CEN

WORKSHOP

AGREEMENT

CWA 16008-5

August 2009

ICS 35.240.40

English version

J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - 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: Avenue Marnix 17, B-1000 Brussels

Contents

Foreword

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

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

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

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

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

This CWA is composed of the following parts:

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

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

History

Main differences to CWA 14923-5:2004 are:

- new method to test cash units
- queryCashUnit allowed while in exchange state
- overworked definitions for thresholds
- new way to end exchange without changes
- more precise meaning of dispense and reject cash unit counters
- method to update banknote identification data
- cash-in now allows multiple currencies
- new exchange and acceptance status
- flexible article 6 categorization representation in cash units
- new property to deliver cash-in related information
- additional feature to limit the cashed-in amount
- improved reset information
- Extended denomination handling for cash-out operations
- creation of article 6 reference signatures in a multivendor way
- extended and redesigned position handling
- additional information on how to handle open safe doors

Main differences to CWA 13937-5:2000 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, IJxfsCashDispenserControl, IJxfsATMControl, IJxfsBaseControl**

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

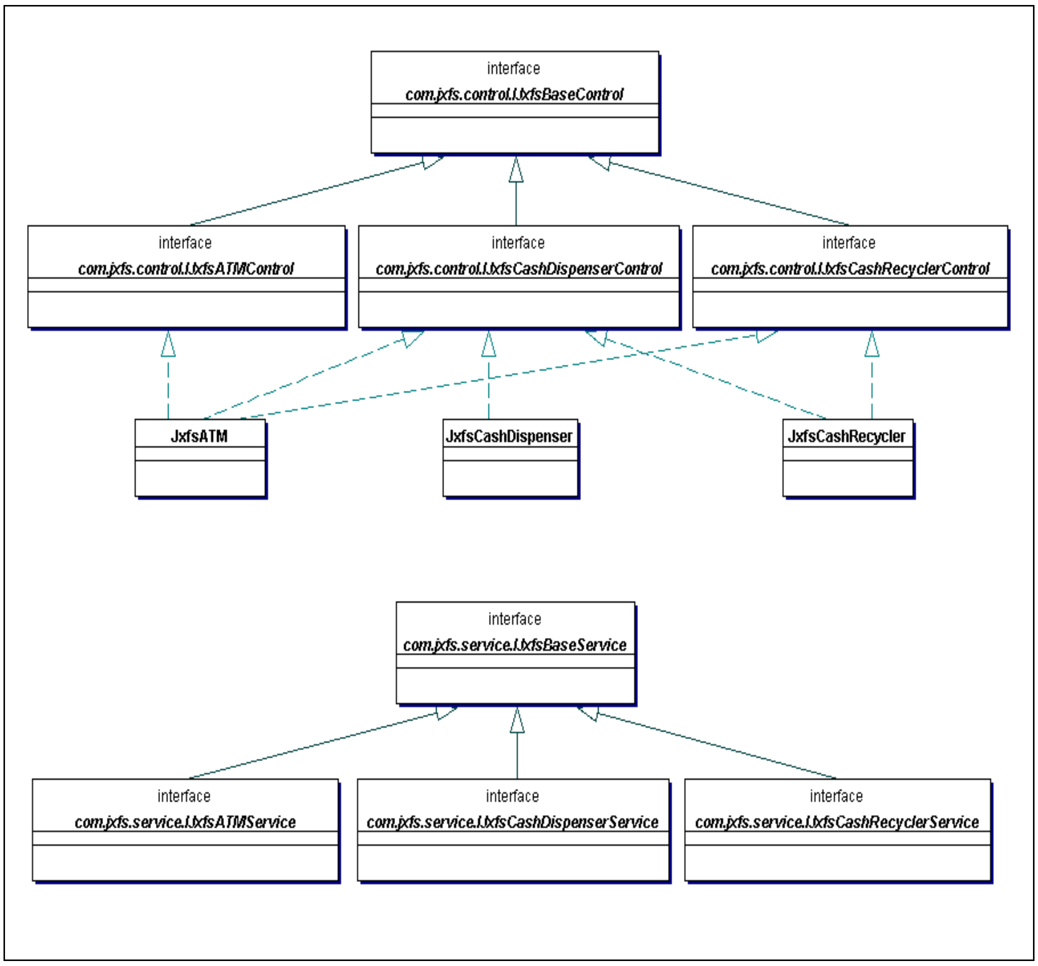
Class or Interface	Name	Description	Extends or Implements
Interface	**IJxfsBaseService**	Base interface for all services.	
Interface	**IJxfsCashDispenserService**	Base interface for all cash dispenser services. Contains method declarations specific to cash dispenser devices.	Extends: **IJxfsBaseService**
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, a *JxfsOperatonCompleteEvent* 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

3.3.1 Summary

Extends	**Implements**
IJxfsBaseControl	

Property	**Type**	**Access**
capabilities	*JxfsCapabilities*	R
mixTable	*java.util.lang.Vector of JxfsMixTable*	RW
uvv	*boolean*	RW
currencies	*java.util.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
endExchange (no parameters)	identificationID
openSafeDoor	identificationID
calibrateCashUnit	identificationID
getDateTime	identificationID
setDateTime	identificationID
queryOrder	identificationID
removeOrder	identificationID
queryCashUnit	identificationID
updateCashUnit	identificationID
reset	identificationID
testCashUnits	identificationID
queryDenominations	identificationID
updateDenominations	identificationID

3.3.2 Properties

3.3.2.1 capabilities (R)

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

3.3.2.2 mixTables (RW)

Type	***java.util.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	***java.util.Vector of JxfsMixTable*** objects Updated property *mixTables*.

3.3.2.3 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.

3.3.2.4 currencies (R)

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

3.3.3 Methods

Following methods are specific to CashDispenser devices.

3.3.3.1 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 *JxfsDenomination*. For 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.

The *denominate()* method calculates the denomination according to the following sequence:

1. The *denomination.cashBox* property is subtracted from the *denomination.amount* property and stored in the *restAmount* variable.
2. The *restAmount* variable is decreased by the amount denoted by the initial denomination.
3. The *returnDenomination* variable is initialized with the content of the initial denomination.

<u>case A:</u> *mixNumber* parameter specifies a denomination (JXFS_C_CDR_MIX_DENOM)

4a the items, amount, and currency passed as input parameters are all checked for consistency. If the items match the amount and currency, and the values requested are dispensable the same JxfsDenomination object is returned as *returnDenomination* (no denomination is performed by the device service in this case).

<u>case B:</u> *mixNumber* parameter specifies an algorithm

4b The *restAmount* is denominated according to the specified algorithm and the result is added to the *returnDenomination* variable.

<u>case C:</u> *mixNumber* parameter specifies a mix table

4c The *restAmount* is denominated according to the specified table as described below and the result is added to the *returnDenomination* variable.

4c.1 if *mixInfo.mixType* property of the table is JXFS_C_CDR_MIX_DENOM an empty denomination is returned.

4c.2 if *mixInfo.mixType* property of the table is JXFS_C_CDR_MIX_TABLE, among all items in the mix table the item with the largest *amount* property which is still less than or equal to the specified amount is selected and used for the returned denomination.

Example: if the table contains items for the amounts of 100, 200, 300 and 400 EUR and the requested *amount* is 320 EUR, the item for the amount of 300 EUR will be selected.

4c.3 if *mixInfo.mixType* property of the table is JXFS_C_CDR_MIX_ALGORITHM, the *returnDenomination* variable is first determined as already described in the case A. Then, the remaining amount is denominated according to the algorithm specified by the *mixInfo.mixAlgorithmType* property of the table and added to the *returnDenomination*.

5. The *JxfsDenomination* object returned in the *JxfsOperationCompleteEvent* is initialized in the following way:
   - The *amount* property is set to the amount property of the *denomination* parameter.
   - The *items* property is set to the *returnDenomination* variable.
   - The *cashBox* property is set to the difference between the *amount* property and the amount specified by the *items* property.

Denominating in the *dispense()* method follows the same rules. The *mixNumber*, *denomination* and *currency* input parameters are bundled together in the *JxfsDispenseRequest* object.

Please, note that if *mixNumber* specifies a table which contains assets which are not defined in the denomination parameter, the operation will fail with JXFS_E_CDR_INVALID_DENOMINATION.

Parameter	**Type**	**Name**	**Description**
	int	mixNumber	Identifies the mix table, algorithm, or denomination verification to use for denomination
	JxfsDenomination	denomination	Specifies the amount to denominate or denomination to verify. It contains the initial (minimal) amounts in the cash box. As already stated in CWA (4.2.12.3.1): *the items included here define the asset used for denominate,* so units included here define the final denomination set with each item indicating the initial (minimal) number of bills/coins which should contribute to the final denomination.
	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* Summary and Description).
data	***JxfsDenomination*** object Specifies the calculated Denomination.

3.3.3.2 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* Summary and Description).
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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.3 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* Summary and Description).
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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.4 startExchange

Syntax ***identificationID startExchange(java.util.Vector units) throws JxfsException;***

Remarks Used to start the exchange of cash units. No other method calls than ***endExchange***, ***close, openSafeDoor, queryCashUnit*** or a *get**Property*** may be performed.

Parameter

Type	**Name**	**Description**
java.util.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* Summary and Description).
data	*JxfsCashUnit* object

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.5 endExchange

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

Remarks On successful completion this method establishes the cash unit configuration, updates the devices associated *JxfsCDRStatus* to reflect the current status of the device and ends the exchange state.

Depending on the capabilities of the attached device, the device service will:

1) validate the supplied *JxfsCashUnit* configuration to ensure that it is consistent
2) validate the supplied *JxfsCashUnit* configuration against the hardware configuration reported by the device
3) assign the supplied *JxfsCashUnit* configuration, if valid, to the device service
4) perform any hardware tests necessary to determine the status of the hardware and/or to allow the cash units to be accessed by the device service. If a hardware error occurs, the operation will complete successfully (i.e. *JXFS_RC_SUCCESSFUL* will be returned). In this case, the *JxfsCDRStatus* object returned by the *IJxfsBaseControl.getStatus( )* method will contain the current status of the hardware component causing the failure.
5) update the devices *JxfsCDRStatus* with the current status of the device

If invalid data is encountered, during the validation tests performed in steps 1) and 2), which can be replaced by the device service using know hardware/software values, the supplied *JxfsCashUnit* will be corrected by the device service and returned in a *JxfsOperationCompleteEvent* with result *JXFS_E_CDR_CASH_UNIT_ERROR*.

If this method does not complete successfully the device remains in an exchange state. It is the responsibility of the operator to correct any problem in order to allow the exchange state to be exited successfully.

On completion the status of the device should be queried, using the *IjxfsBaseControl.getStatus()* method, in order to determine whether the device is operational or not.

If the device service is capable of identifying the available physical cash units, the situation may occur whereby physical units are identified which have no corresponding *JxfsCashUnit* configuration. In this case, the device service will determine as much information as possible from the unconfigured cash unit(s) before appended new cash unit configurations to the list of cash unit configurations given in the supplied *JxfsCashUnit*. Depending on the amount of information which the device service was able to determine these new units may be immediately usable or not. If they are not immediately usable the *JxfsLogicalCashUnit.status* and *JxfsPhysicalCashUnit.status* properties of the new cash unit configurations will be set to *JXFS_C_CDR_LCU_NO_VALUE* signaling that additional configuration is required before the units can be used. The updated configuration will be established as the current cash unit configuration and returned through the *JxfsOperationCompleteEvent.data* property. The presence of unconfigured cash units will not cause the operation to fail.

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	The corresponding ID
result	Common or device dependent error code. (See section on *Error Codes* Summary and Description).
data	*JxfsCashUnit* object Updated cash unit configuration. This information is always returned, regardless of whether the method completes successfully or not.

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
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.6 endExchange

Syntax	***identificationID endExchange() throws JxfsException;***
Remarks	Puts dispenser back into an operational state without modifying the latest known cash unit. It will now accept regular method calls.
Parameter	**none**
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* Summary and Description).
data	*JxfsCashUnit* object Actual 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
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.7 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* Summary and Description).
data	none

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_SAFE_DOOR_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.8 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* Summary and Description).
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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.9 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* Summary and Description).
Data	***java.util.Date*** object Current date and time of device.

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.10 setDateTime

Syntax	***identificationID setDateTime(Date date) throws JxfsException;***
Remarks	Set device date and time. More and more devices are 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* Summary and Description).
data	***java.util.Date*** object Previous date and time of device.

JxfsStatusEvent

JXFS_S_CDR_DATE_TIME_CHANGED
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.11 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* Summary and Description).
data	***java.util.Vector of*** *JxfsDispenseOrder* objects Vector of selected Orders.

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.12 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* Summary and Description).
data	*JxfsDispenseOrder* object Removed Order.

JxfsStatusEvent

JXFS_S_CDR_DELAYED_ORDER_REMOVED
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.13 queryCashUnit

Syntax	***identificationID queryCashUnit() throws JxfsException;***
Remarks	Retrieve the current cash units. Inside an exchange sequence the device service reports the last known cash unit before starting the exchange sequence unless the structure is updated by the device service according to actual hardware knowledge.
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* Summary and Description).
data	*JxfsCashUnit* object Current cash units.

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.14 updateCashUnit

Syntax	***identificationID updateCashUnit(JxfsCashUnit cashUnit) throws JxfsException;***
Remarks	Replace current cash units. When calling this method it is important that the application fills 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* Summary and Description).
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
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.15 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* Summary and Description).
data	none

JxfsIntermediateEvent

JXFS_I_CDR_INPUT_EURART6

This event may only be generated if these two conditions are met: trustedUser is *false* and *reset* is performed within a cash acceptance transaction (between *cashInStart* and *cashInEnd).*

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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.3.3.16 testCashUnits

Syntax	***identificationID testCashUnits (int position) throws JxfsException***
Remarks	This method can be used to test cash units following replenishment. All physical cash units are tested if the dispenser state is JXFS_C_CDR_DIS_OK or JXFS_S_CDR_DIS_CU_STATE and the cash unit is not application locked. The command completes with a JXFS_RC_SUCCESSFUL *JxfsOperationComplete* event if the Device Service successfully manages to test all of the Cash Units which are low or ok regardless of the outcome of the test. This is the case if all the cash units could be tested and a dispense was possible from at least one of the cash units. A JXFS_E_CDR_CASH_UNIT_ERROR Operation Complete event is sent if all the cash unit tests failed. The operation performed to test the cash units is vendor dependent. Items may be dispensed or transported into the reject bin or a recycler bin as a result of this command. This command cannot be used to test cash units that have been application locked. If UVV is activated, this method will behave according to the UVV legislation.

Parameter

Type	**Name**	**Description**
int	position	Specifies the output position to use for presenting money.

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

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

Field	**Value**
operationID	JXFS_O_CDR_TESTCASHUNITS
identificationID	identificationID returned by method.
Result	Common or device dependent error code. (See section on *Error Codes* Summary and Description).
Data	***java.util.Vector of JxfsCashUnitTestError*** Specifies the cash units which failed. Empty if none failed.

3.3.3.17 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 an 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* Summary and Description).
Data	***java.util.Vector*** object A vector of *JxfsDenominationInfo*, one for each different denomination supported by the device.

3.3.3.18 updateDenominations

Syntax	***identificationID updateDenominations(java.util.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

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 an *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* Summary and Description).
Data	***java.util.Vector*** object A vector of *JxfsDenominationInfo* objects. This object contains the list of updated denominations.

3.4 IJxfsCashRecyclerControl

3.4.1 Summary

Extends	Implements
IjxfsBaseControl	

Property	Type	Access
BIMStatus	*int*	R
cashInInfo	*JxfsCDRCashInStatus*	R

Method	Return
cashInStart	identificationID
cashInStart	identificationID
cashIn	identificationID
cashInEnd	identificationID
cashInRollback	identificationID
empty	identificationID
querySignatures	identificationID
updateBIMDataSets	identificationID
createSignature	identificationID

3.4.2 Properties

3.4.2.1 BIMStatus

Type *int*

Remarks Checks the BIM data in the device against the one found in the repository and return

OK_NEWER	BIM data in repository is newer than BIM data in device. Update possible.
OK_OLDER	BIM data in repository is older (!) than BIM data in device. Update possible (but not recommended).
OK_EQUAL	BIM data in the repository is equal to the BIM data in the device. Update possible.
OK_OTHER	BIM data in repository has different currencies, but an update is possible.
NO_SOURCE	Update not possible, no BIM data found in repository.
NO_MATCH	Update not possible, firmware in repository not correct for this device.
NO_SUPPORT	No BIM data update possibility with this device.
INCONSISTENT	The data sets currently stored in the BIM are inconsistent. A possible reason can be that a previous update process was forcedly aborted before completion. This may result in indefinite behaviour concerning recognition of bank notes.

Events If the BIM status changes, no events will be automatically generated.

Exceptions The following exceptions can occur:

JXFS_E_UNREGISTERED	Device is not registered at the DM.
JXFS_E_CLOSED	Device is closed.
JXFS_E_NOHARDWARE	Device is not connected to the workstation.

JXFS_E_REMOTE	Communication error during remote call.

3.4.2.2 cashInInfo

Type	***JxfsCDRCashInStatus***
Remarks	Used to keep complete information about the current/last cash-in transaction. This property is the only way for multivendor purpose to get reliable information about the currently cashed in money.

3.4.3 Methods

Following methods are specific to Recycler devices.

3.4.3.1 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* Summary and Description).
data	None

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.4.3.2 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	Common or device dependent error code. (See section on *Error Codes* Summary and Description).
data	None

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

JxfsIntermediateEvent

JXFS_I_CDR_EURART6_EVENT_POSSIBLE

This is an optional event. If sent by device service it will indicate that not only *cashIn* operation, but also *cashInEnd* operations can fire article 6 events during the *cashIn* transaction that is just starting.

3.4.3.3 cashInStart

Syntax ***identificationID cashInStart(int position, boolean trustedUser, JxfsCDRCashValue maxValues[]) throws JxfsException;***

Remarks

This method is doing the same as the *cashInStart* method without maxValue parameter. The only difference is that this method allows to specify a maximum limit for at least one currency for the money to be cashed in.

If the device service supports limits for accept operations can be queried via *acceptLimit* in the *JxfsCapabilities* class.

If the limit for a certain currency is reached, no more items of this currency are accepted in a *cashIn* operation. The device either stops directly or continues to transport the remaining items into the refund position.

The limit is not counted per *cashIn* execution, but for all accepted items of the cash-in transaction.

If a currency is missing in the *maxValues a*rray no limit applies to that currency.

A JXFS_I_CDR_MAX_VALUE_REACHED intermediate event per *cashIn* command will be generated if
- a banknote will be rejected, because accepting it would exceed one of the given limits or
- after the limit for a currency will be exactly matched by accepted banknotes,

whatever comes first.

If article 6 is activated, some devices cannot guarantee not exceeding the limit, because banknotes that have been recognized as C2/C3 may be deposited and not returned if they are identified after the limit has been reached.

If the execution of a *cashIn* command has resulted in a situation where the limit was reached, no other *cashIn* commands will be executed unless the cash-in transaction has been completed, i. e. ended by *cashInEnd* or cancelled with *cashInRollback*. The *cashIn* command returns with success if there is no other error condition. If the limit is reached any additional *cashIn* commands will be rejected with a JXFS_E_ILLEGAL code.

The reasons for this feature are several national laws and orders like
- the german "Geldwäschegesetz" (money laundering prevention law) that defines special rules if a customer pays in a sum over a limit. To prevent these special rules some banks do not allow a customer to cash in more money as the limit. This limit is dependant on the customer (like in the case private vs. business customer) and therefore is defined by the application.
- insurance contracts that do not allow more than a specified sum in a safe.

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.
JxfsCDRCashValue[]	maxValues	Array of amount limits per currency.

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	Common or device dependent error code. (See section on *Error Codes* Summary and Description).
data	None

JxfsStatusEvent

all status events

JxfsIntermediateEvent

JXFS_I_CDR_EURART6_EVENT_POSSIBLE

This is an optional event. If sent by device service it will indicate that not only *cashIn* operation, but also *cashInEnd* operations can fire article 6 events during the *cashIn* transaction that is just starting.

3.4.3.4 cashIn

Syntax

identificationID cashIn(JxfsCashInOrder order) throws JxfsException

Remarks

Accept cash from the input slot.
This command transports notes from the cashin position 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 refuses 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. If device also has a detector (see *detector c*apability) then some notes may be moved somewhere else (see *cashInInfo* property). Notes in escrow will be held 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.

If *shutterCmd p*roperty in capabilities is *true t*hen:
- If the input is a tray, the application has to ensure the cash is on the tray and the shutter closed before calling *cashIn()* J/XFS operation.
- If the input is a slot, the application must open the shutter and call the *cashIn()* J/XFS operation right after the shutter opened to start cash acceptance.
- When *cashIn* completes, the refused items (if any) are not accessible to the customer and the application has to call *shutterMove* to open/close shutters for the input and refuse positions.

If items remain in the input and refuse position it is the preferred way to clear the input position first and afterwards the refuse position.

Parameter

Type	**Name**	**Meaning**
JxfsCashInOrder	order	Specifies the notes or coins to accept. Depending on the hardware the denomination property may have different values for *cashIn* () operations. For cash-in devices with a banknote identification module the denomination property is not considered by the device service and preferably null. The existence of a banknote identification module is indicated by the detector capability. In case of a cash-in device without detector the properties of the *JxfsDenomination* object will be set as follows: cashbox: The cashbox property is not used and should be set to zero. amount: This value is not used because recyclers without note detector are not able of splitting an amount into denomitation items. It should be set to zero. vector of *JxfsDenominationItem* objects: This property has to be filled accordingly.

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	Common or device dependent error code. (See section on *Error Codes* Summary and Description).
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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_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.

JxfsIntermediateEvent

When the acceptance limit defined by *cashInStart* is reached, this event is sent.

Field	**Value**
OperationID	JXFS_O_CDR_CASH_IN
IdentificationID	The operation's Identification Id.

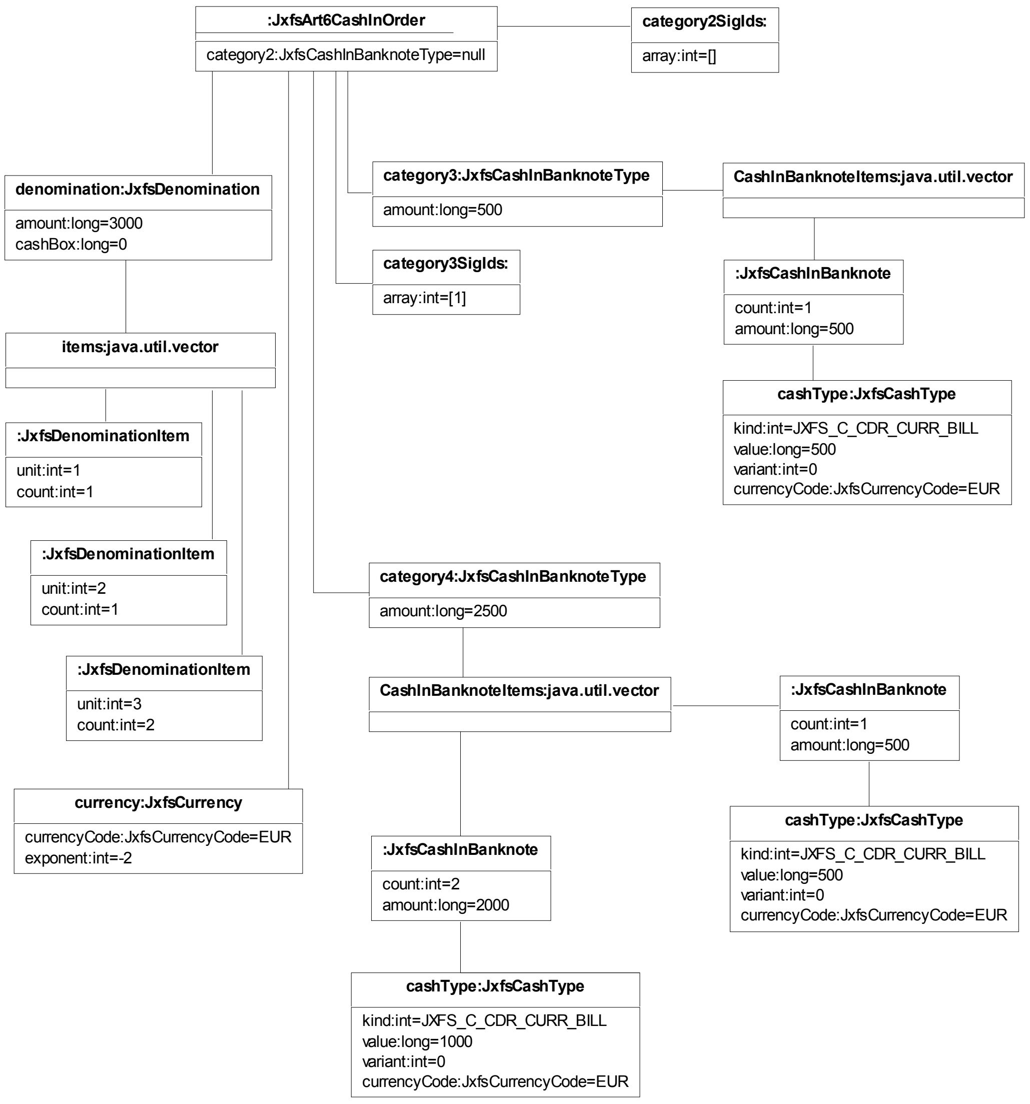
reason:	JXFS_I_CDR_MAX_VALUE_REACHED A banknote will be rejected, because accepting it would exceed the given limit or the limit will be exactly matched by accepted banknotes, whatever comes first.
Data	Always null.

3.4.3.4.1 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

3.4.3.5 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.

<u>**Note:**</u> If a JXFS_I_CDR_INPUT_EURART6 event is generated during execution, the implication is that a customer has agreed on a sum figure which may have since changed if a C2 banknote has been discovered, because a C2 banknote is usually not credited to a customer. An application has to take care that it is vendor and hardware specific, from what moment a sum figure can be taken for granted (either the contents of the ESCROW plus all already deposited money or the deposited money after the end of the *cashIn* operation).

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* Summary and Description).
Data	If the eurArt6Capability capability equals *true*, this will return a ***JxfsArt6CashInOrder*** object with the appropriate information, otherwise, a ***JxfsCashInOrder*** object will be returned. Total amount and Denomination cashed in since *cashInStart*.

JxfsIntermediateEvent

JXFS_I_CDR_INPUT_EURART6

This is an optional event that can be generated depending on vendor and hardware design. See description of the JXFS_I_CDR_EURART6_EVENT_POSSIBLE event for more details on how application can know in advance if this event can be produced or not.

JxfsStatusEvent

JXFS_S_CDR_CASHUNIT_CHANGED
JXFS_S_CDR_CASHUNIT_THRESHOLD
JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.4.3.6 cashInRollback

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

Remarks Moves the cash from the escrow to the rollback position.

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,
In general, if the European article 6 regulations (or other countries equivalent) 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.

If *shutterCmd p*roperty in capabilities is *true,* this command completes when the cash has been moved to the rollback position even when it is not accessible to the customer. Then it is the application responsibility to call *shutterMove* to open/close the shutter.

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	Common or device dependent error code. (See section on *Error Codes* Summary and Description).
data	If the eurArt6Capability capability equals *true*, this field will return a ***JxfsArt6CashInOrder*** object with the appropriate information, otherwise, a ***JxfsCashInOrder*** object will be returned.

see section 9.4 for more details

JxfsIntermediateEvent

JXFS_I_CDR_PARTIAL_DISPENSE
JXFS_I_CDR_INPUT_EURART6

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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.4.3.7 empty – deprecated

Syntax	***identificationID empty(JxfsDispenseRequest dispenseRequest) throws JxfsException;***
Remarks	This method is used to empty the cash device 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* Summary and Description).
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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.4.3.8 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 device.

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* Summary and Description).
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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.4.3.9 querySignatures

Syntax	***identificationID querySignatures(int[] signatureIds) throws JxfsException;***
Remarks	This method queries category 2 and 3 banknote signatures for given signature identification numbers. Since article 6 signatures can accumulate during a deposit transaction, this method should be queried on completion of the transaction in order to ensure the complete set of article 6 signatures are available. This operation succeeds if and only if signatures for all identification numbers specified by the *signatureIds* parameter are available. 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*. 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.

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* Summary and Description).
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.

3.4.3.10 updateBIMDataSets

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

Remarks

A method to trigger a data set update.

If this command is initiated while the BIM status is either NO_SOURCE, NO_MATCH or NO_SUPPORT, the command will fail with JXS_E_CDR_NO_UPDATE_NECESSARY or JXFS_E_CDR_NO_DATA_SET_MATCH (depending on the reason). Some devices may even fail, if the BIM status is OK_OLDER in advance to calling *updateBIMDataSets* () because of security regulations (never update with an older version).

This command may fail, if banknotes are inside the device that are affected by the update. This can be the case if the new data sets do not contain a *JxfsCashType* that has been present in the previous one and at least one banknote of that type is in the device.

After performing this command the configuration of logical cash units may change.

Events

Events, which can be generated by this method.

JxfsOperationCompleteEvent

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

Field	**Value**
operationID	JXFS_O_CDR_UPDATE_BIM_DATA_SETS
identificationID	identificationID returned by method.
result	Common or device dependent error code. (See section on *Error Codes* Summary and Description).
data	***null***

3.4.3.11 createSignature

Syntax ***identificationID createSignature(int inputPosn, int outputPosn) throws JxfsException;***

Remarks This method is used to retrieve detailed information for a single item. This is typically used to obtain the reference signature(s) for an item that is known to be a forgery. These reference signatures can then be compared against those retrieved following one of the methods that create signatures in order to determine whether any forged items were inserted by the customer. The application may have to call this method repeatedly to ensure that all possible signatures are captured.

Check the position capabilities and the capabilities for the signature creation to see if signature creation is supported at all and, if yes, for which positions. The exact processing is dependant on the position design.

Parameter

Type	**Name**	**Description**
int	*inputPosn*	The input position where the reference item should be inserted.
int	*outputPosn*	The output position where the reference item will be presented for removal.

JxfsOperationCompleteEvent

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

Field	**Value**
operationID	JXFS_O_CDR_CREATE_SIGNATURE
identificationID	identificationID returned by method.
result	JXFS_RC_SUCCESSFUL Common or device dependent error code. (See section on Error codes).
data	*JxfsCDRCreateSignatureResult* object. Even if the device in general is able to scan multiple orientations in one operation it may be that not all of these orientations have been scanned in one run due to technical limitations. If not all expected orientations have been scanned the operation will return successfully unless not even one orientation is available.

3.5 IJxfsATMControl

3.5.1 Summary

Extends	Implements
IJxfsBaseControl	

Property	Type	Access	
retractArea	*JxfsRetractArea*	R	deprecated

Method	Return
present	identificationID
reject	identificationID
retract	identificationID
shutterMove	identificationID

3.5.2 Methods

Following methods are specific to ATM devices.

3.5.2.1 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 specific JXFS_S_CDR_CASH_TAKEN status event is generated when the user has removed the bills and position contents are not customer accesible anymore. If no JXFS_S_CDR_POSITION_CHANGE event indicating that position has been emptied 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 JXFS_S_CDR_POSITION_CHANGE status event indicating that position contents are unknown is generated as soon as the bills are available to the user. When shutterCmd property in capabilities is *true*, present method is equivalent to *shutterMove*(*true*,position) for the position used to dispense. Then the application has to close the shutter once the position is empty. Refer to sequence diagrams at the end of the document for usage samples.
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* Summary and Description).
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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.5.2.2 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 ***dispense***, ***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* Summary and Description).
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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.5.2.3 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, cashInRollback or present*** method.

Parameter

Type	**Name**	**Description**
JxfsRetractArea	retractArea	Specifying the retract area to which the notes will be withdrawn. For C2/C3 notes final storage could be different. Check cash unit information after completion to identify final location of notes.

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* Summary and Description).
data	***JxfsCashUnit*** object

JxfsIntermediateEvent

JXFS_I_CDR_INPUT_EURART6

This event may only be generated if these two conditions are met: trustedUser is *false* and *retract* is performed within a cash acceptance transaction (between *cashInStart* and *cashInEnd*).

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
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_CHANGED

3.5.2.4 shutterMove

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

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

If the position is not empty, opening the shutter will move items to a position accessible to the customer, and closing the shutter will move items back if necessary.

This method can only be used if *shutterCmd* property in capabilities is *true*. Otherwise a JXFS_E_NOT_SUPPORTED completion is returned.

In case *shutterCmd* property for a given position is *true*, if application is about to perform a *shutterMove* operation in this position and the physical device decides to implicitely perform the same movement just before, the *shutterMove* job should complete with JXFS_RC_SUCCESSFUL.

Refer to sequence diagrams at the end of the document (chapter *Shutter Handling sequence diagrams*) for usage samples.

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* Summary and Description).
data	none

JxfsStatusEvent

JXFS_S_CDR_DEVICE_STATUS_CHANGED
JXFS_S_CDR_SHUTTER_CHANGED
JXFS_S_CDR_POSITION_CHANGED
JXFS_S_CDR_RESET_STATUS_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.
JxfsCurrencyCode	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.
JxfsCashUnitTestError	Information about failed cash units at test dispense.
JxfsCDRArt6Categories	Indicates present of article 6 categories in cash unit.
JxfsCDRCashInStatus	Information about the current/last cash-in transaction.
JxfsCDRCashValue	Used to specify an amount for a given currency.
JxfsCDRCreateSignatureCapabilities	Capabilities about how to create reference signatures.
JxfsCDRCreateSignatureResult	Result object of a *createSignature* operation.
JxfsCDRReferenceSignature	Object to store the reference signature for one orientation.

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
testCashUnit	*boolean*	R
multipleCurrenciesCashIn Supported	*boolean*	R
acceptLimit	*boolean*	R
deviceOrientation	*JxfsCDRDeviceOrientationEnum*	R
signatureCreation	*JxfsCDRCreateSignatureCapabilities*	R
defaultRollbackPosition	*int*	R
positionsCapabilities	*JxfsCDRPositionCapabilities[]*	R
safeDoorSequence	*JxfsCDRSafeDoorSequenceEnum*	R

Constructor #1	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

Constructor #2	**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
	testCashUnit	boolean
	multipleCurrenciesCashInSupported	boolean
	acceptLimit	boolean
	deviceOrientation	*JxfsCDRDeviceOrientationEnum*
	signatureCreation	*JxfsCDRCreateSignatureCapabilities*
	defaultRollbackPosition	*int*
	positionsCapabilities	*JxfsCDRPositionCapabilities []*
	safeDoorSequence	*JxfsCDRSafeDoorSequenceEnum*

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 explicit shutter handling required. When this property is *true*, the application will be responsible for opening and closing the shutter, using *shutterMove*, for at least one position (see positionsCapabilities for positions). As a device may have positions with different hardware implementations please refer to *JxfsCDRPositionCapabilities.shutterCmd* for guidance for an individual position.

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) - Deprecated

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

4.2.3.3.23 outputPositions (R) - Deprecated

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

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.

4.2.3.3.32 testCashUnit (R)

Type	*boolean*
Remarks	Specifies whether the device service supports the testCashUnit method.

4.2.3.3.33 multipleCurrenciesCashInSupported (R)

Type	***boolean***
Remarks	Indicates if the device supports cash-in of more than one currency in a cash-in operation.

4.2.3.3.34 acceptLimit (R)

Type	***boolean***
Remarks	Specifies if the device service accepts that a cash-in limit will be set by the *cashInStart* method.

4.2.3.3.35 deviceOrientation (R)

Type	***JxfsCDRDeviceOrientationEnum***
Remarks	Provides information how the device processes banknotes. This value is necessary if an application wants to show a customer graphically how to handle the banknotes when performing the *createSignature* handling.

4.2.3.3.36 signatureCreation (R)

Type	*JxfsCDRCreateSignatureCapabilities*
Remarks	Provides the capabilities of the device for creating all necessary reference signature of a category 2 or category 3 banknote.

4.2.3.3.37 defaultRollbackPosition (R)

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

4.2.3.3.38 positionsCapabilities (R)

Type	*JxfsCDRPositionCapabilities[]*
Remarks	Specifies the capabilities of each position supported by the device. An empty array indicates that this value is unknown. Each object in the array reported by this property should contain a unique value for its *position* property, representing a single position. All positions (including default ones) should be part of this array.

4.2.3.3.39 safeDoorSequence (R)

Type	*JxfsCDRSafeDoorSequenceEnum*
Remarks	Valid command sequence for the safe door command.

4.2.3.4 Constructors

4.2.3.4.1 JxfsCapabilities

Syntax	***public JxfsCapabilities(boolean autopresent, int cdType, JxfsEurArt6Capability eurArt6capability, boolean trustedUser, int maxInBills, int maxInCoins, int maxOutBills, int maxOutCoins, boolean compoind, boolean shutterCmd, boolean retract, boolean safeDoorCmd, boolean coins, boolean cylinders, boolean cashBox, boolean refill, boolean dispense, boolean deposit, boolean checkVandalism, boolean intermediateStacker, boolean billsTakenSensor, int inputPositions, int outputPositions, int defaultInputPosition, boolean defaultOutputPosition, boolean silentAlarm, boolean escrow, int escrowSize, boolean detector, boolean baitTrap, java.lang.String vendorData) throws JxfsException***
Remarks	*testCashUnit* will be set to *false*. *multipleCurrenciesCashInSupported* will be set to *false*. *accessLimit* will be set to *false*. *deviceOrientation* will be set to unknown. *signatureCreation* will be set to the default object. *defaultRollbackPosition* will be set to JXFS_C_CDR_POS_NONE. *positionsCapabilities* will be set to an empty array. safeDoorSequence wll be set to unknown.
Exceptions	No additional exceptions are generated by this constructor.

4.2.3.4.2 JxfsCapabilities

Syntax	***public JxfsCapabilities(boolean autopresent, int cdType, JxfsEurArt6Capability eurArt6capability, boolean trustedUser, int maxInBills, int maxInCoins, int maxOutBills, int maxOutCoins, boolean compoind, boolean shutterCmd, boolren retract, boolean safeDoorCmd, boolean coins, boolean cylinders, boolean cashBox, boolean refill, boolean***

dispense, boolean deposit, boolean checkVandalism, boolean intermediateStacker, boolean billsTakenSensor, int inputPositions, int outputPositions, int defaultInputPosition, boolean defaultOutputPosition, boolean silentAlarm, boolean escrow, int escrowSize, boolean detector, boolean baitTrap, java.lang.String vendorData, boolean testCashUnit, boolean multipleCurrenciesCashInSupported, boolean acceptLimit, JxfsCDRDeviceOrientationEnum deviceOrientation, JxfsCDRCreateSignatureCapabilities signatureCreation, int defaultRollbackPosition, JxfsCDRPositionCapabilities positionsCapabilities[], JxfsCDRSafeDoorSequenceEnum safeDoorSequence) throws JxfsException

Remarks

Exceptions Exceptions, which can be generated by this method.

JXFS_E_PARAMETER_INVALID	Generated if one of the following cases applies: - eurArt6Capability is null - deviceOrientation is null - signatureCreation is null - positionsCapabilities is null - safeDoorSequence is null

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	*int*	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	***int***
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	*java.lang.String*	R

Constructor	Parameter	Parameter-Type
JxfsCurrencyCode	*currencyCode*	String

Method	Return
getProperty	*Property*

4.2.10.3 Properties

4.2.10.3.1 currencyCode (R)

Type	***java.lang.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 *JxfsDenominationItems*.
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

JxfsDenominationInfo

4.2.12.5 Usage

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

4.2.12.6 Summary

Extends	**Implements**
JxfsType	

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

Constructor#1	**Parameter**	**Parameter-Type**
JxfsDenominationInfo	*cashType*	*JxfsCashType*
	enableDenomination	boolean

Constructor#2	**Parameter**	**Parameter-Type**
JxfsDenominationInfo	*cashType*	*JxfsCashType*
	enableDenomination	boolean
	enableDenominationDispense	boolean

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

4.2.12.7 Properties:

4.2.12.7.1 cashType (R)

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

4.2.12.7.2 enableDenomination (R/W)

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

4.2.12.7.3 enableDenominationDispense (R/W)

Type	*boolean*
Remarks	Specifies if the denomination is enabled for cash-out or not.

4.2.12.8 Constructors

4.2.12.8.1 JxfsDenominationInfo

Syntax	***public JxfsDenominationInfo(JxfsCashType cashType,boolean enableDenomination) throws JxfsException***
Remarks	*enableDenominationDispense* will be set to *true*.
Exceptions	No additional exceptions are generated by this constructor.

4.2.12.8.2 JxfsDenominationInfo

Syntax	***public JxfsDenominationInfo(JxfsCashType cashType,boolean enableDenomination, boolean enableDenominationDispense) throws JxfsException***
Remarks	
Exceptions	No additional exceptions are generated by this constructor.

4.2.13 JxfsDenominationItem

4.2.13.1 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.

4.2.13.2 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*

4.2.13.3 Properties

4.2.13.3.1 unit (R)

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

4.2.13.3.2 count (R)

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

4.2.14 JxfsDispenseOrder

4.2.14.1 Usage

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

4.2.14.2 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*

4.2.14.3 Properties

4.2.14.3.1 orderID (RW)

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

4.2.14.3.2 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

4.2.14.3.3 denomination (RW)

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

4.2.14.3.4 currency (RW)

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

4.2.14.3.5 when (RW)

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

4.2.14.3.6 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.

4.2.14.3.7 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

4.2.15 JxfsDispenseRequest

4.2.15.1 Usage

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

4.2.15.2 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*

4.2.15.3 Properties

4.2.15.3.1 mixNumber (RW)

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

4.2.15.3.2 denomination (RW)

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

4.2.15.3.3 currency (RW)

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

4.2.15.3.4 position (RW)

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

4.2.16 JxfsEurArt6Capability

4.2.16.1 Usage

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

4.2.16.2 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*

4.2.16.3 Properties

4.2.16.3.1 category2 (R)

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

4.2.16.3.2 category3 (R)

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

4.2.16.3.3 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.

4.2.17 JxfsLogicalCashUnit

4.2.17.1 Usage

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

4.2.17.2 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
containedCategories	*JxfsCDRArt6Categories*	RW

Constructor #1	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

Constructor #2	**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
	containedCategories	JxfsCDRArt6Categori es

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

4.2.17.3 Properties

4.2.17.3.1 cashType (RW)

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

4.2.17.3.2 number (RW)

Type	***int***
Remarks	Logical number of cash unit. Unique number of the cash unit. Once this number is assigned, it identifies the unit along the time; therefore, it can be used to track unit changes, or uniquely reference units in method calls (JxfsDenominationItem unit property is an example).

4.2.17.3.3 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

4.2.17.3.4 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

4.2.17.3.5 unitID (RW)

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

4.2.17.3.6 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.

4.2.17.3.7 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 *endExchange* 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.

4.2.17.3.8 threshold (RW)

Type	***JxfsThreshold***
Remarks	Identifies the software based threshold levels for this logical unit. These levels are compared with the count of the logical unit to evaluate the thresholdStatus. The following rules are applied: • If count >= full, it is considered full. • If full>count>=high, it is considered high. • If low>=count>empty, it is considered low. • If empty>=count, it is considered empty. Otherwise, it is considered ok.

4.2.17.3.9 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.

4.2.17.3.10 devLock (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.

4.2.17.3.11 status (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

4.2.17.3.12 thresholdStatus (RW)

Type	***JxfsThresholdStatus***
Remarks	Specifies the current threshold status as calculated by the device service using the *JxfsLogicalCashUnit.threshold* and *JxfsLogicalCashUnit.count* properties.

4.2.17.3.13 physicalName (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.

4.2.17.3.14 physicalUnits (RW)

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

4.2.17.3.15 depositCount (RW)

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

4.2.17.3.16 dispenseCount (RW)

Type	***int***
Remarks	Number of items from this logical unit which have been presented to the customer. This count will include items retracted from a customer accessible position, using the ***retract*** method, but will not include items which have not been accessible to a customer and are retained from a location within the device using the ***reject*** or ***reset*** methods. This value is persistent on power failure, open, close and system reset. It is initialized by the ***endExchange*** and ***updateCashUnit*** methods.

4.2.17.3.17 rejectCount (RW)

Type	***int***
Remarks	Count of items from this logical unit that have been rejected during any operation which handles items originating from this logical unit. **Note:** Due to the fact that the most common cause of items being rejected is that they are stuck together, this count cannot be guaranteed to be accurate. This value is persistent on power failure, open, close and system reset. It is initialized by the ***endExchange*** and ***updateCashUnit*** methods and updated during any operation which results in items, which have not been accessible to a customer, being rejected.

4.2.17.3.18 containedCategories (RW)

Type	***JxfsCDRArt6Categories***
Remarks	Specifies the categories of notes held by this logical cash unit according to the ECB6 rules. This property is equal to the default JxfsCDRArt6Categories if cuType equals JXFS_C_CDR_LCU_COIN_CYLINDER, JXFS_C_CDR_LCU_COIN_DISPENSER, JXFS_C_CDR_LCU_COUPON, JXFS_C_CDR_LCU_DOCUMENT or JXFS_C_CDR_LCU_ESCROW. See class *JxfsCDRArt6Categories* description for details on how to structure the LCUs.

4.2.17.4 Methods

4.2.17.4.1 addUnit

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

4.2.17.5 Constructors

4.2.17.5.1 JxfsLogicalCashUnit

Syntax	***public JxfsLogicalCashUnit(JxfsCashType cashType, int number, int cuKind, int cuType, java.lang.String unitID, int initialCount, int count, JxfsThreshold threshold, boolean appLock, boolean devLock, int status, JxfsThresholdStatus thresholdStatus, java.lang.String physicalName, java.util.Vector physicalUnits, int depositCount, int dispenseCount, int rejectCount) throws JxfsException***
Remarks	*containedCategories* will be set to the default *JxfsCDRArt6Categories* class.
Exceptions	No additional exceptions are generated by this constructor.

4.2.17.5.2 JxfsLogicalCashUnit

Syntax	***public JxfsLogicalCashUnit(JxfsCashType cashType, int number, int cuKind, int cuType, java.lang.String unitID, int initialCount, int count, JxfsThreshold threshold, boolean appLock, boolean devLock, int status, JxfsThresholdStatus thresholdStatus, java.lang.String physicalName, java.util.Vector physicalUnits, int depositCount, int dispenseCount, int rejectCount, JxfsCDRArt6Categories containedCategories) throws JxfsException***	
Remarks		
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if one of the following cases applies: - *containedCategories* is a null reference

4.2.18 JxfsMixEntry

4.2.18.1 Usage

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

4.2.18.2 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*

4.2.18.3 Properties

4.2.18.3.1 lcu (R)

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

4.2.18.3.2 count (R)

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

4.2.19 JxfsMixInfo

4.2.19.1 Usage

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

4.2.19.2 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**
getProperty	*Property*

4.2.19.3 Properties

4.2.19.3.1 number (R)

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

4.2.19.3.2 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

4.2.19.3.3 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

4.2.19.3.4 name (R)

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

4.2.20 JxfsMixItem

4.2.20.1 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.

4.2.20.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
amount	*long*	RW
entries	*java.util.Vector*	RW

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

Method	**Return**
get*Property*	*Property*
setProperty	*void*

4.2.20.3 Properties

4.2.20.3.1 amount (RW)

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

4.2.20.3.2 entries (RW)

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

4.2.21 JxfsMixTable

4.2.21.1 Usage

Contains complete description of a mix table.

4.2.21.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
mixInfo	*JxfsMixInfo*	RW
items	*java.util.Vector*	RW

Constructor	**Parameter**	**Parameter-Type**
JxfsMixTable	*mixInfo*	*JxfsMixInfo*
	items	java.util.Vector

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

4.2.21.3 Properties

4.2.21.3.1 mixInfo (RW)

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

4.2.21.3.2 items (RW)

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

4.2.22 JxfsPhysicalCashUnit

4.2.22.1 Usage

Information about a physical cash unit.

4.2.22.2 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
configuredCategories	*JxfsCDRArt6Categories*	R

Constructor #1	**Parameter**	**Parameter-Type**
JxfsPhysicalCashUnit	*name*	java.lang.String
	unitID	java.lang.String
	count	int
	threshold	*JxfsThreshold*
	status	int
	thresholdStatus	JxfsThresholdStatus
	lock	boolean

Constructor #2	**Parameter**	**Parameter-Type**
JxfsPhysicalCashUnit	*name*	java.lang.String
	unitID	java.lang.String
	count	int
	threshold	JxfsThreshold
	status	int
	thresholdStatus	JxfsThresholdStatus
	lock	boolean
	configuredCategories	JxfsCDRArt6Categories

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

4.2.22.3 Properties

4.2.22.3.1 name (R)

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

4.2.22.3.2 unitID (R)

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

4.2.22.3.3 count (R)

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

4.2.22.3.4 threshold (R)

Type	*JxfsThreshold*
Remarks	Provides the best estimation for the hardware based threshold levels for this physical unit. If a threshold status cannot be detected by the device, the corresponding level will be returned as -1. Notice that detectable levels have following relationship: full >= high >= low >= empty >= 0.

4.2.22.3.5 status (R)

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

4.2.22.3.6 thresholdStatus (R)

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

4.2.22.3.7 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.

4.2.22.3.8 configuredCategories (R)

Type	***JxfsCDRArt6Categories***
Remarks	Specifies, which article 6 categories this PCU is able to accept. In case of setting this property: If the device service does not support the specified category combination it sets this property to the most probable wanted possible combination. Some devices that are capable of configuring the categories for a cash unit require that this unit is physically empty prior to setting this property.

4.2.22.4 Constructors

4.2.22.4.1 JxfsPhysicalCashUnit

Syntax	***public JxfsPhysicalCashUnit(java.lang.String name, java.lang.String unitID, int count, JxfsThreshold threshold, int status, JxfsThresholdStatus thresholdStatus, boolean lock) throws JxfsException***
Remarks	configuredCategories will be set to the default JxfsCDRArt6Categories class.
Exceptions	No additional exceptions are generated by this constructor.

4.2.22.4.2 JxfsPhysicalCashUnit

Syntax	***public JxfsPhysicalCashUnit(java.lang.String name, java.lang.String unitID, int count, JxfsThreshold threshold, int status, JxfsThresholdStatus thresholdStatus, boolean lock, JxfsCDRArt6Categories configuredCategories) throws JxfsException***	
Remarks		
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if one of the following cases applies: - configuredCategories is a null reference

4.2.23 JxfsRetractArea

4.2.23.1 Usage

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

4.2.23.2 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*

4.2.23.3 Properties

4.2.23.3.1 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

4.2.23.3.2 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

4.2.23.3.3 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).

4.2.24 JxfsThreshold

4.2.24.1 Usage

Defines limits for cassettes.

4.2.24.2 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*

4.2.24.3 Properties

4.2.24.3.1 full (R)

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

4.2.24.3.2 high (R)

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

4.2.24.3.3 low (R)

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

4.2.24.3.4 empty (R)

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

4.2.25 JxfsCashUnitTestError

4.2.25.1 Usage

Information about cash units which failed when a test dispense was attempted.

4.2.25.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
cashUnit	*JxfsPhysicalCashUnit*	R
error	*int*	R

Constructor	Parameter	Parameter-Type
JxfsCashUnitTestError	cashUnit	*JxfsPhysicalCashUnit*
	error	*int*

Method	Return
Ge*tProperty*	*Property*

4.2.25.3 Properties

4.2.25.3.1 cashUnit (R)

Type	***JxfsPhysicalCashUnit***
Remarks	Specifies the physical cash unit which failed when a test dispense was attempted.

4.2.25.3.2 error (R)

Type	*int*
Remarks	Specifies the error which has resulted when a test dispense failed. One of the following values: JXFS_E_CDR_EXCHANGE_ACTIVE JXFS_E_CDR_ NOT_DISPENSABLE JXFS_E_CDR_NO_BILLS JXFS_E_CDR_UNABLE_MOVE_ SHUTTER JXFS_E_CDR_UNIT_LOCKED JXFS_E_CDR_UNIT_FULL JXFS_E_CDR_CASH_DEVICE_ERROR

4.2.26 JxfsCDRArt6Categories

4.2.26.1 Usage

Used in JxfsLogicalCashUnit class to indicate the categories of notes held by the logical cash unit. The corresponding flag is set to *false* if no banknote of the specified category is present in the LCU.

There are two options for representing existance of categories in an LCU:

- per Unit
  This class indicates which categories are present in the LCU per cuType. All LCUs with the same *cuType* value must reference the same *JxfsCDRArt6Categories* values. A typical usage scenario is after deplenishing the unit as the existance of some categories define the further processing of the contents.

- per cashType
  This class reports definite counters per category per cuType per CashType. For each possible category according to the PCU's configuration a single LCU must exist. Using this option may easily lead to very large cash units.

Used in *JxfsPhysicalCashUnit* class to indicate the configured categories of notes that can be held by the physical cash unit. This class may be used to configure the categories to be stored in a PCU if this feature is configurable in the device.

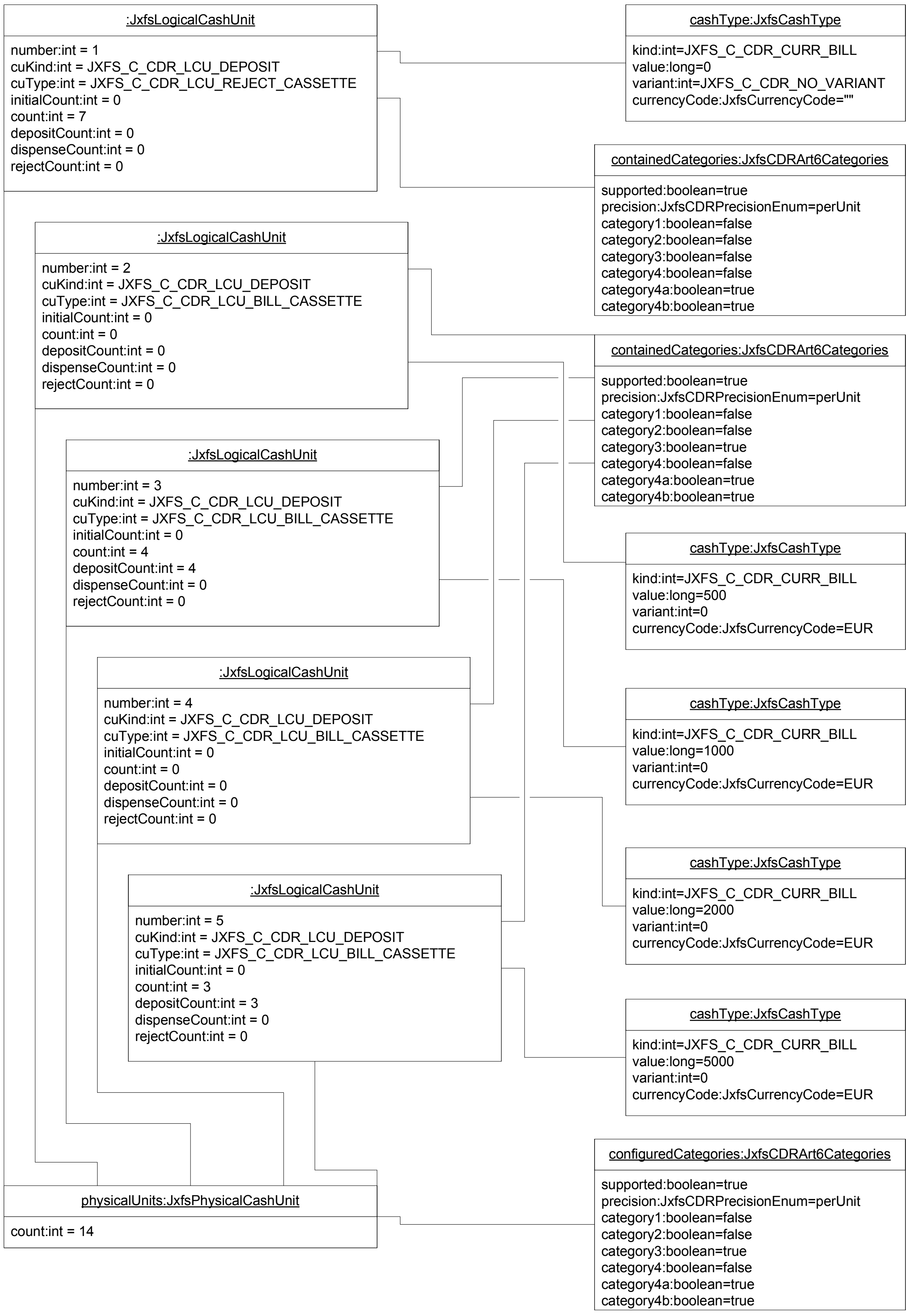
An application should be aware of the possible combinations of the categories in the LCUs and PCUs.

<u>Example #1 (categories per Unit):</u>

We have one PCU that is linked to 5 LCUs (1 for reject, the other 4 for deposit). The PCU received rejected banknotes from a dispense operation and category 3/4a/4b banknotes from a cash-in operation. The device supports fit/unfit categorization.

property	**PCU**	**LCU (REJECT_CASSETTE)**	**LCU (BILL_CASSETTE)**
supported	*true*	*true*	*true*
precision	perUnit	perUnit	perUnit
category1	*false*	*false*	*false*
category2	*false*	*false*	*false*
category3	*true*	*false*	*true*
category4	*false*	*false*	*false*
category4a	*true*	*true*	*true*
category4b	*true*	*true*	*true*

An example of this configuration may look like the following model (not relevant properties are suppressed) that shows an extract of a cash unit.

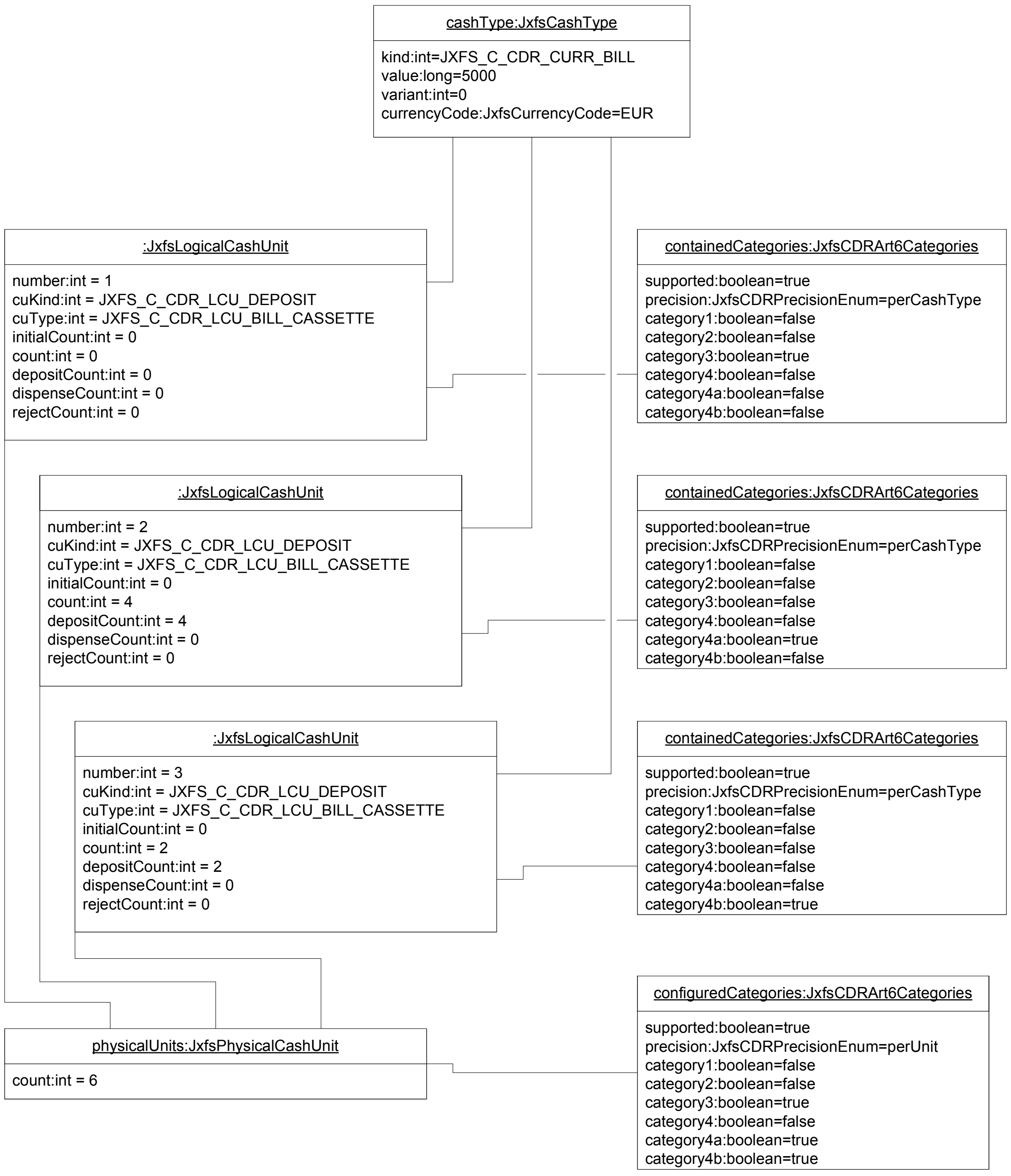

:JxfsLogicalCashUnit
number:int = 1
cuKind:int = JXFS_C_CDR_LCU_DEPOSIT
cuType:int = JXFS_C_CDR_LCU_REJECT_CASSETTE
initialCount:int = 0
count:int = 7
depositCount:int = 0
dispenseCount:int = 0
rejectCount:int = 0
cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=0
variant:int=JXFS_C_CDR_NO_VARIANT
currencyCode:JxfsCurrencyCode=""
containedCategories:JxfsCDRArt6Categories
supported:boolean=true
precision:JxfsCDRPrecisionEnum=perUnit
category1:boolean=false
category2:boolean=false
category3:boolean=false
category4:boolean=false
category4a:boolean=true
category4b:boolean=true
:JxfsLogicalCashUnit
number:int = 2
cuKind:int = JXFS_C_CDR_LCU_DEPOSIT
cuType:int = JXFS_C_CDR_LCU_BILL_CASSETTE
initialCount:int = 0
count:int = 0
depositCount:int = 0
dispenseCount:int = 0
rejectCount:int = 0
containedCategories:JxfsCDRArt6Categories
supported:boolean=true
precision:JxfsCDRPrecisionEnum=perUnit
category1:boolean=false
category2:boolean=false
category3:boolean=true
category4:boolean=false
category4a:boolean=true
category4b:boolean=true
:JxfsLogicalCashUnit
number:int = 3
cuKind:int = JXFS_C_CDR_LCU_DEPOSIT
cuType:int = JXFS_C_CDR_LCU_BILL_CASSETTE
initialCount:int = 0
count:int = 4
depositCount:int = 4
dispenseCount:int = 0
rejectCount:int = 0
cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=500
variant:int=0
currencyCode:JxfsCurrencyCode=EUR
:JxfsLogicalCashUnit
number:int = 4
cuKind:int = JXFS_C_CDR_LCU_DEPOSIT
cuType:int = JXFS_C_CDR_LCU_BILL_CASSETTE
initialCount:int = 0
count:int = 0
depositCount:int = 0
dispenseCount:int = 0
rejectCount:int = 0
cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=1000
variant:int=0
currencyCode:JxfsCurrencyCode=EUR
:JxfsLogicalCashUnit
number:int = 5
cuKind:int = JXFS_C_CDR_LCU_DEPOSIT
cuType:int = JXFS_C_CDR_LCU_BILL_CASSETTE
initialCount:int = 0
count:int = 3
depositCount:int = 3
dispenseCount:int = 0
rejectCount:int = 0
cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=2000
variant:int=0
currencyCode:JxfsCurrencyCode=EUR
cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=5000
variant:int=0
currencyCode:JxfsCurrencyCode=EUR
physicalUnits:JxfsPhysicalCashUnit
count:int = 14
configuredCategories:JxfsCDRArt6Categories
supported:boolean=true
precision:JxfsCDRPrecisionEnum=perUnit
category1:boolean=false
category2:boolean=false
category3:boolean=true
category4:boolean=false
category4a:boolean=true
category4b:boolean=true

Example #2 (LCU counters per category per cashType per cuType):

We have one PCU that is configured to store 50 EUR banknotes either category 3, category 4a or category 4b. The device supports fit/unfit categorization.

property	**PCU**	**LCU #1 (BILL_CASSETTE)**	**LCU #2 (BILL_CASSETTE)**	**LCU #3 (BILL_CASSETTE)**
supported	*true*	*true*	*true*	*true*
precision	perCategory	perCategory	perCategory	perCategory
category1	*false*	*false*	*false*	*false*
category2	*false*	*false*	*false*	*false*
category3	*true*	*true*	*false*	*false*
category4	*false*	*false*	*false*	*false*
category4a	*true*	*false*	*true*	*false*
category4b	*true*	*false*	*false*	*true*

An example of this configuration may look like the following model (not relevant properties are suppressed) that shows an extract of a cash unit.


cashType:JxfsCashType
kind:int=JXFS_C_CDR_CURR_BILL
value:long=5000
variant:int=0
currencyCode:JxfsCurrencyCode=EUR
:JxfsLogicalCashUnit
number:int = 1
cuKind:int = JXFS_C_CDR_LCU_DEPOSIT
cuType:int = JXFS_C_CDR_LCU_BILL_CASSETTE
initialCount:int = 0
count:int = 0
depositCount:int = 0
dispenseCount:int = 0
rejectCount:int = 0
containedCategories:JxfsCDRArt6Categories
supported:boolean=true
precision:JxfsCDRPrecisionEnum=perCashType
category1:boolean=false
category2:boolean=false
category3:boolean=true
category4:boolean=false
category4a:boolean=false
category4b:boolean=false
:JxfsLogicalCashUnit
number:int = 2
cuKind:int = JXFS_C_CDR_LCU_DEPOSIT
cuType:int = JXFS_C_CDR_LCU_BILL_CASSETTE
initialCount:int = 0
count:int = 4
depositCount:int = 4
dispenseCount:int = 0
rejectCount:int = 0
containedCategories:JxfsCDRArt6Categories
supported:boolean=true
precision:JxfsCDRPrecisionEnum=perCashType
category1:boolean=false
category2:boolean=false
category3:boolean=false
category4:boolean=false
category4a:boolean=true
category4b:boolean=false
:JxfsLogicalCashUnit
number:int = 3
cuKind:int = JXFS_C_CDR_LCU_DEPOSIT
cuType:int = JXFS_C_CDR_LCU_BILL_CASSETTE
initialCount:int = 0
count:int = 2
depositCount:int = 2
dispenseCount:int = 0
rejectCount:int = 0
containedCategories:JxfsCDRArt6Categories
supported:boolean=true
precision:JxfsCDRPrecisionEnum=perCashType
category1:boolean=false
category2:boolean=false
category3:boolean=false
category4:boolean=false
category4a:boolean=false
category4b:boolean=true
physicalUnits:JxfsPhysicalCashUnit
count:int = 6
configuredCategories:JxfsCDRArt6Categories
supported:boolean=true
precision:JxfsCDRPrecisionEnum=perUnit
category1:boolean=false
category2:boolean=false
category3:boolean=true
category4:boolean=false
category4a:boolean=true
category4b:boolean=true

4.2.26.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
supported	boolean	R
precision	JxfsCDRPrecisionEnum	R
category1	boolean	R
category2	boolean	R
category3	boolean	R
category4	boolean	R
category4a	boolean	R
category4b	boolean	R

Constructor #1	Parameter	Parameter-Type
JxfsCDRArt6Categories	no Parameter	no Type

Constructor #2	Parameter	Parameter-Type
JxfsCDRArt6Categories	precision	JxfsCDRPrecisionEnum
	category1	boolean
	category2	boolean
	category3	boolean
	category4	boolean
	category4a	boolean
	category4b	boolean

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

4.2.26.3 Properties

4.2.26.3.1 supported (R)

Type	boolean
Remarks	Specifies, if the categorization applies at all. All the other properties are set to *false*, if this property is equal to *false*.

4.2.26.3.2 precision (R)

Type	*JxfsCDRPrecisionEnum*
Remarks	Specifies the precision of the categorization data. For a PCU the value *perUnit* is allowed only. For a LCU the value *perUnit* indicates that the following flags are meant globally per cuType per Unit. For a LCU the value *perCashType* indicates that there are LCUs for each possible category per PCU per cuType per cashType.

4.2.26.3.3 category1 (R)

Type	boolean
Remarks	Specifies that the physical/logical cash unit may contain category1 items.

4.2.26.3.4 category2 (R)

Type	boolean
Remarks	Specifies that the physical/logical cash unit may contain category2 notes.

4.2.26.3.5 category3 (R)

Type	boolean
Remarks	Specifies that the physical/logical cash unit may contain category3 notes. If the machine is not able to distinguish C3 banknotes, this value is always *false*.

4.2.26.3.6 category4 (R)

Type	boolean
Remarks	Specifies that the physical/logical cash unit may contain category4 notes. This should only be used when the validator is unable to identify the 4a and 4b subcategories

4.2.26.3.7 category4a (R)

Type	boolean
Remarks	Specifies that the physical/logical cash unit may contain category4a notes.

4.2.26.3.8 category4b (R)

Type	boolean
Remarks	Specifies that the physical/logical cash unit may contain category4b notes. If the machine is not able to distinguish fit/unit banknotes, this value is always *false*.

4.2.26.4 Constructors

4.2.26.4.1 JxfsCDRArt6Categories

Syntax	***public JxfsCDRArt6Categories() throws JxfsException***
Remarks	Sets all boolean properties to *false*. precision will be set to *perUnit*.
Exceptions	No additional exceptions are generated by this constructor.

4.2.26.4.2 JxfsCDRArt6Categories

Syntax	***public JxfsCDRArt6Categories(JxfsCDRPrecisionEnum precision, boolean category1, boolean category2, boolean category3, boolean category4, boolean category4a, boolean category4b) throws JxfsException***
Remarks	Sets supported to *true*. The other properties will be set according to the contructor parameters.
Exceptions	No additional exceptions are generated by this constructor.

4.2.27 JxfsCDRCashInStatus

4.2.27.1 Usage

This class contains information about the current cash-in transaction or the last cash-in transaction, if no cash-in transaction is currently active.

This value is persistent through power failure. It is always reset with *cashInStart*.

4.2.27.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
cashInStatus	*JxfsCDRCashInEnum*	R
acceptedNoteList	*JxfsArt6CashInOrder*	R
rollbackItems	*JxfsArt6CashInOrder*	R
numOfRefused	*int*	R

Constructor	**Parameter**	**Parameter-Type**
JxfsCDRCashInStatus	cashInStatus	*JxfsCDRCashInEnum*
	acceptedNoteList	*JxfsArt6CashInOrder*
	rollbackItems	*JxfsArt6CashInOrder*
	numOfRefused	*int*

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

4.2.27.3 Properties

4.2.27.3.1 cashInStatus

Type	**JxfsCDRCashInEnum**
Remarks	Information about the current state of the cash-in transaction.

4.2.27.3.2 acceptedNoteList

Type	**JxfsArt6CashInOrder**
Remarks	Accumulated list of all banknotes that have been accepted since the last *cashInStart* operation. This list does not contain refused or rolled back banknotes.

4.2.27.3.3 rollbackItems

Type	**JxfsArt6CashInOrder**
Remarks	List of all banknotes that can be rolled back in the current transaction since cashInStart. If ECB article 6 applies, this list does not contain category 2 or category 3 banknotes. A *cashInRollback* will not necessarily present all banknotes of this list to the customer as there are devices that require several *cashInRollback* operations to return all banknotes. If this list is not empty a *cashInRollback* is possible.

4.2.27.3.4 numOfRefused

Type	**int**
Remarks	Number of items that have been refused in the current or last cashIn transaction. As it is difficult in many cases to give an exact count of items that gave trouble in accepting, an application should not rely on the exact value of this property. It is for statistical reason only. If the number of refused items is not known, the value of this property is JXFS_C_CDR_REFUSED_UNKNOWN. This value is accumulating through several subsequent *cashIn* operations.

4.2.27.3.5 Constructors

Syntax	***public JxfsCDRCashInStatus(JxfsCDRCashInEnum cashInStatus, JxfsArt6CashInOrder acceptedNoteList, JxfsArt6CashInOrder rollbackItems, int numOfRefused) throws JxfsException***	
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if one of the following cases applies: - *cashInStatus* is a null reference - *acceptedNoteList* is a null reference - *rollbackItems* is a null reference - *numOfRefused* is negative and not of the value JXFS_C_CDR_REFUSED_UNKNOWN

4.2.28 JxfsCDRCashValue

4.2.28.1 Usage

Used to specify an amount for a given currency.

4.2.28.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**	**Access**
currencyCode	*JxfsCurrencyCode*	R
amount	*long*	R

Constructor	**Parameter**	**Parameter-Type**
JxfsCurrency	*currencyCode*	JxfsCurrencyCode
	amount	long

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

4.2.28.3 Properties

4.2.28.3.1 currencyCode

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

4.2.28.3.2 amount

Type	***long***
Remarks	Amount in MDUs for this specific currency

4.2.28.4 Constructors

Syntax	***public JxfsCDRCashValue(JxfsCurrencyCode currencyCode, long amount) throws JxfsException***	
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if one of the following cases applies: - currencyCode is a null reference - amount is smaller than 1

4.2.29 JxfsCDRCreateSignatureCapabilities

4.2.29.1 Usage

Provides the capabilities of the device for creating all necessary reference signatures of a category 2 or category 3 banknote.

The default object represents the object to be returned, if it is not (yet) known, what kind of functionality the device supports.

For the effective evaluation of the capabilities the *JxfsCapabilities.deviceOrientation* property is also relevant.

4.2.29.2 Summary

Extends	Implements
JxfsType	

Property	Type	Default Value	Access
supported	*JxfsCDRSupportedEnum*	unknown	R
orientationsToBeScanned	*JxfsCDRNoteOrientation Enum[]*	empty array	R
deviceScanningBothLongside	*boolean*	*false*	R
deviceScanningBothShortside	*boolean*	*false*	R

Constructor 1	Parameter
JxfsCDRCreateSignatureCapa bilities	supported
	orientationsToBeScanned
	deviceScanningBothLongside
	deviceScanningBothShortside

Constructor 2	Parameter
JxfsCDRCreateSignatureCapa bilities	Sets all properties to their default values.

4.2.29.3 Properties

4.2.29.3.1 supported

Type	***JxfsCDRSupportedEnum***
Remarks	Identifies whether the device supports the creation of reference signatures of items. '*supported*' –the device supports the *createSignature* command. '*notSupported*' –the device does not support the *createSignature* command.

4.2.29.3.2 orientationsToBeScanned

Type	***JxfsCDRNoteOrientationEnum []***
Remarks	Array of all orientations of a banknote that have to be scanned in *createSignature* commands for article 6 tracking. This does not refer to the recognition process in a cash-in transaction. This value is preset by the vendor specific validators implementation. One call to *createSignature* may return scans of more than one orientation.

4.2.29.3.3 deviceScanningBothLongside

Type	***boolean***
Remarks	Identifies whether the device supports scans both longside orientations within one *createSignature* call. '*true*' –the device scans all longside orientations in one *createSignature* call. '*false*' –the device provides only one longside orientation scan in a *createSignature* call.

4.2.29.3.4 deviceScanningBothShortside

Type	***boolean***
Remarks	Identifies whether the device supports scans both shortside orientations within one *createSignature* call. '*true*' –the device scans all shortside orientations in one *createSignature* call. '*false*' –the device provides only one shortside orientation scan in a *createSignature* call.

4.2.29.4 Constructors

4.2.29.4.1 JxfsCDRCreateSignatureCapabilities

Syntax	***public JxfsCDRCreateSignatureCapabilities(JxfsCDRSupportedEnum supported, JxfsCDRNoteOrientationEnum orientationsToBeScanned[], boolean deviceScanningBothLongside, boolean deviceScanningBothShortside) throws JxfsException***	
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if one of the following cases applies: - supported is a null reference - orientationsToBeScanned is a null reference - supported is *true* and orientationsToBeScanned is an empty array.

4.2.29.4.2 JxfsCDRCreateSignatureCapabilities

Syntax	***public JxfsCDRCreateSignatureCapabilities() throws JxfsException***
Remarks	This constructor will be used to generate the default object.
Exceptions	No additional exceptions generated.

4.2.30 JxfsCDRCreateSignatureResult

4.2.30.1 Usage

Defines the result of a *createSignature* operation.

4.2.30.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
signatureList	*java.util.List of JxfsCDRReferenceSignature objects*	R
longsideTurned	*boolean*	R
shortsideTurned	*boolean*	R
allOrientationsScanned	*boolean*	R

Constructor	Parameter	Parameter-Type
JxfsCDRCreateSignatureResult	*signatureList*	*java.util.List of JxfsCDRReferenceSignature objects*
	longsideTurned	*boolean*
	shortsideTurned	*boolean*
	allOrientationsScanned	*boolean*

4.2.30.3 Properties

4.2.30.3.1 signatureList

Type	***java.util.List of JxfsCDRReferenceSignature objects***
Remarks	Detailed information for the inserted item. Reference Signature information is included. If no signature could be generated, this is an empty list.

4.2.30.3.2 longsideTurned

Type	***boolean***
Remarks	Identifies whether the device returned the banknote turned via the long side. '*true*' –the device returned a banknote turned via the long side. '*false*' –the device returned the banknote with the same orientation as inserted.

4.2.30.3.3 shortsideTurned

Type	***boolean***
Remarks	Identifies whether the device returned the banknote turned via the short side. '*true*' –the device returned a banknote turned via the short side. '*false*' –the device returned the banknote with the same orientation as inserted.

4.2.30.3.4 allOrientationsScanned

Type	***boolean***
Remarks	Flag if all necessary orientations (see *JxfsCDRCreateSignatureCapabilities*) for this kind of insertion have been scanned accordingly. Reasons for not having all necessary orientations scanned may be technical problems that prevented performing a scan with the required quality or a BIM that requires several runs with the same insertion orientation by design. '*true*' –all orientations for this insertion orientation have been scanned. The application may proceed with another insertion orientation. '*false*' –not all necessary orientations for this insertion have been scanned. The application has to repeat the *createSignature* command with the same insertion orientation.

4.2.30.4 Constructors

4.2.30.4.1 JxfsCDRCreateSignatureResult

Syntax	***public JxfsCDRCreateSignatureResult(java.util.List signatureList, boolean longsideTurned, boolean shortsideTurned, boolean allOrientationsScanned) throws JxfsException***	
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAM ETER_INVALID	Generated if one of the following cases applies: - signatureList is a null reference - signatureList contains other elements as of the class type *JxfsCDRReferenceSignature*

4.2.31 JxfsCDRReferenceSignature

4.2.31.1 Usage

This class represents a record of a scan of one banknote orientation as a result of a *createSignature* operation.

The properties derived from *JxfsCashType* may not represent a banknote (category 2, 3 or 4). Not all validators support reporting a valid cash type in all cases for a reference signature. An example for this case is a currency change like the introduction of the Euro in germany. A 50 DEM banknote that has been reported in a self-service system as category 3 by the cash-in device at 2001-12-28 could be recognized as category 1 after a week, if the system has been switched to EUR currency.

4.2.31.2 Summary

Extends	Implements
JxfsCashType	

Property	Type	Access
orientation	*JxfsCDRNoteOrientationEnum*	R
signature	*byte[]*	R

Constructor	Parameter	Parameter Type
JxfsCDRReference Signature	kind	*int*
	currencyCode	*JxfsCurrencyCode*
	value	*long*
	variant	*int*
	orientation	*JxfsCDRNoteOrientationEnum*
	signature	*byte[]*

4.2.31.3 Properties

4.2.31.3.1 orientation

Type	***JxfsCDRNoteOrientationEnum object***
Remarks	Orientation of the accepted banknote.

4.2.31.3.2 signature

Type	***byte[]***
Remarks	Banknote signature data.

4.2.31.4 Constructors

4.2.31.4.1 JxfsCDRReferenceSignature

Syntax	***public JxfsCDRReferenceSignature(int kind, JxfsCurrencyCode currencyCode, long value, int variant, JxfsCDRNoteOrientationEnum orientation, byte signature[]) throws JxfsException***	
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if one of the following cases applies: - orientation is a null reference - signature is a null reference

4.2.32 JxfsCDRPositionCapabilities

4.2.32.1 Usage

Defines the characteristics of an input/output position.

4.2.32.2 Summary

Extends	**Implements**
JxfsType	

Property	**Type**
position	*int*
shutterStatusSupported	*boolean*
shutterCmd	*boolean*
contentsStatusSupported	*boolean*
maxItems	*int*
mechanicalDesign	*JxfsCDRMechDesignEnum*
input	*boolean*
output	*boolean*
rollback	*boolean*
refusal	*boolean*

Constructor	**Parameter**	**Parameter Type**
JxfsCDRPositionCapabilities	position	*int*
	shutterStatusSupported	*boolean*
	shutterCmd	*boolean*
	contentsStatusSupported	*boolean*
	maxItems	*int*
	mechanicalDesign	*JxfsCDRMechDesignEnum*
	input	*boolean*
	output	*boolean*
	rollback	*boolean*
	refusal	*boolean*

4.2.32.3 Properties

4.2.32.3.1 position (R)

Type	***int***
Remarks	Identification for this position. It can be one of: • JXFS_C_CDR_POS_LEFT • JXFS_C_CDR_POS_CENTER • JXFS_C_CDR_POS_RIGHT • JXFS_C_CDR_POS_FRONT • JXFS_C_CDR_POS_REAR • JXFS_C_CDR_POS_TOP • JXFS_C_CDR_POS_BOTTOM. (Defined as dispense position code)

4.2.32.3.2 shutterStatusSupported (R)

Type	***boolean***
Remarks	Specifies whether shutter status is supported for this position. When this property is *false* the corresponding isNotSupported query will return *true*.

4.2.32.3.3 shutterCmd (R)

Type	***boolean***
Remarks	Defines if the shutter has to be explicitly controlled by the application. When *true*, the application is responsible for opening and closing the shutter using *shutterMove*. If this property is *true* for an output position, then the *autoPresent* capability must 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. Even if *shutterCmd* is *true* a device service may close the shutter automatically. In this case a further close command of the application will return with JXFS_RC_SUCCESSFUL.

4.2.32.3.4 contentsStatusSupported (R)

Type	***boolean***
Remarks	Specifies whether there is a sensor to detect if the position is empty. When this property is *false*, the corresponding isNotSupported query will return *true*.

4.2.32.3.5 maxItems (R)

Type	***int***
Remarks	Maximum number of items which this position can hold. This is not a guaranteed value. It’s an estimation of the number of items that can be held under normal conditions.

4.2.32.3.6 mechanicalDesign (R)

Type	***JxfsCDRMechDesignEnum***
Remarks	Specifies the mechanical design of this position.

4.2.32.3.7 input (R)

Type	***boolean***
Remarks	Specifies whether this position can be used as source for an accept command.

4.2.32.3.8 output (R)

Type	***boolean***
Remarks	Specifies whether this position can be used as target for a *dispense* command.

4.2.32.3.9 rollback (R)

Type	***boolean***
Remarks	Specifies whether this position can be used as target for *cashInRollback* command.

4.2.32.3.10 refusal (R)

Type	***boolean***
Remarks	Specifies whether refused notes can be moved to this position during *cashIn* command.

4.2.32.4 Constructors

4.2.32.4.1 JxfsCDRPositionCapabilities

Syntax	***public JxfsCDRPositionCapabilities(int position, boolean shutterStatusSupported, boolean shutterCmd, boolean contentsStatusSupported, int maxItems, JxfsCDRMechDesignEnum mechanicalDesign, boolean input, boolean output, boolean rollback, boolean refusal) throws JxfsException***	
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if one of the following cases applies: - mechanicalDesign is a null reference.

4.3 Enum Classes

All enumerations are defined in terms of a class. The following describes all enumerated classes.

4.3.1 JxfsCDRPrecisionEnum

This enumerated data type represents the possible reporting modes for article 6 categories in an LCU.

Field	Description
perUnit	LCU: Values of banknotes per category are valid for the same cuType per LCU. PCU: Categorization flags are valid globally for this unit.
perCashType	LCU: Counters are valid per cashType and per cuType per PCU. PCU: This combination is not allowed.

4.3.2 JxfsCDRCashInEnum

This enumerated data type represents the possible states in which an accept transaction can exist.

Field	Description
notActiveItemsAccepted	Transaction completed with items being accepted into the devices' logical/physical unit(s).
notActiveNoItemsAccepted	Transaction completed with no items being accepted into the devices' logical/physical unit(s).
active	Transaction currently active.
activeNoMoreAccept	The transaction is active, but no more items can be accepted. Examples can be when the escrow is full, a necessary category 2 box is full in an ECB 6 configuration, specific error states or a reached cash-in limit.
unknown	The state of the transaction is unknown. This is also the case if there was no cash-in transaction before.

4.3.3 JxfsCDRDeviceOrientationEnum

This enumerated data type represents the hardware capability of the device to process banknotes either short side first or long side first. This value is necessary if an application wants to show a customer graphically how to handle the banknotes.

Field	Description
shortSideFirst	A note is inserted using the short side as the leading edge.
longSideFirst	A note is inserted using the long side as the leading edge.
unknown	The device orientation could not be determined.
notSupported	Neither the device nor the processable items have a predefined orientation (like coin acceptors).

4.3.4 JxfsCDRNoteOrientationEnum

This enumerated data type represents the possible orientations of banknotes entered during an accept transaction.

Field	Description
frontTop	If the note was inserted using the wide side as the leading edge, the note was inserted with the front image facing up and the top edge was inserted first. If the note was inserted using the short side as the leading edge, the note was inserted with the front image facing up and the left edge was inserted first.
frontBottom	If the note was inserted using the wide side as the leading edge, the note was inserted with the front image facing up and the bottom edge was inserted first. If the note was inserted using the short side as the leading edge, the note was inserted with the front image facing up and the right edge was inserted first.
backTop	If the note was inserted using the wide side as the leading edge, the note was inserted with the back image facing up and the top edge was inserted first. If the note was inserted using the short side as the leading edge, the note was inserted with the back image facing up and the left edge was inserted first.
backBottom	If the note was inserted using the wide side as the leading edge, the note was inserted with the back image facing up and the bottom edge was inserted first. If the note was inserted using the short side as the leading edge, the note was inserted with the back image facing up and the right edge was inserted first.
unknown	The orientation of the inserted note could not be determined.
notSupported	The hardware is not capable of determining the orientation.

4.3.5 JxfsCDRSupportedEnum

This enumerated data type represents the possible states to indicate if a certain feature is supported.

Field	Description
supported	Feature is supported.
unknown	It is currently unknown if this feature is supported.
notSupported	Feature is not supported.

4.3.6 JxfsCDRMechDesignEnum

This enumerated data type represents the mechanical design for a given position. For more details on the different position designs see chapter *Position Mechanical Design Notes.*

Field	Description
slot	This position is based on a slot design.
tray	This position is based on a try design.

4.3.7 JxfsCDRContentsStatusEnum

This enumerated data type represents the contents for a given position.

Field	Description
empty	The position is empty.
notEmpty	The position is not empty.
notSupported	The device cannot know if there are any contents in the position.
unknown	The current contents in the position are unknown.

4.3.8 JxfsCDRPositionProcessingProblemsEnum

This enumerated data type represents an indication of any problems that may be affecting a given position

Field	Description
none	There are no problems with the position and it's associated items known.
unknown	Due to a hardware error or other condition, the state of the position cannot be determined.
metallicObjectPresent	The position contains a metallic object (e.g. coin).
foreignObjectPresent	The position contains a foreign object.
tooManyItems	The bunch of items in the position exceeds the capacity of the position and therefore cannot be processed.
mechanicalTrouble	The items at the position cannot be processed because of mechanical problems like jammed banknotes or a bundle wrapped with banderole.
wrongOrientation	Items are inserted, but with a wrong orientation. Banknote acceptors are ususally working either short side first or long side first. Depending on the geometry of the position they may be even entered in a 90 degrees angle where they cannot be processed.

4.3.9 JxfsCDRSafeDoorSequenceEnum

This enumerated data type represents the possible command sequences for the openSafeDoor command.

Field	Description
notSupported	Safe door command not supported.
beforeStartExchange	Safe door must be opened before the exchange operation starts.
afterStartExchange	Safe door must be opened after the exchange operation has started.
beforeOrAfterStartExchange	Safe door can be opened independently of the exchange status of the device.
unknown	It is not known when to call the openSafeDoor command.

4.3.10 JxfsCDRStatusSelectorEnum

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

Extends	Implements
JxfsStatusSelectorEnum	

Field	Returned Type	Description
status	*JxfsStatus*	General status of the device.
currencies	java.util.Vector of *JxfsCurrency*	List of currencies.
cashUnit	*JxfsCashUnitStatus*	The complete cash unit.
BIMStatus	*Integer*	Status of banknote identification module. This status is available only, if the device service implements the cash recycler interface.
cashInInfo	*JxfsCDRCashInStatus*	Information about current acceptance process. This status is available only, if the device service implements the cash recycler interface.
mixtable	java.util.Vector of *JxfsMixTable*	The complete information about all MixTables.
uvv	*Boolean*	Specifies if the UVV is activated or not.
cashTrayStatus	*JxfsCashTrayStatus*	Status of the cash tray (deprecated)
presentStatus	*JxfsPresentStatus* - deprecated	Status of the presenter (deprecated)
deviceStatus	*JxfsDeviceStatus*	Current device status.
dispenseOrderStatus	*JxfsDispenseOrderStatus*	Current dispense order
dispenserStatus	*JxfsDispenserStatus*	Status of the dispenser
intermediateStackerStatus	*JxfsIntermediateStackerStatus*	Intermediate stacker status
safeDoorStatus	*JxfsSafeDoorStatus*	Safe door status
shutterStatus	*JxfsShutterStatus*	Status of the shutter
transportStatus	*JxfsTransportStatus*	Status of the transport unit.
vandalismStatus	*JxfsVandalismStatus*	Vandalism attack status.
exchangeStatus	*JxfsExchangeStatus*	Exchange operation status.
acceptorStatus	*JxfsAcceptorStatus*	Status of the acceptor.
resetStatus	*JxfsCDRResetStatus*	Reset status.
positionsStatus	*JxfsCDRPositionStatus[]*	Status of the positions.

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.
JxfsExchangeStatus	Exchange status.
JxfsAcceptorStatus	Acceptor status.
JxfsCDRResetStatus	Reset status.
JxfsCDRPositionStatus	Status of a position.

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 (deprecated)*	
getCashUnitStatus	*JxfsCashUnitStatus*	
getDeviceStatus	*JxfsDeviceStatus*	
getDispenseOrderStatus	*JxfsDispenseOrderStatus*	
getDispenserStatus	*JxfsDispenserStatus*	
getIntermediateStackerStatus	*JxfsIntermediateStackerStatus*	
getPresentStatus	JxfsPresentStatus	deprecated
getSafeDoorStatus	*JxfsSafeDoorStatus*	
getShutterStatus	*JxfsShutterStatus (deprecated)*	
getTransportStatus	*JxfsTransportStatus*	
getVandalismStatus	*JxfsVandalismStatus*	
getExchangeStatus	*JxfsExchangeStatus*	
getAcceptorStatus	*JxfsAcceptorStatus*	
getResetStatus	*JxfsCDRResetStatus*	
getPositionsStatus	*JxfsCDRPositionStatus[]*	

When there is more than one cash tray, the value returned by getCashTrayStatus is a summary based on the state of each individual tray. The evaluation of this summary must be performed by the device service, based on this table. If no summary evaluation is provided by the device service *JxfsCDRStatus* class should fire a NOT_SUPPORTED exception when accesing getCashTrayStatus.

Summary	Empty	Not empty	Unknown	Not Supported
Not Supported	None	None	None	All
Empty	All	None	None	NA
Unknown	Any	Any	At least one	any
Not empty	Any	At least one	None	any

NA stands for “Not Applicable”

When there is more than one shutter, the value returned by getShutterStatus is a summary evaluated as:

Summary	Closed	Open	Jammed	Unknown
Unknown	Any	Any	Any	At least one
Jammed	Any	Any	At least one	None
Open	Any	At least one	None	None
Closed	All	None	None	None
Not Supported	None	None	None	None

Also the value returned by getPresentStatus is a summary evaluated as:

Summary	Not Presented	Presented	Unknown
Unknown	Any	Any	At least one
Presented	Any	At least one	None
Not Presented	All	None	None
Not Supported	None	None	None

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	NOTE: this value will be *true* whenever the device detects a jam in the shutter. If device is able to report more precise information about this jam, isJammedOpening or isJammedClosing may be *true* as well.
isJammedOpening	boolean	The shutter jammed while trying to open. NOTE: if this value is *true*, then isJammed should return *true*, and isJammedClosing should return *false*.
isJammedClosing	boolean	The shutter jammed while trying to close. NOTE: if this value is *true*, then isJammed should return *true*, and isJammedOpening should return *false*.
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*

5.2.13 JxfsExchangeStatus

Extends	Implements
JxfsType	

Query	Return
isActive	*boolean*
isNotActive	*boolean*
isNotSupported	*boolean*
isUnknown	*boolean*

5.2.14 JxfsAcceptorStatus

5.2.14.1 Usage

Represents the status of the cash acceptor functionality.

5.2.14.2 Summary

Extends	Implements
JxfsType	

Property	Type	Access
acceptorStatus	*int*	R

Constructor	Parameter	Parameter-Type
JxfsAcceptorStatus	*status*	int

Method	Return	Meaning
isOk	*boolean*	The acceptorStatus is JXFS_S_CDR_ACCEPTOR_OK
isCashUnitState	*boolean*	The acceptorStatus is JXFS_S_CDR_ACCEPTOR_CU_STATE
isCashUnitStop	*boolean*	The acceptorStatus is JXFS_S_CDR_ACCEPTOR_CU_STOP
isCashUnitUnknown	*boolean*	The acceptorStatus is JXFS_S_CDR_ACCEPTOR_CU_UNKNOWN
isCashUnitNotSupported	*boolean*	The acceptorStatus is JXFS_S_CDR_ACCEPTOR_CU_NOT_SUPPORTED.

5.2.14.3 Properties

5.2.14.3.1 acceptorStatus

Type	*int*
Remarks	One of the values: JXFS_S_CDR_ACCEPTOR_OK, JXFS_S_CDR_ACCEPTOR_CU_STATE, JXFS_S_CDR_ACCEPTOR_CU_STOP, JXFS_S_CDR_ACCEPTOR_CU_UNKNOWN, JXFS_S_CDR_ACCEPTOR_CU_NOT_SUPPORTED.

5.2.14.4 Constructors

Syntax	***public JxfsAcceptorStatus(int status) throws JxfsException***	
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if the status is not one of: JXFS_S_CDR_ACCEPTOR_OK, JXFS_S_CDR_ACCEPTOR_CU_STATE, JXFS_S_CDR_ACCEPTOR_CU_STOP, JXFS_S_CDR_ACCEPTOR_UNKNOWN.

5.2.15 JxfsCDRResetStatus

5.2.15.1 Usage

Describes whether *reset* is required to return the device to a known operational state and details about the effects of this call. This information can be used by the application to decide if the *reset* can be performed during transaction execution, when the ATM is out of service, or wait for supervisor presence.

5.2.15.2 Summary

Extends	Implements
JxfsType	

.

Property	Type	Access
resetRequired	boolean	R
maxTime	int	R
returnItemsPossible	boolean	R
informationLost	boolean	R

Constructor#1	Parameter	Parameter-Type
JxfsCDRResetStatus	resetRequired	boolean
	maxTime	int
	returnItemsPossible	boolean
	informationLost	boolean

Constructor#2	Parameter	Parameter-Type
JxfsCDRResetStatus	resetRequired	boolean
	returnItemsPossible	boolean
	informationLost	boolean

Method	Return	
getProperty	Property	
isProperty	boolean	

5.2.15.3 Properties

5.2.15.3.1 resetRequired (R)

Type	**boolean**
Remarks	If *true*, the hardware requires a *reset* command which will attempt to return it to a known operational state. Normally, errors are resolved internally by the device service. There are, however, some scenarios in which this automatic recovery may not be performed: • When automatic recovery will cause an observable impact on the customer. In this case, this method allows the application to decide the best time to perform the recovery. • When automatic recovery will cause some valuable information to be lost (e.g. information required to deal with a customer dispute). • When an unrecoverable error has occurred. In this case, the device has to be informed when the error is manually corrected, in order to allow it to perform any device specific activities required to return it to an operational state. This property is set to *true* if and only if such exceptional events occur. If a J/XFS call sends an operation complete event with result = JXFS_E_CDR_RESET_REQUIRED, the *JxfsCDRResetStatus.resetRequired* property will always be *true*. This property could be *true* without a previous operation complete event with result = JXFS_E_CDR_RESET_REQUIRED. If this property is *true* and the device service is not closed or restarted, it will be *true* until a reset command is sent. After calling *reset*, this property becomes *false* if the *reset* performed successfully and the device is operative again or the device requires manual intervention to be recovered.

5.2.15.3.2 maxTime (R)

Type	**int**
Remarks	Maximum estimated time to perform the *reset*, expressed in milliseconds. A value of JXFS_C_CDR_RESET_MAXTIME_UNKNOWN means unknown.

5.2.15.3.3 returnItemsPossible (R)

Type	**boolean**
Remarks	If *true*, the *reset* command may move items to a position accesible by the customer.

5.2.15.3.4 informationLost (R)

Type	**boolean**
Remarks	If *true*, the *reset* command may lose information during the execution and the counters or status could be inaccurate.

5.2.15.4 Constructors

5.2.15.4.1 JxfsCDRResetStatus

Syntax	**JxfsCDRResetStatus(boolean resetRequired, int maxTime, boolean returnItemsPossible, boolean informationLost)**
Exceptions	No exception thrown.

5.2.15.4.2 JxfsCDRResetStatus

Syntax	**JxfsCDRResetStatus(boolean resetRequired, boolean returnItemsPossible, boolean informationLost)**
Exceptions	No exception thrown.
Remarks	Creates a *JxfsCDRResetStatus* with unknown maxTime.

5.2.16 JxfsCDRPositionStatus

5.2.16.1 Summary

Extends	Implements
JxfsType	

.

Property	Type	Access
position	int	R
shutterStatus	*JxfsShutterStatus*	R
contentsStatus	*JxfsCDRContentsStatusEnum*	R
processingProblems	*JxfsCDRPositionProcessingProblemsEnum*	R

Constructor	Parameter	Parameter-Type
JxfsCDRPositionStatus	position	int
	shutterStatuse	*JxfsShutterStatus*
	contentsStatus	*JxfsCDRContentsStatusEnum*
	processingProblems	*JxfsCDRPositionProcessingProblemsEnum*

Method	Return	
getProperty	Property	

5.2.16.2 Properties

5.2.16.2.1 position (R)

Type	int
Remarks	Identification of the position.

5.2.16.2.2 shutter (R)

Type	*JxfsShutterStatus*
Remarks	Status of the shutter.

5.2.16.2.3 contentsStatus (R)

Type	*JxfsCDRContentsStatusEnum*
Remarks	Status of the contents.

5.2.16.2.4 processingProblems (R)

Type	*JxfsCDRPositionProcessingProblemsEnum*
Remarks	Information about problems at the position.

5.2.16.3 Constructors

5.2.16.3.1 JxfsCDRPositionStatus

Syntax	***JxfsCDRPositionStatus(int position, JxfsShutterStatusshutterStatus, JxfsCDRContentsStatusEnum contentsStatus,***

	JxfsCDRPositionProcessingProblemsEnum procesingProblems) throws JxfsException	
Exceptions	Exceptions, which can be generated by this method.	
	JXFS_E_PARAMETER_INVALID	Generated if - shutterStatus is a null reference - contentsStatus is a null reference - processingProblems is a null reference

6 Events

6.1 Intermediate Events

6.1.1 Intermediate Event Code Summary and Description

Value	Description	Value
JXFS_I_CDR_INPUT_EURART6	At least one category 2 or one category 3 banknote has been detected.	6212
JXFS_I_CDR_INPUT_REFUSED	At least one banknote was not recognized during a *cashIn* operation and has been returned to the reject slot.	6209
JXFS_I_CDR_PARTIAL_DISPENSE	A partial dispense occurred.	6144
JXFS_I_CDR_EURART6_EVENT_POSSIBLE	Optional event. Indicates that *cashInEnd* operations can fire article 6 events during the cashin transaction that is just starting.	6801
JXFS_I_CDR_MAX_VALUE_REACHED	Event indicating that a currency limit has been hit.	6802

6.1.2 IJxfsCashDispenserControl Intermediate Events

Methods / Intermediate Events	reset	updateCashUnit	queryCashUnit	removeOrder	queryOrder	setDateTime	getDateTime	calibrateCashUnit	openSafeDoor	endExchange	startExchange	dispenseExec	dispense	denominate
JXFS_I_CDR_PARTIAL_DISPENSE												x	x	
JXFS_I_CDR_INPUT_EURART6	x													

Methods												updateDenominations	queryDenominations	testCashUnits
Intermediate Events														
JXFS_I_CDR_PARTIAL_DISPENSE														x
JXFS_I_CDR_INPUT_EURART6														

6.1.3 IJxfsCashRecyclerControl Intermediate Events

Methods / Error Codes				updateBIMDataSets	querySignatures	empty	cashInRollback	cashInEnd	cashIn	cashInStart
JXFS_I_CDR_INPUT_EURART6									x	
JXFS_I_CDR_INPUT_REFUSED									x	
JXFS_I_CDR_PARTIAL_DISPENSE						x	x			
JXFS_I_CDR_EURART6_EVENT_POSSIBLE										x
JXFS_I_CDR_MAX_VALUE_REACHED						x	x		x	

6.1.4 IJxfsATMControl Intermediate Events

Methods / Error Codes	shutterMove	retract	reject	present
JXFS_I_CDR_INPUT_EURART6		x		

6.1.5 Intermediate Event Details

6.1.5.1 JXFS_I_CDR_INPUT_EURART6

This intermediate event is sent once per operation, when at least a category 2 or category 3 banknote is detected for the first time. It is not regenerated if the same category 2/3 banknote passes the bill validator more than once.
This event can be generated only if these two conditions are met: trustedUser is *false* and operation is executed within a cash acceptance transaction (*cashInStart* and *cashInEnd*).

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	*null* if generated during the execution of an *IJxfsCashRecyclerControl* method, otherwise, a *JxfsArt6CashInOrder* object is returned containing information for all category 2 and category 3 notes (according to this event description above) when no more notes need to be processed.

6.1.5.2 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.

6.1.5.3 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.

6.1.5.4 JXFS_I_CDR_EURART6_EVENT_POSSIBLE

This intermediate event is sent to indicate that Article 6 events may be generated by *cashInEnd* operation within a cashin transaction.

Field	Value
operationID	*operationID* of method initiating this event
identificationID	*identificationID* of method initiating this event
reason	JXFS_I_CDR_EURART6_EVENT_POSSIBLE
data	null

6.1.5.5 JXFS_I_CDR_MAX_VALUE_REACHED

This intermediate event is sent, when inside a *cashIn* operation a banknote will be rejected, because accepting it would exceed the given limit or the limit will be exactly matched by accepted banknotes, whatever comes first.

Field	Value
operationID	*operationID* of method initiating this event
identificationID	*identificationID* of method initiating this event
reason	JXFS_I_CDR_MAX_VALUE_REACHED
data	none

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	Value
JXFS_S_CDR_CASH_AVAILABLE	Cash is available at the device exit slot.	6701
JXFS_S_CDR_CASH_TAKEN	Cash has been removed from the last opened position, and position contents are not accesible (cannot be altered) by the customer.	6192
JXFS_S_CDR_CASH_TRAY_CHANGED	*deprecated* - Content of cash tray changed.	6160
JXFS_S_CDR_CASHUNIT_CHANGED	Cashunit changed.	6153
JXFS_S_CDR_CASHUNIT_CONFIGURATION_CHANGED	The cashunit configuration was changed.	6154
JXFS_S_CDR_CASHUNIT_THRESHOLD	A cashunit threshold was changed.	6155
JXFS_S_CDR_DATE_TIME_CHANGED	Date or time of device changed.	6169
JXFS_S_CDR_DELAYED_DISPENSE	Dispense order delayed.	6156
JXFS_S_CDR_DELAYED_ORDER_CHANGED	Status of delayed dispense order changed.	6702
JXFS_S_CDR_DELAYED_ORDER_REMOVED	A dispense order has been removed from the list of orders.	6703
JXFS_S_CDR_DEVICE_STATUS_CHANGED	Device status changed.	6162
JXFS_S_CDR_DISPENSER_STATUS_CHANGED	Dispenser status changed.	6161
JXFS_S_CDR_INTERMEDIATE_STACKER_CHANGED	Content of intermediate stacker changed.	6163
JXFS_S_CDR_MIXTABLE_CHANGED	Property mixTables has been changed.	6704
JXFS_S_CDR_SAFEDOOR_CHANGED	Status of safe door changed.	6165
JXFS_S_CDR_SHUTTER_CHANGED	*deprecated* - Shutter status has changed.	6158
JXFS_S_CDR_TRANSPORT_CHANGED	Transport mechanism status changed.	6167
JXFS_S_CDR_VANDALISM_CHANGED	Manipulation detected.	6168
JXFS_S_CDR_EXCHANGE_CHANGED	Exchange state changed.	6210
JXFS_S_CDR_ACCEPTOR_STATUS_CHANGED	Status of the acceptor changed.	6311
JXFS_S_CDR_CASH_IN_CHANGED	Information about the current cash-in transaction has changed.	6705
JXFS_S_CDR_RESET_STATUS_CHANGED	Reset status changed.	6189
JXFS_S_CDR_DENOM_INFO_CHANGED	The denomination info changed.	6211
JXFS_S_CDR_POSITION_CHANGED	The status for one of the supported positions has changed.	6213

6.2.2 Status Event Details

6.2.2.1 JXFS_S_CDR_CASH_AVAILABLE

This status event is sent, when cash is available at the device position.

Field	Value
status	JXFS_S_CDR_CASH_AVAILABLE

details	***JxfsDispenseOrderStatus*** object
	For *dispense* operations it contains a dispense order, which can be removed from the output position of the device.
	Property identificationID is used to identify the issuer of the operation.
	For *cashIn* operations it contains a dummy object as no cash information is available at this time:
	`new JxfsDispenseOrderStatus(`
	`new JxfsDispenseOrder(`
	`0, 0, new`
	`JxfsDenomination(new`
	`Vector(), 0, 0), new`
	`JxfsCurrency(new`
	`JxfsCurrencyCode(""), 0), new`
	`Date(), 0),`
	`0);`

6.2.2.2 JXFS_S_CDR_CASH_TAKEN

This status event is sent, when cash is removed from the last opened position.

Field	Value
status	JXFS_S_CDR_CASH_TAKEN

details	***JxfsDispenseOrderStatus*** object
	For *dispense* operations it contains a dispense order, which was removed from the output position of the device.
	Property identificationID is used to identify the issuer of the operation.
	For *cashIn* operations it contains a dummy object as no cash information is available at this time:
	`new JxfsDispenseOrderStatus(`
	`new JxfsDispenseOrder(`
	`0, 0, new`
	`JxfsDenomination(new`
	`Vector(), 0, 0), new`
	`JxfsCurrency(new`
	`JxfsCurrencyCode(""), 0), new`
	`Date(), 0),`
	`0);`

6.2.2.3 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.

6.2.2.4 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.

6.2.2.5 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.

6.2.2.6 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.

6.2.2.7 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.

6.2.2.8 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.

6.2.2.9 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.

6.2.2.10 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

6.2.2.11 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

6.2.2.12 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.

6.2.2.13 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

6.2.2.14 JXFS_S_CDR_MIXTABLE_CHANGED

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

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

6.2.2.15 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)

6.2.2.16 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.

6.2.2.17 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.

6.2.2.18 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.

6.2.2.19 JXFS_S_CDR_EXCHANGE_CHANGED

This status event is sent, if the exchange state changes.

Field	Value
status	JXFS_S_CDR_EXCHANGE_CHANGED
details	*JxfsExchangeStatus* object Current exchange state.

6.2.2.20 JXFS_S_CDR_ACCEPTOR_STATUS_CHANGED

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

Field	Value
status	JXFS_S_CDR_ACCEPTOR_STATUS_CHANGED
details	*JxfsAcceptorStatus* object Current acceptor status.

6.2.2.21 JXFS_S_CDR_CASH_IN_CHANGED

Any information about the current cash-in transaction changed.

Field	**Value**
status	JXFS_S_CDR_CASH_IN_CHANGED
details	*null*

6.2.2.22 JXFS_S_CDR_RESET_STATUS_CHANGED

This status event is sent, if the reset status changes.

Field	**Value**
status	JXFS_S_CDR_RESET_STATUS_CHANGED
details	*JxfsCDRResetStatus* object Current state of reset status.

6.2.2.23 JXFS_S_CDR_DENOM_INFO_CHANGED

This status event is sent, if the denomination info has changed. This is the case if any denomination object has been enabled or disabled for cash-in or cash-out.

Field	**Value**
status	JXFS_S_CDR_DENOM_INFO_CHANGED
details	***null***

6.2.2.24 JXFS_S_CDR_POSITION_CHANGED

This status event is sent, if the state for a position changes.

Field	**Value**
status	JXFS_S_CDR_POSITION_CHANGED
details	*JxfsCDRPositionStatus* object. Status of the position that changes the state.

7 Codes

7.1 Operation Codes

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

7.1.1 IJxfsCashDispenserControl

Value	Method	Value
JXFS_O_CDR_DENOMINATE	*denominate*	6107
JXFS_O_CDR_DISPENSE	*dispense*	6108
JXFS_O_CDR_DISPENSE_EXEC	*dispenseExec*	6109
JXFS_O_CDR_START_EXCHANGE	*startExchange*	6110
JXFS_O_CDR_END_EXCHANGE	*endExchange*	6111
JXFS_O_CDR_OPEN_SAFE_DOOR	*openSafeDoor*	6112
JXFS_O_CDR_CALIBRATE_CASH_UNIT	*calibrateCashUnit*	6113
JXFS_O_CDR_GET_DATE_TIME	*getDateTime*	6119
JXFS_O_CDR_SET_DATE_TIME	*setDateTime*	6120
JXFS_O_CDR_QUERY_ORDER	*queryOrder*	6115
JXFS_O_CDR_REMOVE_ORDER	*removeOrder*	6116
JXFS_O_CDR_QUERY_CASH_UNIT	*queryCashUnit*	6114
JXFS_O_CDR_UPDATE_CASH_UNIT	*updateCashUnit*	6118
JXFS_O_CDR_QUERY_DENOMINATION	*queryDenominations*	6181
JXFS_O_CDR_UPDATE_DENOMINATION	*updateDenominations*	6182
JXFS_O_CDR_RESET	*reset*	6117
JXFS_O_CDR_TESTCASHUNITS	*testCashUnits*	6184

7.1.2 IJxfsCashRecyclerControl

Value	Method	Value
JXFS_O_CDR_CASH_IN_START	*cashInStart*	6121
JXFS_O_CDR_CASH_IN	*cashIn*	6122
JXFS_O_CDR_CASH_IN_END	*cashInEnd*	6123
JXFS_O_CDR_CASH_IN_ROLLBACK	*cashInRollback*	6124
JXFS_O_CDR_EMPTY	*empty*	6125
JXFS_O_CDR_QUERY_SIGNATURES	*querySignatures*	6180
JXFS_O_CDR_UPDATE_BIM_DATA_SETS	*updateBIMDataSets*	6183
JXFS_O_CDR_CREATE_SIGNATURE	*createSignature*	6900

7.1.3 IJxfsATMControl

Value	Method	Value
JXFS_O_CDR_PRESENT	*present*	6126
JXFS_O_CDR_REJECT	*reject*	6127
JXFS_O_CDR_RETRACT	*retract*	6128
JXFS_O_CDR_SHUTTER_MOVE	*shutterMove*	6129

7.2 Error Codes Summary and Description

Value	Description	Value
JXFS_E_CDR_ASSET_UNDEFINED	Due to device error condition the cash unit content can not be determined.	6603
JXFS_E_CDR_CASH_DEVICE_ERROR	An unspecified error occurred.	6073
JXFS_E_CDR_CASH_UNIT_ERROR	A selected cash unit caused an error.	6074
JXFS_E_CDR_CASHIN_ACTIVE	The device has already a *cashInStart* command issued.	6072
JXFS_E_CDR_DELAYED_DISPENSE	Dispense order is delayed.	6077
JXFS_E_CDR_EXCHANGE_ACTIVE	The device is in an exchange state.	6076
JXFS_E_CDR_ILLEGAL_DISPENSE_ORDER	Invalid orderID during *dispenseExec*.	6078
JXFS_E_CDR_ILLEGAL_DISPENSE_REQUEST	Invalid data during *dispense* or *empty*.	6601
JXFS_E_CDR_INPUT_REFUSED	*cashIn* operation failure.	6079
JXFS_E_CDR_INVALID_BILL	Invalid bill detected during *cashIn*.	6082
JXFS_E_CDR_INVALID_CASH_UNIT	Invalid cash unit ID.	6080
JXFS_E_CDR_INVALID_COIN	Invalid coin detected during *cashIn*.	6083
JXFS_E_CDR_INVALID_CURRENCY	*JxfsCurrency* type is not configured.	6081
JXFS_E_CDR_INVALID_DENOMINATION	The sum values for cashbox and cash units do not match the amount specified.	6084
JXFS_E_CDR_INVALID_MIXNUMBER	The number refers to an undefined mix-table or mix-algorithm.	6085
JXFS_E_CDR_INVALID_RETRACT	Retract area is invalid for this system.	6086
JXFS_E_CDR_INVALID_SIGNATURE_ID	A signature Id for which no signature is available is supplied as input parameter.	6144
JXFS_E_CDR_NO_BILLS	There were no items (bills or coins) to handle.	6088
JXFS_E_CDR_NO_CASHIN_STARTED	*cashInStart* was not called.	6089
JXFS_E_CDR_NO_EXCHANGE_ACTIVE	The device is not in an exchange state.	6090
JXFS_E_CDR_NOT_DISPENSABLE	The amount is not dispensable.	6087
JXFS_E_CDR_RESET_REQUIRED	*reset* operation is required.	6091
JXFS_E_CDR_TOO_MANY_BILLS	The request would require too many bills to be dispensed.	6092
JXFS_E_CDR_TOO_MANY_COINS	The request would require too many coins to be dispensed.	6093
JXFS_E_CDR_UNABLE_MOVE_SHUTTER	Shutter could not be moved.	6094
JXFS_E_CDR_UVV_IN_PROCESS	UVV delay is still active for this order.	6095
JXFS_E_CDR_UVV_NOT_DISPENSEABLE	Order is not dispensable due to UVV regulations.	6602

JXFS_E_CDR_NO_UPDATE_NECESSARY	The data sets are up to date. Nothing to do.	6145
JXFS_E_CDR_NO_DATA_SET_MATCH	The device does not allow a download of the provided data sets. Possible reasons are that they are not compatible or the provided data set is older than the one inside the machine.	6146

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.

Constant	Description	Value
JXFS_C_CDR_POS_NONE	No position selected	1
JXFS_C_CDR_POS_DEFAULT	Use configurated position	2
JXFS_C_CDR_POS_LEFT	Use left output side	4
JXFS_C_CDR_POS_CENTER	Use center output side	8
JXFS_C_CDR_POS_RIGHT	Use right output side	16
JXFS_C_CDR_POS_FRONT	Use front output side	32
JXFS_C_CDR_POS_REAR	Use rear output side	64
JXFS_C_CDR_POS_TOP	Use top output side	128
JXFS_C_CDR_POS_BOTTOM	Use bottom output side	256

Constant	Description	Value
JXFS_C_CDR_POS_OVERFLOW	Use overflow cassette	512
JXFS_C_CDR_POS_REJECT	Use reject cassette	1024

8.2 Device Type codes

Constant	Description	Value
JXFS_C_CDR_TYPE_NONE	Device is not defined	6010
JXFS_C_CDR_TYPE_DISPENSER	Device is a Cash Dispenser	6011
JXFS_C_CDR_TYPE_RECYCLER	Device is a Cash Recycler	6012
JXFS_C_CDR_TYPE_ATM	Device is a Automated Teller Machine	6013

8.3 Cash Type codes

Constant	Description	Value
JXFS_C_CDR_CURR_BILL	Item represents a bill	6014
JXFS_C_CDR_CURR_COIN	Item represents a coin	6015

8.4 Cash Type variant code

Constant	Description	Value
JXFS_C_CDR_NO_VARIANT	No cash type variant information available	6050

8.5 CashUnit Kind codes

Constant	Description	Value
JXFS_C_CDR_LCU_NA	Not available; cash unit is missing	6019
JXFS_C_CDR_LCU_DISPENSE	Cash unit can be used for dispense.	6016
JXFS_C_CDR_LCU_DEPOSIT	Cash unit can be used for deposit.	6017
JXFS_C_CDR_LCU_RECYCLE	Cash unit can be used for dispense and deposit.	6018

8.6 CashUnit Type codes

Constant	Description	Value
JXFS_C_CDR_LCU_BAIT_TRAP	Cash unit has bait trap capability.	6020
JXFS_C_CDR_LCU_BILL_CASSETTE	Bill cassette of cash dispenser	6023
JXFS_C_CDR_LCU_COIN_CYLINDER	Cylinder of the coin dispenser	6024
JXFS_C_CDR_LCU_COIN_DISPENSER	Coin dispenser as a whole unit	6025
JXFS_C_CDR_LCU_COUPON	Cassette for coupons or advertising materials	6027
JXFS_C_CDR_LCU_CURRENCY_CASSETTE	Cassette, which may contain various bills with a different denomination for one currency.	6341
JXFS_C_CDR_LCU_DOCUMENT	Cassette for documents	6028
JXFS_C_CDR_LCU_ESCROW	Cassette is an escrow	6029
JXFS_C_CDR_LCU_NA	Not available; cash unit is missing	6019
JXFS_C_CDR_LCU_OVERFLOW_CASSETTE	Overflow cassette of cash dispenser	6022
JXFS_C_CDR_LCU_REJECT_CASSETTE	Reject cassette of cash dispenser	6021
JXFS_C_CDR_LCU_RETRACT_CASSETTE	Retract cassette of cash dispenser	6026

8.7 CashUnit Status codes

Constant	Description	Value
JXFS_C_CDR_LCU_INOP	The cassette or coin cylinder is inoperative.	6036
JXFS_C_CDR_LCU_MISSING	The cassette or coin cylinder is missing.	6037
JXFS_C_CDR_LCU_NO_REF	There is no reference value available for the notes in this cassette. The cash unit needs calibration to be in a usable state.	6039
JXFS_C_CDR_LCU_MANIP	The cash unit is in a state that needs to be confirmed and needs the information to be confirmed via *updateCashUnit* or *endExchange*, by setting this status to JXFS_C_CDR_LCU_OK.	6051
JXFS_C_CDR_LCU_NO_VALUE	The JxfsCashType of the specified cash unit is not available. The application must provide them and set the status to JXFS_C_CDR_LCU_OK via *updateCashUnit* or *endExchange*. If the values of the cash unit are not available and cannot be set by the application using *updateCashUnit* r *endExchange* the status will be	6038

	JXFS_C_CDR_LCU_INOP instead.	
JXFS_C_CDR_LCU_NOT_DISPENSABLE	Cannot dispense from this cassette.	6040
JXFS_C_CDR_LCU_OK	The cash unit is in a good state.	6031
JXFS_C_CDR_LCU_UNKNOWN	The state of the cash unit is unknown.	6030

8.8 Mix Type codes

Constant	Description	Value
JXFS_C_CDR_MIX_ALGORITHM	An algorithm is selected for mixing	6041
JXFS_C_CDR_MIX_TABLE	A table is selected for mixing	6042
JXFS_C_CDR_MIX_DENOM	The current selected *JxfsDenomination* is used.	6043

8.9 Mix Table codes

Constant	Description	Value
JXFS_C_CDR_MXT_NONE	No mix-table specified	6381
JXFS_C_CDR_MXT_TABLE_BASE	Base constant for vendor specific mix tables.	6382

Remark:

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

8.10 Mix Algorithm codes

Constant	Description	Value
JXFS_C_CDR_MXA_NONE	No algorithm selected.	6391
JXFS_C_CDR_MXA_MIN_BILLS	The minimal number of bills is used	6044
JXFS_C_CDR_MXA_EQUAL_EMPTY	All cash units are equally emptied.	6045
JXFS_C_CDR_MXA_ALGORITHM_BASE	Base constant for vendor specific mix algorithm.	6392

Remark:

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

8.11 Retract Area codes

Constant	Description	Value
JXFS_C_CDR_RA_REJECT	Retract to a reject unit.	6511
JXFS_C_CDR_RA_RETRACT	Retract to a retract unit.	6512
JXFS_C_CDR_RA_STACKER	Retract to intermediate stacker.	6513
JXFS_C_CDR_RA_TRANSPORT	Retract to the transport.	6514

8.12 UVV Delayed Order Queue codes

Constant	Description	Value
JXFS_C_CDR_DO_ALL	All orders in all queues.	6049
JXFS_C_CDR_DO_DELAYED	All orders in delay queue.	6047
JXFS_C_CDR_DO_DISPENSABLE	Orders ready for processing.	6046
JXFS_C_CDR_DO_LAQ	All orders in Large Amount Queue.	6048
JXFS_C_CDR_DO_NONE	Order is not in any queue, because of immediate dispense.	6401

8.13 Cash Tray Status codes

Constant	Description	Value
JXFS_S_CDR_CT_EMPTY	Cashtray is empty	6170
JXFS_S_CDR_CT_NOT_EMPTY	Cashtray is not empty	6171
JXFS_S_CDR_CT_NOT_SUPPORTED	A cashtray is not supported	6172
JXFS_S_CDR_CT_UNKNOWN	Cashtray status unknown	6173

8.14 Device Status codes

Constant	Description	Value
JXFS_S_CDR_DS_ON_LINE	Device is online	6174
JXFS_S_CDR_DS_OFF_LINE	Device is offline	6175
JXFS_S_CDR_DS_POWER_OFF	Device has poweroff	6176
JXFS_S_CDR_DS_BUSY	Device is busy	6177
JXFS_S_CDR_DS_NO_DEVICE	No device found	6178
JXFS_S_CDR_DS_USER_ERROR	Device reported an user error	6179
JXFS_S_CDR_DS_HARDWARE_ERROR	Device reported a hardware error	6180

8.15 Dispenser Status codes

Constant	Description	Value
JXFS_S_CDR_DIS_OK	All logical cash units are ok.	6181
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.	6182
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.	6183
JXFS_S_CDR_DIS_CU_UNKNOWN	Due to a hardware error or other condition, the state of the cash units cannot be determined.	6184

8.16 Intermediate Stacker Status codes

Constant	Description	Value
JXFS_S_CDR_IS_EMPTY	Stacker is empty	6185
JXFS_S_CDR_IS_NOT_EMPTY	Stacker is not empty	6186
JXFS_S_CDR_IS_UNKNOWN	Stacker state is unknown	6187
JXFS_S_CDR_IS_NOT_SUPPORTED	A stacker is not supported	6188

8.17 Safe Door Status codes

Constant	Description	Value
JXFS_S_CDR_SD_NOT_SUPPORTED	A safedoor is not supported	6193
JXFS_S_CDR_SD_OPEN	Safedoor is open	6194
JXFS_S_CDR_SD_CLOSED	Safedoor is closed	6195
JXFS_S_CDR_SD_LOCKED	Safedoor is locked	6196
JXFS_S_CDR_SD_UNKNOWN	Safedoor state is unknown	6197

8.18 Shutter Status codes

Constant	Description	Value
JXFS_S_CDR_SHT_CLOSED	Shutter is closed	6198
JXFS_S_CDR_SHT_OPEN	Shutter is open	6199
JXFS_S_CDR_SHT_JAMMED	Shutter is malfunctional	6200
JXFS_S_CDR_SHT_NOT_SUPPORTED	A shutter is not supported	6201
JXFS_S_CDR_SHT_UNKNOWN	Shutter state is unknown	6202

8.19 Transport Status codes

Constant	Description	Value
JXFS_S_CDR_TP_OK	Transport is working	6203
JXFS_S_CDR_TP_INOP	Transport is not working	6204
JXFS_S_CDR_TP_NOT_SUPPORTED	A transport unit is not supported	6205
JXFS_S_CDR_TP_UNKNOWN	State of transport unit is unknown	6206

8.20 Vandalism Status codes

Constant	Description	Value
JXFS_S_CDR_VAN_MANIPULATION	A manipulation was detected	6207
JXFS_S_CDR_VAN_NO_MANIPULATION	No manipulation was detected	6208
JXFS_S_CDR_VAN_NOT_SUPPORTED	A vandalism check is available	6501

8.21 Present Status codes - deprecated

Constant	Description	Value
JXFS_S_CDR_PR_UNKNOWN	It is unknown if the money could be accessed by the customer.	6189
JXFS_S_CDR_PR_NOT_PRESENTED	The money was not presented.	6190
JXFS_S_CDR_PR_PRESENTED	The money was presented. This value is set as soon as the bills are accessible by the customer.	6191
JXFS_S_CDR_CASH_TAKEN	The cash was taken by the user.	6192

8.22 BIM Status codes

Constant	Description	Value
INCONSISTENT	The stored data sets are inconsistent.	914

8.23 JxfsCashInOrder codes

Constant	Description	Value
JXFS_C_CDR_NOT_APPLICABLE	This value is not applicable in this context	6521

8.24 Exchange Status codes

Constant	Description	Value
JXFS_S_CDR_EXC_ACTIVE	Exchange state is active.	6531
JXFS_S_CDR_EXC_NOT_ACTIVE	Exchange state is not active.	6532
JXFS_S_CDR_EXC_NOT_SUPPORTED	Reporting the exchange state is not supported.	6533
JXFS_S_CDR_EXC_UNKNOWN	The current state is not known.	6534

8.25 Acceptor status codes

Constant	Description	Value
JXFS_S_CDR_ACCEPTOR_OK	The acceptor is operational and all cash units that may be involved for deposit are in good state.	6541
JXFS_S_CDR_ACCEPTOR_CU_STATE	The acceptor is operational, notes can still be deposited but one or more of the cash units are not in good state.	6542
JXFS_S_CDR_ACCEPTOR_CU_STOP	The acceptor is not operational, no notes can be deposited.	6543
JXFS_S_CDR_ACCEPTOR_CU_UNKNOWN	Due to a hardware error or other condition, the state cannot be determined.	6544
JXFS_S_CDR_ACCEPTOR_CU_NOTSUPPORTED	The report of acceptor status is not supported. This value has sense only if the deposit capability is *true*.	6545

A unit is considered to be in 'good state' when the thresholdStatus is JXFS_S_BIN_OK, JXFS_S_BIN_LOW or JXFS_S_BIN_EMPTY, and the status is JXFS_C_CDR_LCU OK or JXFS_C_CDR_LCU_NOT_DISPENSABLE

8.26 Cash-In Status codes

Value	Description	Value
JXFS_C_CDR_REFUSED_UNKNOWN	The number of refused banknotes is unknown.	6551

8.27 Reset Status Codes

Value	Description	Value
JXFS_C_CDR_RESET_MAXTIME_UNKNOWN	Unknown Maximum Estimated Time to perform *reset*.	-1

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.

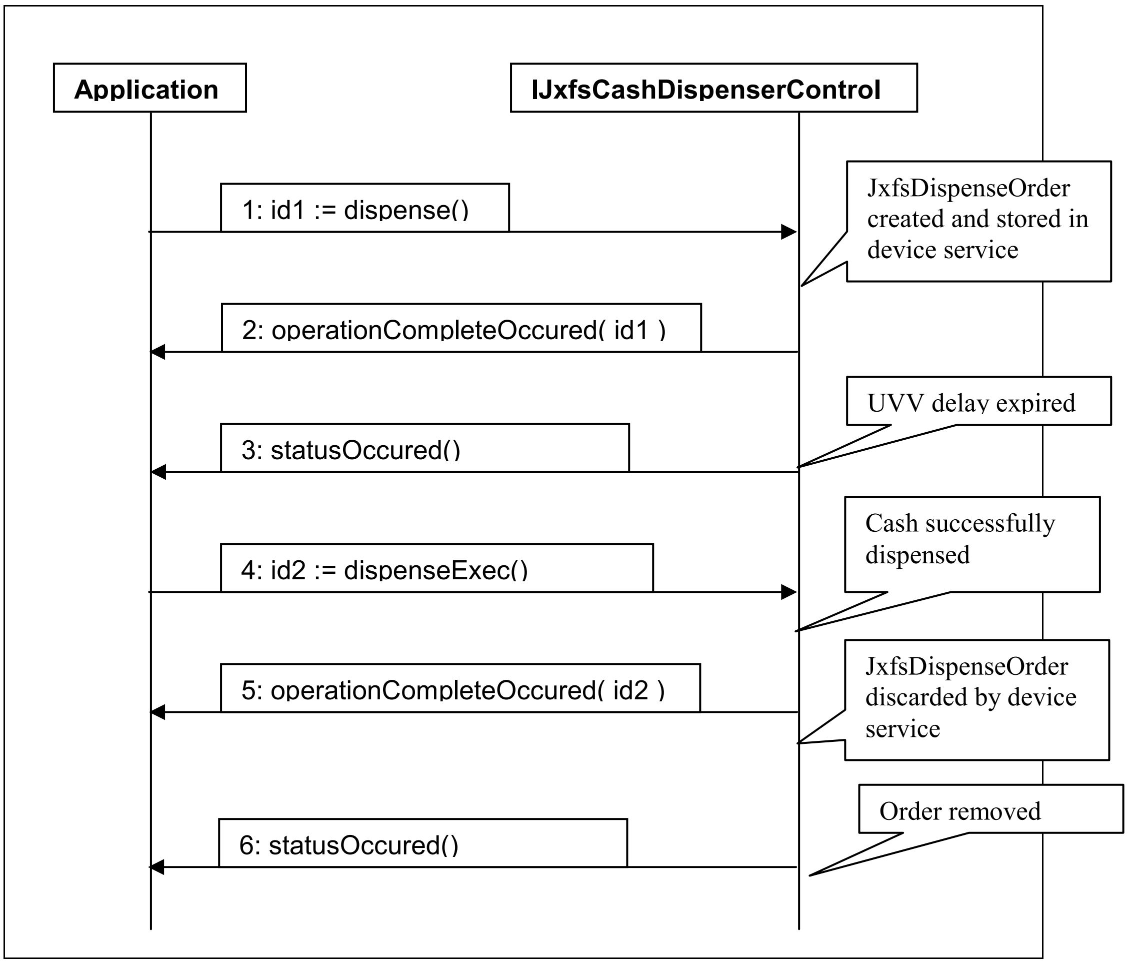
9.2.2 Delayed dispense in J/XFS

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()
JxfsDispenseOrder discarded by device service
5: operationCompleteOccured(id2)
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.

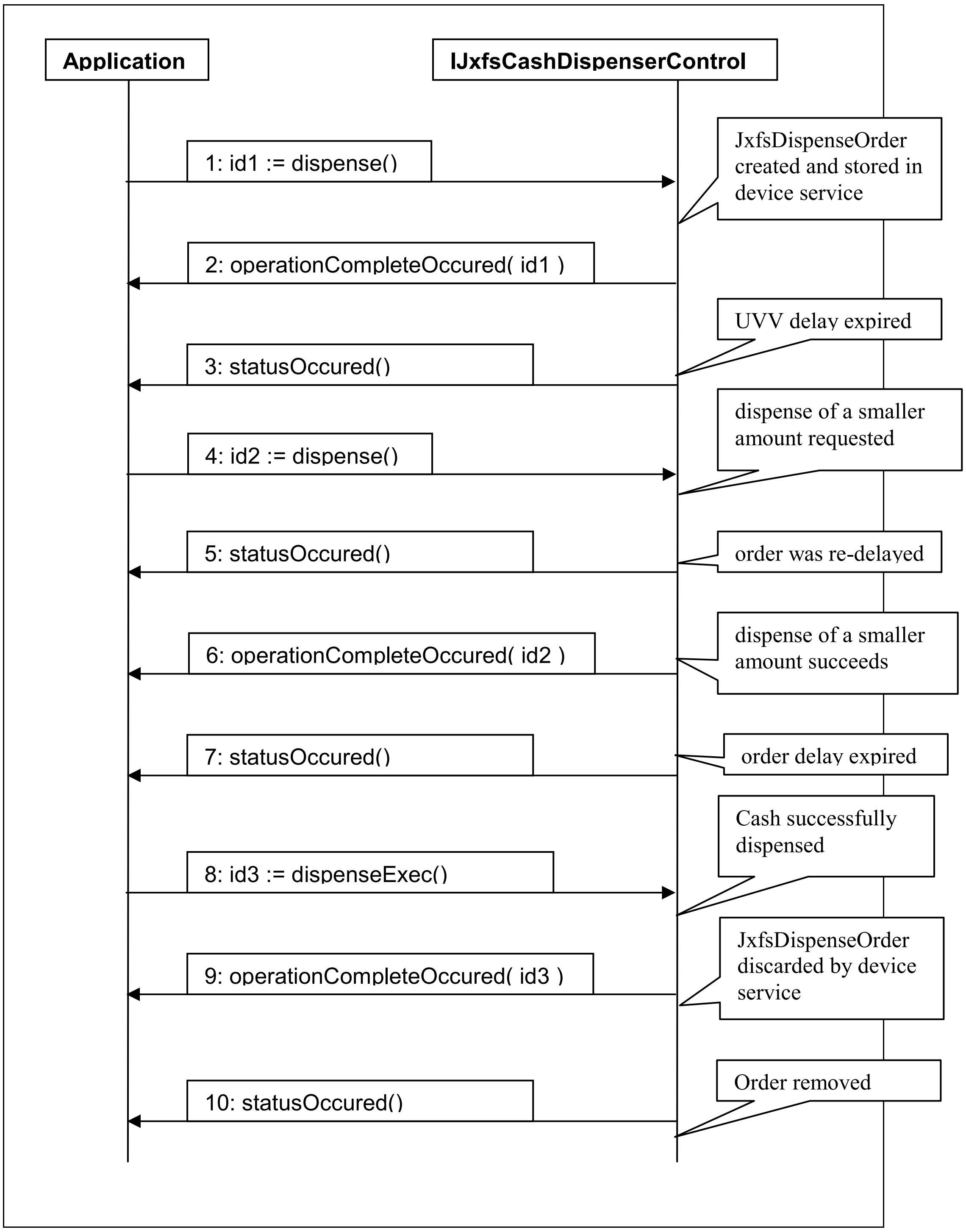
During the execution the cash is dispensed to the exit slot of the device and a *JxfsOperationCompleteEvent* is sent to the caller (5). The *result* property of the event is set to the JXFS_RC_SUCCESSFUL value. The *data* property contains a copy of the *JxfsDispenseOrder* object representing the dispense 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()
9: operationCompleteOccured(id3)
JxfsDispenseOrder discarded by device service
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.

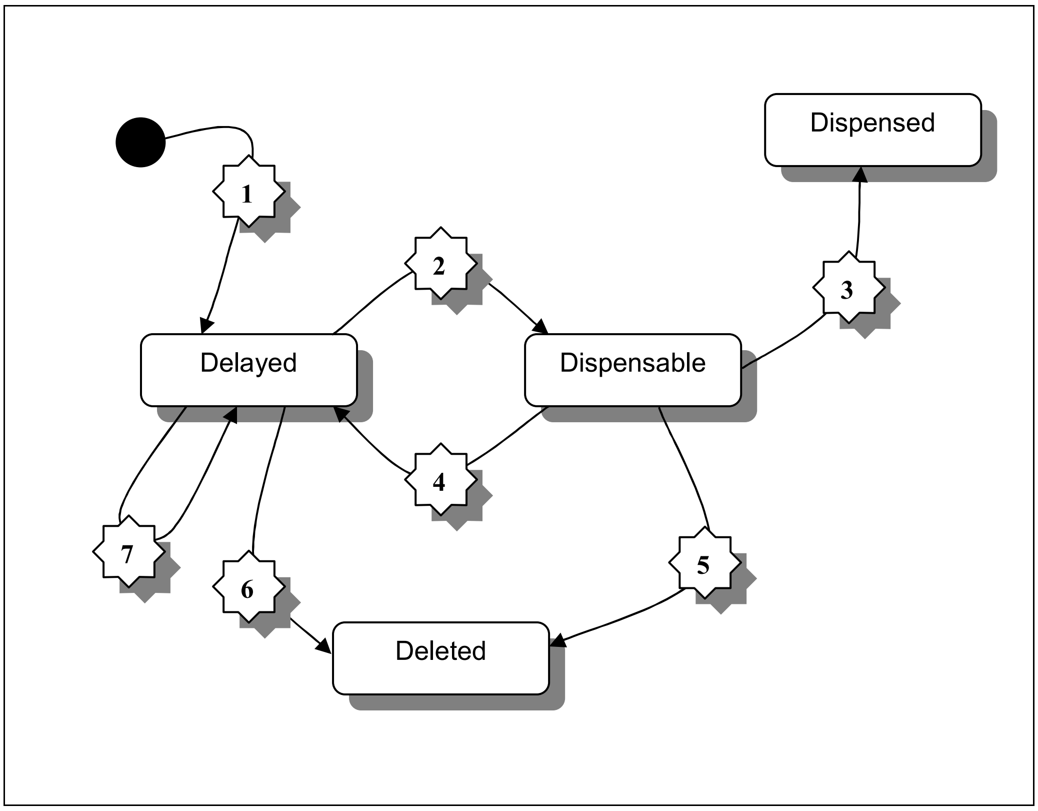
The JXFS_E_CDR_UVV_IN_PROCESS error code can be sent as the *result* property of the *JxfsOperationCompleteEvent* of the *dispenseExec* and indicates that the requested dispense order isn’t dispensable yet. The *data* property of the event contains a copy of the internally stored *JxfsDispenseOrder* 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 "RECYCLING OF EURO BANKNOTES : FRAMEWORK FOR THE DETECTION OF COUNTERFEITS AND FITNESS SORTING BY CREDITINSTITUTIONS AND OTHER PROFESSIONAL CASH HANDLERS" of January 2005. These rules are generally called "Article 6."
European Article 6 defines 4 categories of notes for customer-operated machines and the rules how to handle them:

Category	Classification	Properties	Treatment
1	Not a banknote, not recognised as euro banknote.	Not detected as a banknote because of: • Wrong image or format; • Transportation error. (e.g. double feeds, etc.); • Large dog-ears or missing parts; • Hand-drafted banknnotes, separating cards, etc.; or • Non-euro currency.	Return to customer
2	Objects identified as suspect counterfeit euro banknotes	Image and format recognised, but one or more authentication features missing or clearly out of tolerance.	To be withdrawn from circulation. To be handed over for authentication – together with information on the account holder – to the competent national authorities as soon as possible, in line with national regulations, at the latest 20 working days after deposit in a machine. Not to be credited to account holder.
3	Euro banknotes not clearly authenticated.	Image and format recognised, but not all authentication features recognised because of quality and/or tolerance deviations. In most cases damaged or soiled banknotes.	The banknotes have to be processed separately and transported to the competent national authorities for authentication as soon as possible, in line with national regulations, at the latest 20 working days after deposit in a machine.2) The information on the account holder has to be stored for eight weeks after the banknotes have been detected by the machine. This information shall be made available on request. Alternatively, in agreement with the competent national authorities, the information allowing the traceability of the

			account holder can be handed over together with the category 3 banknotes to the authorities. May be credited to account holder.
4a	Euro banknotes identified as genuine and fit.	All authentication and fitness checks supported by the machine delivered positive results.	Can be used for recycling. To be credited to account holder.
4b	Euro banknotes identified as genuine and unfit.	All authentication checks supported by the machine delivered positive results. Fitness checks supported by the machine delivered negative results.	Shall not be used for recycling and shall be returned to the NCB. 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:

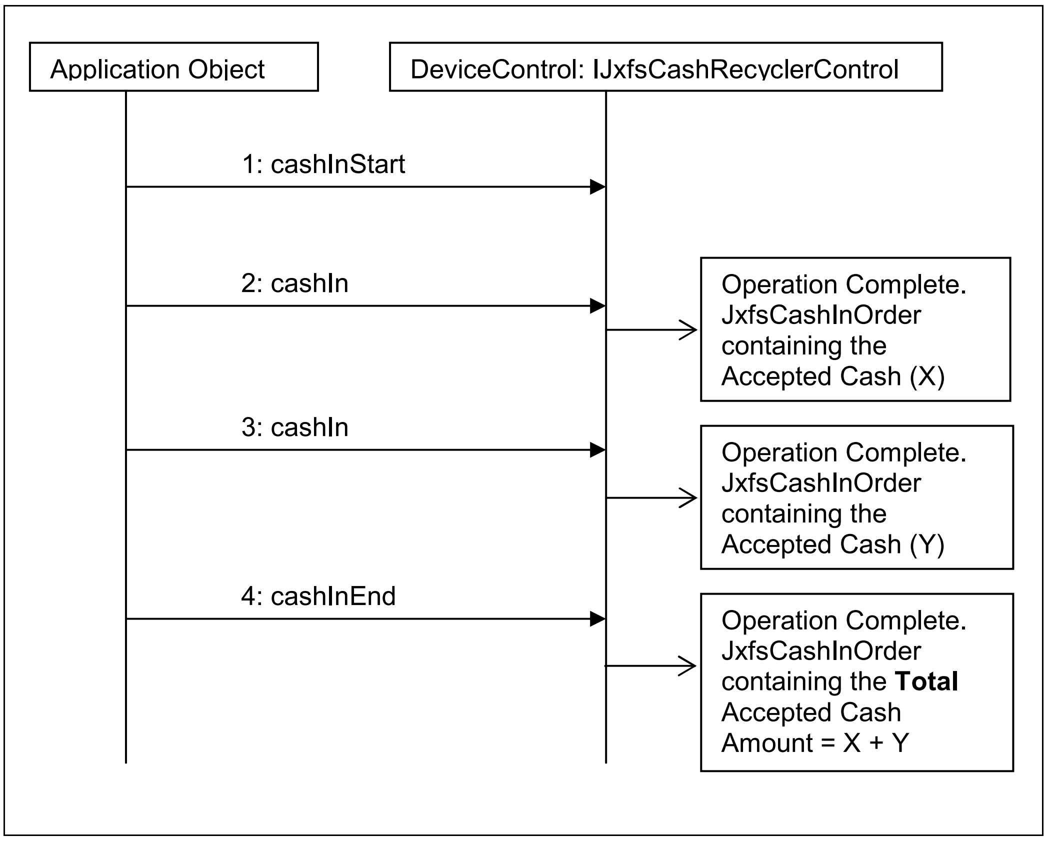
- For each cash deposit and for each category of note, the complete set of a bank note parameters should be returned to the application.
    - After cash deposit operations, the number and kind of category 2 and 3 banknotes must be reported to the application, thus enabling it to perform the corresponding 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.

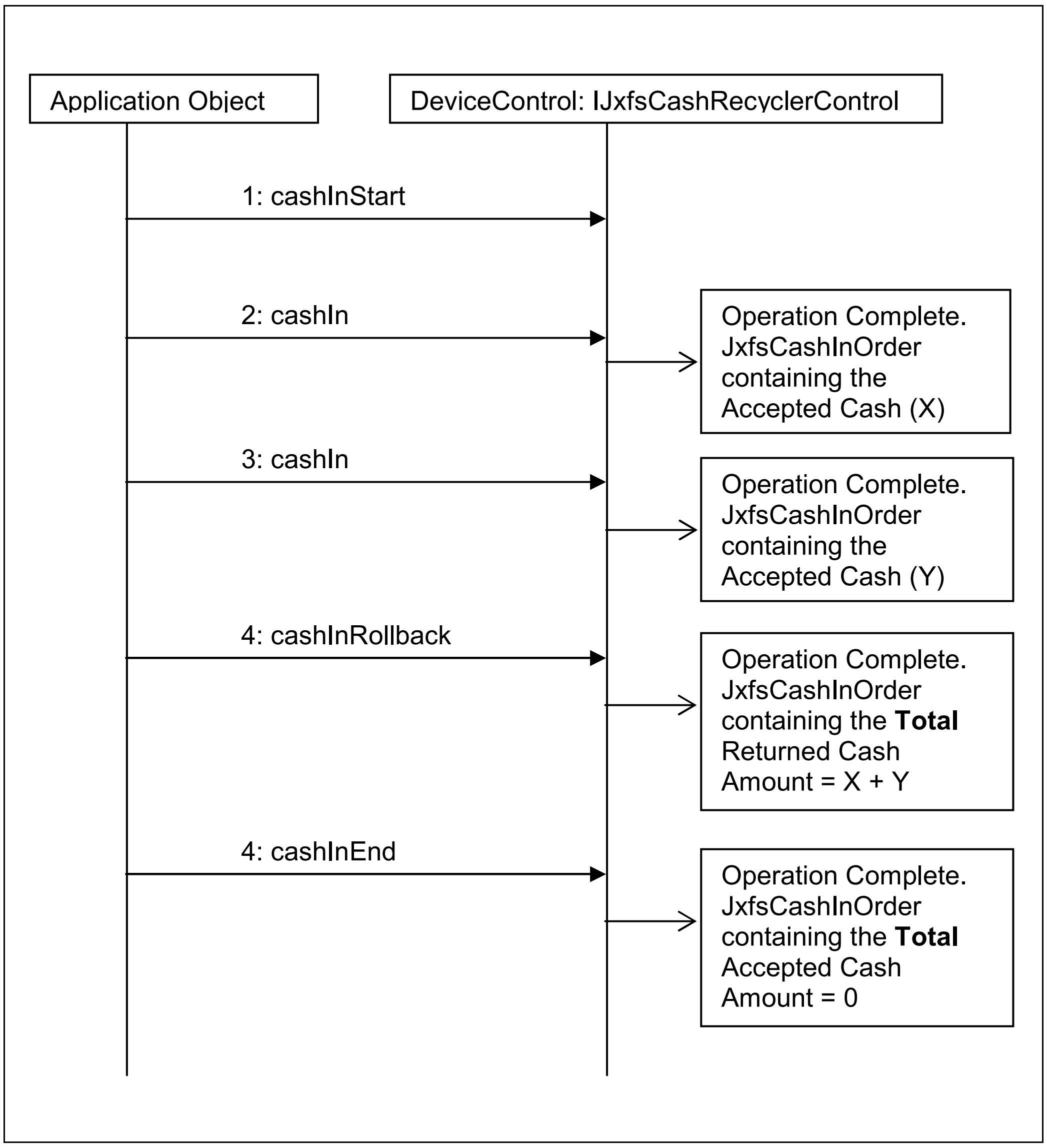
9.4.1 Normal operating

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

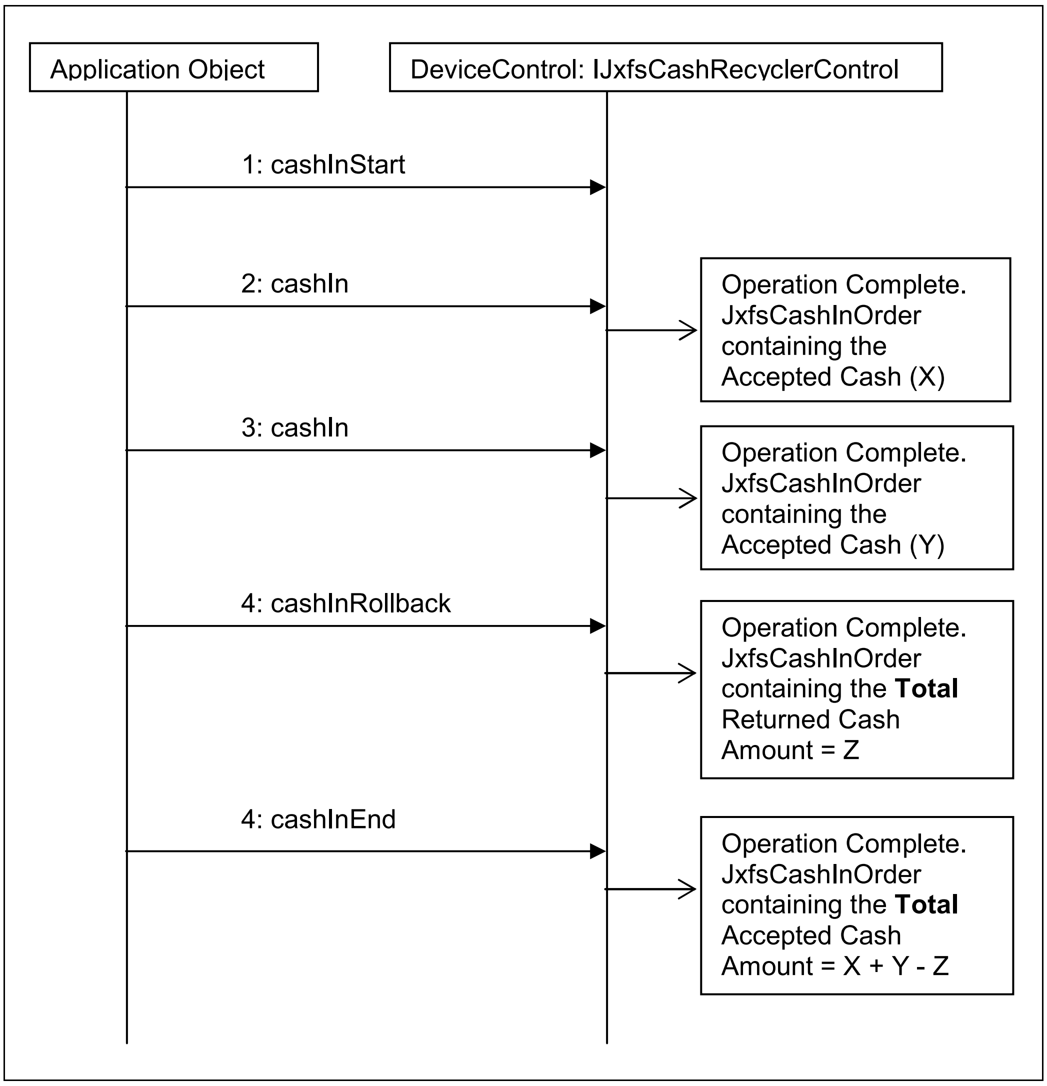
Most of the times, the notes inserted by means of consecutive *cashIn* are stored in the escrow. When performing the *cashInRollback* operation, these notes will be ejected and presented to the customer.


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

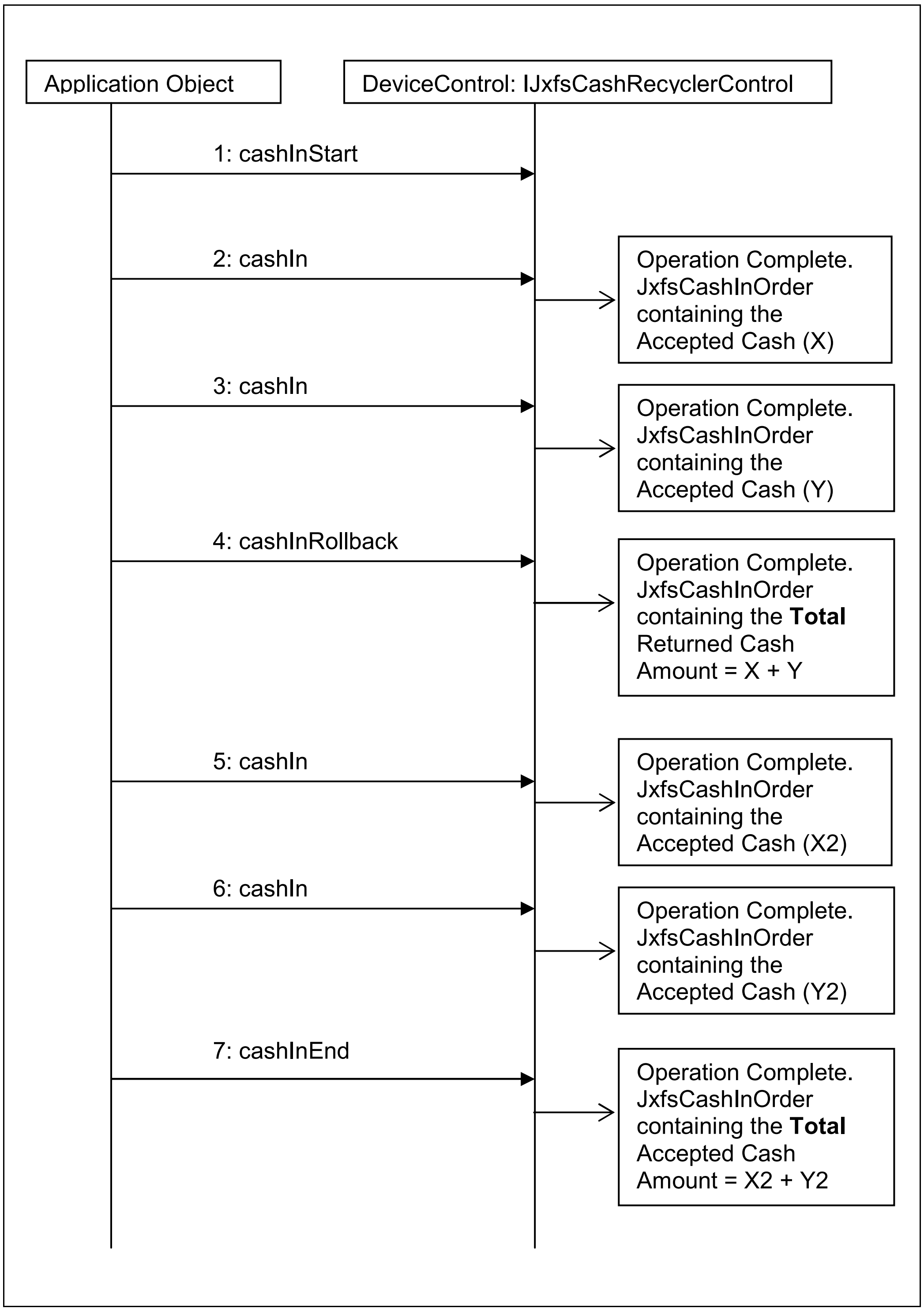
The fact of performing a rollback and not being returned all the notes might occur. This is not likely to happen, but in the specific case of the recyclers without an escrow and those where the rollback process is performed by means of a dispense operation, a dispense error could occur and thus the customer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


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

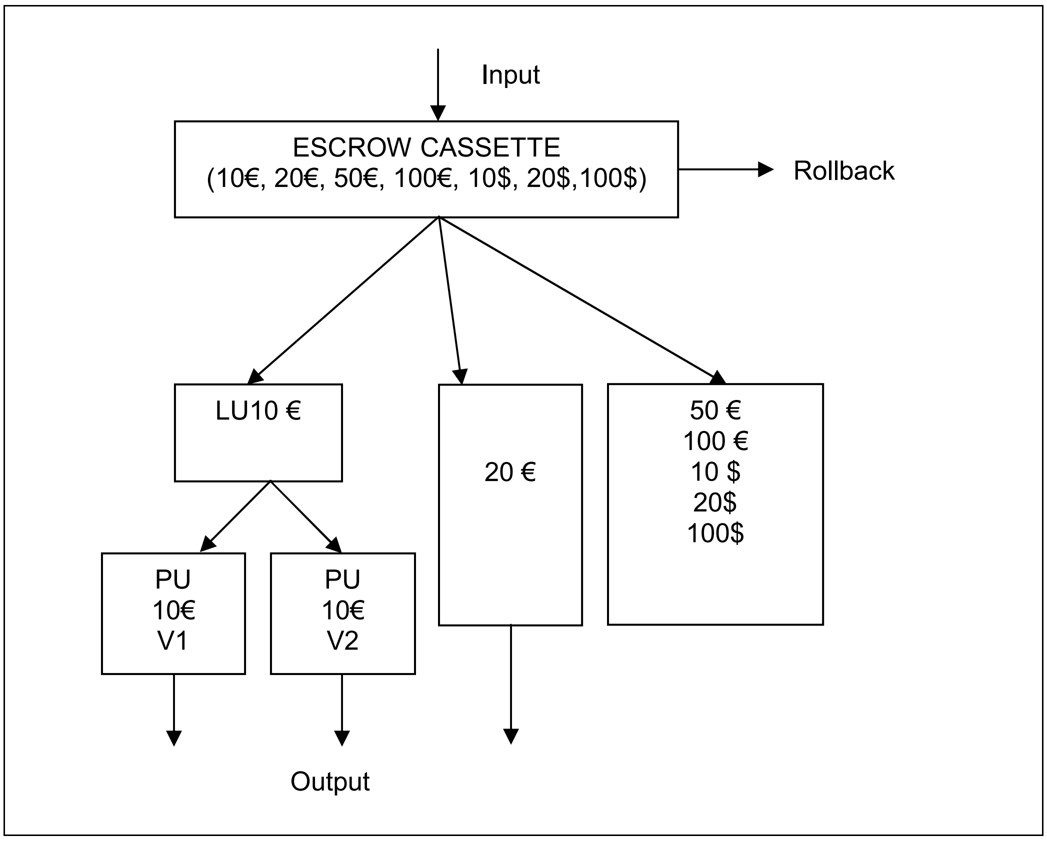
The current specification regarding cash dispensers and recyclers do not clarify the manner a cassette of the escrow type has to be defined; therefore an explanation permitting us to homogenize every manufacturer's device services, as much as possible, is necessary to be given.

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.

Currently different hardware and software implementations of an escrow exist on the market. Therefore the contents of cassettes of the type escrow cannot be assumed to represent their cashed-in money of the current transaction, because it is not clear which of the banknotes are physically present on an escrow and which are merely logically presented. The cash unit does not give guaranteed information if all category 2, 3 or 4 bank notes will be actually stored on the escrow and which of these can be rolled back. Therefore a multivendor application should rely on the *cashInInfo* property for information about cashed-in money of the current transaction.

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 contains.*

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.

9.8 Multiple Currency Cash-In operations

If the device shall process more than one currency in a *cashIn* operation, this requires some additional definitions as the *JxfsCashInOrder* class is limited to one currency (see amount and currency properties). Another requirement is that the device service supports that feature what can be checked via JxfsCapabilities.multipleCurrenciesCashInSupported.

To be able to accept more than one currency in a *cashIn* operation, the following preconditions have to be met:

As usual the acceptable currencies are defined with *updateDenominations*. Devices without banknote validator will not be used.

The properties of the *JxfsCashInOrder* parameter must be set as follows by the application:

currency.currencyCode.currencyCode = "*";
currency.exponent = 0;
denomination.amount = JXFS_C_CDR_NOT_APPLICABLE;

If items of more than one currency have been accepted by a *cashIn* operation, then the resulting *JxfsCashInOrder* / *JxfsArt6CashInOrder* object must be set up as follows:

currency.currencyCode.currencyCode = "*";
currency.exponent = 0;
denomination.amount = JXFS_C_CDR_NOT_APPLICABLE;

The following applies to *JxfsArt6CashInOrder* only:

category2.amount= JXFS_C_CDR_NOT_APPLICABLE;
category3.amount= JXFS_C_CDR_NOT_APPLICABLE;
category4.amount= JXFS_C_CDR_NOT_APPLICABLE;

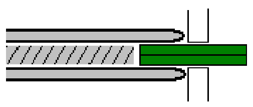
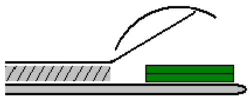
If the application wants to know what money has been accepted, it must either

- parse *JxfsCashInOrder*.denomination.items and get the denomination from the unit numbers plus the general currency exponent or

- analyse the cashInInfo property.

9.9 Position Mechanical Design Notes

Supported mechanical designs for positions on a dispenser/acceptor/recycler device.

Value	Description
JxfsCDRMechDesignEnum.slot	
JxfsCDRMechDesignEnum.tray	

Depending on the position mechanical design, explicit shutter handling has to be performed in the following ways:

- Slot during output operation:
  1) Call *shutterMove*(*true*,…) to open the slot. This method also moves items to a position accessible to the customer if required.
  2) Wait for the customer to take the items or timeout. Cash taken from the position is detected because the contents state changes to empty.
  3) Anyway call *shutterMove* (*false*,…) to ensure the slot is closed (even if the shutter has been closed automatically in some conditions). If there were items, this operation would move them back to allow the shutter to be closed.
  4) Check the final slot status.

- Slot during input operation:
  1) Call *shutterMove* (*true*,…) to open the slot.
  2) Call *cashIn* method right after the shutter opened to start cash acceptance.
  3) Anyway call *shutterMove* (*false*,…) to ensure the slot is closed.
  4) Check the final slot status.
- Tray during output operation:
  1) Call *shutterMove* (*true*,…) to open the tray.
  2) Wait for the customer to take the items or timeout. Cash taken from the position is detected because the contents state changes to empty.
  3) Ask customer confirmation to continue (using the screen).
  4) Call *shutterMove* (*false*,…) to close the tray.
  5) Check the final tray status.

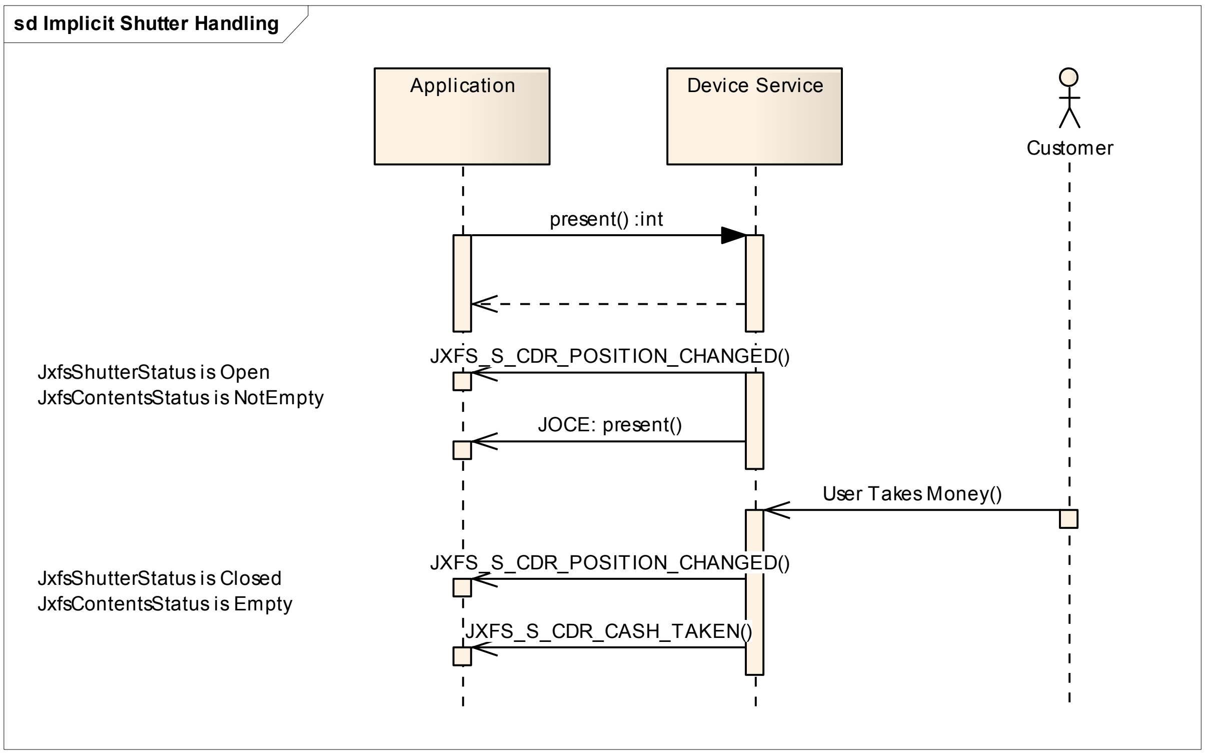
- Tray during input operation:
  1) Call *shutterMove*(*true*,…) to open the tray
  2) Wait for the customer to insert the items or timeout. Cash inserted is detected because the contents state changes to not empty.
  3) Ask customer confirmation to continue (using the screen).
  4) Call *shutterMove*(*false*,…) to closed the tray.
  5) Check the tray status to ensure the device is ready for cash-in.
  6) Call *cashIn*.

9.10 Shutter Handling sequence diagrams

The following diagrams depict the way to handle the *shutterMove* (and present) jobs and how *JxfsCDRPositionStatus* should change through the shutter handling operation.

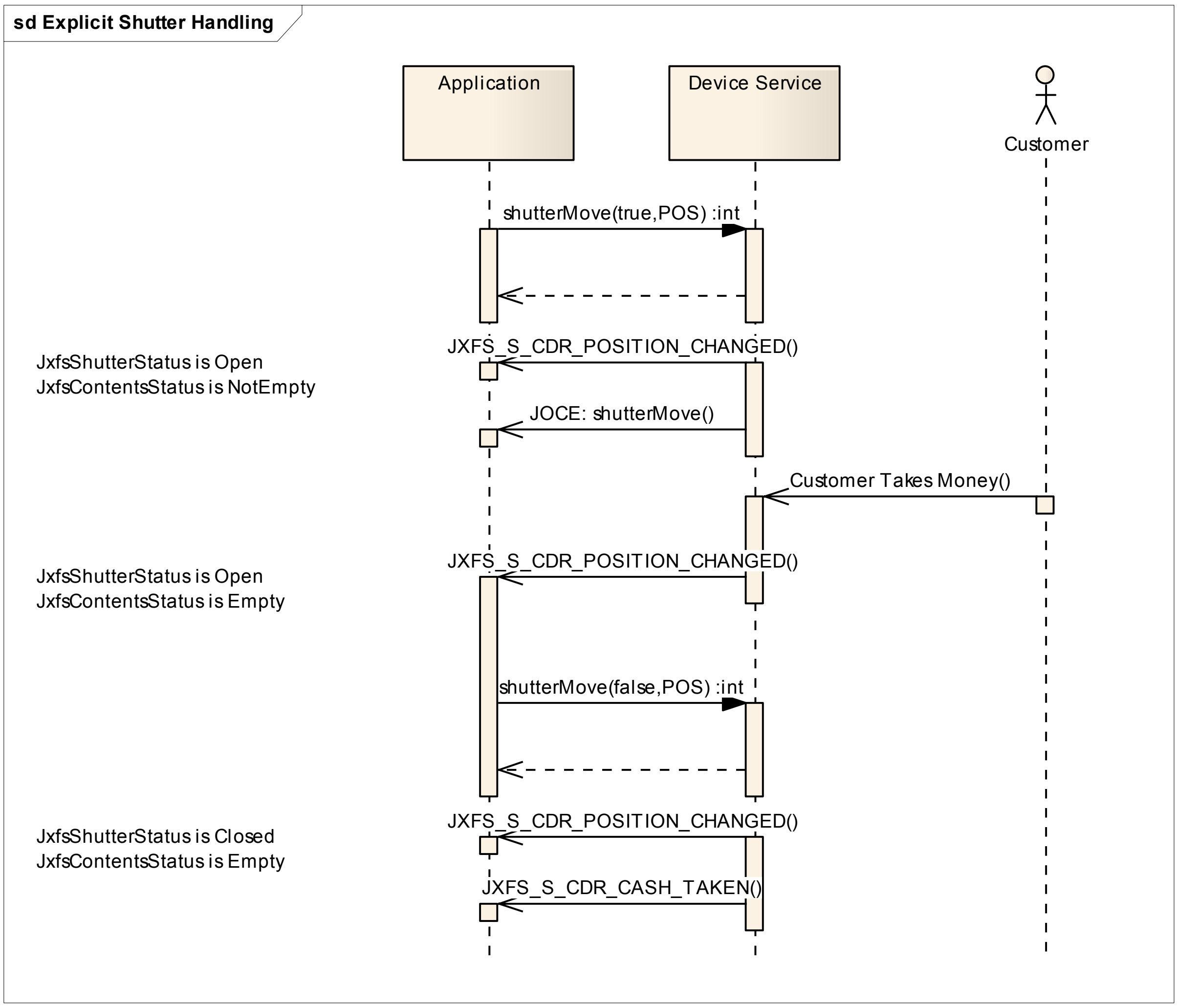
9.10.1 Implicit Shutter Handling

In this case customer takes money after it has been successfully presented in the output position, controlled implicitely:


sd Implicit Shutter Handling
Application
Device Service
Customer
present() :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is NotEmpty
JOCE: present()
User Takes Money()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Closed
JxfsContentsStatus is Empty
JXFS_S_CDR_CASH_TAKEN()

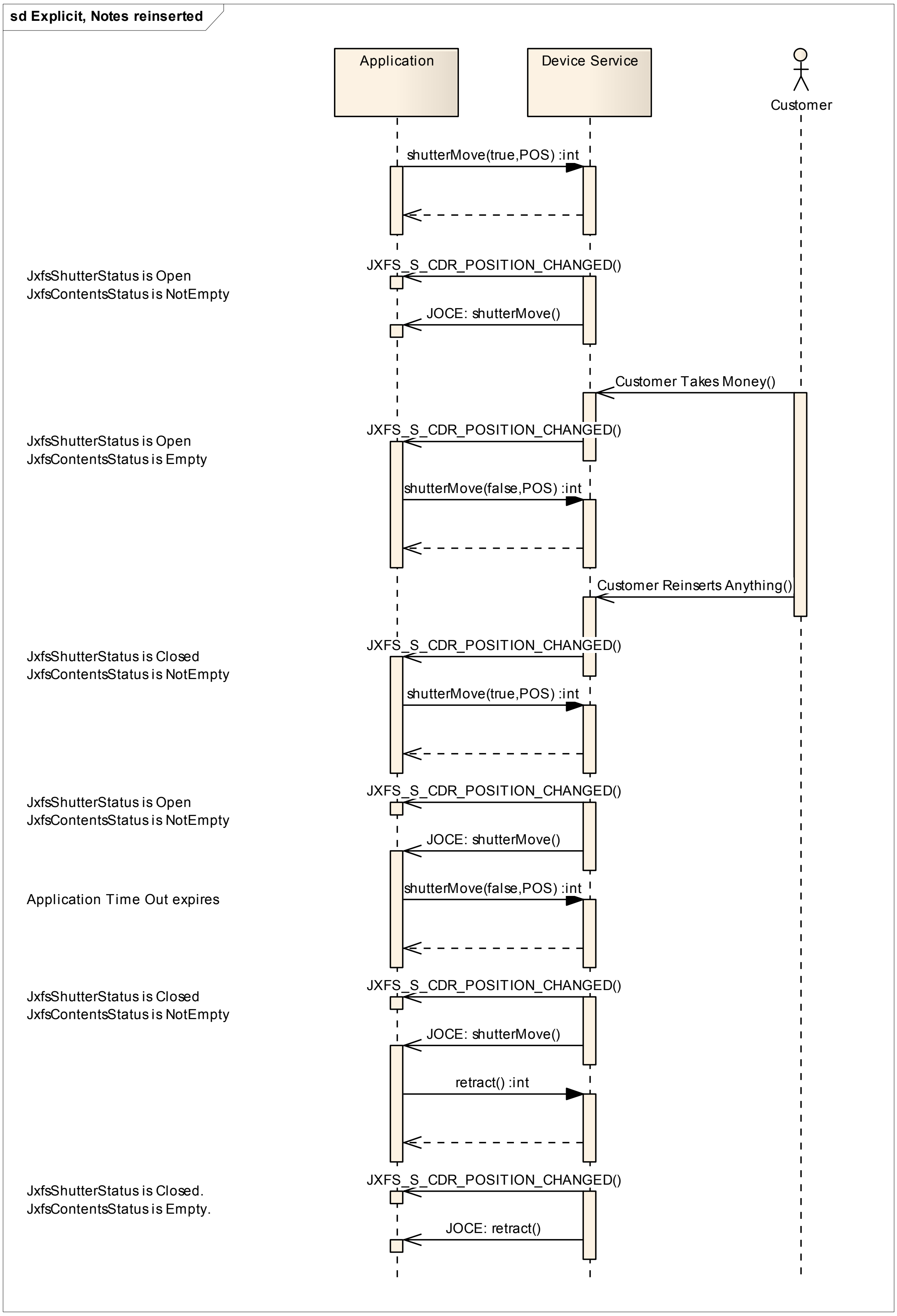
9.10.2 Explicit Shutter Handling

In this case customer takes money after it has been succesfully presented in the output position, controlled explicitely:


sd Explicit Shutter Handling
Application
Device Service
Customer
shutterMove(true,POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
Customer Takes Money()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is Empty
shutterMove(false,POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Closed
JxfsContentsStatus is Empty
JXFS_S_CDR_CASH_TAKEN()

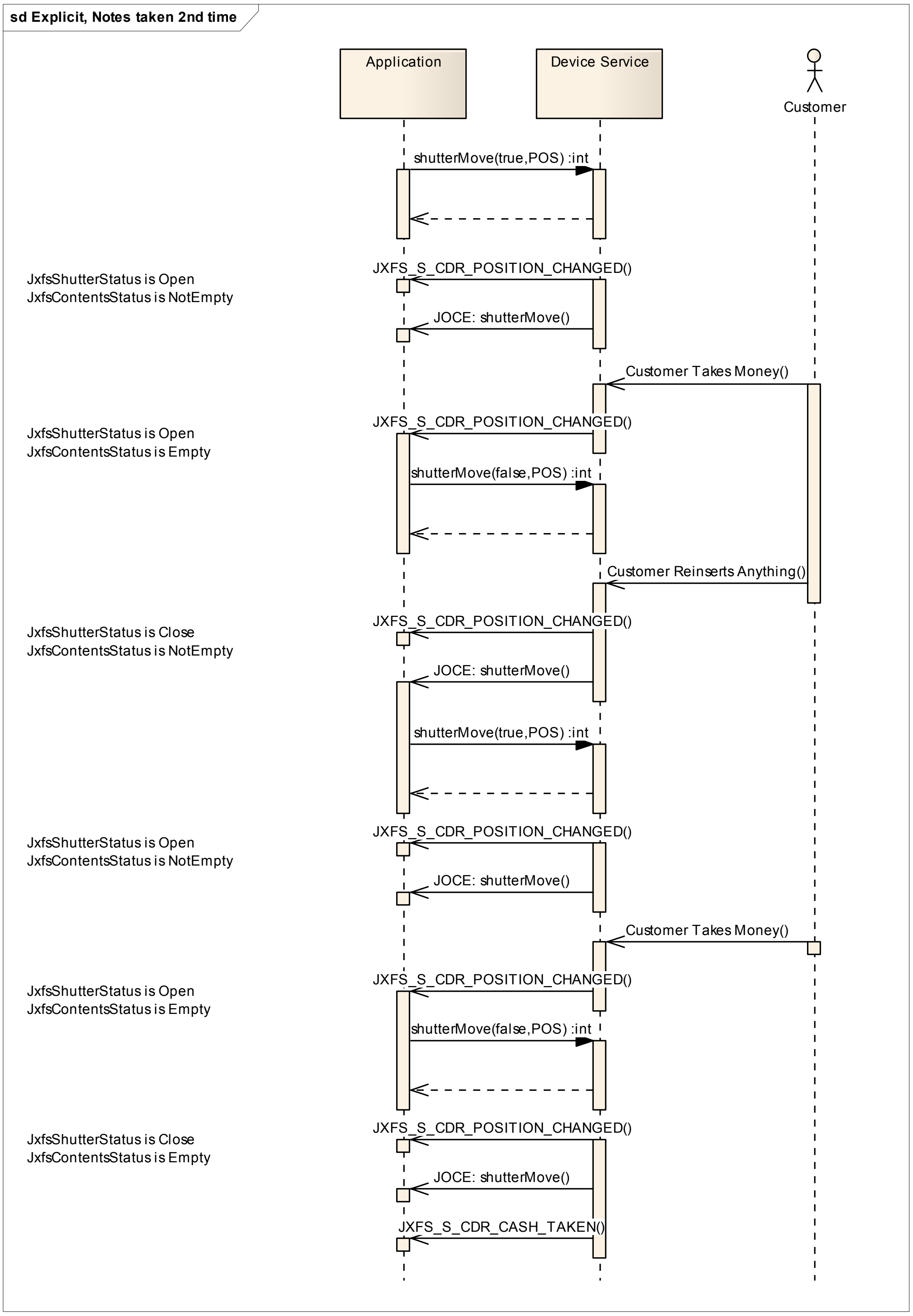
9.10.3 Explicit Shutter Handling, Notes reinserted and never taken

In this case after the notes have been presented for the first time, user takes them, and while position is being closed they are reinserted into it. Then, customer goes and application time out expires, so it proceeds to retract them.


sd Explicit, Notes reinserted
Application
Device Service
Customer
shutterMove(true,POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
Customer Takes Money()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is Empty
shutterMove(false,POS) :int
Customer Reinserts Anything()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Closed
JxfsContentsStatus is NotEmpty
shutterMove(true,POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
Application Time Out expires
shutterMove(false,POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Closed
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
retract() :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Closed.
JxfsContentsStatus is Empty.
JOCE: retract()

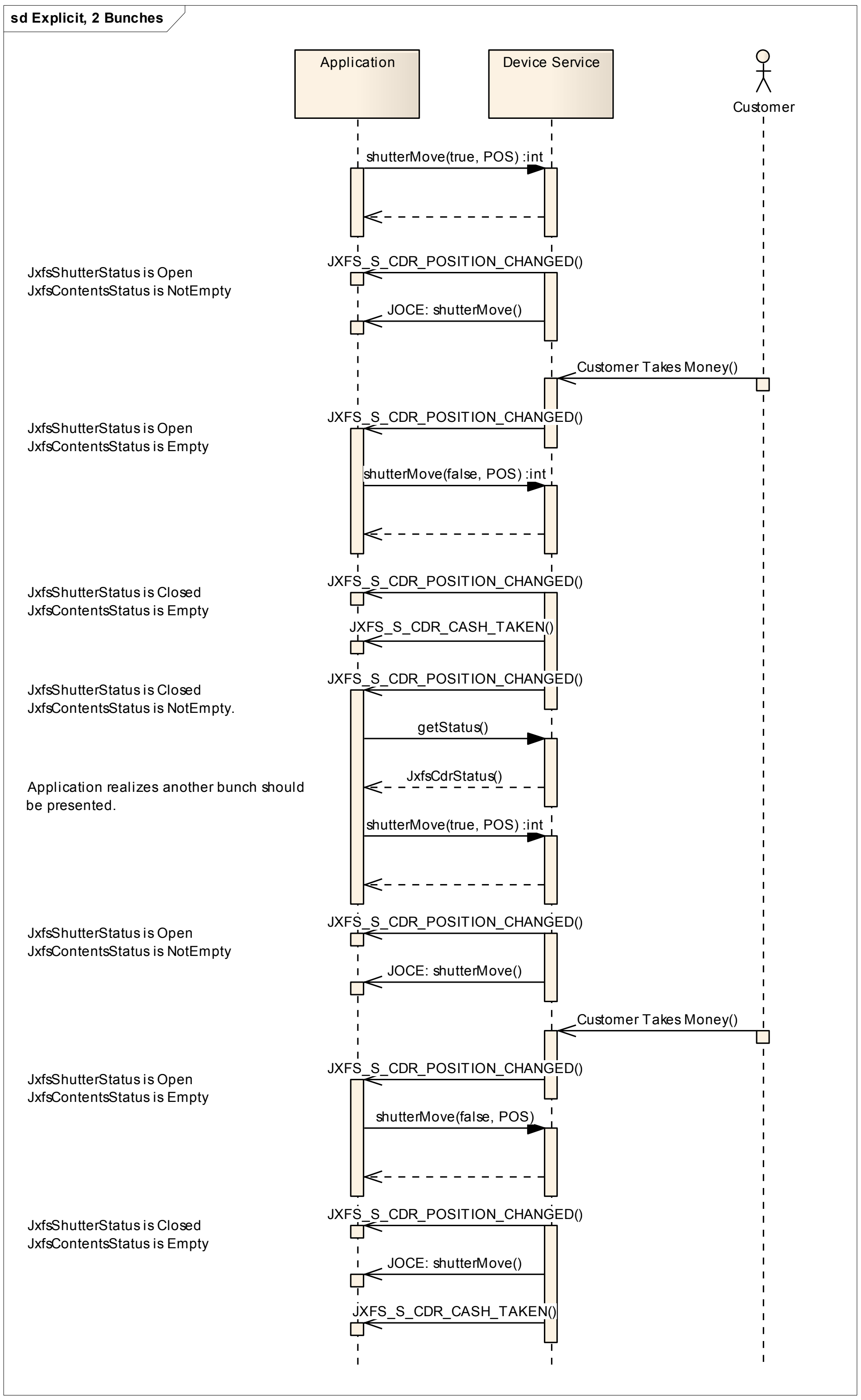
9.10.4 Explicit Shutter Handling, Notes taken in second presentation

In this case after the notes have been presented for the first time, customer takes them, and while position is being closed they are reinserted into it. Then, customer takes them and goes away.


sd Explicit, Notes taken 2nd time
Application
Device Service
Customer
shutterMove(true,POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
Customer Takes Money()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is Empty
shutterMove(false,POS) :int
Customer Reinserts Anything()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Close
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
shutterMove(true,POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
Customer Takes Money()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is Empty
shutterMove(false,POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Close
JxfsContentsStatus is Empty
JOCE: shutterMove()
JXFS_S_CDR_CASH_TAKEN()

9.10.5 Explicit Shutter Handling, Handling of two bunches

Some devices may not be able to present the complete position contents by a single shutter open action. if additional bunches need to be presented just after the first bunch presentation has been retrieved by customer (JXFS_S_CDR_CASH_TAKEN event), the device service should ensure that the position status is changed again to a NotEmpty state so application is able to check the position status and reopen the shutter.


sd Explicit, 2 Bunches
Application
Device Service
Customer
shutterMove(true, POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
Customer Takes Money()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is Empty
shutterMove(false, POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Closed
JxfsContentsStatus is Empty
JXFS_S_CDR_CASH_TAKEN()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Closed
JxfsContentsStatus is NotEmpty.
getStatus()
JxfsCdrStatus()
Application realizes another bunch should be presented.
shutterMove(true, POS) :int
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is NotEmpty
JOCE: shutterMove()
Customer Takes Money()
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Open
JxfsContentsStatus is Empty
shutterMove(false, POS)
JXFS_S_CDR_POSITION_CHANGED()
JxfsShutterStatus is Closed
JxfsContentsStatus is Empty
JOCE: shutterMove()
JXFS_S_CDR_CASH_TAKEN()