATR) data.

If media is present, the operation is performed immediately. Otherwise, the device waits until it is present or the operation is cancelled.

After a successful completion of this operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the result.

Event

JxfsOperationCompleteEvent
When a *chipInit()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by CCD Device Control to all registered operation complete listeners. It will contain the data read.

Field	Value
operationID	JXFS_O_CCD_CHIPINIT
identificationID	Identification Id of complete operation.
result	Common or device dependent error code. (See section on Error codes).
data	A **JxfsCCDData** object. It contains ATR data from chip. In the case that there is no chip card data available in case of an error, the data reference equals null. An application should be aware that data may be available even in case of error in certain situations.

JxfsIntermediateEvent
JxfsIntermediateEvent can be sent by CCD Device Control to all registered intermediate listeners

Field	Value
operationID	JXFS_O_CCD_CHIPINIT
identificationID	Identification Id of operation.
reason	JXFS_I_CCD_NO_MEDIA_PRESENT The read operation request cannot progress because there is no media inserted.

		JXFS_I_CCD_MEDIA_INSERTED The read operation request continues because a media has been inserted.
	data	null

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

chipIO

Syntax ***identificationID chipIO (byte[] chipData, int protocol) throws JxfsException;***

Description This method initiates an input/output operation. The content of *chipData* is sent to the chip card. Replied data from the chip card is returned to the application in a *JxfsOperationCompleteEvent* event. The parameter *protocol* specifies the protocol to use.

After a successful completion of this operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the results.

Parameter	**Type**	**Name**	**Meaning**
	byte[]	chipData	Data to be sent.
	int	protocol	Protocol to be used (0..15).

Event **JxfsOperationCompleteEvent**
When a *chipIO()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by CCD Device Control to all registered operation complete listeners. It will contain the data read.

Field	**Value**
operationID	JXFS_O_CCD_CHIPIO
identificationID	Identification Id of complete operation.
result	Common or device dependent error code. (See section on Error codes).
data	A **JxfsCCDData** object. It contains data returned from chip if operation completed successfully. In the case that there is no chip card data available in case of an error, the data reference equals null. An application should be aware that data may be available even in case of error in certain situations.

JxfsIntermediateEvent
JxfsIntermediateEvent can be sent by CCD Device Control to all registered intermediate listeners

Field	**Value**
operationID	JXFS_O_CCD_CHIPIO
identificationID	Identification Id of operation.
reason:	JXFS_I_CCD_NO_MEDIA_PRESENT The read operation request cannot progress because there is no media inserted. JXFS_I_CCD_MEDIA_INSERTED The read operation request continues because a media has been inserted.
data	null

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

isActivateCardSupported

Syntax ***boolean isActivateCardSupported() throws JxfsException;***

Description This method is used to obtain information if the *activateCard* method is supported.
Returns *true* if the method is supported, *false* otherwise.

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

isDeactivateCardSupported

Syntax ***boolean isDeactivateCardSupported() throws JxfsException;***

Description This method is used to obtain information if the *deactivateCard* method is supported.
Returns *true* if the method is supported, *false* otherwise.

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

isWarmResetCardSupported

Syntax ***boolean isWarmResetCardSupported() throws JxfsException;***

Description This method is used to obtain information if the *warmResetCard* method is supported.
Returns *true* if the method is supported, *false* otherwise.

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

activateCard

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

Description Activates the already contacted card electrically. If the card is already activated, the card will be deactivated and then activated again. This functionality is also known as a cold reset. The difference to the *chipInit*() method is that the *chipInit* () method also decontacts the chip (if the contacting of the chip is motorized) which may help if there are contact problems, but takes more time.

Invoking this command will not move the card. This method will fail, if there is no card in contact position.

After a successful completion of this operation, a JxfsOperationCompleteEvent event is issued to inform the application of the result.

Event **JxfsOperationCompleteEvent**
When an activateCard*()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by CCD Device Control to all registered operation complete listeners. It will contain the data read.

Field	**Value**
operationID	JXFS_O_CCD_ACTIVATE_CARD
identificationID	Identification Id of the completed operation.

	result	Common or device dependent error code. (See section on Error codes).
	data	A **JxfsCCDData** object if no fatal error occurred. It contains ATR data from chip. Otherwise this value is null.

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

deactivateCard

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

Description Deactivates the card electrically. If the card is already deactivated, the method will return at once with a *JxfsOperationCompleteEvent* indicating success.

Invoking this command will not move the card. This method will fail, if there is no card in contact position.

After a successful completion of this operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the result.

Event **JxfsOperationCompleteEvent**
When a *deactivateCard()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by CCD Device Control to all registered operation complete listeners. It will contain the data read.

Field	**Value**
operationID	JXFS_O_CCD_DEACTIVATE_CARD
identificationID	Identification Id of the completed operation.
result	Common or device dependent error code. (See section on Error codes).
data	This value is null.

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

warmResetCard

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

Description Performs a warm reset on the chip card. If the chip card is present, but deactivated, the method will return with a JXFS_E_ILLEGAL error.

Invoking this command will not move the card. This method will fail, if there is no card in contact position.

After a successful completion of this operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the result.

Event **JxfsOperationCompleteEvent**
When a *warmResetCard()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by CCD Device Control to all registered operation complete listeners. It will contain the data read.

Field	**Value**
operationID	JXFS_O_CCD_WARM_RESET_CARD
identificationID	Identification Id of the completed operation.
result	Common or device dependent error code. (See section on Error codes).

	data	A **JxfsCCDData** object if no fatal error occurred. It contains ATR data from chip. Otherwise this value is null.
Exceptions	No additional exceptions are generated. See section on JxfsExceptions for common value codes.	

4.5 IJxfsMotorizedCard

4.5.1 Introduction

This interface contains those properties and functions commonly supported in motorized card devices (such as motorized magnetic card readers/encoder and chip card stations) related with its mechanical capabilities like eject or retain cards.

It is intended that this interface will be implemented by device controls that represent physical devices able to manage cards with chip or magnetic stripes (that is, subclasses of JxfsMagStripe and JxfsChipCard classes) that are equipped with motorized and mechanical capabilities.

Summary

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

Therefore, the IJxfsMotorizedCard consists on the following methods:

- Getters of listed properties.
- Methods listed.

Property	Type	Access	Initialized after
powerOffCapabilities	**int**	R	
powerOnCapabilities	**int**	R	
retainBinStatus	**JxfsThresholdStatus**	R	
retainCardCount	**int**	R	
retainCapability	**boolean**	R	
secureModuleType	**int**	R	

Method	Return	May use after
get*Property*	*Property*	
set*Property*	*void*	
resetRetainCardCount	*void*	
ejectCard	identificationID	
retainCard	identificationID	

4.5.2 Properties

powerOffCapabilities Property (R)

Type ***int***
Initial Value Depends on device.
Description Indicates the action taken by the device at power off if media is present. Depending on the device capabilities it will be set with one of the following values:

Value	Meaning
JXFS_MOTOR_EJECT	Card is ejected.
JXFS_MOTOR_EJECT_THEN_RETAIN	Card is ejected, then, after some seconds, it is retained.
JXFS_MOTOR_NOACTION	No action is taken.
JXFS_MOTOR_READ_POSITION	Card is brought to the read/write position.
JXFS_MOTOR_RETAIN	Card is retained.

powerOnCapabilities Property (R)

Type ***int***

Initial Value Depends on device.

Description Indicates the action taken by the device at power on if media is present. Depending on the device capabilities it will be set with one of the following values:

Value	Meaning
JXFS_MOTOR_EJECT	Card is ejected.
JXFS_MOTOR_EJECT_THEN_RETAIN	Card is ejected, then, after some seconds, it is retained.
JXFS_MOTOR_NOACTION	No action is taken.
JXFS_MOTOR_READ_POSITION	Card is brought to the read/write position.
JXFS_MOTOR_RETAIN	Card is retained.

retainBinStatus Property (R)

Type ***JxfsThresholdStatus***

Initial Value A JxfsThresholdStatus (see related section in Base Architecture document).

Description Indicates the fill status of the retain bin, if supported.

Event If the value of this property changes, the Device Service will send all registered status listeners a *JxfsStatusEvent* with the following value:

Field	Value
status	JXFS_S_MOTOR_BIN_STATUS *retainBinStatus* has changed.
details	A ***JxfsThresholdStatus*** object.

retainCardCount Property (R/W)

Type ***int***

Initial Value Depends on device at open.

Description Number of cards retained. This value is persistent independently of the power/open/close state.
The *resetRetainCardCount* method resets this property to 0.

Event If the value of this property changes (increments), the Device Service will send all registered status listeners a *JxfsStatusEvent* with a status value of:

Field	Value
status	JXFS_S_MOTOR_BIN_CARDRETAINED *retainCardCount* has incremented.
details	None.

retainCapability Property (R)

Type ***boolean***

Initial Value Depends on device.

Description Indicates if device is able to retain cards.
True means it is able to retain, false no retain capability support.

secureModuleType Property (R)

Type ***int***

Initial Value Depends on device.

Description Contains the secure module type, if any being used by the device.

Value	Meaning
JXFS_MSD_SECTYPE_NOTSUPPORTED	No security module available.
JXFS_MSD_SECTYPE_MMBOX	MMBox module.
JXFS_MSD_SECTYPE_CIM86	CIM86 module

4.5.3 Methods

resetRetainCardCount

Syntax ***void resetRetainCardCount ()***

Description Sets *retainCardCount* property to 0.

ejectCard

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

Description Ejects the card allowing card taking from user.

Event

JxfsOperationCompleteEvent
When a *ejectCard()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by the Device Control to all registered operation complete listeners with the following data:

Field	Value
operationID	JXFS_O_MOTOR_EJECTCARD
identificationID	The corresponding Id.
Result	Common or device dependent error code. (See section on Error codes).
data:	null.

JxfsStatusEvent
A *JxfsStatusEvent* can be sent by the Device Control to all registered status listeners

Field	Value
status	JXFS_S_MEDIA_STATUS
details	JxfsMediaStatus mediaStatus The new media status of the device.

Exceptions No additional exceptions are generated. See section on JxfsExceptions for common value codes.

retainCard

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

Description Retains card.

Event **JxfsOperationCompleteEvent**
When a *retainCard()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by the Device Control to all registered operation complete listeners.

Field	Value
OperationID	JXFS_O_MOTOR_RETAINCARD
IdentificationID	The corresponding Id.
Result	Common or device dependent error code. (See section on Error codes).
data	null

JxfsStatusEvent

A *JxfsStatusEvent* can be sent by the Device Control to all registered status listeners

Field	**Value**
status	JXFS_S_MEDIA_STATUS
details	JxfsMediaStatus mediaStatus The new media status of the device.

Exceptions

No additional exceptions are generated. See section on JxfsExceptions for common value codes.

4.6 IJxfsMSDSecure

4.6.1 Introduction

This interface contains properties and functions that may be supported in motorized card MSD devices with a security box installed.

It is intended that this interface will be implemented by device controls that represent physical devices with the security feature.

Summary

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

Therefore, the IJxfsMSDSecure consists on the following methods:

- Getters of listed properties.
- Methods listed.

Property	Type	Access	Initialized after
secureModuleKey	**byte[]**	R/W	
secureModuleStatus	**int**	R	

Method	Return	May be used after
get*Property*	*Property*	
set*Property*	*void*	
readData	identificationID	
readWMtrack	identificationID	

4.6.2 Properties

secureModuleKey Property (R/W)

Type ***byte []***
Initial Value null
Description Contains the secure module key with parity. Its value should be introduced once and be kept after power off.

secureModuleStatus Property (R)

Type ***int***
Initial Value Depends on device at open.
Description Indicates the status of the security module, if any.

Value	Meaning
JXFS_S_MSD_SEC_READY	Security module ready.
JXFS_S_MSD_SEC_NOTREADY	Security module not ready.
JXFS_S_MSD_SEC_UNKNOWN	State of the security module cannot be determined with the device in its current state.

Event If the value of this property changes, the Device Service will send all registered status listeners a *JxfsStatusEvent* with a status value of:

Field	Value
status	JXFS_S_MSD_SEC_STATUS *secureModuleStatus* has changed.
details	None.

4.6.3 Methods

readData

Syntax

identificationID readData (JxfsMSDTrackSelection tracksToRead, JxfsMSDSecureMode secureMode) throws JxfsException;

Description

This method overloads the normal readData method.

It launches a read operation to obtain the data contained in the tracks specified by the *tracksToRead* parameter.

If media is present, the read operation is performed immediately. Otherwise, the device waits until it is present or the operation is cancelled.
After a successful completion of this input operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the results.

Parameter

Type	Name	Meaning
JxfsMSDTrackSelection	tracksToRead	Tracks to be read.
JxfsMSDSecureMode	secureMode	Required settings for secure operation.

Event

JxfsOperationCompleteEvent
When a *readData()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by MSD Device Control to all registered operation complete listeners. It will contain the data read.

Field	Value
operationID	JXFS_O_MSD_READDATA
identificationID	Identification ID of complete operation.
result	Common or device dependent error code. (See section on Error codes).
data	A **JxfsMSDReadDataSecure** object.

JxfsIntermediateEvent
JxfsIntermediateEvent can be sent by MSD Device Control to all registered intermediate listeners

Field	Value
operationID	JXFS_O_MSD_READDATA
identificationID	Identification ID of operation.
reason	JXFS_I_MSD_NO_MEDIA_PRESENT The read operation request cannot progress because there is no media inserted. JXFS_I_MSD_MEDIA_INSERTED The read operation request continues because a media has been inserted.
data	null

Exceptions

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

Value	Meaning
JXFS_E_MSD_NOTSUPPORTEDTRACK	At least one track specified in *tracksToRead* parameter is not supported by the device.
JXFS_E_MSD_NOTRACKS	No tracks specified in *tracksToRead* parameter.

JXFS_E_MSD_NOTSUPPORTEDCAP	The service does not have secure capability.

readWMtrack

Syntax

identificationID readWMtrack () throws JxfsException;

Description

This method launches a read operation to obtain the data contained in the Watermark.

If media is present, the read operation is performed immediately. Otherwise, the device waits until it is present or the operation is cancelled.

After a successful completion of this input operation, a *JxfsOperationCompleteEvent* event is issued to inform the application of the results.

Event

JxfsOperationCompleteEvent

When a *readData()* operation is completed a *JxfsOperationCompleteEvent* event will be sent by MSD Device Control to all registered operation complete listeners. It will contain the data read.

Field	**Value**
operationID	JXFS_O_MSD_READDATA
identificationID	Identification ID of complete operation.
result	Common or device dependent error code. (See section on Error codes).
data	A **JxfsMSDWmData** with Watermark data.

Exceptions

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

Value	**Meaning**
JXFS_E_MSD_NOTSUPPORTEDTRACK	Watermark is not supported.

5 Support Classes

5.1 JxfsMSDTracks

This class provides properties and methods to query which tracks of a MSD device have been selected, are active or have been written.

Used by readData method.

Summary

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

Property	Type	Access	Initialized after
track1	boolean	R	
track2	boolean	R	
track3	boolean	R	

Method	Return	May use after
is*Property*	*Property*	
allTracks	boolean	
noTracks	boolean	
JxfsMSDTracks (boolean track1, boolean track2, boolean track3)	(constructor of the class)	

5.1.1 Properties

track1 Property (R)

Type ***boolean***
Initial Value false
Description Indicates if track1 is selected.

Value	Meaning
false	Track1 is not selected.
true	Track1 is selected.

track2 Property (R)

Type ***boolean***
Initial Value false
Description Indicates if track2 is selected.

Value	Meaning
false	Track2 is not selected.
true	Track2 is selected.

track3 Property (R)

Type ***boolean***
Initial Value false
Description Indicates if track3 is selected.

Value	Meaning
false	Track3 is not selected.
true	Track3 is selected.

5.1.2 Methods

isTrack1 .. isTrack3 Methods

Syntax ***boolean isTrack1 () .. boolean isTrack3 ()***
Description Returns *true* if specific track property is set to *true*.

allTracks Method

Syntax ***boolean allTracks ()***
Description Returns *true* if all tracks (*track1, track2* and *track3*) are set to *true*.

noTracks Method

Syntax ***boolean noTracks ()***
Description Returns *true* if all tracks (*track1, track2* and *track3*) are set to *false*.

JxfsMSDTracks Constructor

Syntax ***JxfsMSDTracks (boolean track1, boolean track2, boolean track3)***
Description Constructor of the class.

5.2 JxfsMSDTrackSelection

This class provides properties and methods to query and select the active tracks of a MSD device.

Summary

Implements : **--** **Extends :** **JxfsMSDTracks**

Property	Type	Access	Initialized after
No additional properties.			

Method	Return	May use after
set*Property*	void	
setAllTracks	void	
setNoTracks	void	
JxfsMSDTrackSelection (boolean track1, boolean track2, boolean track3)	(constructor of the class)	

5.2.1 Properties

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

5.2.2 Methods

setTrack1 .. setTrack3 Methods

Syntax ***void setTrack1 () .. void setTrack3 ()***
Description Set specific track property to *true*.

setAllTracks Method

Syntax ***void setAllTracks ()***
Description Sets all tracks (*track1, track2 and track3* properties) to *true*.

setNoTracks Method

Syntax ***void setNoTracks ()***
Description Sets all tracks (*track1, track2 and track3* properties) to *false*.

JxfsMSDTrackSelection Constructor

Syntax ***JxfsMSDTrackSelection (boolean track1, boolean track2, boolean track3)***
Description Constructor of the class.

5.3 JxfsMSDReadData

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

Summary

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

Property	Type	Access	Initialized after
DataRead	**java.util.Vector**	R	
tracksRead	**JxfsMSDTracks**	R	
resultReadTrack1	**int**	R	
resultReadTrack2	**int**	R	
resultReadTrack3	**int**	R	

Method	Return	May use after
get*Property*	*Property*	
JxfsMSDReadData (java.util.Vector dataRead, JxfsMSDTracks tracksRead, int resultReadTrack1, int resultReadTrack2, int resultReadTrack3)	(constructor of the class)	

5.3.1 Properties

dataRead Property (R)

Type ***java.util.Vector***
Description Vector of three byte []. Each one contains the raw data read from a track. Vector element 0 contains data for track 1, vector element 1 contains data for track 2, and so on.
If no data has been read for a given track, the corresponding vector element contains **null**.
The track data has no hardware control characters or BCC included (like SS, SE, or BCC). The data for ISO track #1 (6 bits per character) is transformed in the range of 0x20 to 0x5F and the data for the ISO tracks #2 and #3 (4 bits per character) are transformed in the range from 0x30 to 0x3F.

tracksRead Property (R)

Type ***JxfsMSDTracks***
Description Indicates which tracks were effectively read.

resultReadTrack1, resultReadTrack2, resultReadTrack3 Properties (R)

Type ***int***
Initial Value Depends on device type.
Description Holds the error code resulting from the read operation for the tracks that could not be read. Should be consulted when a global read error JXFS_E_MSD_READFAILURE has been reported.

Applications must not rely on specific error codes since these may depend on the specific device for a given faulty card.
They will be set with one of the following values:

Value	Meaning
JXFS_E_MSD_NOTSUPPORTEDTRACK	Track not supported by device.
JXFS_E_MSD_READFAILURE	Read error on track.
JXFS_E_MSD_PARITY	Parity read error.
JXFS_E_MSD_READ_EOF	Only SS,SE,BCC on track.
JXFS_E_MSD_NO_STRIPE	No magnetic stripe or flux on stripe detected (if device has capability to detect this situation).
JXFS_E_MSD_READ_OTHER	Any other type of error.

5.3.2 Methods

JxfsMSDReadData Constructor

Syntax ***JxfsMSDReadData (java.util.Vector dataRead, JxfsMSDTracks tracksRead, int resultReadTrack1, int resultReadTrack2, int resultReadTrack3)***

Description Constructor of the class.

5.4 JxfsCCDData

This class contains the data returned by an *OperationCompleteEvent* event for *chipInit*() and *chipIO()* operations.

Summary

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

Property	Type	Access	Initialized after
chipData	**byte[]**	R	

Method	Return	May use after
get*Property*	*Property*	
JxfsCCDData (byte[] chipData)	(constructor of the class)	

5.4.1 Properties

chipData Property (R)

Type ***byte[]***
Description Contains the data returned by the chip card after a successfull completion of an I/O operation.
If operation completed is *chipInit ()* , then it contains the ATR data from the chip.
If operation completed is *chipIO ()* , then it contains the data replied by the chip.

5.4.2 Methods

JxfsCCDData Constructor

Syntax ***JxfsCCDData (byte[] chipData)***
Description Constructor of the class.

5.5 JxfsMSDWmData

This class contains the data returned by a *JxfsOperationCompleteEvent* event for *readWMtrack()* operation.

Summary

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

Property	Type	Access	Initialized after
wmData	**byte[]**	R	

Method	Return	May use after
get*Property*	*Property*	
JxfsMSDWmData (byte[] wmData)	(constructor of the class)	

5.5.1 Properties

wmData Property (R)

Type ***byte[]***
Description Contains the raw Watermark data read

5.5.2 Methods

JxfsMSDWmData Constructor

Syntax ***JxfsMSDWmData (byte[] wmData)***
Description Constructor of the class.

5.6 JxfsMSDSecureMode

This class provides required properties for *readData()* operation in secure mode.

Summary

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

Property	Type	Access	Initialized after
securityCheck	boolean	R/W	
secureTestCard	boolean	R/W	

Method	Return	May be used after
is*Property*	*Property*	
set*Property*	*void*	
JxfsMSDSecureMode (boolean securityCheck, boolean secureTestCard)	(constructor of the class)	

5.6.1 Properties

securityCheck Property (R/W)

Type ***boolean***

Description Indicates whether a security check has to be requested in read operation. Since the overloaded method will normally be used when security check is desired, this property will usually be *true*.

Value	Meaning
true	Security check requested.
false	No security check requested.

securityTestCard Property (R/W)

Type ***boolean***

Description Indicates whether the card to be read is an ecCard or a Test Card.

Value	Meaning
true	Test card to be read.
false	Normal card to be read.

5.6.2 Methods

JxfsMSDSecureMode Constructor

Syntax ***JxfsMSDSecureMode (boolean securityCheck, boolean secureTestCard)***

Description Constructor of the class.

5.7 JxfsMSDReadDataSecure

This class contains the data returned by a *JxfsOperationCompleteEvent* event for readData*()* method in secure mode.

Summary

Implements : -- **Extends :** JxfsType

Property	Type	Access	Initialized after
readData	JxfsMSDReadData	R	
securityInfo	int	R	
testResult	byte	R	
cim86Info	byte []	R	

Method	Return	May be used after
get*Property*	*Property*	
JxfsMSDReadDataSecure (JxfsMSDReadData readData, int securityInfo, byte testResult, byte[] cim86Info)	(constructor of the class)	

5.7.1 Properties

readData Property (R)

Type ***JxfsMSDReadData***
Description This class contains the data obtained from *readData()* operation as in the unsecure mode. *See JxfsMSDReadData* class for details.

securityInfo Property (R)

Type ***int***
Description Indicates the result of the security check in the read operation, that could be one of the following values:

Value	Meaning
JXFS_MSD_SEC_NOCHECK	No security check was requested.
JXFS_MSD_SEC_NOTREADY	Security module was not ready.
JXFS_MSD_SEC_SECFAIL	Security module failed reading media security sign.
JXFS_MSD_SEC_SECOK	Successful security check.

testResult Property (R)

Type ***byte***
Description In case of a MM module:
Holds the result of the given customer card or the result of the given test card. See MM module specifications.

In case of a CIM-86 module:
Holds the test result for the given test card. See CIM-86 specifications.
In case of customer cards the content of this property is not defined.

cim86Info Property (R)

Type	***byte[]***
Description	In case of a MM module the content of this property is an empty array as the MM module does not provide additional information. In case of a CIM-86 module: Contains detailed result of the security check in the read operation for CIM-86 modules. The first byte contains the result of the security check for the given customer card. See CIM-86 specifications.

5.7.2 Methods

JxfsMSDReadDataSecure Constructor

Syntax	***JxfsMSDReadDataSecure (JxfsMSDReadData readData, int securityInfo, byte testResult, byte[] cim86Info)***
Description	Constructor of the class.

5.8 JxfsCCDCardStatus

This class defines the state of a present chip card. Every change of this state is reported with a JXFS_S_CCD_CARD_STATUS event.

Summary

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

Method	Return	Meaning
JxfsCCDCardStatus(boolean notSupported, boolean unknown, boolean inContactPosition, boolean contacted, boolean activated)	void	Constructs a new object, card state is set accordingly.
isNotSupported()	boolean	This functionality is not supported.
isUnknown()	boolean	The current state of the chip card is not known.
isInContactPosition()	boolean	A chip card is present in contact position.
isContacted()	boolean	A chip card is present and contacted.
isActivated()	boolean	A chip card is activated.
toString()	java.lang.String	Returns a short textual representation of the contents of this object.

5.8.1 Methods

JxfsCCDCardStatus Constructor

Syntax ***JxfsCCDCardStatus(boolean notSupported, boolean unknown, boolean inContactPosition, boolean contacted, boolean activated)***

Description Constructor of the class.
The following combinations of the states are allowed:

- All states false (either no card or card not in chip card contacting position)
- isInContactPosition
- isInContactPosition & isContacted
- isInContactPosition & isContacted & isActivated
- isUnknown
- isNotSupported

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

Value	Meaning
JXFS_E_PARAMETER_INVALID	An invalid combination of values was provided.

isInContactPosition Method

Syntax ***boolean isInContactPosition ()***

Description Returns *true* if card is present in chip card contacting position.

isContacted Method

Syntax ***boolean isContacted()***
Description Returns *true* if card is physically contacted.

isActivated Method

Syntax ***boolean isActivated()***
Description Returns *true* if card is electrically activated.

isUnknown Method

Syntax ***boolean isUnknown()***
Description Returns *true* if the status of the card cannot be determined with the device in its current state.

IsNotSupported Method

Syntax ***boolean isNotSupported()***
Description Returns *true* if this functionality is not supported by this device.

In a motorized reader a card will first be present (mediaStatus), then a chipInit() will transport it into the contact position (isInContactPosition()==true), the card will be contacted (isContacted()==true) and activated (isActivated()==true).

For a JXFS_CCD_TYPE_PERMANENT card reader isInContactPosition() and isContacted() will always return the same value. As most unintelligent JXFS_CCD_TYPE_PERMANENT card readers do not have sensors to distinguish if there is a card at all, isInContactPosition() and isContacted() will always return true on those readers.

6 Enum Classes

6.1 JxfsMSDStatusSelectorEnum

This enumeration class is used for the base *getStatus(java.util.List)* method.

Implements : -- **Extends :** **JxfsStatusSelectorEnum**

Field	Returned Type	Description
status	JxfsStatus	General status of the device
mediaStatus	JxfsMediaStatus	Status of the current media.
cardStatus	JxfsCCDCardStatus	Status of the chip card. This status is available only if the device implements the *IJxfsChipCardControl* interface.
manipulationStatus	JxfsManipulationStatusEnum	Speficies the state of any present anti fraud feature.
retainBinStatus	JxfsThresholdStatus	Indicates the fill status of the retain bin, if supported. This status is only available if the device service implements the *IJxfsMotorizedCard* interface.
secureModuleStatus	Integer	Indicates the status of the security module, if any. This status is only available if the device service implements the *IJxfsMSDSecure* interface.

6.2 JxfsManipulationStatusEnum

This enumerated data type represents the possible states of a present anti fraud feature.

Implements : -- **Extends :** **JxfsEnum**

Field	Description
presentWorking	Anti fraud feature is present and is working correctly. If a device has more than one security feature, this means that all of them are present and working. This status does not necessarily mean that the manipulated status of the base status will ever be armed as there are anti fraud features that prevent an attack, but cannot report it.
presentNotWorking	Anti fraud feature is present but not working correctly. If a device has more than one security feature, this means that at least one of them is not working correctly.
unknown	It is not known if an anti fraud feature is present.
notSupported	No anti fraud feature is present.

7 Codes

7.1 Error Codes

Value	Meaning
JXFS_E_MSD_READFAILURE	No read conditions were satisfied (that is, not all tracks specified in *tracksToRead* parameter have been read or no Watermark was read). It is possible, however, that some tracks could be read. Check any existing data object for extended information on tracks actually read.
JXFS_E_MSD_NOMEDIA	Media was removed before operation completion.
JXFS_E_MSD_INVALIDMEDIA	No appropriated media was found.
JXFS_E_MSD_MEDIAJAMMED	Media is jammed.
JXFS_E_MSD_SHUTTERFAIL	Shutter could not be opened.
JXFS_E_MSD_NOTSUPPORTEDTRACK	At least one track specified in *tracksToRead* parameter is not supported by the device.
JXFS_E_MSD_NOTRACKS	No tracks specified in tracksToRead parameter.
JXFS_E_MSD_WRITEFAILURE	No write conditions were satisfied.
JXFS_E_MSD_BADDATA	Data is invalid.
JXFS_E_MSD_NOTSUPPORTEDCAP	The service does not have secure capability.
JXFS_E_MSD_PARITY	Parity read error.
JXFS_E_MSD_READ_EOF	Only SS,SE,BCC on track.
JXFS_E_MSD_NO_STRIPE	No magnetic stripe or flux on stripe detected (if device has capability to detect this situation).
JXFS_E_MSD_READ_OTHER	Any other type of read error.

Value	Meaning
JXFS_E_CCD_IOERROR	IO error occurred. No ATR data is available.
JXFS_E_CCD_NOMEDIA	Media was removed before operation completion.
JXFS_E_CCD_INVALIDMEDIA	No appropriated media was found.
JXFS_E_CCD_MEDIAJAMMED	Media is jammed.
JXFS_E_CCD_SHUTTERFAIL	Shutter could not be opened.
JXFS_E_CCD_BADDATA	Chip reported data was bad.
JXFS_E_CCD_BADPROTOCOL	Protocol not supported.
JXFS_E_CCD_MISSINGMEDIA	No media available when operation was initiated.
JXFS_E_CCD_INVALID_SEQUENCE	The chipIO() method was issued even if there was no card available or the present card was not yet activated.

Value	Meaning
JXFS_E_MOTOR_MEDIAJAMMED	Media is jammed.
JXFS_E_MOTOR_SHUTTERFAIL	Shutter could not be opened.
JXFS_E_MOTOR_NOMEDIA	There is no media to eject.
JXFS_E_MOTOR_BINFULL	Retain bin is full.

7.2 Status Codes

Value	Meaning
JXFS_S_MSD_MEDIA_STATUS	*mediaStatus* property has changed.
JXFS_S_MSD_MANIPULATION_STATUS	*manipulationStatus* property has changed.

Value	Meaning
JXFS_S_CCD_MEDIA_STATUS	*mediaStatus* property has changed.
JXFS_S_CCD_CARD_STATUS	*cardStatus* property has changed.
JXFS_S_CCD_MANIPULATION_STATUS	*manipulationStatus* property has changed.

Value	Meaning
JXFS_S_MOTOR_BIN_STATUS	*retainBinStatus* property has changed.
JXFS_S_MOTOR_BIN_CARDRETAINED	*retainCardCount* property has incremented.

Value	Meaning
JXFS_S_MSD_SEC_STATUS	*secureModuleStatus* property has changed.
JXFS_S_MSD_SEC_READY	Security module ready.
JXFS_S_MSD_SEC_NOTREADY	Security module not ready.
JXFS_S_MSD_SEC_UNKNOWN	State of the security module cannot be determined with the device in its current state.

7.3 Operation Codes

The following codes identify the operation that generated a *JxfsOperationCompleteEvent* or *JxfsIntermediateEvent*:

Value	Method
JXFS_O_MSD_READDATA	*readData, readWMtrack*
JXFS_O_MSD_WRITEDATA	*writeData*

Value	Method
JXFS_O_CCD_CHIPINIT	*chipInit*
JXFS_O_CCD_CHIPIO	*chipIO*
JXFS_O_CCD_ACTIVATE_CARD	*activateCard*
JXFS_O_CCD_DEACTIVATE_CARD	*deactivateCard*
JXFS_O_CCD_WARM_RESET_CARD	*warmResetCard*

Value	Method
JXFS_O_MOTOR_EJECTCARD	*ejectCard*
JXFS_O_MOTOR_RETAINCARD	*retainCard*

The following codes identify the reason for a *JxfsIntermediateEvent*:

Value	Meaning
JXFS_I_MSD_NO_MEDIA_PRESENT	The read operation request cannot progress because there is no media inserted.
JXFS_I_MSD_MEDIA_INSERTED	The read operation request continues because a media has been inserted.

Value	Meaning
JXFS_I_CCD_NO_MEDIA_PRESENT	The read operation request cannot progress because there is no media inserted.
JXFS_I_CCD_MEDIA_INSERTED	The read operation request continues because a media has been inserted.

7.4 Constants

Value	Meaning
JXFS_MSD_TYPE_SWIPE	Swipe/pull through magnetic stripe reader/encoder.
JXFS_MSD_TYPE_DIP	Dip magnetic card reader/encoder.
JXFS_MSD_TYPE_MOTOR	Motorized card reader.
JXFS_MSD_SECTYPE_NOTSUPPORTED	No security module available.
JXFS_MSD_SECTYPE_MMBOX	MMBox module available.
JXFS_MSD_SECTYPE_CIM86	CIM86 module available.

Value	Meaning
JXFS_CCD_TYPE_SWIPE	Swipe/pull through chip card device.
JXFS_CCD_TYPE_DIP	Dip chip card device.
JXFS_CCD_TYPE_MOTOR	Motorized chip card device.
JXFS_CCD_TYPE_CONTACTLESS	Contactless chip card device.
JXFS_CCD_TYPE_DIP_LATCHED	Dip chip card device with a latch to fix the card.
JXFS_CCD_TYPE_PERMANENT	Permanent chip card.

Value	Meaning
JXFS_MOTOR_EJECT	At power off /on card is ejected.
JXFS_MOTOR_EJECT_THEN_RETAIN	At power off /on card is ejected, then, after some seconds, it is retained.
JXFS_MOTOR_NOACTION	At power off /on no action is taken.
JXFS_MOTOR_READ_POSITION	At power off /on card is brought to the read/write position.
JXFS_MOTOR_RETAIN	At power off /on card is retained.

Value	Meaning
JXFS_MSD_SEC_NOCHECK	No security check was requested.
JXFS_MSD_SEC_NOTREADY	Security module was not ready.
JXFS_MSD_SEC_SECFAIL	Security module failed reading media security sign.
JXFS_MSD_SEC_SECOK	Successful security check.

7.5 Code Values

7.5.1 MSD Device types

Constant	Numerical Value
JXFS_MSD_TYPE_SWIPE	1
JXFS_MSD_TYPE_DIP	2
JXFS_MSD_TYPE_MOTOR	4

7.5.2 MSD Operation Complete codes

Constant	Numerical Value
JXFS_O_MSD_READDATA	4006
JXFS_O_MSD_WRITEDATA	4008

7.5.3 MSD Intermediate Event codes

Constant	Numerical Value
JXFS_I_MSD_NO_MEDIA_PRESENT	4014
JXFS_I_MSD_MEDIA_INSERTED	4015

7.5.4 MSD Status codes

Constant	Numerical Value
JXFS_S_MSD_MEDIA_STATUS	4005
JXFS_S_MSD_BIN_STATUS	4040
JXFS_S_MSD_SEC_READY	4052
JXFS_S_MSD_SEC_NOTREADY	4053
JXFS_S_MSD_SEC_UNKNOWN	4054
JXFS_S_MSD_SEC_STATUS	4055
JXFS_S_MSD_MANIPULATION_STATUS	4065

7.5.5 MSD Error codes

Constant	Numerical Value
JXFS_E_MSD_READFAILURE	4007
JXFS_E_MSD_WRITEFAILURE	4009
JXFS_E_MSD_NOMEDIA	4010
JXFS_E_MSD_INVALIDMEDIA	4011
JXFS_E_MSD_MEDIAJAMMED	4012
JXFS_E_MSD_SHUTTERFAIL	4013
JXFS_E_MSD_NOTSUPPORTEDTRACK	4016
JXFS_E_MSD_NOTRACKS	4017
JXFS_E_MSD_BADDATA	4018
JXFS_E_MSD_NOTSUPPORTEDCAP	4056
JXFS_E_MSD_PARITY	4057
JXFS_E_MSD_READ_EOF	4058
JXFS_E_MSD_NO_STRIPE	4059
JXFS_E_MSD_READ_OTHER	4060

7.5.6 MSD Secure Module types

Constant	Numerical Value
JXFS_MSD_SECTYPE_NOTSUPPORTED	4042
JXFS_MSD_ SECTYPE_MMBOX	4043
JXFS_MSD_ SECTYPE_CIM86	4044

7.5.7 MSD Security information codes

Constant	Numerical Value
JXFS_MSD_SEC_NOCHECK	4061
JXFS_MSD_SEC_NOTREADY	4062
JXFS_MSD_SEC_SECFAIL	4063
JXFS_MSD_SEC_SECOK	4064

7.5.8 CCD Device types

Constant	Numerical Value
JXFS_CCD_TYPE_SWIPE	1
JXFS_CCD_TYPE_DIP	2
JXFS_CCD_TYPE_MOTOR	4
JXFS_CCD_TYPE_CONTACTLESS	8
JXFS_CCD_TYPE_DIP_LATCHED	16
JXFS_CCD_TYPE_PERMANENT	32

7.5.9 CCD Operation Complete codes

Constant	Numerical Value
JXFS_O_CCD_CHIPINIT	4108
JXFS_O_CCD_CHIPIO	4109
JXFS_O_CCD_ACTIVATE_CARD	4114
JXFS_O_CCD_DEACTIVATE_CARD	4115
JXFS_O_CCD_WARM_RESET_CARD	4116

7.5.10 CCD Intermediate Event codes

Constant	Numerical Value
JXFS_I_CCD_NO_MEDIA_PRESENT	4106
JXFS_I_CCD_MEDIA_INSERTED	4107

7.5.11 CCD Status codes

Constant	Numerical Value
JXFS_S_CCD_MEDIA_STATUS	4113
JXFS_S_CCD_CARD_STATUS	4117
JXFS_S_CCD_MANIPULATION_STATUS	4119

7.5.12 CCD Error codes

Constant	Numerical Value
JXFS_E_CCD_NOMEDIA	4102
JXFS_E_CCD_INVALIDMEDIA	4103
JXFS_E_CCD_MEDIAJAMMED	4104
JXFS_E_CCD_SHUTTERFAIL	4105
JXFS_E_CCD_IOERROR	4110
JXFS_E_CCD_BADPROTOCOL	4111
JXFS_E_CCD_BADDATA	4112
JXFS_E_CCD_INVALID_SEQUENCE	4113
JXFS_E_CCD_MISSINGMEDIA	4118

7.5.13 Motorized Card Operation Complete codes

Constant	Numerical Value
JXFS_O_MOTOR_EJECTCARD	4245
JXFS_O_MOTOR_RETAINCARD	4250

7.5.14 Motorized Card Status codes

Constant	Numerical Value
JXFS_S_MOTOR_BIN_STATUS	4242
JXFS_S_MOTOR_BIN_CARDRETAINED	4241

7.5.15 Motorized Card Error codes

Constant	Numerical Value
JXFS_E_MOTOR_MEDIAJAMMED	4246
JXFS_E_MOTOR_SHUTTERFAIL	4247
JXFS_E_MOTOR_NOMEDIA	4248
JXFS_E_MOTOR_BINFULL	4251

7.5.16 Motorized Card Capabilities

Constant	Numerical Value
JXFS_MOTOR_EJECT	4230
JXFS_MOTOR_EJECT_THEN_RETAIN	4231
JXFS_MOTOR_NOACTION	4232
JXFS_MOTOR_READ_POSITION	4233
JXFS_MOTOR_RETAIN	4234

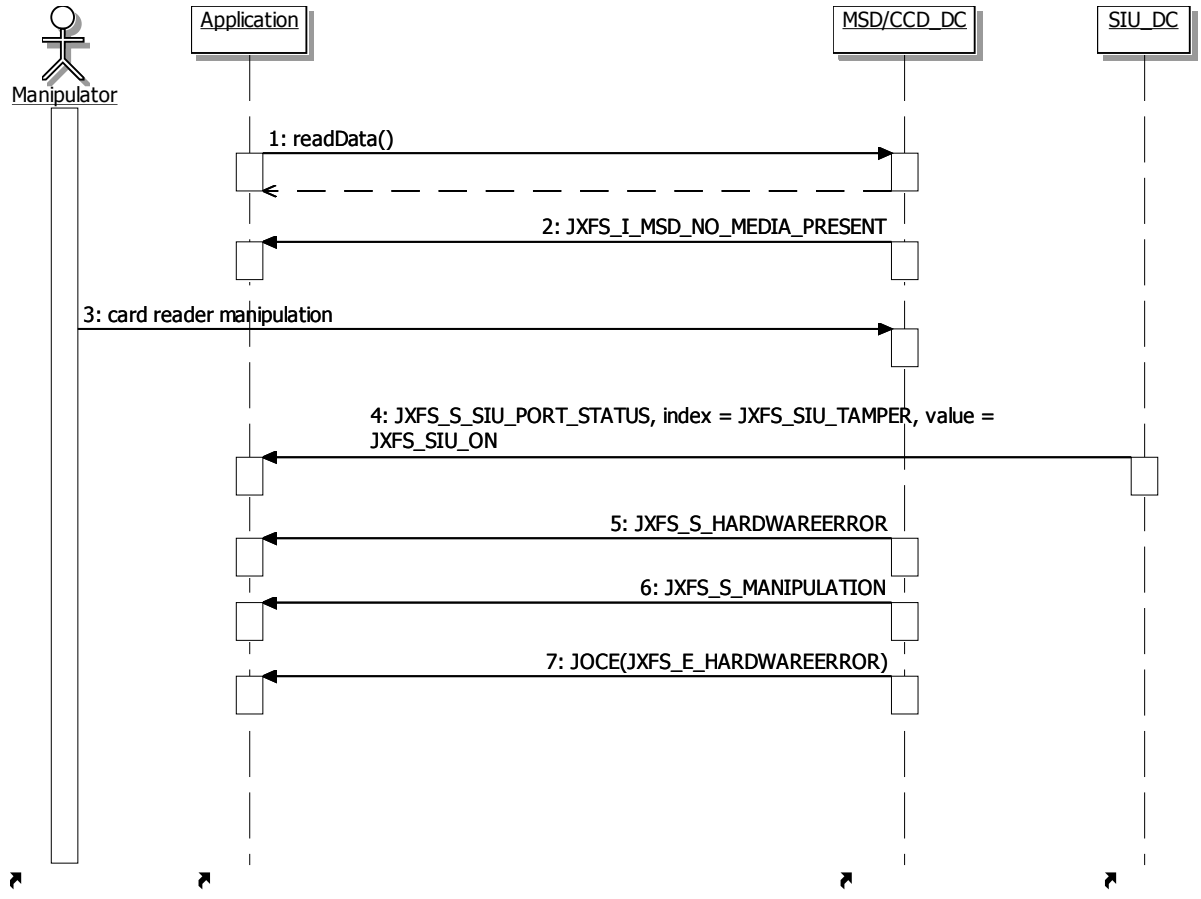
8 Device Service Interface Methods

The Device Service interface is common to all device services of this device type. It is used by the Device Controls to access the functionality of the device. This interface has to be implemented by any J/XFS Device Service.
The device type specific Device Service interface is similar to the Device Control interface. All device specific method calls are extended by an additional parameter (int control_id). This is always added as the last parameter in every operation.

9 Appendix A: Manipulation of Card Reader

In the case that a card reader is manipulated there are in general two options how to treat this situation. Either the DS does not allow further operation and completes with a hardware error (case #1) or the device service only flags the manipulation, but allows the application to continue (case #2).

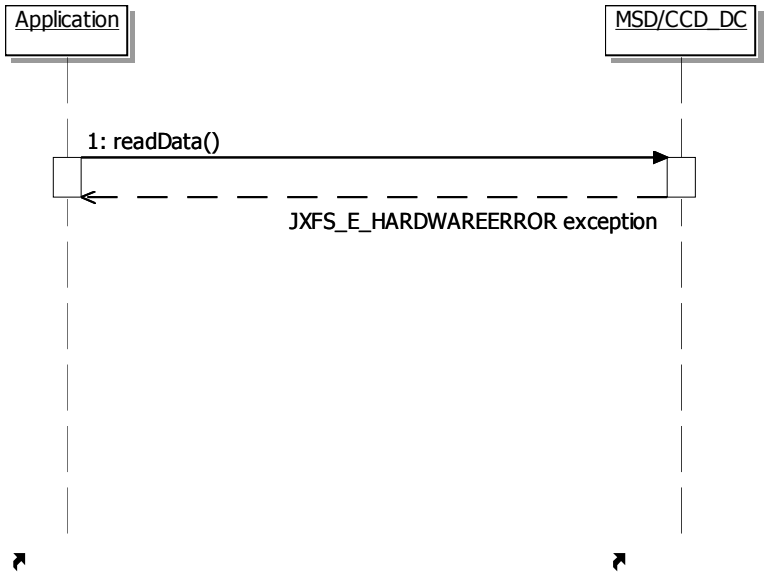
Case #1: DS Auto Completes With the Result Code: JXFS_E_HARDWAREERROR



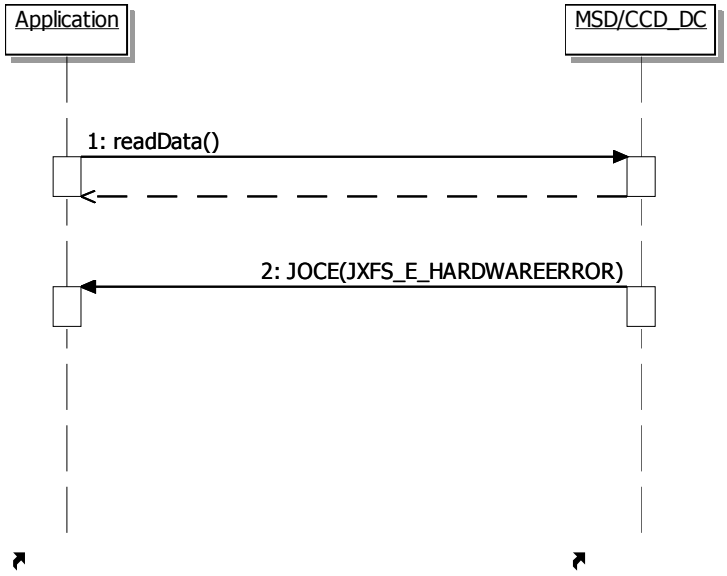
Summary

- ***readData( )*** is initiated successfully; the device waits for a card to be entered
- the card reader is manipulated:
    - a JXFS_S_SIU_PORT_STATUS event, if supported, is generated to indicate that tampering has been detected
    - a 'hardware error' status event plus a manipulation event is generated
- to secure the device, the device service automatically completes the ***readData( )*** operation with the return code ***JXFS_E_HARDWAREERROR***. If the device service receives a cancel(599) command before it initiates the automatic completion it returns
    - the result code: JXFS_E_CANCELLED

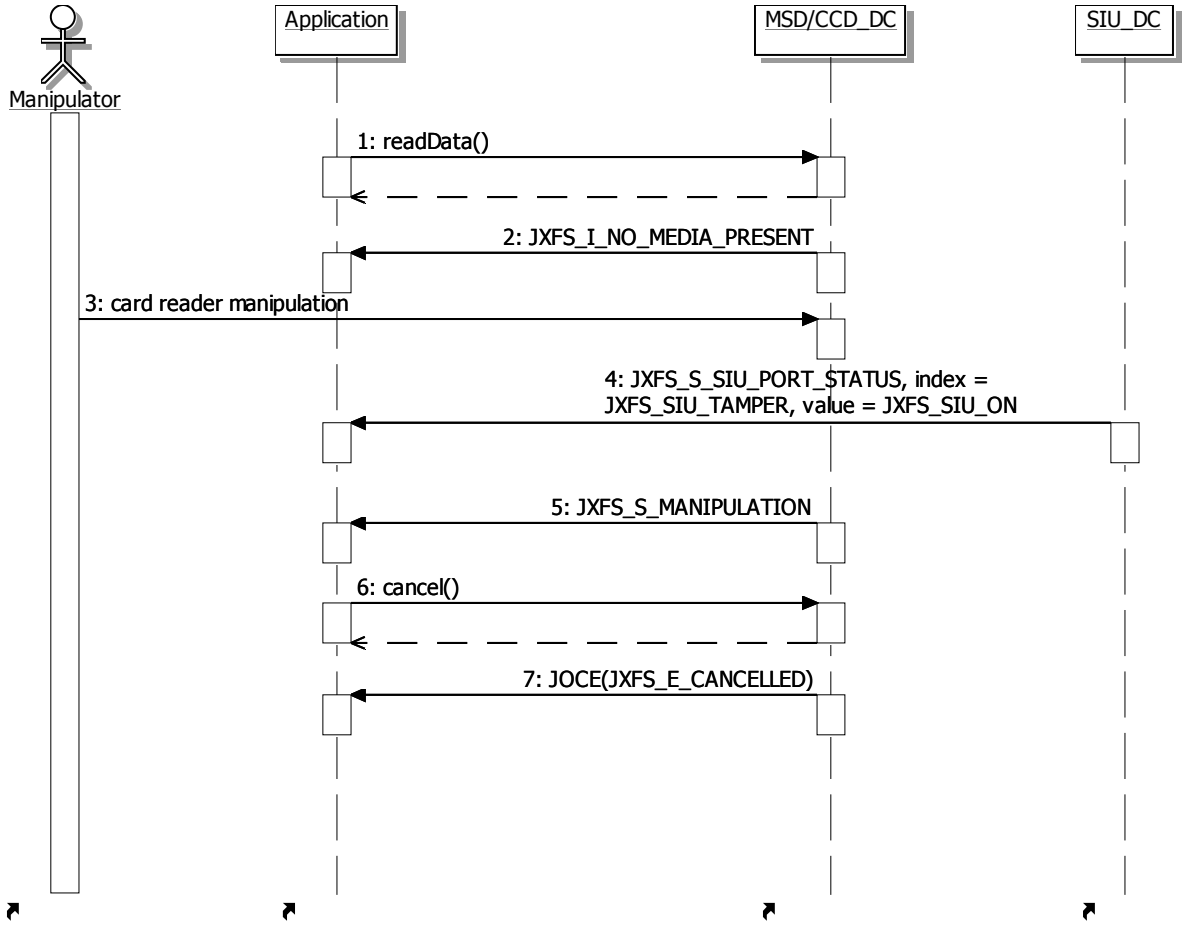
- Subsequent invocations of ***readData( )*** will result in one of the sequences shown below:
    - if no communication takes place between the device service and the physical device, a ***JXFS_E_HARDWAREERROR*** exception will be thrown.



    - if the device service communicates with the physical device, an operation complete event with result code ***JXFS_E_HARDWAREERROR*** will be returned.



Case #2: DS continues operation, but reports manipulation status



Summary

- ***readData( )*** is initiated successfully; the device waits for a card to be entered
- the card reader is manipulated:
    - a JXFS_S_SIU_PORT_STATUS event, if supported, is generated to indicate that tampering has been detected
    - a manipulation event is generated
    - the device service usually cancels the operation to prevent that a bank customer inserts a card that can be illegally read or used

- the application does not issue additional read requests as long as the manipulation status is pending or it is accepting the risk of manipulated cards of bank customers.