CEN

WORKSHOP

AGREEMENT

CWA 14923-5

May 2004

ICS 35.240.40

Supersedes CWA 13937-5:2003

English version

J/eXtensions for Financial Sevices (J/XFS) for the Java Platform - Part 5: Cash Dispenser, Recycler and ATM Device Class Interface - Programmer's Reference

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

Management Centre: rue de Stassart, 36 B-1050 Brussels

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/ISSS 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/ISSS 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/ISSS Secretariat. The specification was agreed upon by the J/XFS Workshop Meeting of 2004-02-10/11 in Saint-Denis (Paris) and a subsequent electronic review by the Workshop participants, and the final version was sent to CEN for publication on 2004-03-24.

The specification is continuously reviewed and commented in the CEN/ISSS J/XFS Workshop. The information published in this CWA is furnished for informational purposes only. CEN/ISSS makes no warranty expressed or implied, with respect to this document. Updates of the specification will be available from the CEN/ISSS J/XFS Workshop public web pages pending their integration in a new version of the CWA (see: http://www.cenorm.be/cenorm/businessdomains/businessdomains/informationsocietystandardizationsystem/applying+technologies/j-xfs+workshop/index.asp).

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

Questions and comments can also be submitted to the members of the J/XFS Forum, who are all CEN/ISSS J/XFS Workshop members, through the J/XFS Forum web-site http://www.jxfs.com

This CWA is composed of the following parts:

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

CWA 14923-5:2004 replaces CWA 13937-5:2003 and should be read in conjunction with CWA 13937-5:2000, which contains the previous release of the J/XFS specification

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://java.sun.com/nav/business/trademark_guidelines.html.
All other trademarks are trademarks of their respective owners.

History

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

- Article 6 added
- class diagram now include interfaces
- intermediateEvent re-introduced
- mixAlgorithm now Read-Only, corresponding statusevent removed
- new parameter for empty-method: JxfsCashUnit
- new cuType-constant in logical cash units: JXFS_C_CDR_LCU_CURRENCY_CASSETTE
- JXFS_S_CDR_ORDER_REMOVED renamed JXFS_S_CDR_DELAYED_ORDER_REMOVED
- Several constants marked as deprecated
- Mmissing constant codes added
- Reworked class diagram
- Chapter on Denominate removed
- Mixing redesigned again
- New chapter on Physical Escrow
- New chapter on Delayed Dispense
- New chapter on Recycler Rollback
- Document layout modified
- Mixing redesigned
- New constants added
- New chapter on Null value handling

1 Scope

This document describes the Cash Dispenser, Recycler and ATM device 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 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 Cash Dispenser, Recycler and ATM's 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

Cash Device Support within the J/XFS – API is available for the following device types:

- **Dispenser**
  General dispense devices consist of components that allow the dispensing of cash, either bills or coins. This interface provides common functionality that is although used by the following device types.

- **Recycler**
  A Recycler is primarily a Dispenser plus additional components that allow acceptance of cash as input to the device. This specification for Recyclers is intended for branch-teller environments and not for use in self-service environments.

- **ATM**
  ATM's (Automated Teller Machine) inherit their functional behaviour from Dispenser and Recycler. They also have functions to support ATM-specific hardware.

3 Classes and Interfaces

The following interfaces and classes are used by the J/XFS Cash Dispenser Device Controls.

Class or Interface	Name	Description	Extends or Implements
Interface	**IJxfsBaseControl**	Base interface for all device controls. Contains method declarations specific to all device controls.	
Interface	**IJxfsCashDispenserControl**	Base interface for all cash dispenser controls. Contains method declarations specific to cash dispenser controls.	Extends: **IJxfsBaseControl**
Interface	**IJxfsCashRecyclerControl**	Base interface for all cash recycler controls. Contains method declarations specific to cash recycler controls.	Extends: **IJxfsBaseControl**
Interface	**IJxfsATMControl**	Base interface for all ATM controls. Contains method declarations specific to ATM controls.	Extends: **IJxfsBaseControl**
Class	**JxfsCashDispenser**	Class for cash dispenser control.	Implements: **IJxfsCashDispenserControl, IJxfsBaseControl**
Class	**JxfsCashRecycler**	Class for cash recycler control.	Implements: **IJxfsCashDispenserControl, IJxfsCashRecyclerControl, IJxfsBaseControl**
Class	**JxfsATM**	Class for ATM control.	Implements: **IJxfsCashDispenserControl, IJxfsCashRecyclerControl, IJxfsATMControl, IJxfsBaseControl**

The following interfaces are used by the J/XFS Cash Dispenser Device Services.

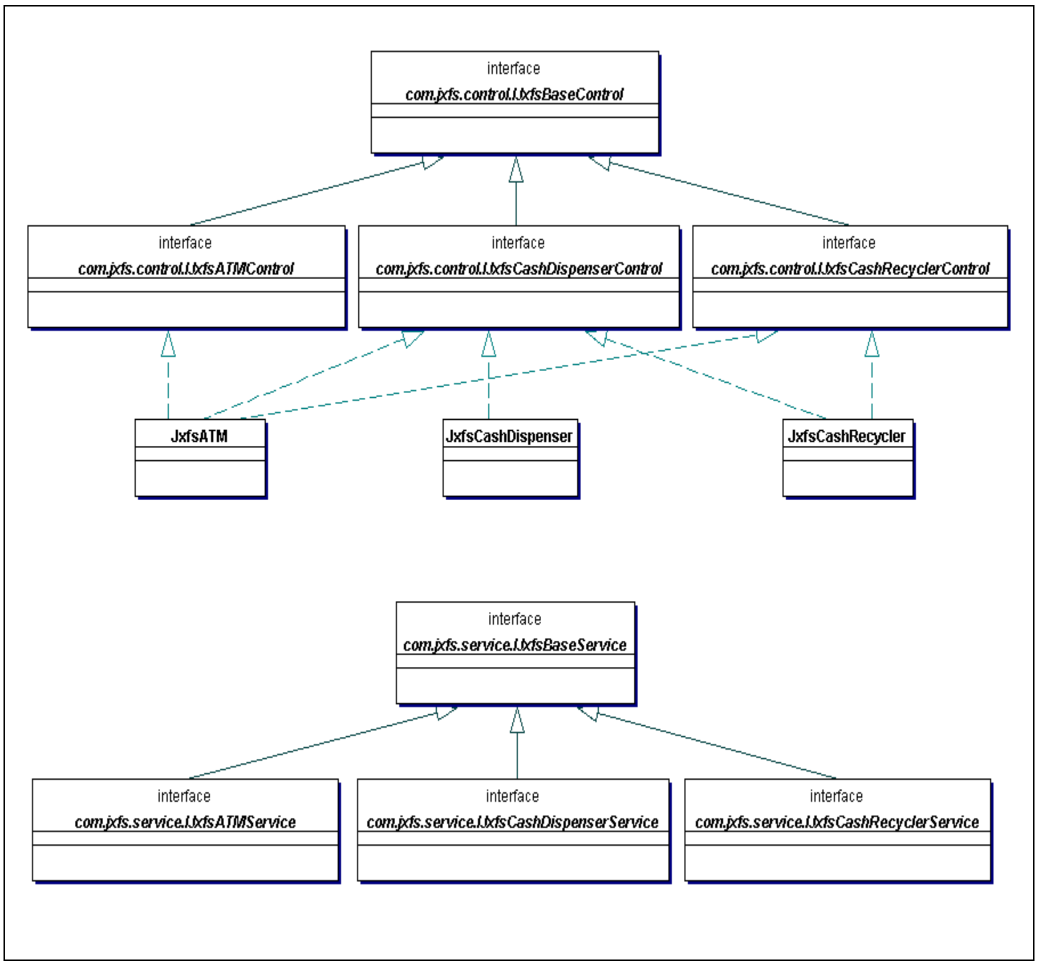
Class or Interface	**Name**	**Description**	**Extends or Implements**
Interface	**IJxfsBaseService**	Base interface for all services.	
Interface	**IJxfsCashDispenserService**	Base interface for all cash dispenser services. Contains method declarations specific to cash dispenser devices.	Extends: **IJxfsBaseService**
Interface	**IJxfsCashRecyclerService**	Base interface for all cash recycler services. Contains method declarations specific to cash recycler devices.	Extends: **IJxfsBaseService**
Interface	**IJxfsATMService**	Base interface for all ATM services. Contains method declarations specific to ATM devices.	Extends: **IJxfsBaseService**

Remark on Device Services

The Device Service interface is common for all device services of a specific 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 controlID). This is always added as the last parameter in every operation.

3.1 Class Diagram

The following class diagram shows the overall layout of the Cash Dispenser, Recycler and ATM interfaces and classes provided by J/XFS.


interface
com.jxfs.control.IJxfsBaseControl
interface
com.jxfs.control.IJxfsATMControl
interface
com.jxfs.control.IJxfsCashDispenserControl
interface
com.jxfs.control.IJxfsCashRecyclerControl
JxfsATM
JxfsCashDispenser
JxfsCashRecycler
interface
com.jxfs.service.IJxfsBaseService
interface
com.jxfs.service.IJxfsATMService
interface
com.jxfs.service.IJxfsCashDispenserService
interface
com.jxfs.service.IJxfsCashRecyclerService

3.2 Class and Interface Details

All operation methods return an identificationID. If a method cannot be processed immediately a JxfsException is thrown.

After processing has taken place, an OutputComplete – Event 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.

3.2.1 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 read or write a property the application must use the appropriate methods as defined in the JavaBeans specification.

3.2.1.1 get*Property*

Syntax	**Property *get*Property*(void) throws JxfsException;***
Description	Returns the requested property.
Parameter	**None**
Event	No additional events are generated.
Exceptions	Some possible JxfsException *value codes*. See section on JxfsExceptions for other JxfsException value codes. JXFS_E_CLOSED JXFS_E_REMOTE JXFS_E_UNREGISTERED

3.2.1.2 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*. See section on JxfsExceptions for other JxfsException value codes. JXFS_E_CLOSED JXFS_E_PARAMETER_INVALID JXFS_E_REMOTE JXFS_E_UNREGISTERED

3.3 IJxfsCashDispenserControl Summary

Extends	**Implements**
IJxfsBaseControl	

Property	**Type**	**Access**
capabilities	*JxfsCapabilities*	R
mixTable	*Vector of JxfsMixTable*	RW
uvv	*boolean*	RW
currencies	*Vector of JxfsCurrency*	R

Method	**Return**
get*Property*	*Property*
set*Property*	*void*
is*Property*	*boolean*
denominate	identificationID
dispense	identificationID
dispenseExec	identificationID
startExchange	identificationID
endExchange	identificationID
openSafeDoor	identificationID
calibrateCashUnit	identificationID
getDateTime	identificationID
setDateTime	identificationID
queryOrder	identificationID
removeOrder	identificationID
queryCashUnit	identificationID
updateCashUnit	identificationID
reset	identificationID

3.4 IJxfsCashDispenserControl Properties

3.4.1 capabilities (R)

Type	***JxfsCapabilities***
Remarks	Used to keep complete information about all device Capabilities.

3.4.2 mixTables (RW)

Type	***Vector of JxfsMixTable***
Remarks	Used to keep complete information about all MixTables.
Events	If the value of this property changes a *JxfsStatusEvent* is sent to all registered listeners with following data:

Field	**Value**
status	JXFS_S_CDR_MIXTABLE_CHANGED
details	***Vector of JxfsMixTable*** objects Updated property *mixTables*.

3.4.3 uvv (RW)

Type	***boolean***
Remarks	UVV is a german abreviation for „Unfallverhütungsvorschrift Kassen“. This is a regulation which describes the processing of dispensing cash according to german security rules. Defines the current mode for dispense operations. If set to true, delayed dispense (according to german security rules) is activated.

3.4.4 currencies (R)

Type	***Vector of JxfsCurrency***
Remarks	Contains a vector of supported currencies.

3.5 IJxfsCashDispenserControl Methods

Following methods are specific to CashDispenser devices.

3.5.1 denominate

Syntax ***identificationID denominate(int mixNumber, JxfsDenomination denomination, JxfsCurrency currency) throws JxfsException;***

Remarks Denominates a specified amount of money. Cash can be retrieved from different sources:

- cash dispenser
- coin dispenser
- teller's cash box

The configuration specifies the sources to be used in the JxfsDenominationFor a Dispenser all three can be used.
If the device used is an ATM, only the cash dispenser and, optionally, the coin dispenser can be available.

Parameter

Type	**Name**	**Description**
int	mixNumber	ID of mixtable or algorithm to use.
JxfsDenomination	denomination	Specifies the amount to denominate.
JxfsCurrency	currency	Specifies the Currency to use.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *denominate* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_DENOMINATE
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsDenomination*** object Specifies the calculated Denomination.

3.5.2 dispense

Syntax	***identificationID dispense(JxfsDispenseRequest dispenseRequest) throws JxfsException;***
Remarks	Dispenses the amount of money which is specified by the JxfsDenomination. The cash is dispensed at the side specified with the *position* property.

Parameter

Type	**Name**	**Description**
JxfsDispenseRequest	dispenseRequest	Contains all parameter used for dispensing cash.

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *dispense* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_DISPENSE
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsDispenseOrder*** object Amongst other information, this carries a JxfsDenomination property. If a successful immediate dispense, or an error occurs, then this will return details of the actual cash dispensed. If the dispense is delayed (JXFS_E_CDR_DELAYED_DISPENSE result is returned by the event), then this will return details of the cash that will be dispensed following a successful call for the dispense order to the dispenseExec method. If the dispense is delayed, then the when property of the JxfsDispenseOrder will be set to the time from which the delay is started, and the delay property will give the total delay time in ms. When the operation is canceled during a partial dispense, the returned JxfsDispenseOrder contains the total amount of cash dispensed before cancel occurred.

see section 9.2 for more details

JxfsIntermediateEvent

JXFS_I_CDR_PARTIAL_DISPENSE

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DELAYED_DISPENSE
JXFS_S_CDR_DELAYED_ORDER_CHANGED
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED
JXFS_S_CDR_TRANSPORT_CHANGED

3.5.3 dispenseExec

Syntax ***identificationID dispenseExec(JxfsDispenseOrder dispenseOrder) throws JxfsException;***

Remarks Accepts an order, which should be ready for dispense.

Parameter

Type	**Name**	**Description**
JxfsDispenseOrder	dispenseOrder	Contains all parameter used for dispensing cash.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *dispenseExec* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_DISPENSE_EXEC
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsDispenseOrder*** object Amongst other information, this carries a JxfsDenomination property. If a successful dispense, or an error occurs, then this will return details of the actual cash dispensed. When the operation is canceled during a partial dispense, the returned JxfsDispenseOrder contains the total amount of cash dispensed before cancel occurred.

see section 9.2 for more details

JxfsIntermediateEvent

JXFS_I_CDR_PARTIAL_DISPENSE

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DELAYED_ORDER_CHANGED
JXFS_S_CDR_DELAYED_ORDER_REMOVED
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED
JXFS_S_CDR_TRANSPORT_CHANGED

3.5.4 startExchange

Syntax	***identificationID startExchange(Vector units) throws JxfsException;***
Remarks	Used to start the exchange of cash units. No other method calls than ***endExchang***e, ***close*** or a *get**Property*** may be performed.

Parameter

Type	**Name**	**Description**
Vector of Integer	units	Vector of Integer which specify the logical cash units to exchange.

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *startExchange* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_START_EXCHANGE
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsCashUnit*** object

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.5.5 endExchange

Syntax ***identificationID endExchange(JxfsCashUnit cashUnit) throws JxfsException;***

Remarks Set actual Cash Unit and put dispenser back into an operational state. It will now accept regular method calls.

Parameter

Type	Name	Description
JxfsCashUnit	cashUnit	Update information for the cash units.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When an *endExchange* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	Value
operationID	JXFS_O_CDR_END_EXCHANGE
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsCashUnit*** object Updated cash units.

JxfsStatusEvent

JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_CONFIGURATION_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED

3.5.6 openSafeDoor

Syntax ***identificationID openSafeDoor() throws JxfsException***

Remarks This command controls the time lock for the safe door. It sends the currently configured value for the safe door timer to the device. This configuration parameter is vendor dependent.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When an *openSafeDoor* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_OPEN_SAFE_DOOR
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	none

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_SAFE_DOOR_CHANGED

3.5.7 calibrateCashUnit

Syntax

identificationID calibrateCashUnit(JxfsCalibrateItem calibrateItem) throws JxfsException;

Remarks

This command is used to initialize the reference value of a cash unit. It will action a vendor dependent sequence of hardware events which will calibrate the physical cash unit. This is necessary if a new type of bank note is put into the cash unit. By this command the cash unit gets the new measures of the bank notes.

Parameter

Type	**Name**	**Description**
JxfsCalibrateItem	calibrateItem	CalibrateItem to use.

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *calibrateCashUnit* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_CALIBRATE_CASH_UNIT
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***java.util.Vector*** object Updated CalibrateItems.

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_TRANSPORT_CHANGED

3.5.8 getDateTime

Syntax	***identificationID getDateTime() throws JxfsException;***
Remarks	Get device date and time.
Events	Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *getDateTime* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_GET_DATE_TIME
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	*java.util.**Date*** object Current date and time of device.

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.5.9 setDateTime

Syntax	***identificationID setDateTime(Date date) throws JxfsException;***
Remarks	Set device date and time. More and more devices were equipped with computer systems that have their own real time clock. The usage of this command is to synchronize this internal device clock with other systems.

Parameter

Type	**Name**	**Description**
java.util.Date		

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *setDateTime* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_SET_DATE_TIME
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	*java.util.**Date*** object Previous date and time of device.

JxfsStatusEvent

JXFS_S_CDR_DATE_TIME_CHANGED
JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.5.10 queryOrder

Syntax ***identificationID queryOrder(int orderType) throws JxfsException;***

Remarks This method is used to retrieve four different lists of dispense orders.

Parameter

Type	**Name**	**Description**
int	orderType	specifies the list to retrieve.

Value	**Description**
JXFS_C_CDR_DO_DISPENSABLE	Orders ready for processing.
JXFS_C_CDR_DO_DELAYED	All orders in delay queue.
JXFS_C_CDR_DO_LAQ	All orders in Large Amount Queue.
JXFS_C_CDR_DO_ALL	All orders in all queues.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *queryOrder* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_QUERY_ORDER
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***Vector of JxfsDispenseOrder*** objects Vector of selected Orders.

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.5.11 removeOrder

Syntax ***identificationID removeOrder(JxfsDispenseOrder dispenseOrder) throws JxfsException;***

Remarks This method is used to remove a dispense order from the lists of dispense orders.

Parameter

Type	Name	Description
JxfsDispenseOrder	dispenseOrder	specifies the dispenseOrder to remove from one of the queues: LAQ, Dispensable or Delayed.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *removeOrder* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	Value
operationID	JXFS_O_CDR_REMOVE_ORDER
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsDispenseOrder*** object Removed Order.

JxfsStatusEvent

JXFS_S_CDR_DELAYED_ORDER_REMOVED
JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.5.12 queryCashUnit

Syntax	***identificationID queryCashUnit() throws JxfsException;***
Remarks	Retrieve the current cash units.
Events	Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *queryCashUnit* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_QUERY_CASHUNIT
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsCashUnit*** object Current cash units.

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.5.13 updateCashUnit

Syntax ***identificationID updateCashUnit(JxfsCashUnit cashUnit) throws JxfsException;***

Remarks Replace current cash units. When calling this method it is important that the application fill in the whole structure including all *JxfsLogicalCashUnits* and *JxfsPhysicalCashUnits*.

Parameter

Type	**Name**	**Description**
JxfsCashUnit	cashUnit	unit of device.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When an *updateCashUnit* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_UPDATE_CASHUNIT
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsCashUnit*** object Current cash units.

JxfsStatusEvent

JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_CONFIGURATION_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.5.14 reset

Syntax	***identificationID reset() throws JxfsException;***
Remarks	This method is used to reset the device and put it into a defined operational state.
Events	Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *reset* operation is completed , this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_RESET
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	none

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_CONFIGURATION_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED
JXFS_S_CDR_SAFE_DOOR_CHANGED
JXFS_S_CDR_TRANSPORT_CHANGED

3.6 IJxfsCashRecyclerControl Summary

Extends	Implements
IJxfsBaseControl	

Method	Return
cashInStart	identificationID
cashIn	identificationID
cashInEnd	identificationID
cashInRollback	identificationID
empty	identificationID
querySignatures	identificationID
queryDenominations	identificationID
updateDenominations	identificationID

3.7 IJxfsCashRecyclerControl Methods

Following methods are specific to Recycler devices.

3.7.1 cashInStart - deprecated

Syntax ***identificationID cashInStart(int position) throws JxfsException;***

Remarks Each cash in procedure has to be handled as a transaction that can be rolled back, if a difference occurs between the amount counted by the device and the amount the teller inserted. This command is used to start the cash in transaction at the defined input position.

Parameter

Type	**Name**	**Description**
int	Position	Input position used during *cashIn*. For position codes see output position codes description in Constants section.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *cashInStart* operation is completed , this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_CASH_IN_START
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	None

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.7.2 cashInStart

Syntax ***identificationID cashInStart(int position, boolean trustedUser) throws JxfsException;***

Remarks Each cash in procedure has to be handled as a transaction that can be rolled back in any case if a difference occurs between the amount counted by the device and the amount the teller inserted. This command is used to start the cash in transaction at the defined input position.

If the device does not support the "trusted user mode" and the *trustedUser* parameter is set to *true*, a *JxfsException* with the error code JXFS_E_NOT_SUPPORTED is thrown.
This method deletes the signatures from internal data structures of the device service.

Parameter

Type	**Name**	**Description**
int	position	Input position used during *cashIn*. For position codes see output position codes description in Constants section.
boolean	trustedUser	If set to *true*, it specifies that this operation is performed by a trusted user. That means that category 2 and / or 3 banknotes (according to European article 6 regulations) detected during cash deposit operations within this transaction should be treated as not recognized. The device should dispense them at its reject slot instead of retracting them.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *cashInStart* operation is completed, a*JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_CASH_IN_START
identificationID	identificationID returned by method.
result	
	JXFS_RC_SUCCESSFUL
	JXFS_E_CDR_RESET_REQUIRED
	JXFS_E_CDR_CASH_UNIT_ERROR
	JXFS_E_CDR_EXCHANGE_ACTIVE
	JXFS_E_CDR_CASHIN_ACTIVE
data	None

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.7.3 cashIn

Syntax ***identificationID cashIn(JxfsCashInOrder order) throws JxfsException***

Remarks Accept cash from the input slot.
This command transports notes from the cashin slot to the cashin module. The notes may pass through the banknote reader for identification. Failure to identify notes does not mean that the command has failed - even if the banknote reader rejects some or all of the notes, the command may return JXFS_RC_SUCCESSFUL. In this case a JXFS_I_CDR_INPUT_REFUSED intermediate event will be sent to listeners.
If the device has an escrow then this command will cause inserted notes to be moved there. Notes will be held on the escrow until the current Cash-In Transaction is either cancelled by cashInRollback or confirmed by cashInEnd commands. If there is no escrow then this command will move notes directly to the cash units.

Parameter

Type	Name	Meaning
JxfsCashInOrder	order	Specifies the notes or coins to accept.

Events Additional events can be generated

JxfsOperationCompleteEvent

When a *cashIn* operation is completed a *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	Value
OperationID	JXFS_O_CDR_CASH_IN
IdentificationID	The corresponding ID
Result	JXFS_RC_SUCCESSFUL JXFS_E_CDR_RESET_REQUIRED JXFS_E_CDR_CASH_UNIT_ERROR JXFS_E_CDR_EXCHANGE_ACTIVE JXFS_E_CDR_INVALID_CURRENCY JXFS_E_CDR_INVALID_DENOMINATION JXFS_E_CDR_NO_CASHIN_STARTED JXFS_E_CDR_TOO_MANY_BILLS JXFS_E_CDR_TOO_MANY_COINS JXFS_E_CDR_NO_BILLS JXFS_E_CDR_NO_COINS JXFS_E_CDR_INVALID_BILL JXFS_E_CDR_INVALID_COIN JXFS_E_CDR_INPUT_REFUSED JXFS_E_CDR_OUTPUT_NOT_EMPTY
Data	If the eurArt6Capability capability is set to TRUE then this field will contain a ***JxfsArt6CashInOrder*** object with the appropriate information. Otherwise a ***JxfsCashInOrder*** object will be returned.

JxfsStatusEvent

Note: If there are only category 1 banknotes, then they are returned immediately to the teller/customer and are not stored on the escrow.

Therefore the cash unit status is not changed, and the JXFS_S_CDR_CASHUNIT_CHANGED JxfsStatusEvent is not sent.

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED
JXFS_S_CDR_TRANSPORT_CHANGED

JxfsIntermediateEvent

When a category 2 or category 3 banknote is detected, it generates a *JxfsIntermediateEvent*. A single *JxfsIntermediateEvent* is sent per *cashIn* operation.
JxfsIntermediateEvent events are sent by CDR Device Control to all registered IntermediateListeners.
This *JxfsIntermediateEvent* is generated only when the ***eurArt6Capability*** capability is set to TRUE

Field	**Value**
OperationID	JXFS_O_CDR_CASH_IN
IdentificationID	The corresponding ID.
reason:	JXFS_I_CDR_INPUT_EURART6 At least one category 2 or one category 3 banknote has been detected.
Data	None; the information will be contained in ***JxfsArt6CashInOrder*** of the JxfsOperationCompleteEvent .

JxfsIntermediateEvent

When a deposited banknote is detected as category 1, it generates a *JxfsIntermediateEvent*. A single *JxfsIntermediateEvent* is sent per *cashIn* operation.
JxfsIntermediateEvent events are sent by CDR Device Control to all registered IntermediateListeners.

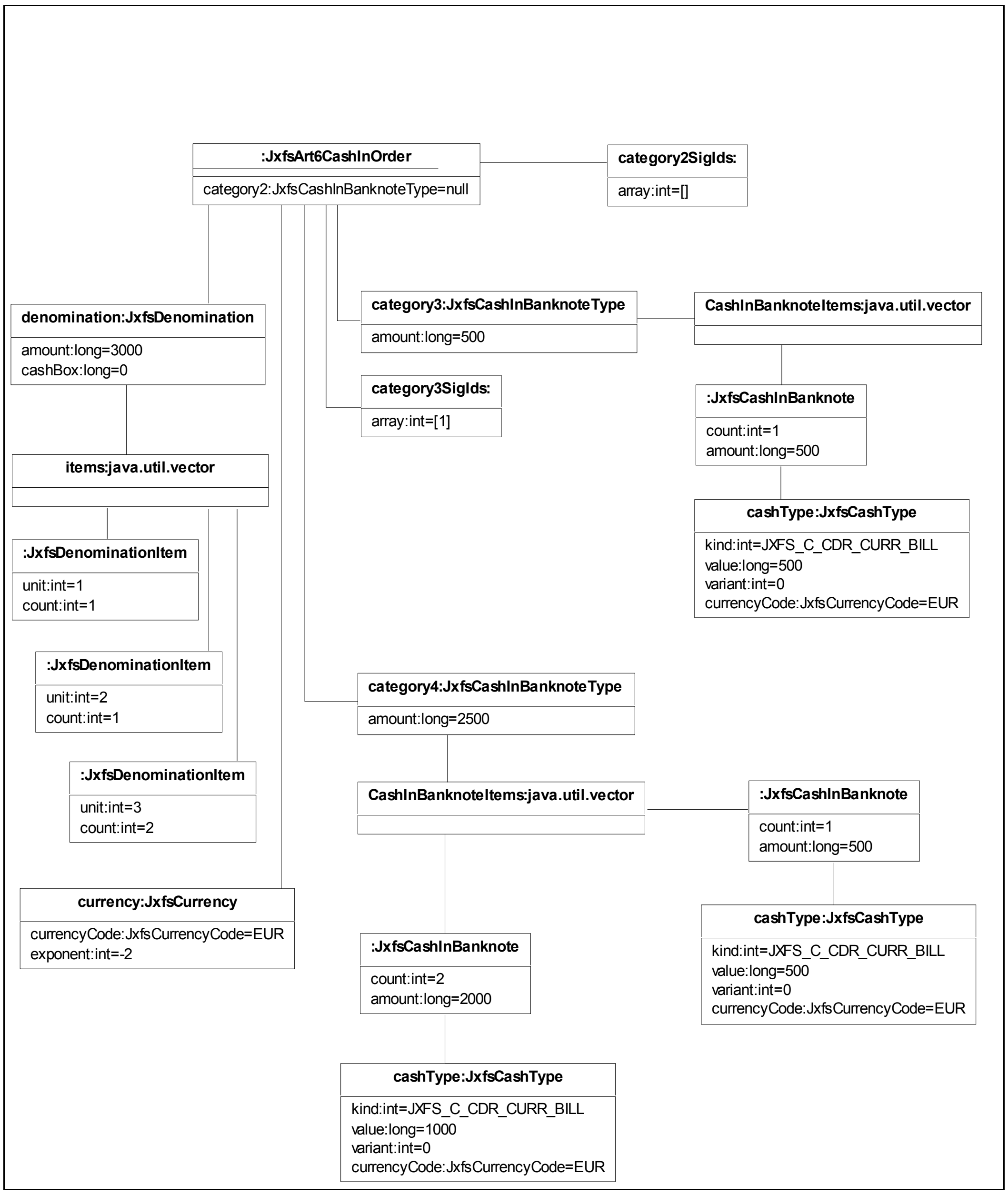
Field	**Value**
OperationID	JXFS_O_CDR_CASH_IN
IdentificationID	The operation's Identification Id.
reason:	JXFS_I_CDR_INPUT_REFUSED At least one banknote was not recognized and returned to the reject slot.
Data	Always null. Category 1 banknotes are returned immediately to the teller/customer.

3.7.3.1 Example

For the example below, it is assumed that the following bank notes have been put into the device:

- one US dollar bank note (category 1 as the BIM does not know anything about dollars)
- two 5 € bank notes (one category 3 and one category 4 bank note)
- two 10 € bank notes (category 4)

Then the following data structure is returned as the result of the cashIn operation:


:JxfsArt6CashInOrder
category2:JxfsCashInBanknoteType=null
category2SigIds:
array:int=[]
denomination:JxfsDenomination
amount:long=3000
cashBox:long=0
category3:JxfsCashInBanknoteType
amount:long=500
CashInBanknoteItems:java.util.vector
category3SigIds:
array:int=[1]
:JxfsCashInBanknote
count:int=1
amount:long=500
items:java.util.vector
cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=500
variant:int=0
currencyCode:JxfsCurrencyCode=EUR
:JxfsDenominationItem
unit:int=1
count:int=1
:JxfsDenominationItem
unit:int=2
count:int=1
category4:JxfsCashInBanknoteType
amount:long=2500
:JxfsDenominationItem
unit:int=3
count:int=2
CashInBanknoteItems:java.util.vector
:JxfsCashInBanknote
count:int=1
amount:long=500
currency:JxfsCurrency
currencyCode:JxfsCurrencyCode=EUR
exponent:int=-2
cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=500
variant:int=0
currencyCode:JxfsCurrencyCode=EUR
:JxfsCashInBanknote
count:int=2
amount:long=2000
cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=1000
variant:int=0
currencyCode:JxfsCurrencyCode=EUR

3.7.4 cashInEnd

Syntax	***identificationID cashInEnd( ) throws JxfsException;***
Remarks	Each cash in procedure has to be handled as a transaction that can be rolled back if a difference occurs between the amount counted by the device and the amount the teller / customer inserted. This command is used to end the cash in transaction. If the device has an escrow then this command will move the notes from the escrow to the cash in units. If the European article 6 regulations are applicable, then the category 2 and 3 notes must be transported to the appropriate area, with the following exception: if the "trusted usermode" is set then all the category 2 and category 3 notes are returned to the customer/teller, category 4 notes are transported to the appropriate cashin units. If there are no notes in the escrow an error code JXFS_E_CDR_NO_BILLS is returned on the **JxfsOperationCompleteEvent** Event and the cashin operation is completed.
Events	Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *cashInEnd* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_CASH_IN_END
identificationID	identificationID returned by method.
Result	Common or device dependent error code. (See section on Error codes).
Data	***JxfsCashInOrder*** object Total amount and Denomination cashed in since *cashInStart*.

JxfsStatusEvent

JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED

3.7.5 cashInRollback

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

Remarks Each cash in procedure has to be handled as a transaction that can be rolled back if a difference occurs between the amount counted by the device and the amount the teller/customer inserted.
If the device has a cash-in escrow then this command is used to rollback the notes that are in the escrow to the teller/customer. If there are no notes in the escrow an error code JXFS_E_CDR_NO_BILLS is returned on the **JxfsOperationCompleteEvent** event and the cashInRollback operation is completed.
If the European article 6 regulations **are not applicable**, then all the notes cashed in since the last *cashInStart* command are returned to the teller / customer,
If the European article 6 regulations **are applicable**, only category 4 notes are returned to the customer/teller; with the following exception: If the "trusted user mode" is set then **all the notes** are returned to the customer/teller

It is assumed that the category 1 notes are returned immediately to the teller/ customer and are not stored in the escrow.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *cashInRollback* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_CASH_IN_ROLLBACK
identificationID	The corresponding ID.
result	
	JXFS_RC_SUCCESSFUL
	JXFS_E_CDR_RESET_REQUIRED
	JXFS_E_CDR_CASH_UNIT_ERROR
	JXFS_E_CDR_NO_CASHIN_STARTED
	JXFS_E_CDR_NO_BILLS
	JXFS_E_CDR_NO_COINS
	JXFS_E_CDR_OUTPUT_NOT_EMPTY
data	***JxfsCashInOrder*** object This represents the amount of cash that is returned by this action.

see section 9.4 for more details

JxfsIntermediateEvent

JXFS_I_CDR_PARTIAL_DISPENSE

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED

JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED
JXFS_S_CDR_TRANSPORT_CHANGED

3.7.6 empty - deprecated

Syntax ***identificationID empty(JxfsDispenseRequest dispenseRequest) throws JxfsException;***

Remarks This method is used to empty the cash dispenser of a particular Denomination of bills.

Parameter

Type	**Name**	**Description**
JxfsDispenseRequest	dispenseRequest	Contains all parameter used to empty the device.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When an *empty* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_EMPTY
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsDispenseOrder*** object Dispensed cash. When the operation is canceled during a partial dispense, the returned JxfsDispenseOrder contains the total amount of cash dispensed before cancel occurred.

JxfsIntermediateEvent

JXFS_I_CDR_PARTIAL_DISPENSE

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DELAYED_DISPENSE
JXFS_S_CDR_DELAYED_ORDER_CHANGED
JXFS_S_CDR_DELAYED_ORDER_REMOVED
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED
JXFS_S_CDR_TRANSPORT_CHANGED

3.7.7 empty

Syntax	***identificationID empty(java.util.Vector names) throws JxfsException;***
Remarks	This method is used to empty one or more physical cash units of the dispenser.

Parameter

Type	**Name**	**Description**
java.util.Vector	names	A vector of Strings containing the name attribute of the physical cash units to empty.

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

When an empty operation is completed, this JxfsOperationCompleteEvent is sent to all registered listeners with following data:

Field	**Value**
OperationID	JXFS_O_CDR_EMPTY
IdentificationID	IdentificationID returned by method.
Result	Common or device dependent error code. (See section on Error codes).
Data	JxfsDispenseOrder object Dispensed cash. When the operation is canceled during a partial dispense, the returned JxfsDispenseOrder contains the total amount of cash dispensed before cancel occurred.

JxfsIntermediateEvent

JXFS_I_CDR_PARTIAL_DISPENSE

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DELAYED_DISPENSE
JXFS_S_CDR_DELAYED_ORDER_CHANGED
JXFS_S_CDR_DELAYED_ORDER_REMOVED
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED
JXFS_S_CDR_TRANSPORT_CHANGED

3.7.8 querySignatures

Syntax ***identificationID querySignatures(int[] signatureIds) throws JxfsException;***

Remarks This method queries category 2 and 3 banknote signatures for given signature identification numbers.
This operation succeeds if and only if signatures for all identification numbers specified by the *signatureIds* parameter are available.
The signatures are stored by the Device Service in persistent mode in such a way that they may be recovered after application, Device Service or power failure or system restart. The signatures are deleted from internal data structures of the device service by the cashInStart method.

If there are no signatures available for one of the given *signatureIds* the code JXFS_E_CDR_INVALID_SIGNATURE_ID is returned on the **JxfsOperationCompleteEvent**.

Parameter

Type	**Name**	**Description**
int[]	signatureIds	List of signature identification numbers. One should use numbers contained in *category 2* and *category 3 SigIds* properties *JxfsArt6CashInOrder* objects returned by cashin command.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *querySignatures* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with the following data:

Field	**Value**
operationID	JXFS_O_CDR_QUERY_SIGNATURES
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***java.util.Hashtable*** object This associative map contains signature identification numbers (represented by *java.lang.Integer* objects) as keys and signature information (represented by a *byte[]* objects) as values.

3.7.9 queryDenominations

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

Remarks This method is used to query information about denominations supported by the device. In the JxfsOperationCompleteEvent event, it returns a vector of denominations with their current settings. Each element of the returned vector is an object of type JxfsDenominationInfo, which contains information on the settings of the validation unit for the denomination.

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *queryDenominations* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with the following data:

Field	**Value**
operationID	JXFS_O_CDR_QUERY_DENOMINATIONS
identificationID	identificationID returned by method.
Result	Common or device dependent error code. (See section on Error codes).
Data	***java.util.Vector*** object A vector of JxfsDenominationInfo, one for each different denomination supported by the device.

3.7.10 updateDenominations

Syntax

identificationID updateDenominations(Vector denomInfo) throws JxfsException;

Remarks

This method is used to update the settings for a list of denominations. For each JxfsDenominationInfo element of the vector, the application can update its validation settings (if the device is a cash recycler).

Parameter

Type	**Name**	**Description**
java.util.Vector	denomInfo	A vector of JxfsDenominationInfo objects. This object should be a modified version of the one obtained from the queryDenominations call.

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *updateDenominations* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with the following data:

Field	**Value**
operationID	JXFS_O_CDR_UPDATE_DENOMINATIONS
identificationID	identificationID returned by method.
Result	Common or device dependent error code. (See section on Error codes).
Data	A vector of JxfsDenominationInfo objects. This object contains the list of updated denominations.

3.8 IJxfsATMControl Summary

Extends	Implements
IJxfsBaseControl	

Property	Type	Access	
retractArea	*JxfsRetractArea*	R	deprecated

Method	Return
present	identificationID
reject	identificationID
retract	identificationID
shutterMove	identificationID

3.9 IJxfsATMControl Methods

Following methods are specific to ATM devices.

3.9.1 present

Syntax	***identificationID present( ) throws JxfsException;***
Remarks	This command causes presentation of the cash. It can be used only following the ***dispense*** method. The command completes when the bills are positioned at the exit slot of the device. A status event is generated to report the user has removed the bills. If no event is received within a reasonable time period, the application should send a ***retract*** method to clear the bills from the exit. On devices which do not have the ability to detect when bills are taken the service event is generated as soon as the bills are available to the user.
Events	Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *present* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_PRESENT
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	none

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED

3.9.2 reject

Syntax ***identificationID reject(boolean present) throws JxfsException;***

Remarks Specifies if the rejected cash should be presented to the user at the position specified by the preceding ***dispens***e, ***dispenseExec*** or ***calibrateCashUnit*** method (present = true) or, whether the cash should be moved to the reject bin.

Parameter

Type	**Name**	**Description**
boolean	present	Specifies if the cash should be presented to user using the specified position (=true) or, if the money should only be transported to the stacker (=false).

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *reject* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_REJECT
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsCashUnit*** object

JxfsStatusEvent

JXFS_S_CDR_CASH_AVAILABLE
JXFS_S_CDR_CASH_TAKEN
JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED

3.9.3 retract

Syntax

identificationID retract(JxfsRetractArea retractArea) throws JxfsException;

Remarks

This command allows the application to force cash that has been presented to be retracted. Not all ATMs support this capability. This method may only be called following a ***dispense***,***dispenseExec or present*** method.

Parameter

Type	**Name**	**Description**
JxfsRetractArea	retractArea	Specifying the retract area to which the notes will be withdrawn.

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *retract* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_RETRACT
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	***JxfsCashUnit*** object

JxfsStatusEvent

JXFS_S_CDR_CASH_TRAY_CHANGED
JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_DISPENSER_STATUS_CHANGED
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED

3.9.4 shutterMove

Syntax ***identificationID shutterMove(boolean open, int position) throws JxfsException;***

Remarks This method allows the calling application to open and close the dispense shutter. The open parameter specifies in which direction the shutter should be moved. The position parameter determines for which dispense position the shutter is moved.

Parameter

Type	**Name**	**Description**
boolean	open	true – open shutter false – close shutter
int	position	Specifies the output position to which side to move.

Value	**Description**
JXFS_C_CDR_POS_NONE	No position selected
JXFS_C_CDR_POS_DEFAULT	Use configurated position
JXFS_C_CDR_POS_LEFT	Use left output side
JXFS_C_CDR_POS_CENTER	Use center output side
JXFS_C_CDR_POS_RIGHT	Use right output side
JXFS_C_CDR_POS_FRONT	Use front output side
JXFS_C_CDR_POS_REAR	Use rear output side
JXFS_C_CDR_POS_TOP	Use top output side
JXFS_C_CDR_POS_BOTTOM	Use bottom output side
JXFS_C_CDR_POS_REJECT	Use reject cassette

Events Events, which can be generated by this method.

JxfsOperationCompleteEvent

When a *shutterMove* operation is completed, this *JxfsOperationCompleteEvent* is sent to all registered listeners with following data:

Field	**Value**
operationID	JXFS_O_CDR_SHUTTER_MOVE
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on Error codes).
data	none

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_SHUTTER_CHANGED

4 Support Classes

4.1 Summary

Class	Description
JxfsArt6CashInOrder	Subclass of JxfsCashInOrder. Contains additional information regarding Article 6 handling.
JxfsCalibrateItem	Data used for initialization and calibration of cash units.
JxfsCapabilities	Contains the Capabilities of a cash dispenser.
JxfsCashInBanknote	Used by JxfsCashInBanknoteType to store Article 6 infomormation of deposited banknotes.
JxfsCashInBanknoteType	Contains Article 6 information about deposited banknotes.
JxfsCashInOrder	This class specifies all data required for cashIn operations.
JxfsCashType	Used to differentiate between bills and coins.
JxfsCashUnit	Information about the status and contents of the logical and physical cash units.
JxfsCurrency	Defines a Currency.
JxfsCurrenyCode	Contains a 3-character string detailing a currency code as defined by the ISO standard.
JxfsDelay	Used for OpenSafeDoor operation
JxfsDenomination	The JxfsDenomination holds a collection of JxfsDenominationItems that sum up to an amount of cash.
JxfsDenominationInfo	Stores the validation settings for a given denomination or cash type.
JxfsDenominationItem	A JxfsDenominationItem specifies a logical cash unit and the number of bills or coins that were dispensed from this unit or that should be deposited into this unit.
JxfsDispenseOrder	This class specifies all data required to perform a dispense operation.
JxfsDispenseRequest	This class specifies all data required for requesting a dispense or an empty operation.
JxfsEurArt6Capability	Denotes the capability of a device to handle the european article 6 rules.
JxfsLogicalCashUnit	Logical information about a cash unit.
JxfsMixEntry	Contains a reference to the logical cash unit and the number of bills/coins used in mixing.
JxfsMixInfo	Type for identifying mix algorithm and/or house mix tables.
JxfsMixItem	Specifies an amount used in a JxfsMixTable. The amount is expressed in MDU's.
JxfsMixTable	Contains complete description of one house mix table.
JxfsPhysicalCashUnit	Information about a physical cash unit.
JxfsRetractArea	Contains information about positions to be used during retract.
JxfsThreshold	Defines cassette thresholds.

4.2 Details

4.2.1 JxfsArt6CashInOrder

4.2.1.1 Usage

This class specifies data about deposited notes and their classification according to the European article 6 rules.
It is a subclass of the JxfscashInOrder
The information contained in this class are only relevant if the *eurArt6Capability* is set to true.

4.2.1.2 Summary

Extends	**Implements**
JxfsCashInOrder	

Property	**Type**	**Access**
category2	JxfsCashInBanknoteType	R
category2SigIds	int[]	R
category3	JxfsCashInBanknoteType	R
category3SigIds	int[]	R
category4	JxfsCashInBanknoteType	R

Constructors	**Parameter**	**Parameter-Type**
JxfsArt6CashInOrder	denomination	JxfsDenomination
	currency	JxfsCurrency
	category2	JxfsCashInBanknoteType
	category2SigIds	int[]
	category3	JxfsCashInBanknoteType
	category3 SigIds	int[ ]
	category4	JxfsCashInBanknoteType

Method	**Return**
get*Property*	*Property*

4.2.1.3 Properties

4.2.1.3.1 category2 (R)

Type	JxfsCashInBanknoteType
Remarks	Contains information about the deposited banknotes detected as category 2 banknotes.

4.2.1.3.2 category2SigIds (R)

Type	***int [ ]***
Remarks	Signature identification of category 2 banknotes. The array is empty, if no signatures are available.

4.2.1.3.3 category3 (R)

Type	JxfsCashInBanknoteType
Remarks	Contains information about the deposited banknotes detected as category 3 banknotes.

4.2.1.3.4 category3 SigIds(R)

Type	***int [ ]***
Remarks	Signature identification of category 3 banknotes. The array is empty, if no signatures are available.

4.2.1.3.5 category4 (R)

Type	JxfsCashInBanknoteType
Remarks	Contains information about the deposited banknotes detected as category 4 banknotes.

4.2.2 JxfsCalibrateItem

4.2.2.1 Usage

Data used for initialization and calibration of cash units. The vendor supplied service control is responsible for mapping from logical to physical cash units.

4.2.2.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
logicalNumber	*int*	RW
billsCount	*int*	RW
position	*int*	RW

Constructor	Parameter	Parameter-Type
JxfsCalibrateItem	*logicalNumber*	int
	billsCount	int
	position	int

Method	Return
get*Property*	*Property*
set*Property*	*void*

4.2.2.3 Properties

4.2.2.3.1 logicalNumber (RW)

Type	***int***
Remarks	This value specifies the number of the logical cash unit to be used during the initialization.

4.2.2.3.2 billsCount (RW)

Type	***int***
Remarks	On input this value specifies the number of bills to dispense.

4.2.2.3.3 position (RW)

Type	***int***
Remarks	Specifies the output position to dispense the note. (Defined as *dispense position code*).

4.2.3 JxfsCapabilities

4.2.3.1 Usage

Used to query the JxfsCapabilities of a cash dispenser, recycler and ATM.

4.2.3.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
autoPresent	*boolean*	R
cdType	*int*	R
eurArt6capability	*JxfsEurArt6Capability*	R
trustedUser	*boolean*	R
maxInBills	*int*	R
maxInCoins	*int*	R
maxOutBills	*int*	R
maxOutCoins	*int*	R
compound	*boolean*	R
shutterCmd	*boolean*	R
retract	*boolean*	R
safeDoorCmd	*boolean*	R
coins	*boolean*	R
cylinders	*boolean*	R
cashBox	*boolean*	R
refill	*boolean*	R
dispense	*boolean*	R
deposit	*boolean*	R
checkVandalism	*boolean*	R
intermediateStacker	*boolean*	R
billsTakenSensor	*boolean*	R
inputPositions	*int*	R
outputPositions	*int*	R
defaultInputPosition	*int*	R
defaultOutputPosition	*int*	R
silentAlarm	*boolean*	R
escrow	*boolean*	R
escrowSize	*int*	R
detector	*boolean*	R
baitTrap	*boolean*	R
vendorData	*java.lang.String*	R

Constructor	**Parameter**	**Parameter-Type**
JxfsCapabilities	*autoPresent*	boolean
	cdType	int
	eurArt6Capability	JxfsEurArt6Capability
	trustedUser	boolean
	maxInBills	int
	maxInCoins	int
	maxOutBills	int
	maxOutCoins	int
	compound	boolean
	shutterCmd	boolean
	retract	boolean
	safeDoorCmd	boolean
	coins	boolean
	cylinders	boolean
	cashBox	boolean
	refill	boolean
	dispense	boolean
	deposit	boolean
	checkVandalism	boolean
	intermediateStacker	boolean
	billsTakenSensor	boolean
	inputPositions	int
	outputPositions	int
	defaultInputPosition	int
	defaultOutputPosition	int
	silentAlarm	boolean
	escrow	boolean
	escrowSize	int
	detector	boolean
	baitTrap	boolean
	vendorData	java.lang.String

Method	**Return**
get*Property*	*Property*
is*Property*	*boolean*

4.2.3.3 Properties

4.2.3.3.1 autoPresent (R)

Type	***boolean***
Remarks	This specifies whether cash will be automatically presented to the user on execution of a dispense (autoPresent set to true), or whether the cash will only be transported to the stacker. In the latter case, a present command will need to be issued following the dispense (or following each part of a multi-partition dispense). If this property is set to true, then the shutterCmd capability will be false, as it would not be possible for the calling application to determine when it should open the dispense shutter, due to the possibility for a dispense to be delayed.

4.2.3.3.2 cdType (R)

Type	***int***
Remarks	Type of device. One of the following values: JXFS_C_CDR_TYPE_NONE JXFS_C_CDR_TYPE_DISPENSER JXFS_C_CDR_TYPE_RECYCLER JXFS_C_CDR_TYPE_ATM

4.2.3.3.3 eurArt6Capability (R)

Type	JxfsEurArt6Capability
Remarks	This specifies whether this cash recycler device is able to handle banknotes according to European article 6 regulations or not.

4.2.3.3.4 trustedUser (R)

Type	***boolean***
Remarks	If set to *true*, then this property specifies that the cash recycler is able to handle the special “trusted user” mode in *cashInEnd* and *cashInRollback* operations. This property makes sense only if the device supports the *European article 6*.

4.2.3.3.5 maxInBills (R)

Type	***int***
Remarks	Maximum number of bills to be accepted by one command.

4.2.3.3.6 maxInCoins (R)

Type	***int***
Remarks	Maximum number of coins to be accepted by one command.

4.2.3.3.7 maxOutBills (R)

Type	***int***
Remarks	Maximum number of bills to be dispensed by one command.

4.2.3.3.8 maxOutCoins (R)

Type	***int***
Remarks	Maximum number of coins to be dispensed by one command.

4.2.3.3.9 compound (R)

Type	***boolean***
Remarks	Is logical device part of compound physical device.

4.2.3.3.10 shutterCmd (R)

Type	***boolean***
Remarks	Defines, if the shutter be accessed by commands. If this property is set

to true, then the autoPresent capability will be false, as it would not be possible for the calling application to determine when it should open the dispense shutter, due to the possibility for a dispense to be delayed.

4.2.3.3.11 retract (R)

Type	***boolean***
Remarks	The cash dispenser can retract presented bills.

4.2.3.3.12 safeDoorCmd (R)

Type	***boolean***
Remarks	This device supports a safe door command.

4.2.3.3.13 coins (R)

Type	***boolean***
Remarks	The device includes a coin dispenser.

4.2.3.3.14 cylinders (R)

Type	***boolean***
Remarks	The coin dispenser can accept a number of coins per cylinder as input or only totals are allowed.

4.2.3.3.15 cashBox (R)

Type	***boolean***
Remarks	The service can handle a cash box.

4.2.3.3.16 refill (R)

Type	***boolean***
Remarks	Can the device be refilled by placing bills on the stack.

4.2.3.3.17 dispense (R)

Type	***boolean***
Remarks	The device can dispense cash.

4.2.3.3.18 deposit (R)

Type	***boolean***
Remarks	The device can deposit cash.

4.2.3.3.19 checkVandalism (R)

Type	***boolean***
Remarks	The device can detect vandalism.

4.2.3.3.20 intermediateStacker (R)

Type	***boolean***
Remarks	The device has a temporary storage before presenting bills.

4.2.3.3.21 billsTakenSensor (R)

Type	***boolean***
Remarks	The device has a bills taken sensor.

4.2.3.3.22 inputPositions (R)

Type	***int***
Remarks	Specifies the possible input positions to accept cash. (Defined as *dispense position code*s)

4.2.3.3.23 outputPositions (R)

Type	***int***
Remarks	Specifies the possible output positions to dispense cash. (Defined as *dispense position code*s)

4.2.3.3.24 defaultInputPosition (R)

Type	***int***
Remarks	Specifies the default input position to accept cash. (Defined as *dispense position code*)

4.2.3.3.25 defaultOutputPosition (R)

Type	***int***
Remarks	Specifies the default output position to dispense cash. (Defined as *dispense position code*)

4.2.3.3.26 silentAlarm (R)

Type	***boolean***
Remarks	The device supports a silent alarm feature.

4.2.3.3.27 escrow (R)

Type	***boolean***
Remarks	The device supports an escrow.

4.2.3.3.28 escrowSize (R)

Type	***int***
Remarks	Specifies the maximum number of bills on the escrow.

4.2.3.3.29 detector (R)

Type	***boolean***
Remarks	The device supports a detector to verify accepted cash.

4.2.3.3.30 baitTrap (R)

Type	***boolean***
Remarks	The device supports functionality to emit marked notes during dispense.

4.2.3.3.31 vendorData (R)

Type	***java.lang.String***
Remarks	Vendor specific data.

4.2.4 JxfsCashInBanknote

4.2.4.1 Usage

Used to query the information of the cashed in banknote.

4.2.4.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
cashType	JxfsCashType	R
count	int	R
amount	long	R

Constructor	Parameter	Parameter-Type
JxfsCashInBanknote	cashType	JxfsCashType
	count	long
	amount	long

Method	Return
get*Property*	*Property*

4.2.4.3 Properties

4.2.4.3.1 cashType (R)

Type	JxfsCashType
Remarks	Information about the note type. See the JxfsCashType class.

4.2.4.3.2 count (R)

Type	***int***
Remarks	Total number of this type of note and for this category cashed in.

4.2.4.3.3 amount (R)

Type	***long***
Remarks	Total amount of this type of note and for this category cashed in, expressed in MDUs.

4.2.5 JxfsCashInBanknoteType

4.2.5.1 Usage

This class contains information about the deposited banknote.

4.2.5.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
amount	long	R
cashInBanknoteItems	java.util.Vector of JxfsCashInBanknote	R

Constructor	Parameter	Parameter-Type
JxfsCashInBanknoteType	amount	long
	cashInBanknoteItems	java.util.Vector of JxfsCashInBanknote

Method	Return
get*Property*	*Property*

4.2.5.3 Properties

4.2.5.3.1 amount (R)

Type	***long***
Remarks	Total cashed in amount in this category expressed in MDUs.

4.2.5.3.2 cashInBanknoteItems (R)

Type	***java.util.Vector***
Remarks	Data information about the banknotes cashed in.

4.2.6 JxfsCashInOrder

4.2.6.1 Usage

This class specifies all data required for cash-in operations.

4.2.6.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
denomination	*JxfsDenomination*	RW
currency	*JxfsCurrency*	RW

Constructor	Parameter	Parameter-Type
JxfsCashInOrder	*denomination*	JxfsDenomination
	currency	JxfsCurrency

Method	Return
get*Property*	*Property*
set*Property*	*void*

4.2.6.3 Properties

4.2.6.3.1 denomination (RW)

Type	***JxfsDenomination***
Remarks	Specifies the amount to cash-in or the amount accepted.

4.2.6.3.2 currency (RW)

Type	***JxfsCurrency***
Remarks	Specifies the currency to use.

4.2.7 JxfsCashType

4.2.7.1 Usage

This class is used to carry all the information that is required to uniquely define a cash item (e.g.: a bank note or coin).

4.2.7.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
kind	*int*	R
currencyCode	*JxfsCurrencyCode*	R
value	*long*	R
variant	*int*	R

Constructor	Parameter	Parameter-Type
JxfsCashType	*kind*	int
	currencyCode	JxfsCurrencyCode
	value	int
	variant	int

Method	Return
get*Property*	*Property*

4.2.7.3 Properties

4.2.7.3.1 kind (R)

Type	***int***
Remarks	The type of the value, a note or a coin. One of the following values: JXFS_C_CDR_CURR_BILL JXFS_C_CDR_CURR_COIN

4.2.7.3.2 currencyCode (R)

Type	***JxfsCurrencyCode***
Remarks	Defines the currency code for this type of cash.

4.2.7.3.3 value (R)

Type	***long***
Remarks	Value of cash items expressed in MDUs.

4.2.7.3.4 variant (R)

Type	***int***
Remarks	The variant of the cash item represented. The constant JXFS_C_CDR_NO_VARIANT may be used to express that the variant information is not supported. Other values may be vendor specific.

4.2.8 JxfsCashUnit

4.2.8.1 Usage

Information about the status and contents of the logical and physical cash units. Each logical bill or coin type cash unit can be composed of one or more physical cash units. All counters are pure software counters. Due to this fact these values can differ from the actual physical cash counts.

4.2.8.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
rejectCount	*int*	RW

Constructor	Parameter	Parameter-Type
JxfsCashUnit	*rejectCount*	int

Method	Return
get*Property*	*Property*
set*Property*	*void*
addLogicalUnit	*boolean*
getLogicalUnits	*java.util.Vector*

4.2.8.3 Properties

4.2.8.3.1 rejectCount (RW)

Type	***int***
Remarks	Counter for all reject actions in the device.

4.2.8.4 Methods

4.2.8.4.1 addLogicalUnit

Syntax ***boolean addLogicalUnit(JxfsLogicalCashUnit logicalCashUnit)***
Remarks Add a logical cash unit.

Parameter	**Type**	**Name**	**Description**
	JxfsLogicalCashUnit	logicalCashUnit	Add a logical cash unit to the internal list of cash units.

4.2.8.4.2 getLogicalUnits

Syntax	***java.util.Vector getLogicalUnits()***
Remarks	Returns vector of *JxfsLogicalCashUnit*.

4.2.9 JxfsCurrency

4.2.9.1 Usage

Objects of this class are used to define a supported currency. Each currency has a currency identifier (a three character code) and a currency exponent.

4.2.9.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
currencyCode	*JxfsCurrencyCode*	R
exponent	*int*	R

Constructor	**Parameter**	**Parameter-Type**
JxfsCurrency	*currencyCode*	JxfsCurrencyCode
	exponent	int

Method	**Return**
get*Property*	*Property*

4.2.9.3 Properties

4.2.9.3.1 currencyCode (R)

Type	***JxfsCurrencyCode***
Remarks	A 3-character length upper case string detailing a currency code as defined by the ISO standard, ISO 4217.

4.2.9.3.2 exponent (R)

Type	***int***
Remarks	JxfsCurrency exponent.

4.2.10 JxfsCurrencyCode

4.2.10.1 Usage

Used to specify the country specific code (3-character string) for a given currency.

4.2.10.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
currencyCode	*String*	R

Constructor	**Parameter**	**Parameter-Type**
JxfsCurrency	*currencyCode*	String

Method	**Return**
get*Property*	*Property*

4.2.10.3 Properties

4.2.10.3.1 currencyCode (R)

Type	***String***
Remarks	A 3-character length upper case string detailing a currency code as defined by the ISO standard, ISO 4217.

4.2.11 JxfsDelay

4.2.11.1 Usage

A JxfsDelay object stores the time the opening of the safedoor is delayed.

4.2.11.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
delay	*int*	R

Constructor	**Parameter**	**Parameter-Type**
JxfsDelay	*delay*	int

Method	**Return**
get*Property*	*Property*

4.2.11.3 Properties

4.2.11.3.1 delay (R)

Type	***int***
Remarks	Specifies the time to delay in milliseconds.

4.2.12 JxfsDenomination

4.2.12.1 Usage

The JxfsDenomination holds a collection of JxfsDenominationItems that sum up to an amount of cash.

4.2.12.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
items	*java.lang.Vector*	RW
amount	*long*	RW
cashBox	*long*	RW

Constructor	**Parameter**	**Parameter-Type**
JxfsDenomination	*items*	java.lang.Vector
	amount	long
	cashBox	long

Method	**Return**
get*Property*	*Property*
set*Property*	*void*
addItem	*boolean*

4.2.12.3 Properties

4.2.12.3.1 items (RW)

Type	***java.lang.Vector***
Remarks	A list of *JxfsDenominationItem*s.
Note for denominate	These items define the asset used for *denominate*.

4.2.12.3.2 amount (RW)

Type	***long***
Remarks	Amount expressed in MDUs.
Note for denominate	This is the amount to be denominated.

4.2.12.3.3 cashBox (RW)

Type	***long***
Remarks	Cashbox amount expressed in MDUs.
Note for denominate	On return of the *denominate*-operation, this defines an amount, that could not be denominated.

4.2.12.4 Methods

4.2.12.4.1 addItem

Syntax	***boolean addItem(JxfsDenominationItem item)***	
Remarks	Add a *JxfsDenominationItem* to this denomination.	
Parameter	**Type**	**Name**
	JxfsDenominationItem	item

4.2.13 JxfsDenominationInfo

4.2.13.1 Usage

The JxfsDenominationInfo object holds the validation settings for a given denomination or cash type.

4.2.13.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
cashType	*JxfsCashType*	R
enableDenomination	*boolean*	RW

Constructor	Parameter	Parameter-Type
JxfsDenominationInfo	*cashType*	JxfsCashType
	enableDenomination	boolean

Method	Return
get*Property*	*Property*
set*Property*	*void*
isProperty	*boolean*

4.2.13.3 Properties:

4.2.13.3.1 cashType (R)

Type	*JxfsCashType*
Remarks	Specifies the details of the denomination, which is being informed in this JxfsDenominationInfo structure.

4.2.13.3.2 enableDenomination (R/W)

Type	*boolean*
Remarks	Specifies if the denomination is enabled (accepted by the BIM) or not.

4.2.14 JxfsDenominationItem

4.2.14.1 Usage

A JxfsDenominationItem specifies a logical cash unit and the number of bills or coins that were dispensed from this unit or that should be deposited into this unit.

4.2.14.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
unit	*int*	R
count	*int*	R

Constructor	**Parameter**	**Parameter-Type**
JxfsDenominationItem	*unit*	int
	count	int

Method	**Return**
get*Property*	*Property*

4.2.14.3 Properties

4.2.14.3.1 unit (R)

Type	***int***
Remarks	Number of logical cash unit.

4.2.14.3.2 count (R)

Type	***int***
Remarks	Number of bills/coins to dispense/deposit.

4.2.15 JxfsDispenseOrder

4.2.15.1 Usage

This class specifies all data required for *dispense, dispenseExec, queryOrder* and *removeOrder* operations.

4.2.15.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
orderID	*int*	RW
queueID	*int*	RW
denomination	*JxfsDenomination*	RW
currency	*JxfsCurrency*	RW
when	*java.util.Date*	RW
delay	*long*	RW
position	*int*	RW

Constructor	Parameter	Parameter-Type
JxfsDispenseOrder	*orderID*	int
	queueID	int
	denomination	JxfsDenomination
	currency	JxfsCurrency
	when	java.util.Date
	delay	long
	position	int

Method	Return
get*Property*	*Property*
set*Property*	*void*

4.2.15.3 Properties

4.2.15.3.1 orderID (RW)

Type	***int***
Remarks	Used to identify a dispense order.

4.2.15.3.2 queueID (RW)

Type	***int***
Remarks	Specifies the queue the dispense order was inserted in. One of the following values: (UVV Delayed Order Queue codes) JXFS_C_CDR_DO_DELAYED JXFS_C_CDR_DO_DISPENSABLE JXFS_C_CDR_DO_LAQ JXFS_C_CDR_DO_NONE

4.2.15.3.3 denomination (RW)

Type	***JxfsDenomination***
Remarks	Specifies the amount of cash to dispense.

4.2.15.3.4 currency (RW)

Type	***JxfsCurrency***
Remarks	Specifies the currency to use.

4.2.15.3.5 when (RW)

Type	***java.util.Date***
Remarks	Time the operation was requested.

4.2.15.3.6 delay (RW)

Type	***long***
Remarks	Delay in ms from *whe*n. If *delay* equals 0, then the dispense order was processed immediately, else, if *delay* is greater 0, then the order is delayed for *delay* milliseconds.

4.2.15.3.7 position (RW)

Type	***int***
Remarks	Specifies the output position to use for presenting money. One of the following values: JXFS_C_CDR_POS_NONE JXFS_C_CDR_POS_DEFAULT JXFS_C_CDR_POS_LEFT JXFS_C_CDR_POS_CENTER JXFS_C_CDR_POS_RIGHT JXFS_C_CDR_POS_TOP JXFS_C_CDR_POS_BOTTOM JXFS_C_CDR_POS_FRONT JXFS_C_CDR_POS_REAR

4.2.16 JxfsDispenseRequest

4.2.16.1 Usage

This class specifies all data required for a *dispense* or an *empty* operation.

4.2.16.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
mixNumber	*int*	RW
denomination	*JxfsDenomination*	RW
currency	*JxfsCurrency*	RW
position	*int*	RW

Constructor	**Parameter**	**Parameter-Type**
JxfsDispenseRequest	*mixNumber*	int
	denomination	JxfsDenomination
	currency	JxfsCurrency
	position	int

Method	**Return**
get*Property*	*Property*
set*Property*	*void*

4.2.16.3 Properties

4.2.16.3.1 mixNumber (RW)

Type	***int***
Remarks	Specifies kind of mixing.

4.2.16.3.2 denomination (RW)

Type	***JxfsDenomination***
Remarks	Specifies the amount of cash to dispense.

4.2.16.3.3 currency (RW)

Type	***JxfsCurrency***
Remarks	Specifies the currency to use.

4.2.16.3.4 position (RW)

Type	***int***
Remarks	Specifies the output position to use for presenting money. Same values as in *JxfsDispenseOrder*

4.2.17 JxfsEurArt6Capability

4.2.17.1 Usage

Used to query the capability of the device to handle the european article 6 rules.

4.2.17.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
category2	boolean	R
category3	boolean	R
unfit	boolean	R

Constructor	**Parameter**	**Parameter-Type**
JxfsEurArt6Capability	category2	boolean
	category3	boolean
	unfit	boolean

Method	**Return**
is*Property*	*boolean*

4.2.17.3 Properties

4.2.17.3.1 category2 (R)

Type	boolean
Remarks	Specifies if the cash recycler is able to sort category 2 notes and store them separately.

4.2.17.3.2 category3 (R)

Type	boolean
Remarks	Specifies if the cash recycler is able to sort category 3 notes and store them separately.

4.2.17.3.3 unfit (R)

Type	boolean
Remarks	Specifies if the cash recycler is able to sort unfit notes from category 3 notes and store them separately. The unfit notes are notes that are detected as genuine notes but due to the poor quality they are not allowed to be in circulation. European article 6 mandates to handle these notes as category3 notes.

4.2.18 JxfsLogicalCashUnit

4.2.18.1 Usage

Logical information about a cash unit. Each logical unit can be composed of multiple physical units.

4.2.18.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
cashType	*JxfsCashType*	RW
number	*int*	RW
cuKind	*int*	RW
cuType	*int*	RW
unitID	*java.lang.String*	RW
initialCount	*int*	RW
count	*int*	RW
threshold	*JxfsThreshold*	RW
appLock	*boolean*	RW
devLock	*boolean*	RW
status	*int*	RW
thresholdStatus	*JxfsThresholdStatus*	RW
physicalName	*java.lang.String*	RW
physicalUnits	*java.util.Vector*	RW
depositCount	*int*	RW
dispenseCount	*int*	RW
rejectCount	*int*	RW

Constructor	Parameter	Parameter-Type
JxfsLogicalCashUnit	*cashType*	JxfsCashType
	number	int
	cuKind	int
	cuType	int
	unitID	java.lang.String
	initialCount	int
	count	int
	threshold	JxfsThreshold
	appLock	boolean
	devLock	boolean
	status	int
	thresholdStatus	JxfsThresholdStatus
	physicalName	java.lang.String
	physicalUnits	java.util.Vector
	depositCount	int
	dispenseCount	int
	rejectCount	int

Method	Return
get*Property*	*Property*
set*Property*	*void*
is*Property*	*boolean*
addUnit	*boolean*

4.2.18.3 Properties

4.2.18.3.1 cashType (RW)

Type	***JxfsCashType***
Remarks	Defines the type of cash used by this cash unit.

4.2.18.3.2 number (RW)

Type	***int***
Remarks	Logical number of cash unit. Starting with a value of one (1) for the first cash unit. Incremented by one for the next units.

4.2.18.3.3 cuKind (RW)

Type	***int***
Remarks	Specifies, if cash unit can dispense, deposit cash or both. One of the following values: JXFS_C_CDR_LCU_NA JXFS_C_CDR_LCU_DEPOSIT JXFS_C_CDR_LCU_DISPENSE JXFS_C_CDR_LCU_RECYCLE

4.2.18.3.4 cuType (RW)

Type	***int***
Remarks	Type of cash unit. One of the following values: JXFS_C_CDR_LCU_BAIT_TRAP JXFS_C_CDR_LCU_BILL_CASSETTE JXFS_C_CDR_LCU_COIN_CYLINDER JXFS_C_CDR_LCU_COIN_DISPENSER JXFS_C_CDR_LCU_COUPON JXFS_C_CDR_LCU_CURRENCY_CASSETTE JXFS_C_CDR_LCU_DOCUMENT JXFS_C_CDR_LCU_ESCROW JXFS_C_CDR_LCU_NA JXFS_C_CDR_LCU_OVERFLOW_CASSETTE JXFS_C_CDR_LCU_REJECT_CASSETTE JXFS_C_CDR_LCU_RETRACT_CASSETTE

4.2.18.3.5 unitID (RW)

Type	***java.lang.String***
Remarks	Identification value for a cash unit.

4.2.18.3.6 initialCount (RW)

Type	***int***
Remarks	This property represents the sum of all counts in JxfsPhysicalCashUnits attached to this JxfsLogicalCashUnit. This value is persistent on power failure, open, close and system reset. It is set during *endExchange* and *updateCashUnit* and not modified during any other operation.

4.2.18.3.7 count (RW)

Type	***int***
Remarks	This property represents the sum of all count fields in JxfsPhysicalCashUnits attached to this JxfsLogicalCashUnit. This value is persistent on power failure, open, close and system reset. It is set during *endExchang*e and *updateCashUnit*. It will be adjusted by *dispense* or *deposit* actions.
Note	If this is a reject cassette, this value gives the number of rejected notes or coins. If this is a retract cassette, this value gives the numbers of retracted notes or coins.

4.2.18.3.8 threshold (RW)

Type	***JxfsThreshold***
Remarks	This property gives the total for all associated JxfsPhysicalCashUnits.

4.2.18.3.9 appLock (RW)

Type	***boolean***
Remarks	If set to TRUE, the cash unit is locked by the application and can not be used until unlocked by the application. If appLock is set for a logical cash unit, then it must also have been set for all containing physical cash units.

4.2.18.3.10devLock (RW)

Type	***boolean***
Remarks	If set to TRUE, the cash unit is locked by the device and can not be used until unlocked by the device service. If devLock is set for a logical cash unit, then it must also have been set for all containing physical cash units.

4.2.18.3.11status (RW)

Type	***int***
Remarks	Cash unit status. If all physical cash units are OK, the logical cash unit must also set this property to JXFS_C_CDR:LCU_OK. In all other cases the JxfsLogicalCashUnit.status should be set to the value with highest priority of the containing JxfsPhysicalCashUnit.status properties. One of the following values: JXFS_C_CDR_LCU_INOP JXFS_C_CDR_LCU_MISSING JXFS_C_CDR_LCU_NO_VALUE JXFS_C_CDR_LCU_NO_REF JXFS_C_CDR_LCU_NOT_DISPENSEABLE JXFS_C_CDR_LCU_OK JXFS_C_CDR_LCU_UNKNOWN

4.2.18.3.12thresholdStatus (RW)

Type	***JxfsThresholdStatus***
Remarks	Cash unit threshold status.

4.2.18.3.13physicalName (RW)

Type	***java.lang.String***
Remarks	Name of the physical location of the cash unit in the dispenser device. This field is only used when logical unit equals physical unit.

4.2.18.3.14physicalUnits (RW)

Type	***java.util.Vector***
Remarks	Return vector of *JxfsPhysicalCashUnit*.

4.2.18.3.15depositCount (RW)

Type	***int***
Remarks	Number of bills, that were deposited.

4.2.18.3.16dispenseCount (RW)

Type	***int***
Remarks	Number of bills, that were dispensed.

4.2.18.3.17rejectCount (RW)

Type	***int***
Remarks	This property contains the number of all reject operations.

4.2.18.4 Methods

4.2.18.4.1 addUnit

Syntax	***boolean addUnit(JxfsPhysicalCashUnit unit)***	
Remarks	Add a *JxfsPhysicalCashUnit* to this logical cash unit.	
Parameter	**Type**	**Name**
	JxfsPhysicalCashUnit	unit

4.2.19 JxfsMixEntry

4.2.19.1 Usage

One entry in a JxfsMixItem. It contains a reference to the logical cash unit and the number of bills/coins used in mixing.

4.2.19.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
lcu	*int*	R
count	*int*	R

Constructor	Parameter	Parameter-Type
JxfsMixEntry	*lcu*	int
	count	int

Method	Return
get*Property*	*Property*

4.2.19.3 Properties

4.2.19.3.1 lcu (R)

Type	***int***
Remarks	Number of logical cash unit.

4.2.19.3.2 count (R)

Type	***int***
Remarks	Number of bills or coins.

4.2.20 JxfsMixInfo

4.2.20.1 Usage

Type for identifying mix algorithms and/or house mix tables.

4.2.20.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
number	*int*	R
mixType	*int*	R
mixAlgorithmType	*int*	R
name	*java.lang.String*	R

Constructor	Parameter	Parameter-Type
JxfsMixInfo	*number*	int
	mixType	int
	mixAlgorithmType	int
	name	java.lang.String

Method	Return
get*Property*	*Property*

4.2.20.3 Properties

4.2.20.3.1 number (R)

Type	***int***
Remarks	Number of this mixtype item.

4.2.20.3.2 mixType (R)

Type	***int***
Remarks	Specifies that an algorithm or a mix table should be used. One of the following values: JXFS_C_CDR_MIX_ALGORITHM JXFS_C_CDR_MIX_TABLE JXFS_C_CDR_MIX_DENOM

4.2.20.3.3 mixAlgorithmType (R)

Type	***int***
Remarks	This selects the type of algorithm or mix table. One of the following values: JXFS_C_CDR_MXA_MIN_BILLS JXFS_C_CDR_MXA_EQUAL_EMPTY

4.2.20.3.4 name (R)

Type	***java.lang.String***
Remarks	Name of algorithm or mix table.

4.2.21 JxfsMixItem

4.2.21.1 Usage

Specifies an amount used in a JxfsMixTable (in Minimum Dispense Units, MDU). It also contains a list of entries that specify the logical cash units and the number of bills/coins used.

4.2.21.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
amount	*long*	RW
entries	*Vector*	RW

Constructor	**Parameter**	**Parameter-Type**
JxfsMixItem	*amount*	long
	entries	Vector

Method	**Return**
getProperty	*Property*
setProperty	*void*

4.2.21.3 Properties

4.2.21.3.1 amount (RW)

Type	***long***
Remarks	Amount used in the mix table in MDUs.

4.2.21.3.2 entries (RW)

Type	***Vector of JxfsMixEntry***
Remarks	List of *JxfsMixEntry*.

4.2.22 JxfsMixTable

4.2.22.1 Usage

Contains complete description of a mix table.

4.2.22.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
mixInfo	*JxfsMixInfo*	RW
items	*Vector*	RW

Constructor	**Parameter**	**Parameter-Type**
JxfsMixTable	*mixInfo*	JxfsMixInfo
	items	Vector

Method	**Return**
get*Property*	*Property*
set*Property*	*void*

4.2.22.3 Properties

4.2.22.3.1 mixInfo (RW)

Type	***JxfsMixInfo***
Remarks	Identification of mix table.

4.2.22.3.2 items (RW)

Type	***Vector of JxfsMixItem***
Remarks	Specifies amounts used in the JxfsMixTable.

4.2.23 JxfsPhysicalCashUnit

4.2.23.1 Usage

Information about a physical cash unit.

4.2.23.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
name	*java.lang.String*	R
unitID	*java.lang.String*	R
count	*int*	R
threshold	*JxfsThreshold*	R
status	*int*	R
thresholdStatus	*JxfsThresholdStatus*	R
lock	*boolean*	R

Constructor	Parameter	Parameter-Type
JxfsPhysicalCashUnit	*name*	java.lang.String
	unitID	java.lang.String
	count	int
	threshold	JxfsThreshold
	status	int
	thresholdStatus	JxfsThresholdStatus
	lock	boolean

Method	Return
get*Property*	*Property*
is*Property*	*boolean*

4.2.23.3 Properties

4.2.23.3.1 name (R)

Type	***java.lang.String***
Remarks	Name of the physical location in the dispenser device where this cash unit is installed.

4.2.23.3.2 unitID (R)

Type	***java.lang.String***
Remarks	Cash unit ID.

4.2.23.3.3 count (R)

Type	***int***
Remarks	Actual count of bills or coins in the physical cash unit.

4.2.23.3.4 threshold (R)

Type	***JxfsThreshold***
Remarks	This property specifies the threshold values for one cash unit.

4.2.23.3.5 status (R)

Type	***int***
Remarks	Status of the physical cash unit. May have the same range of values as LogicalCashUnit.status.

4.2.23.3.6 thresholdStatus (R)

Type	***JxfsThresholdStatus***
Remarks	Thresholdstatus of the physical cash unit.

4.2.23.3.7 lock (R)

Type	***boolean***
Remarks	Lock status of the physical cash unit. Can be used from application and device service. Usually used for hot swap of cassettes.

4.2.24 JxfsRetractArea

4.2.24.1 Usage

Information about areas where to retract cash items that may have been in customer access.

4.2.24.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
outputPosition	*int*	R
retractArea	*int*	R
logicalPosition	*int*	R

Constructor	**Parameter**	**Parameter-Type**
JxfsRetractArea	outputPosition	*int*
	retractArea	*int*
	logicalPosition	*int*

Method	**Return**
get*Property*	*Property*

4.2.24.3 Properties

4.2.24.3.1 outputPosition (R)

Type	*int*
Remarks	Specifies the output position from which to retract bills. One of the following values: JXFS_C_CDR_POS_NONE JXFS_C_CDR_POS_DEFAULT JXFS_C_CDR_POS_LEFT JXFS_C_CDR_POS_CENTER JXFS_C_CDR_POS_RIGHT JXFS_C_CDR_POS_TOP JXFS_C_CDR_POS_BOTTOM JXFS_C_CDR_POS_FRONT JXFS_C_CDR_POS_REAR

4.2.24.3.2 retractArea (R)

Type	*int*
Remarks	Specifies the area to which the bills are to be retracted. One of the following values: JXFS_C_CDR_RA_REJECT JXFS_C_CDR_RA_RETRACT JXFS_C_CDR_RA_STACKER JXFS_C_CDR_RA_TRANSPORT

4.2.24.3.3 logicalPosition (R)

Type	*int*
Remarks	If *retractArea* is set to JXFS_C_CDR_RA_RETRACT this field is the logical retract position inside the container into which cash is to be retracted, otherwise this field is ignored. Logical positions start with a value of one (1).

4.2.25 JxfsThreshold

4.2.25.1 Usage

Defines limits for cassettes.

4.2.25.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
full	*int*	R
high	*int*	R
low	*int*	R
empty	*int*	R

Constructor	Parameter	Parameter-Type
JxfsThreshold	*full*	int
	high	int
	low	int
	empty	int

Method	Return
get*Property*	*Property*

4.2.25.3 Properties

4.2.25.3.1 full (R)

Type	***int***
Remarks	Specifies the full level for the cash unit

4.2.25.3.2 high (R)

Type	***int***
Remarks	Specifies the high level for the cash unit.

4.2.25.3.3 low (R)

Type	***int***
Remarks	Specifies the low level for the cash unit.

4.2.25.3.4 empty (R)

Type	***int***
Remarks	Specifies the empty level for the cash unit.

5 Status Event Classes

If a device status changes one of the following classes is returned via a ***JxfsStatusEvent***. This *xxxStatus*-Class is passed with the *details* property of the ***JxfsStatusEvent***. Each *xxxStatus*-Class provides several methods to query the changed device status.

The status ***JxfsCdrStatus*** is an exception to this rule: it is only delivered on a ***getStatus()*** method call and can't be sent due to a status change.

5.1 Summary

Status Event	**Description**
JxfsCashTrayStatus	Status of cash tray.
JxfsCashUnitStatus	Current cashunit status.
JxfsCdrStatus	Collection of all device status.
JxfsDeviceStatus	Current device status.
JxfsDispenseOrderStatus	Current dispense order.
JxfsDispenserStatus	Status of dispenser.
JxfsIntermediateStackerStatus	Intermediate stacker status.
JxfsSafeDoorStatus	Safe door status.
JxfsShutterStatus	Status of shutter.
JxfsTransportStatus	Status of transport unit.
JxfsVandalismStatus	Vandalism attack status.

5.2 Details

5.2.1 JxfsCashTrayStatus

Extends	Implements
JxfsType	

Query	Return
isEmpty	*boolean*
isNotEmpty	*boolean*
isNotSupported	*boolean*
isUnknown	*boolean*

5.2.2 JxfsCashUnitStatus

Extends	Implements
JxfsType	

Query	Return
getCashUnit	*JxfsCashUnit*

5.2.3 JxfsCdrStatus

Extends	Implements
JxfsStatus	

Query	Return	
getCashTrayStatus	JxfsCashTrayStatus	
getCashUnitStatus	JxfsCashUnitStatus	
getDeviceStatus	JxfsDeviceStatus	
getDispenseOrderStatus	JxfsDispenseOrderStatus	
getDispenserStatus	JxfsDispenserStatus	
getIntermediateStackerStatus	JxfsIntermediateStackerStatus	
getPresentStatus	JxfsPresentStatus	deprecated
getSafeDoorStatus	JxfsSafeDoorStatus	
getShutterStatus	JxfsShutterStatus	
getTransportStatus	JxfsTransportStatus	
getVandalismStatus	JxfsVandalismStatus	

5.2.4 JxfsDeviceStatus

Extends	Implements
JxfsType	

Query	Return
isOnLine	*boolean*
isOffLine	*boolean*
isPowerOff	*boolean*
isBusy	*boolean*
isNoDevice	*boolean*
isUserError	*boolean*
isHardwareError	*boolean*

5.2.5 JxfsDispenseOrderStatus

Extends	Implements
JxfsType	

Query	Return
getDispenseOrder	*JxfsDispenseOrder*
getIdentificationID	*int*

5.2.6 JxfsDispenserStatus

Extends	Implements
JxfsType	

Query	Return
isOk	*boolean*
isJxfsCashUnitState	*boolean*
isJxfsCashUnitStop	*boolean*
isJxfsCashUnitUnknown	*boolean*

5.2.7 JxfsIntermediateStackerStatus

Extends	Implements
JxfsType	

Query	Return	
isEmpty	*boolean*	
isNotEmpty	*boolean*	*deprecated*
isUnknown	*boolean*	
isNotSupported	*boolean*	

5.2.8 JxfsSafeDoorStatus

Extends	Implements
JxfsType	

Query	Return
isNotSupported	*boolean*
isOpen	*boolean*
isClosed	*boolean*
isLocked	*boolean*
isUnknown	*boolean*
getDelay	*JxfsDelay*
getIdentificationID	*int*

Note:

Due to device characteristics status queries *isOpen() eq. true and isLocked() eq. true* are not possible at the same time, *while isClosed() eq. true and isLocked() eq true* are possible at the same time.

5.2.9 JxfsShutterStatus

Extends	Implements
JxfsType	

Query	Return
isClosed	*boolean*
isOpen	*boolean*
isJammed	*boolean*
isNotSupported	*boolean*
isUnknown	*boolean*

5.2.10 JxfsTransportStatus

Extends	Implements
JxfsType	

Query	Return
isOk	*boolean*
isInOp	*boolean*
isNotSupported	*boolean*
isUnknown	*boolean*

5.2.11 JxfsVandalismStatus

Extends	Implements
JxfsType	

Query	Return
isManipulation	*boolean*
isNotSupported	*boolean*

5.2.12 JxfsPresentStatus - deprecated

Extends	Implements
JxfsType	

Query	Return
isUnknown	*boolean*
isPresented	*boolean*

6 Events

6.1 Intermediate Events

6.1.1 Intermediate Event Code Summary and Description

Value	Description
JXFS_I_CDR_INPUT_EURART6	At least one category 2 or one category 3 banknote has been detected during a cashIn operation.
JXFS_I_CDR_INPUT_REFUSED	At least one banknote was not recognized during a cashIn operation and has been returned to the reject slot.
JXFS_I_CDR_PARTIAL_DISPENSE	A partial dispense occurred.

6.1.2 IJxfsCashDispenserControl Intermediate Events

Intermediate Events	reset	updateCashUnit	queryCashUnit	removeOrder	queryOrder	setDateTime	getDateTime	calibrateCashUnit	openSafeDoor	endexchange	startexchange	dispenseExec	dispense	denominate
JXFS_I_CDR_PARTIAL_DISPENSE												x	x	

6.1.3 IJxfsCashRecyclerControl Intermediate Events

Methods			updateDenomination	queryDenominations	querySignatures	empty	cashInRollback	cashInEnd	cashIn	cashInStart
Error Codes										
JXFS_I_CDR_INPUT_EURART6									x	
JXFS_I_CDR_INPUT_REFUSED									x	
JXFS_I_CDR_PARTIAL_DISPENSE						x	x			

6.1.4 Intermediate Event Details

6.1.4.1 JXFS_I_CDR_INPUT_EURART6

This intermediate event is sent, when a category2 or category 3 banknote is detected on a cashIn.

Field	Value
operationID	*operationID* of the method initiating this event
identificationID	*identificationID* of the method initiating this event.
reason	JXFS_I_CDR_INPUT_EURART6
data	Always null

6.1.4.2 JXFS_I_CDR_INPUT_REFUSED

This intermediate event is sent, when at least one banknote was not recognized and has been returned to the reject slot.

Field	Value
operationID	*operationID* of method initiating this event
identificationID	*identificationID* of method initiating this event
reason	JXFS_I_CDR_INPUT_REFUSED
data	Always null.

6.1.4.3 JXFS_I_CDR_PARTIAL_DISPENSE

This intermediate event is sent, when a partial dispense occurs.

Field	Value
operationID	*operationID* of method initiating this event
identificationID	*identificationID* of method initiating this event
reason	JXFS_I_CDR_PARTIAL_DISPENSE
data	***JxfsDispenseOrderStatus*** object Contains a dispense order, which is part of multiple dispenses.

6.2 Status Events

The following tables specify, which *JxfsStatusEvents* can be generated during a method call.

6.2.1 Status Event Code Summary and Description

Value	**Description**
JXFS_S_CDR_CASH_AVAILABLE	Cash is available at the device exit slot.
JXFS_S_CDR_CASH_TAKEN	Cash has been removed from the device exit slot.
JXFS_S_CDR_CASH_TRAY_CHANGED	Content of cash tray changed.
JXFS_S_CDR_CASHUNIT_CHANGED	Cashunit changed.
JXFS_S_CDR_CASHUNIT_CONFIGURATION_CHANGED	The cashunit configuration was changed.
JXFS_S_CDR_CASHUNIT_THRESHOLD	A cashunit threshold was changed.
JXFS_S_CDR_DATE_TIME_CHANGED	Date or time of device changed.
JXFS_S_CDR_DELAYED_DISPENSE	Dispense order delayed.
JXFS_S_CDR_DELAYED_ORDER_CHANGED	Status of delayed dispense order changed.
JXFS_S_CDR_DELAYED_ORDER_REMOVED	A dispense order has been removed from the list of orders.
JXFS_S_CDR_DEVICE_STATUS_CHANGED	Device status changed.
JXFS_S_CDR_DISPENSER_STATUS_CHANGED	Dispenser status changed.
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED	Content of intermediate stacker changed.
JXFS_S_CDR_MIXTABLE_CHANGED	Property mixTables has been changed.
JXFS_S_CDR_SAFE_DOOR_CHANGED	Status of safe door changed.
JXFS_S_CDR_SHUTTER_CHANGED	Shutter status has changed.
JXFS_S_CDR_TRANSPORT_CHANGED	Transport mechanism status changed.
JXFS_S_CDR_VANDALISM_CHANGED	Manipulation detected.

6.2.2 IJxfsCashDispenserControl Status Events

Status Events	reset	updateCashUnit	queryCashUnit	removeOrder	queryOrder	setDateTime	getDateTime	calibrateCashUnit	openSafeDoor	endexchange	startexchange	dispenseExec	dispense	denominate
JXFS_S_CDR_CASH_AVAILABLE	x							x				x	x	
JXFS_S_CDR_CASH_TAKEN	x							x				x	x	
JXFS_S_CDR_CASH_TRAY_CHANGED	x							x				x	x	
JXFS_S_CDR_CASHUNIT_CHANGED	x	x						x		x		x	x	
JXFS_S_CDR_CASHUNIT_CONFIGURATION_CHANGED	x							x	x	x				
JXFS_S_CDR_CASHUNIT_THRESHOLD	x	x						x		x		x	x	
JXFS_S_CDR_DATE_TIME_CHANGED						x								
JXFS_S_CDR_DELAYED_DISPENSE													x	
JXFS_S_CDR_DELAYED_ORDER_CHANGED													x	
JXFS_S_CDR_DELAYED_ORDER_REMOVED				x								x		
JXFS_S_CDR_DEVICE_STATUS_CHANGED	x	x	x	x	x	x	x	x	x	x	x	x	x	
JXFS_S_CDR_DISPENSER_STATUS_CHANGED	x									x		x	x	
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED	x											x	x	
JXFS_S_CDR_SAFE_DOOR_CHANGED	x								x					
JXFS_S_CDR_TRANSPORT_CHANGED	x							x				x	x	

6.2.3 IJxfsCashRecyclerControl Status Events

Methods / Error Codes			updateDenomination	queryDenominations	querySignatures	empty	cashInRollback	cashInEnd	cashIn	cashInStart
JXFS_S_CDR_CASH_AVAILABLE						x	x		x	
JXFS_S_CDR_CASH_TAKEN						x	x		x	
JXFS_S_CDR_CASH_TRAY_CHANGED						x	x		x	
JXFS_S_CDR_CASHUNIT_CHANGED						x	x	x	x	
JXFS_S_CDR_CASHUNIT_THRESHOLD						x	x	x	x	
JXFS_S_CDR_DELAYED_DISPENSE						x				
JXFS_S_CDR_DELAYED_ORDER_CHANGED						x				
JXFS_S_CDR_DELAYED_ORDER_REMOVED						x				
JXFS_S_CDR_DEVICE_STATUS_CHANGED						x	x	x	x	x
JXFS_S_CDR_DISPENSER_STATUS_CHANGED						x	x		x	
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED						x	x		x	
JXFS_S_CDR_TRANSPORT_CHANGED						x	x		x	

6.2.4 IJxfsATMControl Status Events

Methods / Status Events	shutterMove	retract	reject	present
JXFS_S_CDR_CASH_AVAILABLE			x	x
JXFS_S_CDR_CASH_TAKEN			x	x
JXFS_S_CDR_CASH_TRAY_CHANGED		x	x	x
JXFS_S_CDR_CASHUNIT_CHANGED		x	x	
JXFS_S_CDR_CASHUNIT_THRESHOLD		x	x	
JXFS_S_CDR_DEVICE_STATUS_CHANGED	x	x	x	x
JXFS_S_CDR_DISPENSER_STATUS_CHANGED		x	x	
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED		x	x	x
JXFS_S_CDR_SHUTTER_CHANGED	x			

6.2.5 Status Event Details

6.2.5.1 JXFS_S_CDR_CASH_AVAILABLE

This status event is sent, when cash is available at the exit-slot of the device.

Field	Value
status	JXFS_S_CDR_CASH_AVAILABLE
details	***JxfsDispenseOrderStatus*** object Contains a dispense order, which can be removed from the exit-slot of the device. Property identificationID is used to identify the issuer of the operation.

6.2.5.2 JXFS_S_CDR_CASH_TAKEN

This status event is sent, when cash is removed from the exit-slot of the device.

Field	Value
status	JXFS_S_CDR_CASH_TAKEN
details	***JxfsDispenseOrderStatus*** object Contains a dispense order, which was removed from the exit-slot of the device. Property identificationID is used to identify the issuer of the operation.

6.2.5.3 JXFS_S_CDR_CASH_TRAY_CHANGED

This status event is sent, when the status of the cash tray changes.

Field	Value
status	JXFS_S_CDR_CASH_TRAY_CHANGED
details	***JxfsCashTrayStatus*** object. Current cash tray status.

6.2.5.4 JXFS_S_CDR_CASHUNIT_CHANGED

This status event is sent, if the cashunit content changed.

Field	Value
status	JXFS_S_CDR_CASHUNIT_CHANGED
details	***JxfsCashUnitStatus*** object. Represents the updated cash units.

6.2.5.5 JXFS_S_CDR_CASHUNIT_CONFIGURATION_CHANGED

This status event is sent, if the cashunit configuration changed.

Field	**Value**
status	JXFS_S_CDR_CASHUNIT_CONFIGURATION_CHANGED
details	***JxfsCashUnitStatus*** object Represents the modified cash units.

6.2.5.6 JXFS_S_CDR_CASHUNIT_THRESHOLD

This status event is sent, if a threshold change occurred for one or more cassettes.

Field	**Value**
status	JXFS_S_CDR_CASHUNIT_THRESHOLD
details	***JxfsCashUnitStatus*** object Represents the modified cash units.

6.2.5.7 JXFS_S_CDR_DATE_TIME_CHANGED

This status event is sent, when date or time for a device was changed.

Field	**Value**
status	JXFS_S_CDR_DATE_TIME_CHANGED
details	***Date*** object Previous device date and time.

6.2.5.8 JXFS_S_CDR_DELAYED_DISPENSE

This status event is sent, if the dispense order is delayed for later dispense.

Field	**Value**
status	JXFS_S_CDR_DELAYED_DISPENSE
details	***JxfsDispenseOrderStatus*** object Specifies among other data the time to delay in ms.

6.2.5.9 JXFS_S_CDR_DELAYED_ORDER_CHANGED

This status event is sent, when the status of a dispense order changes. The state of the order can change from delayed to dispensable, or vice versa; or the order can be redelayed because of other dispenses meanwhile.

Field	Value
status	JXFS_S_CDR_DELAYED_ORDER_CHANGED
details	***JxfsDispenseOrderStatus*** object Contains dispense order with state changed.. Property identificationID is used to identify the issuer of the operation.

6.2.5.10 JXFS_S_CDR_DELAYED_ORDER_REMOVED

This status event is sent, when a dispense order was removed from the internal list of orders.

Field	Value
status	JXFS_S_CDR_DELAYED_ORDER_REMOVED
details	***JxfsDispenseOrderStatus*** object. Contains the order, which was removed, either by an explicit call to *removeOrder* or when the order was dispensed or is removed from the internal list because of other reasons

6.2.5.11 JXFS_S_CDR_DEVICE_STATUS_CHANGED

This status event is sent, when the device status changes.

Field	Value
status	JXFS_S_CDR_DEVICE_STATUS_CHANGED
details	***JxfsDeviceStatus*** object Contains information about current device status

6.2.5.12 JXFS_S_CDR_DISPENSER_STATUS_CHANGED

On changes of the dispenser status, this event is sent.

Field	Value
status	JXFS_S_CDR_DISPENSER_STATUS_CHANGED
details	***JxfsDispenserStatus*** object Current dispenser status.

6.2.5.13 JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED

This status event is sent, when the status of the stacker changes.

Field	Value
status	JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED
details	***JxfsIntermediateStackerStatus*** object Contains information about the intermediate stacker

6.2.5.14 JXFS_S_CDR_MIXTABLE_CHANGED

This status event is sent, when the mixTables were changed.

Field	Value
status	JXFS_S_CDR_MIXTABLE_CHANGED
details	***Vector of JxfsMixTable*** objects Updated property *mixTables*.

6.2.5.15 JXFS_S_CDR_SAFE_DOOR_CHANGED

If the safe-door is operated or its status changes, this event is sent.

Field	Value
status	JXFS_S_CDR_SAFE_DOOR_CHANGED
details	***JxfsSafeDoorStatus*** object Actual safe-door status. Contains the delay until the safe door can be opened or will be closed. (in ms)

6.2.5.16 JXFS_S_CDR_SHUTTER_CHANGED

This status event is sent, if the shutter status changed.

Field	Value
status	JXFS_S_CDR_SHUTTER_CHANGED
details	***JxfsShutterStatus*** object. New shutter status.

6.2.5.17 JXFS_S_CDR_TRANSPORT_CHANGED

This status event is sent, if the state of the transport mechanism changes.

Field	Value
status	JXFS_S_CDR_TRANSPORT_CHANGED
details	***JxfsTransportStatus*** object Current transport mechanism status.

6.2.5.18 JXFS_S_CDR_VANDALISM_CHANGED

This status event is sent, if the vandalism detector reports a manipulation.

Field	Value
status	JXFS_S_CDR_VANDALISM_CHANGED
details	***JxfsVandalismStatus*** object Current state of vandalism detector.

7 Codes

7.1 Operation Codes

Following codes specify the method, which generated a *JxfsOperationCompleteEvent*.

7.1.1 IJxfsCashDispenserControl

Value	Method
JXFS_O_CDR_DENOMINATE	*denominate*
JXFS_O_CDR_DISPENSE	*dispense*
JXFS_O_CDR_DISPENSE_EXEC	*dispenseExec*
JXFS_O_CDR_START_EXCHANGE	*startExchange*
JXFS_O_CDR_END_EXCHANGE	*endExchange*
JXFS_O_CDR_OPEN_SAFE_DOOR	*openSafeDoor*
JXFS_O_CDR_CALIBRATE_CASH_UNIT	*calibrateCashUnit*
JXFS_O_CDR_GET_DATE_TIME	*getDateTime*
JXFS_O_CDR_SET_DATE_TIME	*setDateTime*
JXFS_O_CDR_QUERY_ORDER	*queryOrder*
JXFS_O_CDR_REMOVE_ORDER	*removeOrder*
JXFS_O_CDR_QUERY_CASH_UNIT	*queryCashUnit*
JXFS_O_CDR_UPDATE_CASH_UNIT	*updateCashUnit*
JXFS_O_CDR_RESET	*reset*

7.1.2 IJxfsCashRecyclerControl

Value	Method
JXFS_O_CDR_CASH_IN_START	*cashInStart*
JXFS_O_CDR_CASH_IN	*cashIn*
JXFS_O_CDR_CASH_IN_END	*cashInEnd*
JXFS_O_CDR_CASH_IN_ROLLBACK	*cashInRollback*
JXFS_O_CDR_EMPTY	*empty*
JXFS_O_CDR_QUERY_SIGNATURES	*querySignatures*
JXFS_O_CDR_QUERY_DENOMINATION	*queryDenominations*
JXFS_O_CDR_UPDATE_DENOMINATION	*updateDenominations*

7.1.3 IJxfsATMControl

Value	Method
JXFS_O_CDR_PRESENT	*present*
JXFS_O_CDR_REJECT	*reject*
JXFS_O_CDR_RETRACT	*retract*
JXFS_O_CDR_SHUTTER_MOVE	*shutterMove*

7.2 Exception Codes

Following tables specify exception codes that might occur as result of a method call. Exception codes are delivered to a caller during a method call.

The exception codes are already defined in the J/XFS-Base-Architecture and repeated here for easy access only.

7.2.1 Exception Code Summary and Description

Value	Description
JXFS_E_CLAIMED	Device is already claimed.
JXFS_E_CLOSED	Device has not been opened yet.
JXFS_E_FAILURE	The operation failed.
JXFS_E_ILLEGAL	Illegal request.
JXFS_E_IO	Errors during IO-operation.
JXFS_E_NO_HARDWARE	No Device is connected to the workstation.
JXFS_E_NOT_CLAIMED	Device is not claimed by caller.
JXFS_E_NOT_SUPPORTED	Operation is not supported by device.
JXFS_E_OFFLINE	Device is offline.
JXFS_E_PARAMETER _INVALID	An invalid parameter was given to the operation.
JXFS_E_REMOTE	Communication error during remote call.
JXFS_E_TIMEOUT	A timeout has occurred.

7.2.2 IJxfsCashDispenserControl Exception Codes

Methods / **Exception Codes**	reset	updateCashUnit	queryCashUnit	removeOrder	queryOrder	setDateTime	getDateTime	calibrateCashUnit	openSafeDoor	endexchange	startexchange	dispenseExec	dispense	denominate
JXFS_E_CLOSED	x	x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_FAILURE	x	x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_ILLEGAL	x	x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_IO	x	x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_NO_HARDWARE	x	x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_NOT_SUPPORTED	x	x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_OFFLINE	x	x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_PARAMETER_INVALID		x		x	x	x		x		x	x	x	x	x
JXFS_E_REMOTE	x	x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_TIMEOUT	x	x	x	x	x	x	x	x	x	x	x	x	x	x

7.2.3 IJxfsCashRecyclerControl Exception Codes

Error Codes			updateDenomination	queryDenominations	querySignatures	empty	cashInRollback	cashInEnd	cashIn	cashInStart
JXFS_E_CLOSED			x	x	x	x	x	x	x	x
JXFS_E_FAILURE			x	x	x	x	x	x	x	x
JXFS_E_ILLEGAL			x	x	x	x	x	x	x	x
JXFS_E_IO			x	x	x	x	x	x	x	x
JXFS_E_NO_HARDWARE			x	x	x	x	x	x	x	x
JXFS_E_NOT_SUPPORTED			x	x	x	x	x	x	x	x
JXFS_E_OFFLINE			x	x	x	x	x	x	x	x
JXFS_E_PARAMETER_INVALID			x		x	x			x	x
JXFS_E_REMOTE			x	x	x	x	x	x	x	x
JXFS_E_TIMEOUT			x	x	x	x	x	x	x	x

7.2.4 IJxfsATMControl Exception Codes

Exception Codes	shutterMove	retract	reject	present
JXFS_E_CLOSED	x	x	x	x
JXFS_E_FAILURE	x	x	x	x
JXFS_E_ILLEGAL	x	x	x	x
JXFS_E_IO	x	x	x	x
JXFS_E_NO_HARDWARE	x	x	x	x
JXFS_E_NOT_SUPPORTED	x	x	x	x
JXFS_E_OFFLINE	x	x	x	x
JXFS_E_PARAMETER_INVALID	x	x	x	
JXFS_E_REMOTE	x	x	x	x
JXFS_E_TIMEOUT	x	x	x	x

7.3 Error Codes

Following tables specify error codes that might occur as result of a method call. Error codes are delivered to a caller in field *result* of a *JxfsOperationCompleteEvent*.

7.3.1 Common Codes for all operations

Following codes can always occur as *result* of a *JxfsOperationCompleteEvent:*

Value	Description
JXFS_RC_SUCCESSFUL	Operation completed without error.
JXFS_E_TIMEOUT	A timeout during method execution occurred.

7.3.2 Error Code Summary and Description

Value	Description
JXFS_E_CDR_ASSET_UNDEFINED	Due to device error condition the cash unit content can not be determined.
JXFS_E_CDR_CASH_DEVICE_ERROR	An unspecified error occurred.
JXFS_E_CDR_CASH_UNIT_ERROR	A selected cash unit caused an error.
JXFS_E_CDR_CASHIN_ACTIVE	The device has already a *cashInStart* command issued.
JXFS_E_CDR_DELAYED_DISPENSE	Dispense order is delayed.
JXFS_E_CDR_EXCHANGE_ACTIVE	The device is in an exchange state.
JXFS_E_CDR_ILLEGAL_DISPENSE_ORDER	Invalid orderID during *dispenseExec*.
JXFS_E_CDR_ILLEGAL_DISPENSE_REQUEST	Invalid data during *dispense* or *empty*.
JXFS_E_CDR_INPUT_REFUSED	CashIn operation failure.
JXFS_E_CDR_INVALID_BILL	Invalid bill detected during *cashin*.
JXFS_E_CDR_INVALID_CASH_UNIT	Invalid cash unit ID.
JXFS_E_CDR_INVALID_COIN	Invalid coin detected during *cashin*.
JXFS_E_CDR_INVALID_CURRENCY	JxfsCurrency type is not configured.
JXFS_E_CDR_INVALID_DENOMINATION	The sum values for cashbox and cash units do not match the amount specified.
JXFS_E_CDR_INVALID_MIXNUMBER	The number refers to an undefined mix-table or mix-algorithm.
JXFS_E_CDR_INVALID_RETRACT	Retract area is invalid for this system.
JXFS_E_CDR_INVALID_SIGNATURE_ID	A signature Id for which no signature is available is supplied as input parameter.
JXFS_E_CDR_NO_BILLS	There were no bills on the stacker to present.
JXFS_E_CDR_NO_CASHIN_STARTED	*cashInStart* was not called.
JXFS_E_CDR_NO_EXCHANGE_ACTIVE	The device is not in an exchange state.
JXFS_E_CDR_NOT_DISPENSABLE	The amount is not dispensable.
JXFS_E_CDR_RESET_REQUIRED	*Reset* operation is required.
JXFS_E_CDR_TOO_MANY_BILLS	The request would require too many bills to be dispensed.
JXFS_E_CDR_TOO_MANY_COINS	The request would require too many coins to be dispensed.
JXFS_E_CDR_UNABLE_MOVE_ SHUTTER	Shutter could not be moved.
JXFS_E_CDR_UVV_IN_PROCESS	UVV delay is still active for this order.
JXFS_E_CDR_UVV_NOT_DISPENSEABLE	Order is not dispensable due to UVV regulations.

Note:

As defined in J/XFS-Base-Architecture, ANY operation can generate a JXFS_E_NOT_SUPPORTED exception.

For a cash-dispenser, sending this error during a dispense, doesn't make much sense, so exception JXFS_E_NOT_SUPPORTED is not shown in the following tables. Of course, some devices might generate the exception for some operations and applications must be aware of this behaviour.

7.3.3 IJxfsCashDispenserControl Error Codes

Methods / Error Codes	reset	updateCashUnit	queryCashUnit	removeOrder	queryOrder	setDateTime	getDateTime	calibrateCashUnit	openSafeDoor	endexchange	startexchange	dispenseExec	dispense	denominate
JXFS_E_CDR_ASSET_UNDEFINED												x	x	
JXFS_E_CDR_CASH_DEVICE_ERROR	x	x							x	x	x	x	x	
JXFS_E_CDR_CASH_UNIT_ERROR	x	x		x	x			x	x	x	x	x	x	x
JXFS_E_CDR_CASHIN_ACTIVE		x						x	x		x	x	x	
JXFS_E_CDR_DELAYED_DISPENSE												x	x	
JXFS_E_CDR_EXCHANGE_ACTIVE	x	x	x	x	x	x	x	x	x		x	x	x	x
JXFS_E_CDR_ILLEGAL_DISPENSE_ORDER			x									x		
JXFS_E_CDR_ILLEGAL_DISPENSE_REQUEST													x	
JXFS_E_CDR_INVALID_BILL		x								x				
JXFS_E_CDR_INVALID_CASH_UNIT		x								x				
JXFS_E_CDR_INVALID_COIN		x								x				
JXFS_E_CDR_INVALID_CURRENCY		x								x		x	x	x
JXFS_E_CDR_INVALID_DENOMINATION				x								x	x	x
JXFS_E_CDR_INVALID_MIXNUMBER												x	x	x
JXFS_E_CDR_NO_BILLS												x	x	x
JXFS_E_CDR_NO_EXCHANGE_ACTIVE										x				
JXFS_E_CDR_NOT_DISPENSABLE												x	x	x
JXFS_E_CDR_RESET_REQUIRED		x	x	x	x	x	x	x	x	x	x	x	x	x
JXFS_E_CDR_TOO_MANY_BILLS												x	x	x
JXFS_E_CDR_TOO_MANY_COINS												x	x	x
JXFS_E_CDR_UVV_IN_PROCESS												x		
JXFS_E_CDR_UVV_NOT_DISPENSEABLE													x	

7.3.4 IJxfsCashRecyclerControl Error Codes

Methods / Error Codes			updateDenomination	queryDenominations	querySignatures	empty	cashInRollback	cashInEnd	cashIn	cashInStart
JXFS_E_CDR_ASSET_UNDEFINED						x	x			
JXFS_E_CDR_CASH_DEVICE_ERROR			x	x		x	x	x	x	
JXFS_E_CDR_CASH_UNIT_ERROR			x	x		x	x	x	x	x
JXFS_E_CDR_CASHIN_ACTIVE			x	x		x				x
JXFS_E_CDR_DELAYED_DISPENSE						x				
JXFS_E_CDR_EXCHANGE_ACTIVE			x	x		x	x	x	x	x
JXFS_E_CDR_ILLEGAL_DISPENSE_REQUEST						x				
JXFS_E_CDR_INPUT_REFUSED									x	
JXFS_E_CDR_INVALID_BILL						x			x	
JXFS_E_CDR_INVALID_CASH_UNIT			x	x		x			x	
JXFS_E_CDR_INVALID_COIN						x			x	
JXFS_E_CDR_INVALID_CURRENCY			x	x		x			x	
JXFS_E_CDR_INVALID_DENOMINATION			x	x		x			x	
JXFS_E_CDR_INVALID_MIXNUMBER						x				
JXFS_E_CDR_INVALID_SIGNATURE_ID					x					
JXFS_E_CDR_NO_CASHIN_STARTED							x	x		
JXFS_E_CDR_NOT_DISPENSABLE						x				
JXFS_E_CDR_RESET_REQUIRED				x		x	x	x	x	x
JXFS_E_CDR_UVV_IN_PROCESS						x				

7.3.5 IJxfsATMControl Error Codes

Methods / Error Codes	shutterMove	retract	reject	present
JXFS_E_CDR_CASH_DEVICE_ERROR	x	x	x	x
JXFS_E_CDR_CASH_UNIT_ERROR				x
JXFS_E_CDR_CASHIN_ACTIVE	x	x	x	x
JXFS_E_CDR_EXCHANGE_ACTIVE	x	x	x	x
JXFS_E_CDR_INVALID_RETRACT		x		
JXFS_E_CDR_NO_BILLS				x
JXFS_E_CDR_RESET_REQUIRED	x	x	x	x
JXFS_E_CDR_UNABLE_MOVE_ SHUTTER	x			

8 Constants

8.1 Output position codes

Following output position codes can be or'ed groupwise. This is possible for a capability query. These codes are mainly used by dispense, retract and shutter operations.

Value	Description
JXFS_C_CDR_POS_NONE	No position selected
JXFS_C_CDR_POS_DEFAULT	Use configurated position
JXFS_C_CDR_POS_LEFT	Use left output side
JXFS_C_CDR_POS_CENTER	Use center output side
JXFS_C_CDR_POS_RIGHT	Use right output side
JXFS_C_CDR_POS_FRONT	Use front output side
JXFS_C_CDR_POS_REAR	Use rear output side
JXFS_C_CDR_POS_TOP	Use top output side
JXFS_C_CDR_POS_BOTTOM	Use bottom output side

Value	Description
JXFS_C_CDR_POS_OVERFLOW	Use overflow cassette
JXFS_C_CDR_POS_REJECT	Use reject cassette

8.2 Device Type codes

Value	Description
JXFS_C_CDR_TYPE_NONE	Device is not defined
JXFS_C_CDR_TYPE_DISPENSER	Device is a Cash Dispenser
JXFS_C_CDR_TYPE_RECYCLER	Device is a Cash Recycler
JXFS_C_CDR_TYPE_ATM	Device is a Automated Teller Machine

8.3 Cash Type codes

Value	Description
JXFS_C_CDR_CURR_BILL	Item represents a bill
JXFS_C_CDR_CURR_COIN	Item represents a coin

8.4 Cash Type variant code

Value	Description
JXFS_C_CDR_NO_VARIANT	No cash type variant information available

8.5 CashUnit Kind codes

Value	Description
JXFS_C_CDR_LCU_NA	Not available; cash unit is missing
JXFS_C_CDR_LCU_DISPENSE	Cash unit can be used for dispense.
JXFS_C_CDR_LCU_DEPOSIT	Cash unit can be used for deposit.
JXFS_C_CDR_LCU_RECYCLE	Cash unit can be used for dispense and deposit.

8.6 CashUnit Type codes

Value	Description
JXFS_C_CDR_LCU_BAIT_TRAP	Cash unit has bait trap capability.
JXFS_C_CDR_LCU_BILL_CASSETTE	Bill cassette of cash dispenser
JXFS_C_CDR_LCU_COIN_CYLINDER	Cylinder of the coin dispenser
JXFS_C_CDR_LCU_COIN_DISPENSER	Coin dispenser as a whole unit
JXFS_C_CDR_LCU_COUPON	Cassette for coupons or advertising materials
JXFS_C_CDR_LCU_CURRENCY_CASSETTE	Cassette, which may contain various bills with a different denomination for one currency.
JXFS_C_CDR_LCU_DOCUMENT	Cassette for documents
JXFS_C_CDR_LCU_ESCROW	Cassette is an escrow
JXFS_C_CDR_LCU_NA	Not available; cash unit is missing
JXFS_C_CDR_LCU_OVERFLOW_CASSETTE	Overflow cassette of cash dispenser
JXFS_C_CDR_LCU_REJECT_CASSETTE	Reject cassette of cash dispenser
JXFS_C_CDR_LCU_RETRACT_CASSETTE	Retract cassette of cash dispenser

8.7 CashUnit Status codes

Value	Description
JXFS_C_CDR_LCU_INOP	The cassette or coin cylinder is inoperative.
JXFS_C_CDR_LCU_MISSING	The cassette or coin cylinder is missing.
JXFS_C_CDR_LCU_NO_REF	There is no reference value available for the notes in this cassette.
JXFS_C_CDR_LCU_NO_VALUE	The values of the specified cash unit are not available. This could happen to be, if the cassette was changed without J/XFS calls.
JXFS_C_CDR_LCU_NOT_DISPENSEABLE	Cannot dispense from this cassette.
JXFS_C_CDR_LCU_OK	The cash unit is in a good state.
JXFS_C_CDR_LCU_UNKNOWN	The state of the cash unit is unknown.

8.8 Mix Type codes

Value	Description
JXFS_C_CDR_MIX_ALGORITHM	An algorithm is selected for mixing
JXFS_C_CDR_MIX_TABLE	A table is selected for mixing
JXFS_C_CDR_MIX_DENOM	The current selected JxfsDenomination is used.

8.9 Mix Table codes

Value	Description
JXFS_C_CDR_MXT_NONE	No mix-table specified
JXFS_C_CDR_MXT_TABLE_BASE	Base constant for vendor specific mix tables.

Remark:

Vendor specific mix tables are specified by a value of JXFS_C_CDR_MXT_TABLE_BASE + 1..n.

8.10 Mix Algorithm codes

Value	Description
JXFS_C_CDR_MXA_NONE	No algorithm selected.
JXFS_C_CDR_MXA_MIN_BILLS	The minimal number of bills is used
JXFS_C_CDR_MXA_EQUAL_EMPTY	All cash units are equally emptied.
JXFS_C_CDR_MXA_ALGORITHM_BASE	Base constant for vendor specific mix algorithm.

Remark:

Vendor specific mix algorithms are specified by a value of JXFS_C_CDR_MXA_ALGORITHM_BASE + 1..n.

8.11 Retract Area codes

Value	Description
JXFS_C_CDR_RA_REJECT	Retract to a reject unit.
JXFS_C_CDR_RA_RETRACT	Retract to a retract unit.
JXFS_C_CDR_RA_STACKER	Retract to intermediate stacker.
JXFS_C_CDR_RA_TRANSPORT	Retract to the transport.

8.12 UVV Delayed Order Queue codes

Value	Description
JXFS_C_CDR_DO_ALL	All orders in all queues.
JXFS_C_CDR_DO_DELAYED	All orders in delay queue.
JXFS_C_CDR_DO_DISPENSABLE	Orders ready for processing.
JXFS_C_CDR_DO_LAQ	All orders in Large Amount Queue.
JXFS_C_CDR_DO_NONE	Order is not in any queue, because of immediate dispense.

8.13 Cash Tray Status codes

Value	Description
JXFS_S_CDR_CT_EMPTY	Cashtray is empty
JXFS_S_CDR_CT_NOT_EMPTY	Cashtray is not empty
JXFS_S_CDR_CT_NOT_SUPPORTED	A cashtray is not supported
JXFS_S_CDR_CT_UNKNOWN	Cashtray status unknown

8.14 Device Status codes

Value	Description
JXFS_S_CDR_DS_ON_LINE	Device is online
JXFS_S_CDR_DS_OFF_LINE	Device is offline
JXFS_S_CDR_DS_POWER_OFF	Device has poweroff
JXFS_S_CDR_DS_BUSY	Device is busy
JXFS_S_CDR_DS_NO_DEVICE	No device found
JXFS_S_CDR_DS_USER_ERROR	Device reported an user error
JXFS_S_CDR_DS_HARDWARE_ERROR	Device reported a hardware error

8.15 Dispenser Status codes

Value	Description
JXFS_S_CDR_DIS_OK	All logical cash units are ok.
JXFS_S_CDR_DIS_CU_STATE	One of the logical cash units present is in an abnormal state. The dispenser is operational, but one or more of the cash units is in a low, empty or inoperative condition. Bills can still be dispensed from at least one of the cash units.
JXFS_S_CDR_DIS_CU_STOP	Due to a cash unit failure dispensing is impossible. The dispenser is operational, but no bills can be dispensed because all of the cash units are in an empty or inoperative condition. This state occurs when a reject cash unit is full or no reject cassette is present.
JXFS_S_CDR_DIS_CU_UNKNOWN	Due to a hardware error or other condition, the state of the cash units cannot be determined.

8.16 Intermediate Stacker Status codes

Value	Description
JXFS_S_CDR_IS_EMPTY	Stacker is empty
JXFS_S_CDR_IS_NOT_EMPTY	Stacker is not empty
JXFS_S_CDR_IS_UNKNOWN	Stacker state is unknown
JXFS_S_CDR_IS_NOT_SUPPORTED	A stacker is not supported

8.17 Safe Door Status codes

Value	Description
JXFS_S_CDR_SD_NOT_SUPPORTED	A safedoor is not supported
JXFS_S_CDR_SD_OPEN	Safedoor is open
JXFS_S_CDR_SD_CLOSED	Safedoor is closed
JXFS_S_CDR_SD_LOCKED	Safedoor is locked
JXFS_S_CDR_SD_UNKNOWN	Safedoor state is unknown

8.18 Shutter Status codes

Value	Description
JXFS_S_CDR_SHT_CLOSED	Shutter is closed
JXFS_S_CDR_SHT_OPEN	Shutter is open
JXFS_S_CDR_SHT_JAMMED	Shutter is malfunctional
JXFS_S_CDR_SHT_NOT_SUPPORTED	A shutter is not supported
JXFS_S_CDR_SHT_UNKNOWN	Shutter state is unknown

8.19 Transport Status codes

Value	Description
JXFS_S_CDR_TP_OK	Transport is working

JXFS_S_CDR_TP_INOP	Transport is not working
JXFS_S_CDR_TP_NOT_SUPPORTED	A transport unit is not supported
JXFS_S_CDR_TP_UNKNOWN	State of transport unit is unknown

8.20 Vandalism Status codes

Value	Description
JXFS_S_CDR_VAN_MANIPULATION	A manipulation was detected
JXFS_S_CDR_VAN_NO_MANIPULATION	No manipulation was detected
JXFS_S_CDR_VAN_NOT_SUPPORTED	A vandalism check is available

8.21 Present Status codes - deprecated

Value	Description
JXFS_S_CDR_PR_UNKNOWN	It is unknown if the money could be accessed by the customer.
JXFS_S_CDR_PR_NOT_PRESENTED	The money was not presented.
JXFS_S_CDR_PR_PRESENTED	The money was presented. This value is set as soon as the bills are accessible by the customer.
JXFS_S_CDR_CASH_TAKEN	The cash was taken by the user.

9 Device Service Characteristics

9.1 MDU - Minimum Dispense Unit

Each monetary amount is expressed in terms of multiples of “Minimum Dispense Units“ (MDU).

9.1.1 Definitions

Abbreviation	*Description*
MDU	Minimum Dispense Unit
CU	Currency Unit, defined in ISO 4217
CE	Currency Exponent
MAP	Money Amount Parameter. Amount of cash expressed in MDUs.

Currency Unit (CU) for ...	*Country Code*	*Description*
European money	EUR	1 Euro
Former Italian money	LIT	1 Italian Lira

Currency Exponent (CE) for ...	*Description*	*MDU equals*
European money	-2	1 Cent
Former Italian money	+2	100 Lire

A MDU is equal to CU *times 10* ^ CE.
A MAP relates to the amount of cash like: *Amount of cash = MAP * 10* ^ *C*E.

9.1.2 Example

Europe:

Country code	EUR
CU	1 Euro (= 100 Cent)
CE	-2
MAP	10050
Amount of cash	MAP * 10 ^ CE
€ 100,50	10050 * 10 ^ -2

9.2 Delayed Dispense

9.2.1 Introduction

The delayed dispense concept is based on German security rules (also called “UVV”) which define the manner in which a cash dispensing device should dispense cash, in order to minimize losses in the event of bank robbery.

Those security rules define [1]:

- maximum values for total amount of cash allowed to be dispensed within certain time periods, and
- minimum dispense delay times for amounts which exceed certain values.

The cash dispenser software / hardware used in German financial institutes must conform to those rules in order to be officially approved for legal usage.

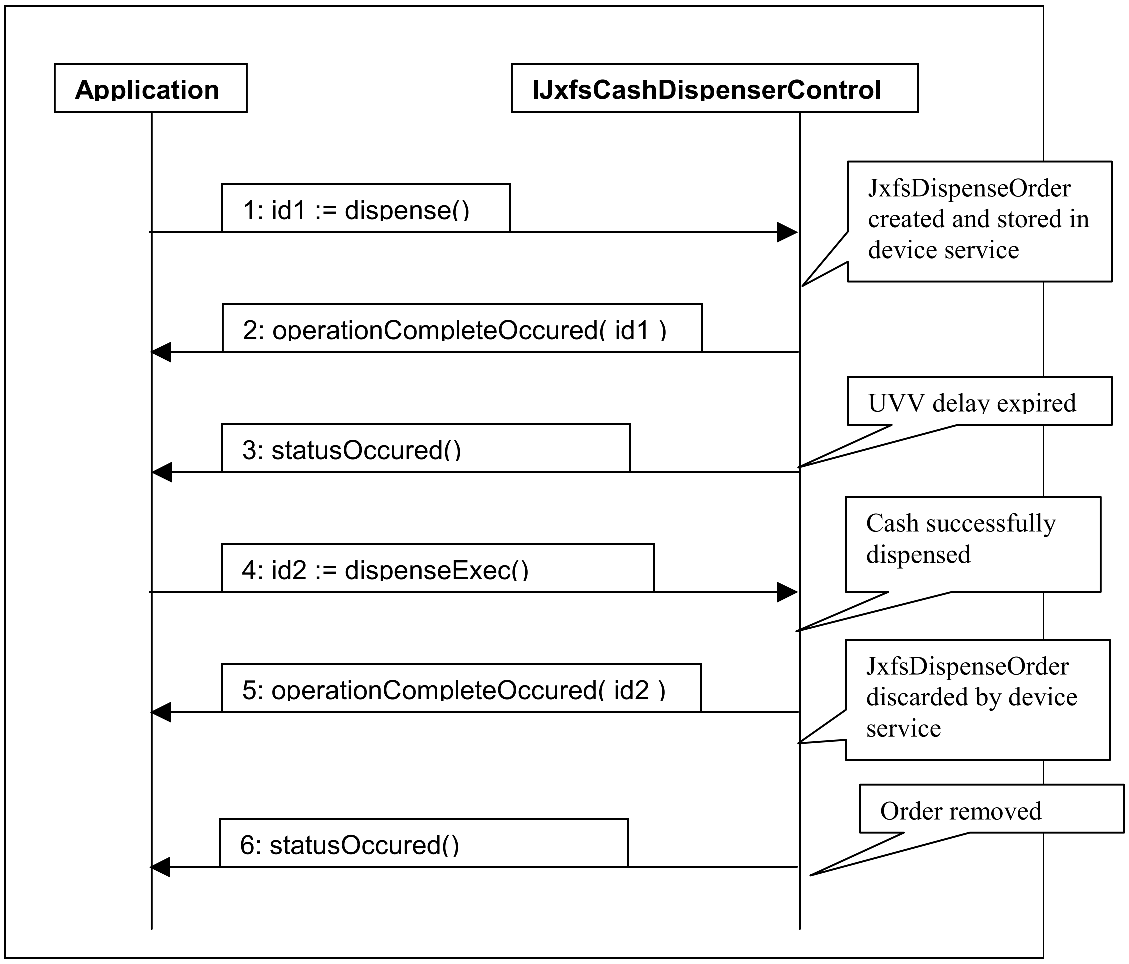
9.2.2 Delayed dispense in J/XFS

J/XFS supports the “UVV” security rules by defining:

- the set of classes, interfaces, properties and constants used for delayed dispense
- the appropriate protocol between the application and the J/XFS device control which enables the handling of delayed dispense transactions

9.2.3 Delayed dispense protocol

The following sequence diagram presents the communication between the application and the J/XFS device control defined by the delayed dispense protocol:


Application
IJxfsCashDispenserControl
1: id1 := dispense()
JxfsDispenseOrder created and stored in device service
2: operationCompleteOccured(id1)
UVV delay expired
3: statusOccured()
Cash successfully dispensed
4: id2 := dispenseExec()
5: operationCompleteOccured(id2)
JxfsDispenseOrder discarded by device service
Order removed
6: statusOccured()

The delayed dispense protocol starts by calling the *dispense()* method of the J/XFS device control implementing the *IJxfsCashDispenserControl* interface (1). The dispense request will be put in the service job queue within the J/XFS device service and an identification number will be returned to the caller immediately, according to the asynchronous nature of J/XFS service jobs.

During the execution of the service job the device service checks if the UVV rules allow an immediate dispense of the requested cash amount. If not, the J/XFS device service creates a *JxfsDispenseOrder* object representing the delayed dispense order and stores it internally. See the description of the *JxfsDispenseOrder* class for information how to initialize the *JxfsDispenseOrder* object properties. The J/XFS device control also sends a *JxfsOperationCompleteEvent* object in order to inform the caller that the dispense order has been delayed (2). The *result* property of the event is set to the JXFS_E_CDR_DELAYED_DISPENSE value. The *data* property contains a copy of the corresponding *JxfsDispenseOrder* object.

When the delay time defined by the UVV rules expires, the device service changes the *queueID* property of the *JxfsDispenseOrder* object to the JXFS_C_CDR_DO_DISPENSABLE value and sends spontaneously a *JxfsStatusEvent* object to all registered listeners (3). The *status* property of the event is set to the JXFS_S_CDR_DELAYED_ORDER_READY value and the *details* property contains a copy of the *JxfsDispenseOrder* object which has changed.

The application requests an immediate dispense of the previously delayed dispense order by calling the *dispenseExec()* method of the device control (4). The dispense request will be sent to the device service and an identificationID will be returned to the caller immediately.

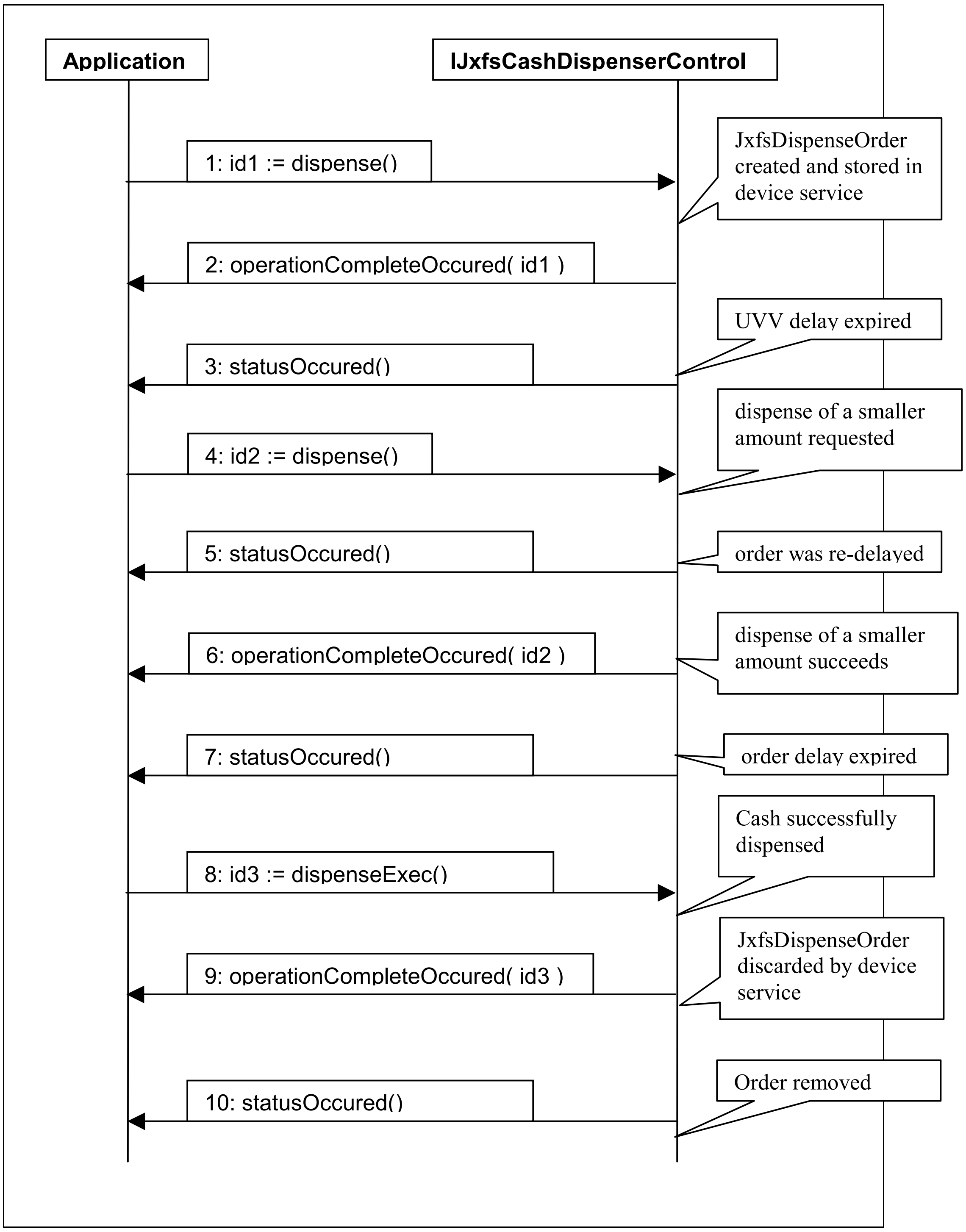
During the execution the cash is dispensed to the exit slot of the device and a *JxfsOperationCompleteEvent* is sent to the caller (5). The *result* property of the event is set to the JXFS_RC_SUCCESSFUL value. The *data* property contains a copy of the *JxfsDispenseOrder* object representing the dispense order which was successfully executed. The device service discards the internally stored *JxfsDispenseOrder* object and sends a JxfsStatusEvent with JXFS_S_CDR_DELAYED_ORDER_REMOVED (6) to all registered listeners.

9.2.4 Re-delaying orders

According to the delayed dispense protocol, the application is responsible for calling the *dispenseExec* method explicitly to dispense cash after the delay period has expired. Depending on the application logic, the application may decide to dispense smaller amounts of money immediately (using the *dispense* method) before calling *dispenseExec*.Those additional dispenses may cause the device service to re-delay an order which was currently ready for dispense in order to comply to UVV rules (especially to the rule (a), see Introduction). The same situation may also happen when two device controls are using the same device service concurrently.

Re-delaying of orders is also required to prevent attacks by enemy client applications. Such an application would create many delayed orders using the *dispense* method. After all delay times for those orders have expired, the application would try to dispense them as quick as possible using *dispenseExec()* method calls. Allowing such scenarios in the device service would violate UVV security rules.

The following sequence diagram presents the communication between the application and the J/XFS device control in such a scenario:


Application
IJxfsCashDispenserControl
1: id1 := dispense()
JxfsDispenseOrder created and stored in device service
2: operationCompleteOccured(id1)
UVV delay expired
3: statusOccured()
dispense of a smaller amount requested
4: id2 := dispense()
5: statusOccured()
order was re-delayed
6: operationCompleteOccured(id2)
dispense of a smaller amount succeeds
7: statusOccured()
order delay expired
Cash successfully dispensed
8: id3 := dispenseExec()
JxfsDispenseOrder discarded by device service
9: operationCompleteOccured(id3)
Order removed
10: statusOccured()

The steps (1)-(3) are the same as in the previous chapter.

In the step (4) the application logic decides to postpone handling of the status event (3) and dispense a smaller amount instead, using the *dispense()* method. The device service dispenses this smaller amount and decides to re-delay the order in order to meet the UVV requirements. The *queueID* property of the *JxfsDispenseOrder* object is changed to JXFS_C_CDR_DO_DELAYED or value (depending on the order kind) and the *delay* property is recalculated.

A *JxfsStatusEvent* object is sent to all registered listeners (5). The *status* property of the event is set to the JXFS_S_CDR_DELAYED_ORDER_CHANGED value and the *details* property contains a copy of the *JxfsDispenseOrder* object which has changed. After the dispensing of the smaller amount succeeds, a *JxfsOperationCompleteEvent* object is sent to the calling application (6). The *result* property of the event is set to the JXFS_RC_SUCCESSFUL value. The *data* property contains a *JxfsDispenseOrder* object representing the amount which was successfully dispensed.

The steps (7)-(10) correspond to the steps (3)-(6) in the previous chapter.

9.2.5 Support methods

The *IJxfsCashDispenserControl* interface provides some support methods for query and manipulation of dispense orders internally stored by the device service.

The *queryOrder* method is used retrieve all orders of the given type. The *removeOrder* method is used to request the device service to discard a dispense order.

The method *getUvv* returns *true* if the order delaying mechanism is currently active, *false* if it is not. If inactive, no order delaying will happen, regardless of requested cash amounts and/or times when the requests are sent. The *setUvv* method can be used to enable or disable order delaying mechanism. Disabling the order delaying mechanism is allowed if and only if there are no dispense orders internally stored in the device service.

For further information about support methods please consult the *IJxfsCashDispenserControl* interface specification.

9.2.6 Error handling

The JXFS_E_CDR_ILLEGAL_DISPENSE_ORDER error code can be sent as the *result* property of the *JxfsOperationCompleteEvent* of any operation which requires a JxfsDispenseOrder object as parameter. It indicates the incorrectness of a *JxfsDispenseOrder* parameter. A *JxfsDispenseOrder* parameter is incorrect if:

- the device service can not find any order with the corresponding *orderID* property
- the *denomination* properties of the internal *JxfsDispenseOrder* object and the parameter don't have the same content
- the *currency* properties of the internal *JxfsDispenseOrder* object and the parameter don't have the same content

The JXFS_E_CDR_DELAYED_DISPENSE error code can be sent as the *result* property of the *JxfsOperationCompleteEvent* of the *dispense*. It indicates that a dispense order was delayed. The *data* property of the event contains a copy of the internally stored *JxfsDispenseOrder* object representing the delayed dispense order.

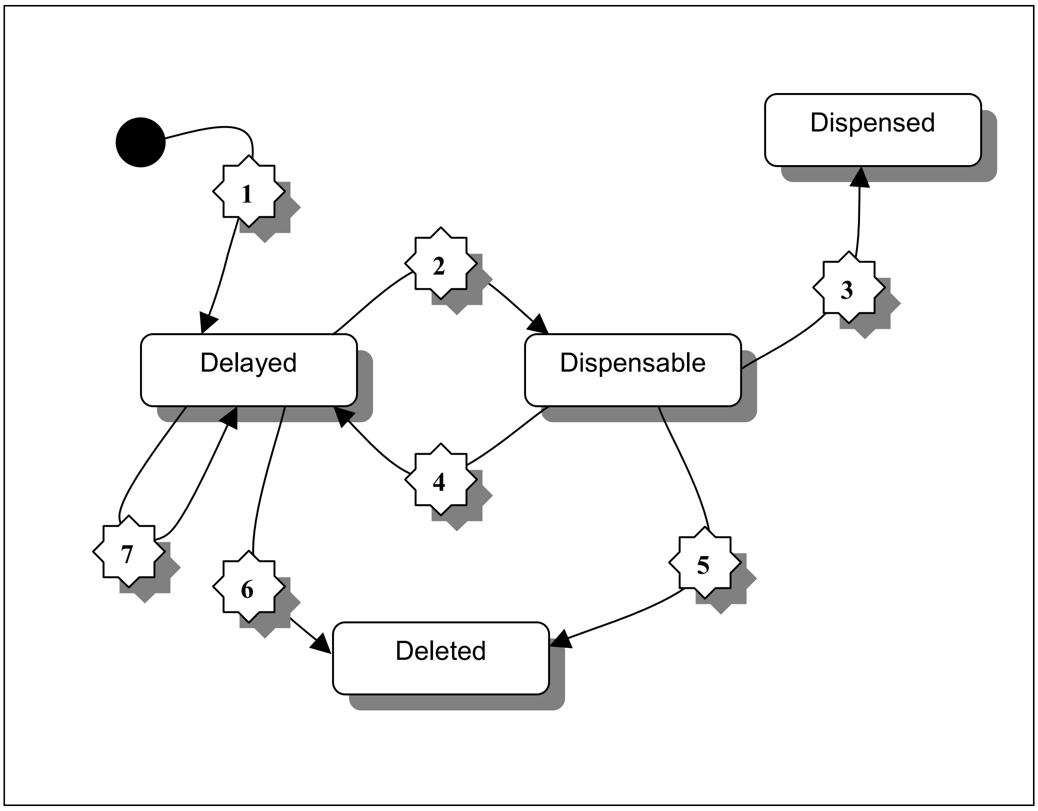
The JXFS_E_CDR_UVV_IN_PROCESS error code can be sent as the *result* property of the *JxfsOperationCompleteEvent* of the *dispenseExec* and indicates that the requested dispense order isn't dispensable yet. The *data* property of the event contains a copy of the internally stored *JxfsDispenseOrder* object representing the delayed dispense order.

The JXFS_E_CDR_UVV_NOT_DISPENSABLE error code can be sent as the *result* property of the *JxfsOperationCompleteEvent* of the *dispense* and indicates that the requested dispense order isn't dispensable due to UVV regulations. The *data* property of the event contains a copy of the rejected *JxfsDispenseOrder* object.

The JXFS_E_ILLEGAL value can be sent as the error code within the *JxfsException* in the *setUvv* method if disabling the order delaying mechanism was requested and there are dispense orders internally stored in the device service.

9.2.7 State changes of a dispense order during delayed dispense

The following diagram shows state transitions of a delayed dispense order and all events transmitted during state transitions.



Legend:

Transition	*Reason*	*Event*
1	dispense	OC: JXFS_E_CDR_DELAYED_DISPENSE SE: JXFS_S_CDR_DELAYED_DISPENSE
2	delay expired	SE: JXFS_S_CDR_DELAYED_ORDER_CHANGED
3	dispenseExec completed	OC: JXFS_RC_SUCCESSFUL SE: JXFS_S_CDR_DELAYED_ORDER_REMOVED
4	redelay	SE: JXFS_S_CDR_DELAYED_ORDER_CHANGED
5	removeOrder	OC: JXFS_RC_SUCCESSFUL SE: JXFS_S_CDR_DELAYED_ORDER_REMOVED
6	removeOrder	OC: JXFS_RC_SUCCESSFUL SE: JXFS_S_CDR_DELAYED_ORDER_REMOVED
7	redelay	SE: JXFS_S_CDR_DELAYED_ORDER_CHANGED

9.2.8 Timing

J/XFS doesn’t define algorithms or strategies for calculating delay times for delayed orders. The only requirement is that the device service implementation has to calculate those delay times in such a way that dispensing the cash conforms to currently active UVV security rules.

For example, let us consider 2 different device service implementations: A and B. Let’s suppose that the application calls the *dispense()* method three times, with the amounts of €2500, €2600 and €100 respectively. According to current UVV security rules [1], the second request should be delayed for at least 30 s after the first one has been fulfilled, so both device services decide to delay it. But, the device service A dispenses the third request immediately, where the device service B delays it to be dispensed after the second amount.

Device services A and B are both conform to J/XFS because they implement the delayed dispense protocol and also ensure that cash dispensing conforms to the UVV security rules.

9.2.9 References

[1] BG-Vorschrift Kassen vom 1. Oktober 1988 in der Fassung vom 1. Januar 1997 mit Durchführungsanweisungen vom Oktober 1988

9.3 European Article 6 regulations support

9.3.1 Background Information

To accept and / or recycle Euro notes, cash recyclers must comply with the rules of banknotes acceptance as defined in "THE EUROSYSTEM'S TERMS OF REFERENCE FOR THE USE OF CASH- RECYCLING MACHINES BY CREDIT INSTITUTIONS AND OTHER EURO AREA INSTITUTIONS ENGAGED IN THE SORTING AND DISTRIBUTION OF BANKNOTES TO THE PUBLIC AS A PROFESSIONAL ACTIVITY". These rules are generally called "Article 6."
European Article 6 defines 4 categories of notes and the rules to handle them:

Category	Classification	Properties	Treatment
1	**Not** recognized as **a banknote**,	Not detected as a banknote because of: • Wrong image or format. • Transportation error. (e.g. double feeds, etc.) • Large dog-eared or missing sections. • Handwritten notes, separating cards, etc. • Wrong currency.	Return to customer
2	Element(s) identified as **counterfeit**.	Image and format recognized, but one or more authentication features (IR, UV, magnetism, security thread, etc.) missing or clearly out of tolerance.	This kind of notes must be withdrawn from circulation. To be handed over – together with information on the account holder – when confirmed as counterfeit to the competent national authorities. **Not to be credited to account holder**.
3	Elements not clearly authenticated. **Suspect banknotes**.	Image, format and authentication features (IR, UV, magnetism, security thread, etc.) recognized, but quality and/or tolerance deviations. **In most cases unfit or soiled banknotes**	The banknotes must be processed separately and transported to an NCB for authentication. The information on the account holder has to be stored for four weeks at least and made available on request. **May be credited to account holder**.
4	Banknotes fully authenticated as **genuine** ones.	All authentication checks delivered positive results	Can be used for recycling. To be credited to account holder.

9.3.2 Requirements

A bank note is defined with the following parameters:

- Currency: defines the currency of the note (EUR, USD,…)
- Value: denomination value (1, 10, 20, 50, …)
- Release: release of note (1, 2, ...)
- Category: category of note 2, 3 or 4. Category 1 notes are always returned to the customer.

For each cashin transaction the following rules should be applied:

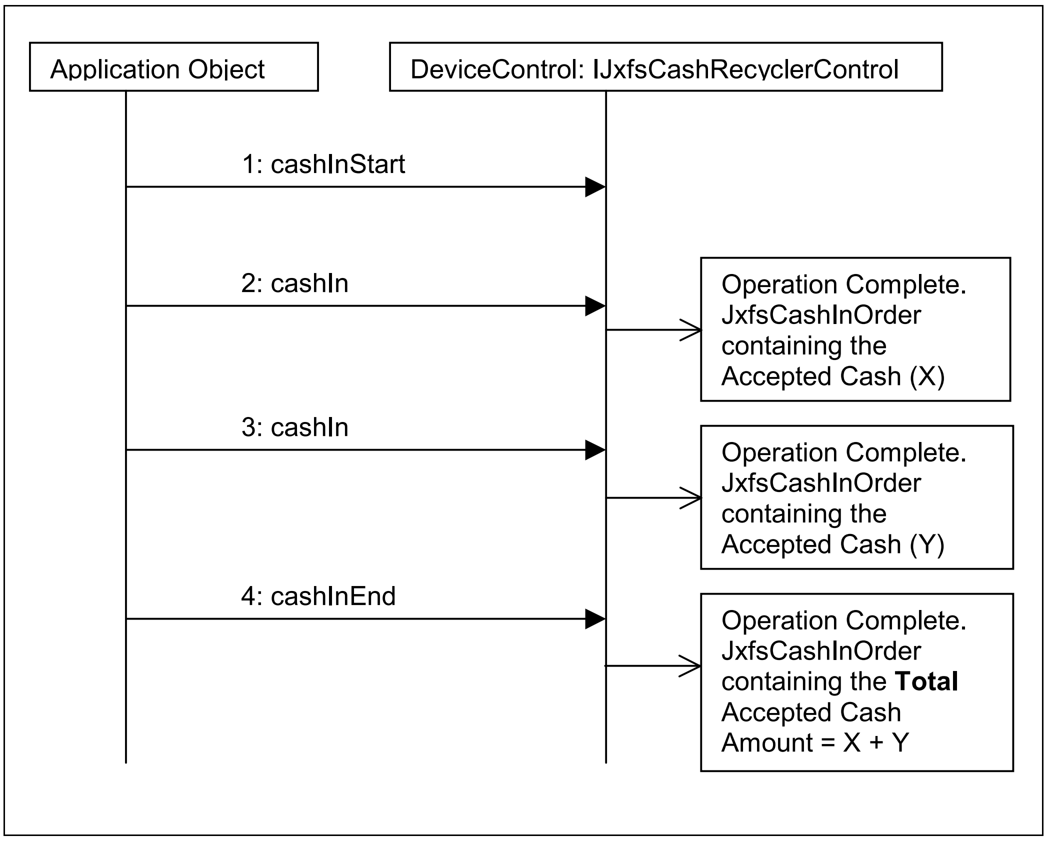
- For each cash deposit and for each category of note, the complete set of a bank note parameters should be returned to the application.
    - After cash deposit operations, the number and kind of category 2 and 3 banknotes must be reported to the application, thus enabling it to perform the corresponding tasks according to the European article 6 regulations.
    - For each category 2 and 3 banknote detected by the device, the corresponding signature information must be reported to the application in order to enable the application to assign it to the customer who has deposited it. A signature is a unique identifier for a banknote. It is used together with the transaction data like an account number (PAN) and transaction number to identify the customer who has deposited this bank note. The format and the content of a signature is vendor dependent.
    - For cash deposit operations, some kind of “trusted user mode” should be provided. This mode may be used by a trusted operator (cashier) for note checking or counting. In this mode the category 2 and category 3 notes will not be retained but returned and no signature will be generated.
- Additional device capabilities must be provided, enabling applications to query the device service about its ability to support European article 6 regulations.

9.4 Recycler Rollback Procedure

The following paragraphs and diagrams show the flow of operation for deposit operations used by cash recycler devices.

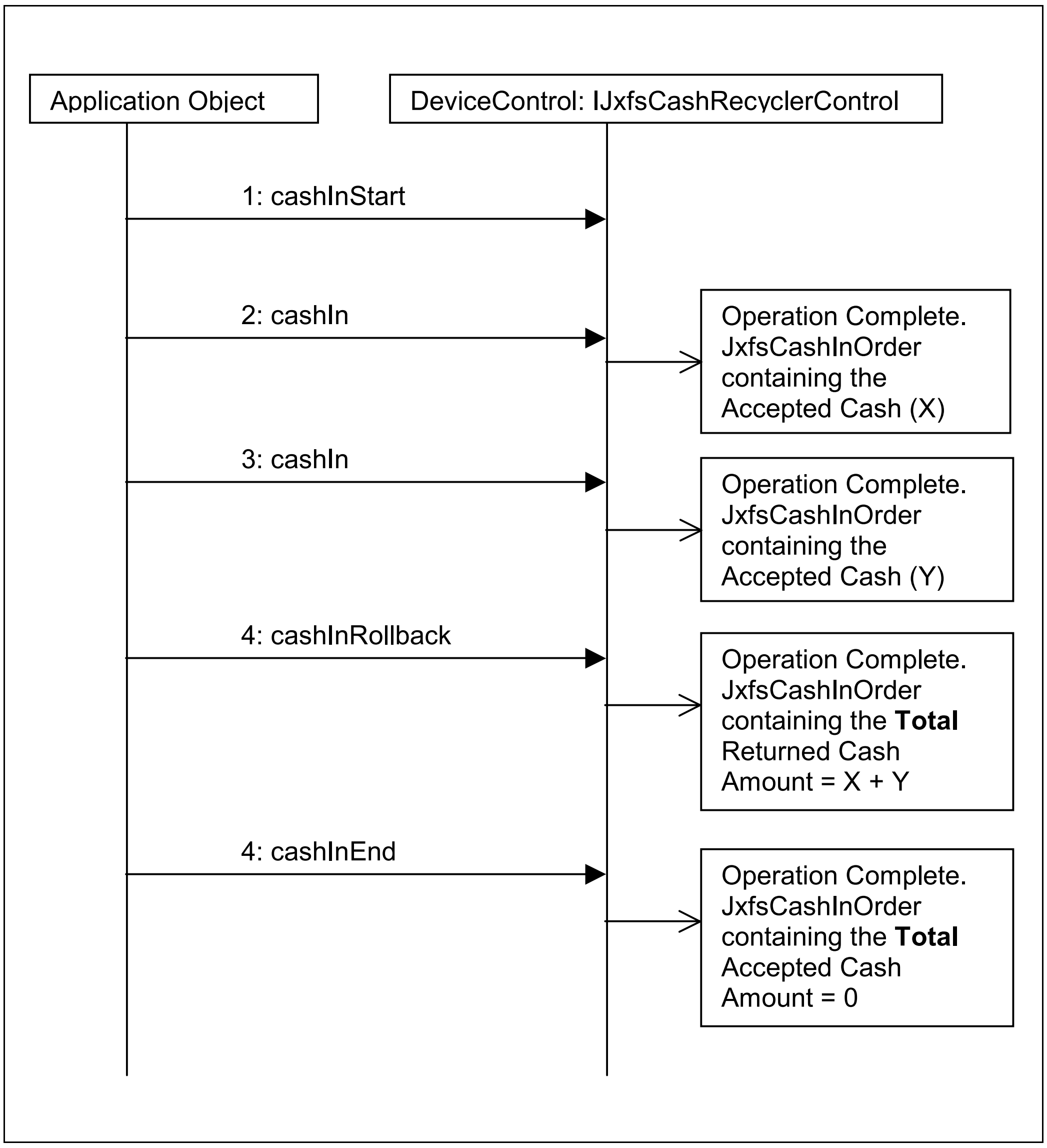
9.4.1 Normal operating

An example of an ordinary deposit operation is displayed below:


Application Object
DeviceControl: IJxfsCashRecyclerControl
1: cashInStart
2: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (X)
3: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (Y)
4: cashInEnd
Operation Complete. JxfsCashInOrder containing the **Total** Accepted Cash Amount = X + Y

9.4.2 Rollback without errors

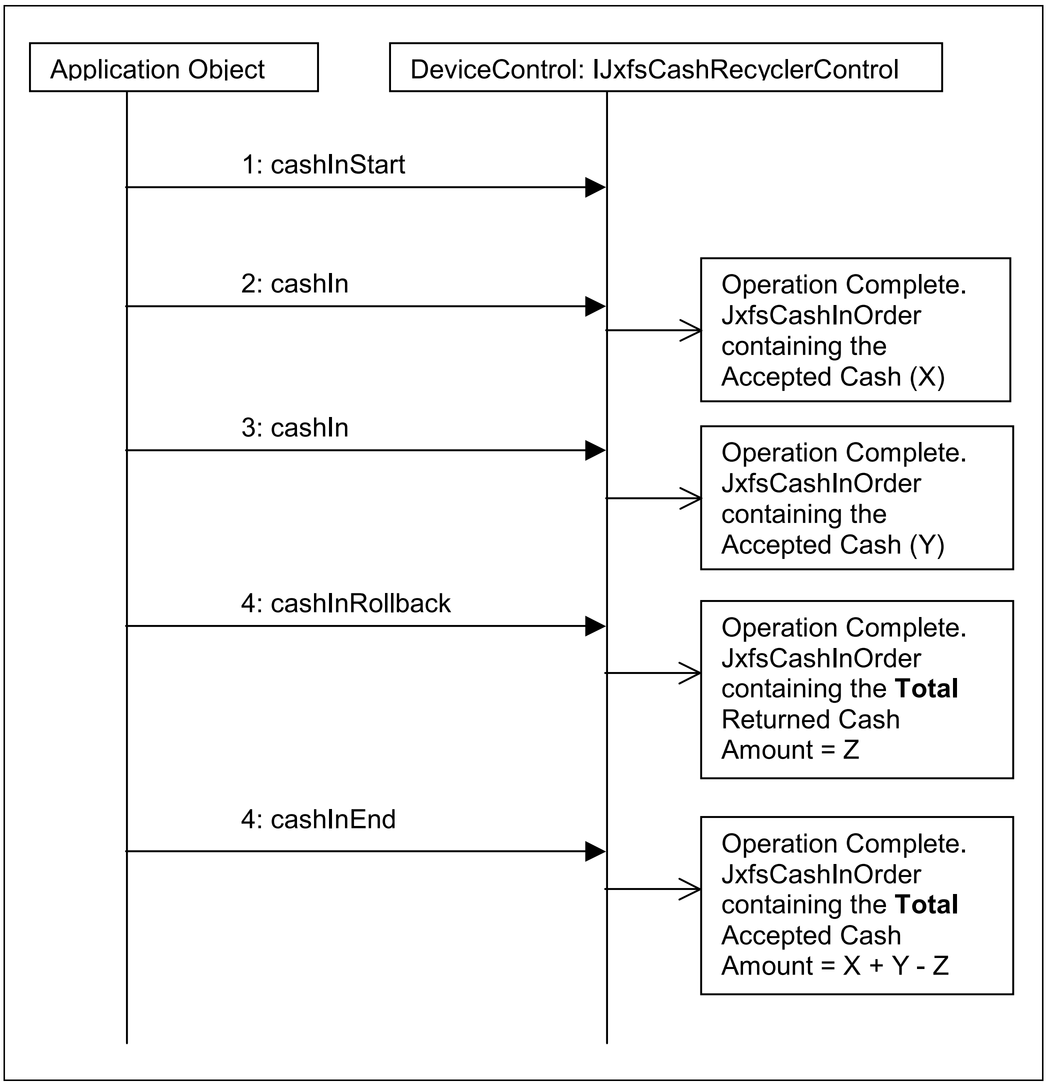
Most of the times, the notes inserted by means of consecutive *cashIn* are stored in the escrow. When performing the rollback operation, these notes will be ejected and presented to the customer.


Application Object
DeviceControl: IJxfsCashRecyclerControl
1: cashInStart
2: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (X)
3: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (Y)
4: cashInRollback
Operation Complete. JxfsCashInOrder containing the Total Returned Cash Amount = X + Y
4: cashInEnd
Operation Complete. JxfsCashInOrder containing the Total Accepted Cash Amount = 0

9.4.3 Rollback with errors

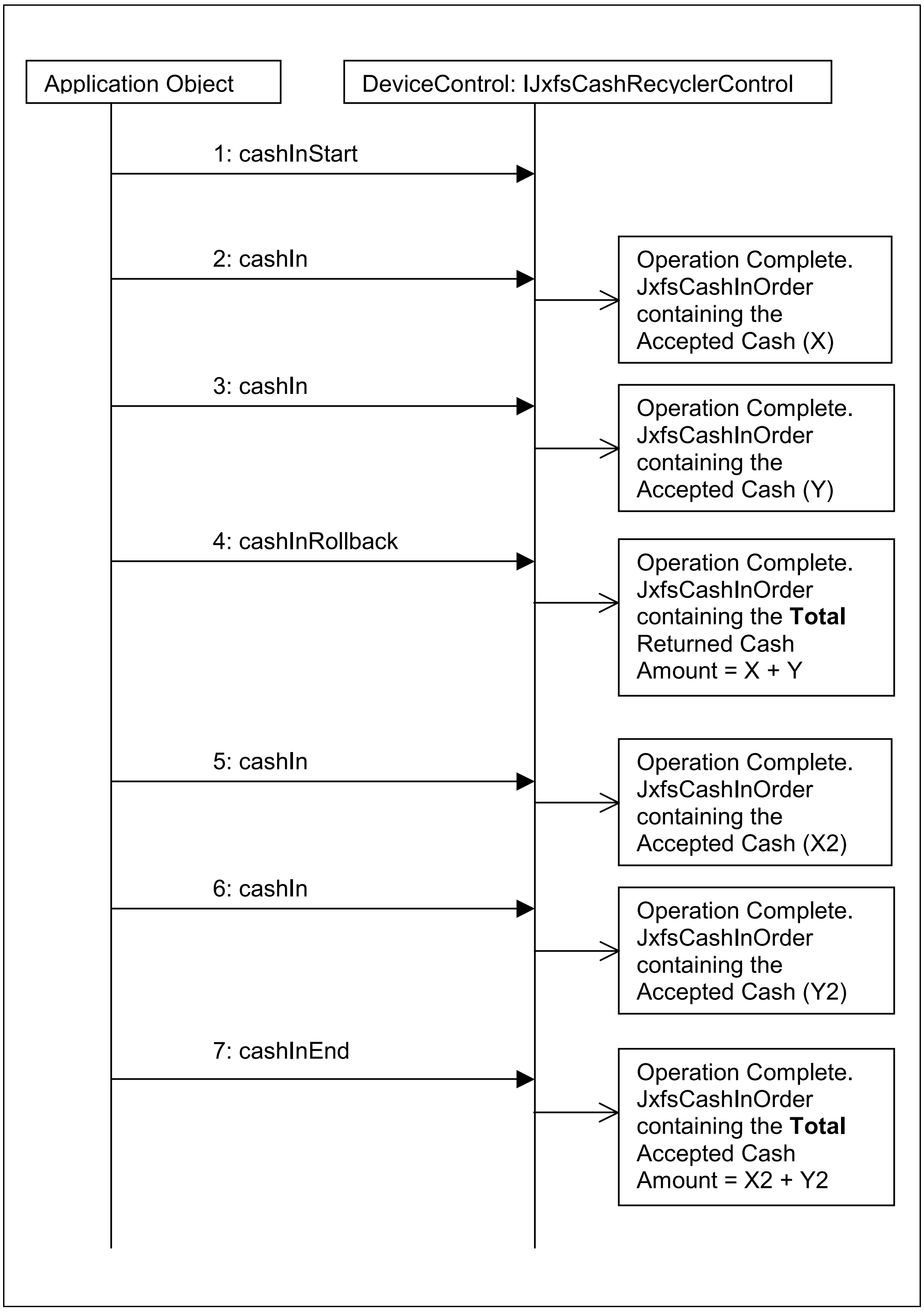
The fact of performing a rollback and not being returned all the notes might occur. This is not likely to happen, but in the specific case of the recyclers without an escrow and those where the rollback process is performed by means of a dispense operation, a dispense error could occur and thus the customer might be presented a smaller amount of cash.

The manner of operating would be the following:


Application Object
DeviceControl: IJxfsCashRecyclerControl
1: cashInStart
2: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (X)
3: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (Y)
4: cashInRollback
Operation Complete. JxfsCashInOrder containing the Total Returned Cash Amount = Z
4: cashInEnd
Operation Complete. JxfsCashInOrder containing the Total Accepted Cash Amount = X + Y - Z

9.4.4 CashIn after rollback

After a rollback operation it is allowed to send more *cashIn*s.


Application Object
DeviceControl: IJxfsCashRecyclerControl
1: cashInStart
2: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (X)
3: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (Y)
4: cashInRollback
Operation Complete. JxfsCashInOrder containing the **Total** Returned Cash Amount = X + Y
5: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (X2)
6: cashIn
Operation Complete. JxfsCashInOrder containing the Accepted Cash (Y2)
7: cashInEnd
Operation Complete. JxfsCashInOrder containing the **Total** Accepted Cash Amount = X2 + Y2

9.4.5 Conclusion

All deposit operations will be started with a *cashInStart* and ended with a *cashInEnd*, regardless whether a *cashInRollback* was performed or not.

The application will be in charge of the possible partial rollbacks. This must be checked by examining the data returned from *cashInRollback* and *cashInEnd*.

Although a *cashInEnd* would not be necessary to be sent when in a *cashInRollback* operation all notes are returned, the operation will not be considered finished by the device service until a *cashInEnd* is received.

It is possible to send more *cashIn* transactions after a *cashInRollback* operation.

It is not allowed to call the *dispense* method between a *cashInStart* and a *cashInEnd*. In this case, a *JxfsOperationCompleteEvent* with JXFS_E_CDR_CASH_IN_ACTIVE will be returned by the *dispense* method.

9.5 Representation of Physical Escrow

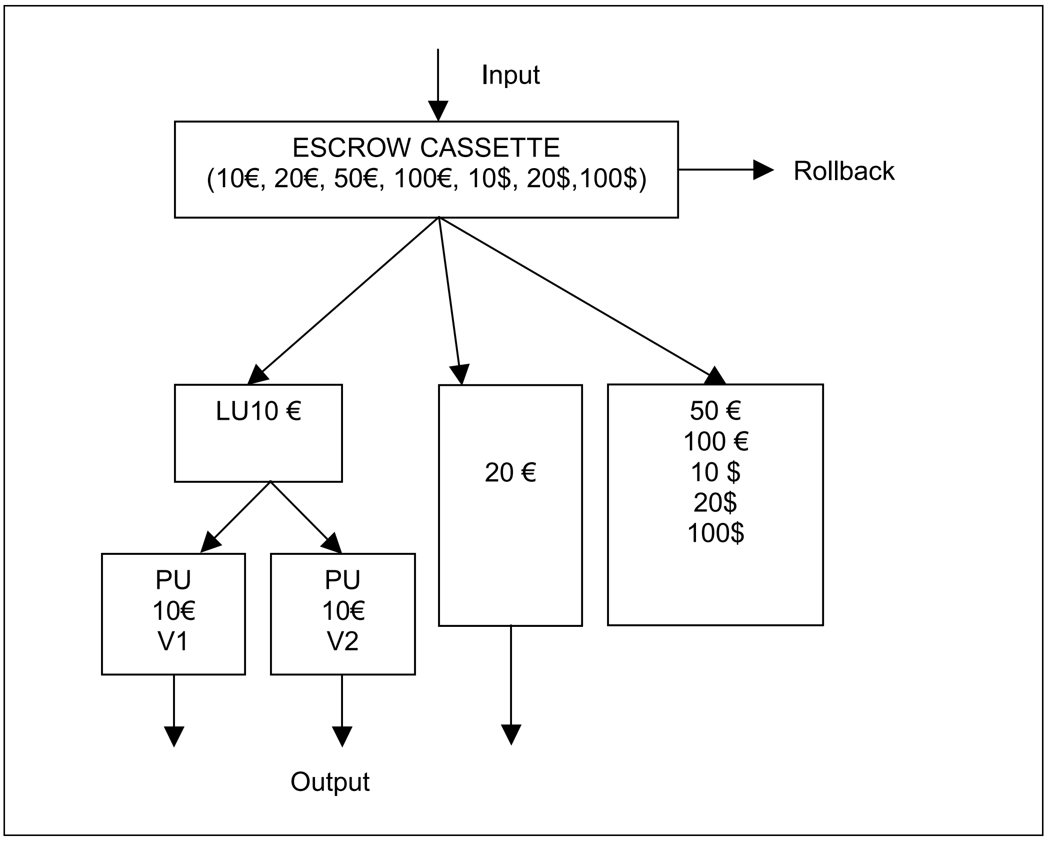
9.5.1 Overview

The current specification regarding cash dispensers and recyclers do not clarify the manner a cassette of the escrow type has to be defined; therefore an explanation permitting us to homogenize every manufacturer's device services, as much as possible, is necessary to be given.

The main objective is to provide a definition concerning this cassette type as complete as it might possible be, for us to know the exact status of this cassette type having the most detailed information available.

9.5.2 Example Recycler

In order to help us with the explanation, the recycler displayed below will be used in the next example. This recycler includes the following cassettes:



This recycler's characteristics are the following:

- A reader for recognition of 10€ variant 1 & 2, 20€, 50€, 100€, 10$, 20$ and 100$ notes
- An escrow cassette where all the notes belonging to the aforementioned types can be stored
- A dispense and deposit cassette (recycler) for 10€ notes (variant 1 & 2)
- A dispense and deposit cassette (recycler) for 20€ notes
- A deposit cassette for the remaining denominations

9.5.3 Physical Cassettes

The recycler will include the following physical cassettes

- P1 Escrow Cassette
- P2 Cassette for 10€ notes variant 1
- P3 Cassette for 10€ notes variant 2
- P4 Cassette for 20€ notes
- P5 Cassette for the remaining denominations

9.5.4 Logical Cassettes

The most meaningful fields corresponding to the JxfsLogicalCashUnit class for the different logical cassettes of this recycler are viewed in the table below:

Number	Kind	Type	*CashType	PhysicalUnit
1	NA	ESCROW	NULL	P1
2	NA	ESCROW	10€ Var.1	P1
3	NA	ESCROW	10€ Var.2	P1
4	NA	ESCROW	20€	P1
5	NA	ESCROW	50€	P1
6	NA	ESCROW	100€	P1
7	NA	ESCROW	10$	P1
8	NA	ESCROW	20$	P1
9	NA	ESCROW	100$	P1
10	RECYCLE	BILL	10€	P2 & P3
11	RECYCLE	BILL	20€	P4
12	DEPOSIT	BILL	NULL	P5
13	DEPOSIT	BILL	50€	P5
14	DEPOSIT	BILL	100€	P5
15	DEPOSIT	BILL	10$	P5
16	DEPOSIT	BILL	20$	P5
17	DEPOSIT	BILL	100$	P5

**CashType: Although the structure is more complex, in the table above, the said structure is summarized to indicate the type of notes each cassette containes.*

In this case, it could be known both the total amount of notes contained in the Escrow (by the Escrow's counter field) and the detailed amount of each type of notes within the Escrow. The result of adding the counter fields of the L2..L9 cassettes will be L1's.

The application will be capable of distinguishing whether a generic Escrow cassette is being dealt with, by checking if the CashType field is NULL or not. Whether the Escrow cassettes will be implemented in detail will be decided by the device service's developer, not being mandatory. However, the generic cassette will be absolutely necessary to be taken into consideration, that is to say, the cassette whose CashType field's value is set to NULL.

The Status field will be the same for all the cassettes of the Escrow type.

Regarding the DEPOSIT cassettes (L12..L17), the generic one (L12) should be optional since the exact amount of notes within the recycler is necessary to be known. The same goes for the Escrow, when the CashType field is set to NULL, indicating that the cassette type is generic.

9.6 Handling of *null* parameters

If *null* is passed as a method parameter or contained within a parameter class, 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.

9.7 Handling of *null* return values

A value *null* returned as result of a method call or contained within a parameter class, is not allowed, unless explicitly specified for a particular reason.

10 APPENDIX A: CEN/ISSS WORKSHOP 14923:2004 CORE MEMBERS :

DELARUE

DIEBOLD

DYNASTY

IBM

KAL

KEBA

LUTZ WOLF GRUPPE

NCR

NEXUS

SEIKO EPSON CORPORATION

WINCOR - NIXDORF