CEN

WORKSHOP

AGREEMENT

CWA 16008-12

August 2009

ICS 35.240.40

English version

J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Part 12: Vendor Dependant Mode Specification - 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 ™ 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 - Release 2009 - Camera Device Class Interface - Programmer's Reference
- Part 12: J/eXtensions for Financial Services (J/XFS) for the Java Platform - Release 2009 - Vendor Dependant Mode Specification - Programmer's Reference
- Part 13: J/eXtensions for Financial Services (J/XFS) for the Java Platform – Scanner Device Class Interface - Programmer's Reference (recommended replacement for Part 10)

History

Main differences to CWA 14923-12:2004 are:

- openAcknowledges and device List properties contains additional information
- revised *Description* chapter and some other texts

Main differences to CWA 13937-12:2003 are:

- Remark that the VDM device service is not a normal device service as it does not control any hardware.
- Added remark to the sequence diagram where an application starts during VDM mode that this application may issue the getDevice(...) and the adding of the listeners more early.
- Changed occurrences of JXFS_S_VDM_ACTIVE to JXFS_VDM_ACTIVE and JXFS_S_VDM_INACTIVE to JXFS_VDM_INACTIVE
- Added a little bit more description to openAcknowledges property.
- New exception JXFS_VDM_NOT_IN_VDM when exitVDM() is called while VDM is not active by this application.
- Additional remark at enterVDM() that these calls will be queued.
- New property deviceList that holds an array of the logical names of all controlled devices.
- Specified IJxfsVDMConst as the name of the interface holding the constants.
- Clarification of scenarios with more than one VDM.
- Claraification about compound devices.
- Exchanged "VDM control application" by "VDM application"
- Explanation what "VDM application" and "application" means.
- Added description of VDM Entry/Exit triggered by external hardware
- Added list of numerical values.

1 Scope

This document describes the Vendor Dependant Mode class 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 VDM devices the basic Device Control structure is extended with various properties and methods specific to this device which are described on the following pages.

2 Overview

2.1 Description

This specification describes the functionality provided by the Vendor Dependent Mode (VDM) services under J/XFS, by defining the service-specific commands that can be issued.

In all device classes there needs to be some method of going into a vendor specific mode to allow for capabilities which go beyond the scope of the current J/XFS specifications. A typical usage of such a mode might be to handle some configuration or diagnostic type of function or perhaps perform some 'off-line' testing of the device. These functions are normally available on Self-Service devices in a mode traditionally referred to as Maintenance Mode or Supervisor Mode and usually require operator intervention. It is those vendor-specific functions not covered by (and not required to be covered by) J/XFS device services that will be available once the device is in Vendor-Dependent mode.

This service provides the mechanism for switching to and from Vendor Dependent Mode. The VDM device service can be seen as the central point through which all Enter and Exit VDM requests are synchronised.

In the following text the expression "VDM application" refers to an application that requests the VDM mode. The expression "application" refers to an application that uses the J/XFS interface to access devices (and registers at the VDM device service). "VDM application" and "application" are not mutually exclusive. That means that both expressions may refer to the same program.

Entry into, or exit from, Vendor Dependent Mode will be initiated by a VDM application. If initiated by a VDMapplication, this application needs to issue the appropriate command to request entry or exit.

Once the entry request has been made, all registered applications will be notified of the entry request by an event message. These applications must attempt to close all open J/XFS Device Controls other than VDM specified in the event as soon as possible and then issue an acknowledgement command to the VDM device service when ready. Once all applications (including the application that initiated the request) have acknowledged, the VDM device service will issue status event messages to these applications to indicate that the system is in Vendor Dependent Mode. The application that made the request will receive an operation complete event indicating that this application is now in charge of all the requested devices.

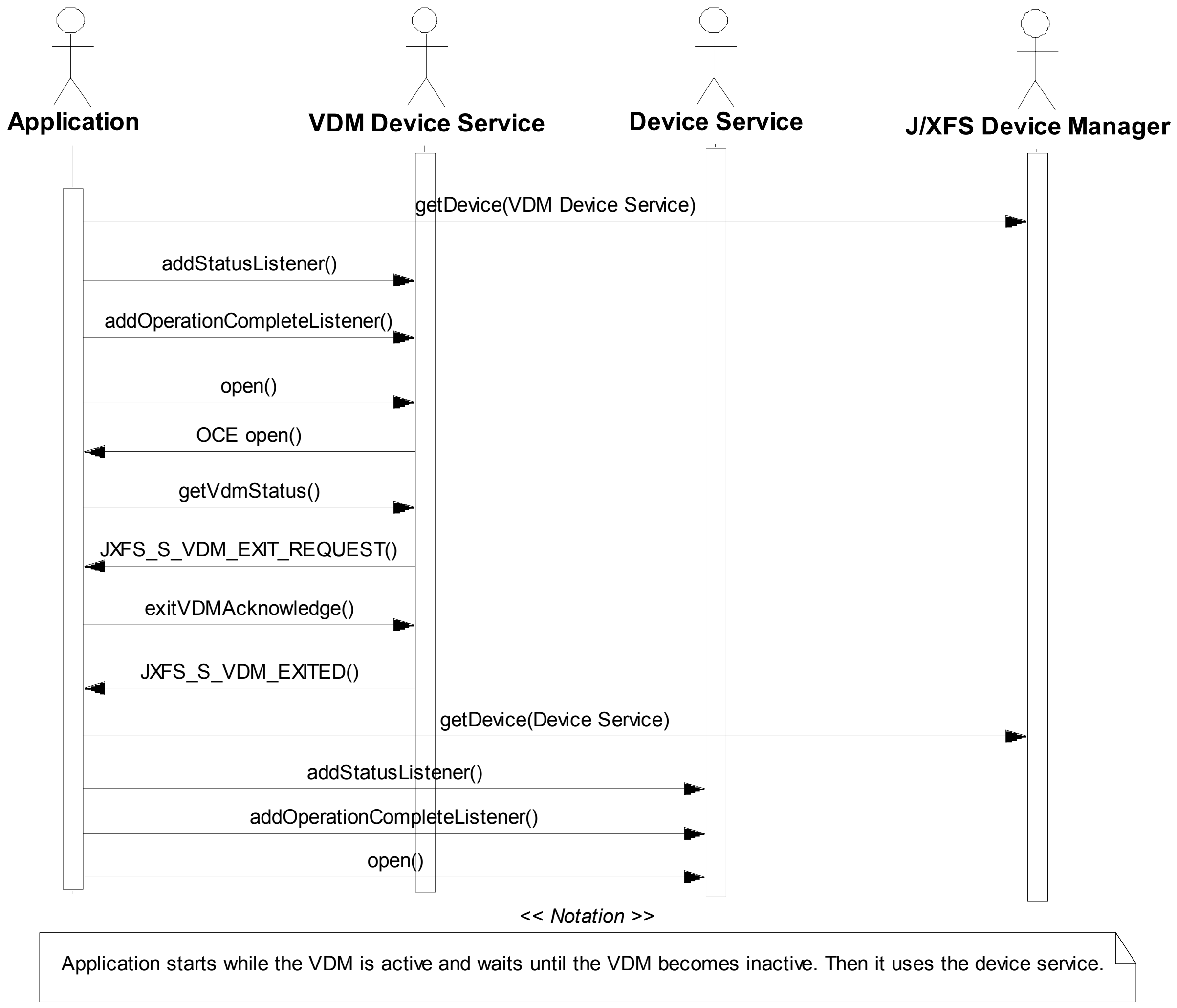
The VDM device service is not a general device service as other device services because it does not directly control any hardware. If a VDM shall be initiated by a hardware trigger, an application has to be created controlling the hardware and making the appropriate calls to the VDM device service.

Similarly, once the exit request has been made, all registered applications will be notified of the exit request by an event message (including the application that made the request). These applications must then issue an acknowledgement command to the VDM device service immediately. Once all applications have acknowledged, the VDM device service will issue event messages to these applications to indicate that the system has exited from Vendor Dependent Mode. The application that made the request will receive an operation complete event indicating that the VDM has successfully exited for all the requested devices. The application must react only on the operation complete event and not on the status event because several VDM requests may be queued by the VDM device service.

The design allows that more than one VDM application may request devices from one VDM device service instance.

Thus, J/XFS compliant applications that do not need the system to be in Vendor Dependent Mode, must comply with the following:

- Every J/XFS application should obtain a device control of the VDM device service, register for all VDM entry and exit events as well as open the VDM device service.
- Before calling open() of a J/XFS Device Control, check the status of the VDM Service. If Vendor Dependent Mode is not "Inactive", do not open a device control Nevertheless the application can already register for events with the device service.
- When getting a VDM entry notice, close all opened J/XFS Device Controls as soon as possible and issue an acknowledgement for the entry to VDM.
- When getting a VDM exit notice, acknowledge at once.
- When getting a VDM exited notice, re-open any required J/XFS Device Controls.


Application
VDM Device Service
Device Service
J/XFS Device Manager
getDevice(VDM Device Service)
addStatusListener()
addOperationCompleteListener()
open()
OCE open()
getVdmStatus()
JXFS_S_VDM_EXIT_REQUEST()
exitVDMAcknowledge()
JXFS_S_VDM_EXITED()
getDevice(Device Service)
addStatusListener()
addOperationCompleteListener()
open()
<< Notation >>
Application starts while the VDM is active and waits until the VDM becomes inactive. Then it uses the device service.

The above sequence diagram demonstrates the correct approach that the getDevice(Device Service) should be issued after the VDM mode has been exited. However, the application is permitted to issue the getDevice(Device Service) and to add listeners earlier than shown above because these calls do not involve device access. However, the application must not call open() before the VDM has been exited.

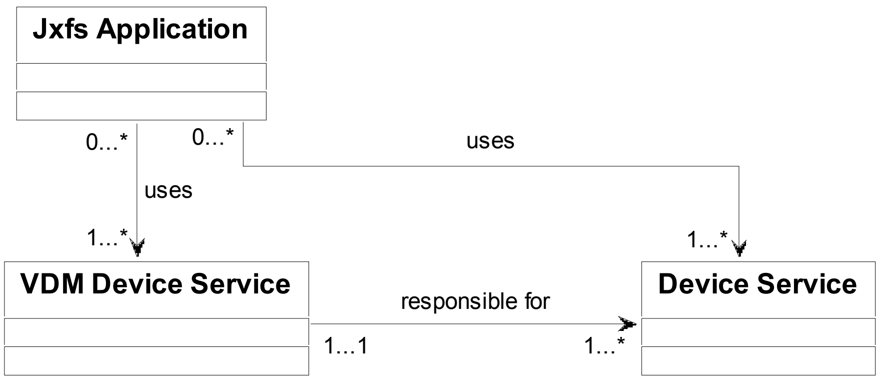
A J/XFS compliant device service must comply to the following:

- It must unlock and close all physical devices when it receives the last close from the Device Controls.

A J/XFS infrastructure may be scattered in a larger network and include several VDM device service instances. A reasonable granularity should be one VDM device service instance per workstation. A VDM device service can guarantee the exclusive access not only to all devices under its control, but to a selection of its devices. Every device should have not more than one VDM instance to guard over a Vendor Dependant Mode for this device. More than one VDM instance per device service would require additional synchronisation between the VDM instances. This is not covered by this specification and therefore illegal.

In a self-service installation there must be no more than one VDM device service instance per workstation. This restriction may not apply in a front office environment where the infrastructure may consist of different organizational units. One example could be a workstation with two physically connected cash dispensers that are logically connected to other workstations. In this case it might make more sense to use one VDM device service instance per organizational unit.

As a recommendation a hardware device used by more than one device service must not be under control of more than one VDM device service. That means that in the case that we have one hardware device that is driven by two or more device services then each request for the VDM for one of these device services must result in a request for a VDM for all the device services driving the hardware device. This has to be ensured by the VDM device service by configuration means.



The expression "responsible for" in the chart does not mean that the VDM Device Service calls any methods of the Device Service it is responsible for.

A VDM device service will only allow one VDM to be active at a time. This active VDM is associated with one, requesting, application and can be for 1..n devices where n represents all devices associated with the VDM device service.

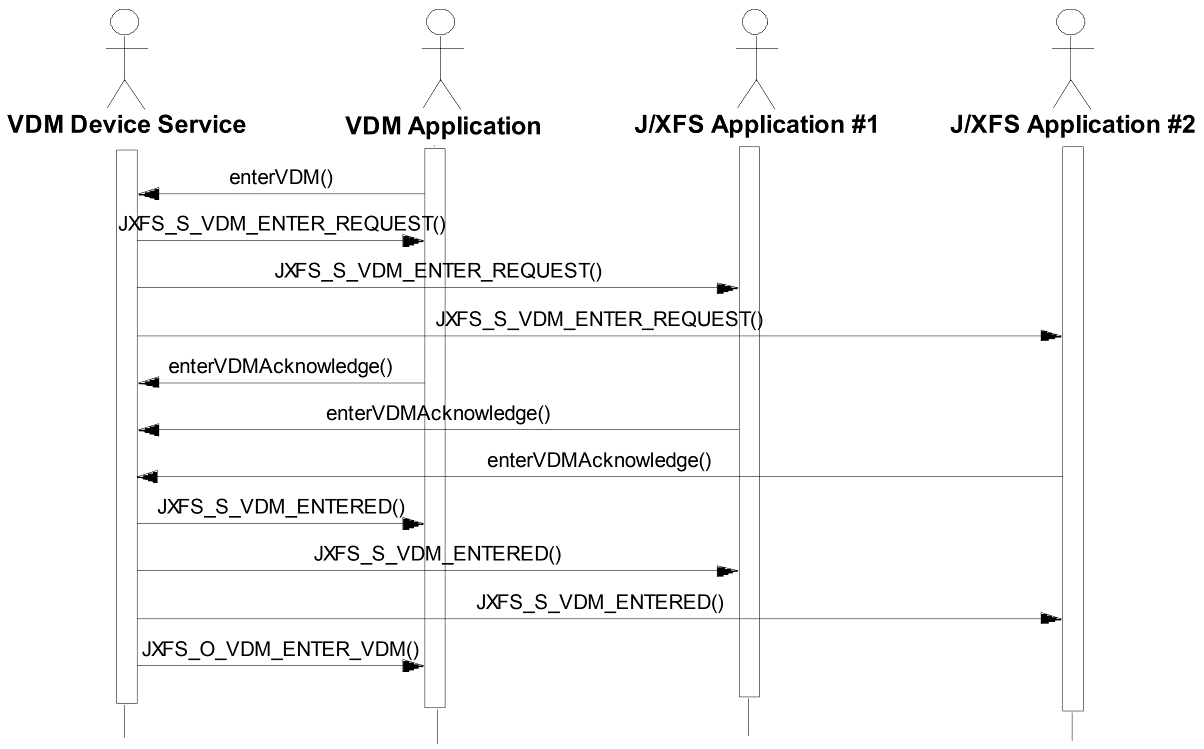
That means that in a self-service device we have one main computer (usually a PC) that runs several device services and application. In such a device we have one instance of the VDM that controlls all devices connected to this workstation.

If such a Vendor Dependant Mode is used in a Front Office area where more than one independant devices are connected to one server PC it can be that all of these independant devices are controlled by another VDM device service instance. In such a case we can have several dispensers connected to one PC. In this case every dispenser is controlled by another VDM device service instance. In such a case an application has normally to be configured which VDM device service to be used with which device. An alternative for an application is to check all available VDM device services for their deviceList to analyse which device services they control.

From the view of a VDM device service a VDM application is the same as a normal J/XFS application. Every application has to open the VDM device service(s) for all devices to be used by the application.

2.1.1 VDM Entry triggered by J/XFS Application

In the following example all applications have opened the VDM device service and registered for status and operation complete events. At the beginning, the VDM is not active.



The application that wants the VDM mode issues an asynchronous *enterVDM*() command to the VDM device service.

This changes the status of the VDM device service from JXFS_VDM_INACTIVE to JXFS_VDM_ENTER_PENDING. All applications that have opened the VDM device service are notified of this change by a JXFS_S_VDM_ENTER_REQUEST event.

Now all applications have to release all of the claimed devices specified by the status event and close them. After an application has closed all of the relevant devices it has to call the synchronous *enterVDMAcknowledge*() method of the VDM device control.

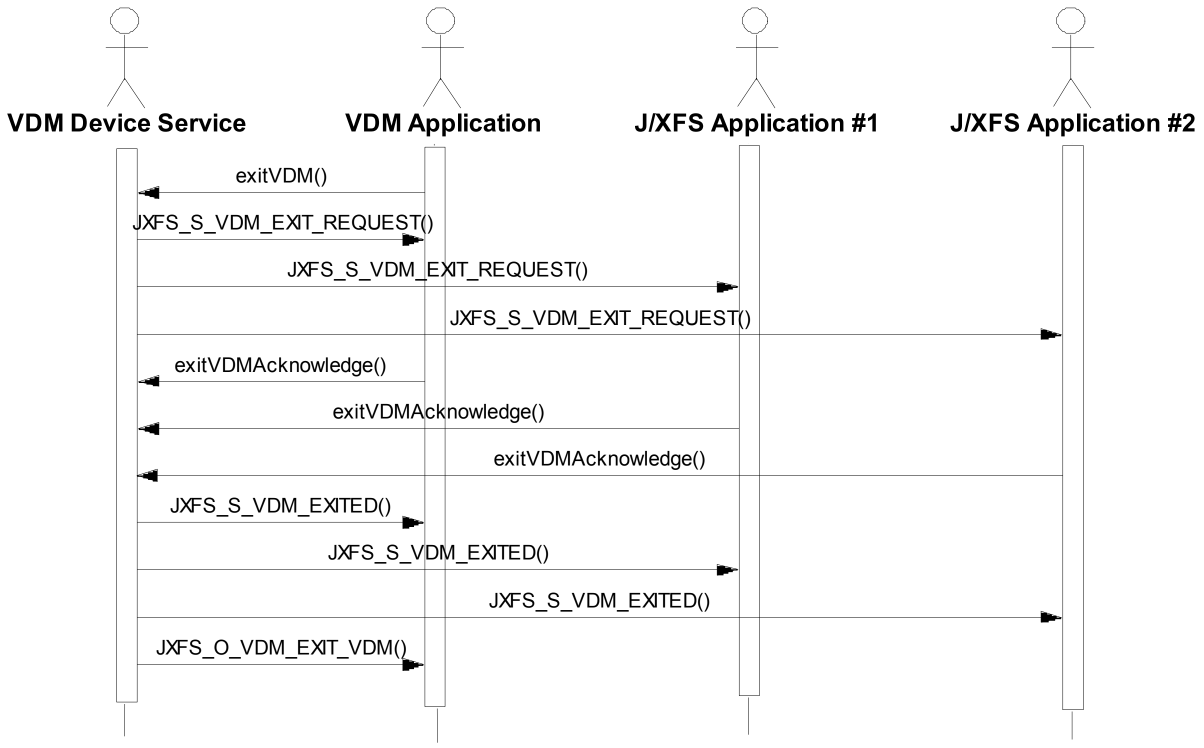
After all applications have notified the VDM device service that they have closed their devices, the internal state of the VDM device service changes to JXFS_VDM_ACTIVE. All applications will be notified of this change by a JXFS_S_VDM_ENTERED event.

Finally the application that originally requested the VDM will be notified by an operation complete event JXFS_O_VDM_ENTER_VDM that it has now exclusive access of all the requested devices. The exlusive access is granted only by the operation complete event and not by the JXFS_S_VDM_ENTERED event because it can be that two applications may request an exclusive VDM mode. In such a case the VDM would be granted to each of the applications sequentially. In such a case the status event would not be sufficient for a VDM application to decide if it was granted the VDM for itself or the other application.

The system is now in Vendor Dependent Mode and the Vendor Dependent application can exclusively use the system devices in a Vendor Dependent manner.

2.1.2 VDM Exit triggered by J/XFS Application

In the following example all applications have opened the VDM device service and registered for status and operation complete events. At the beginning, the VDM is active. The exit request must come from the same application that initiated the entry request.



The application that wants to exit the VDM mode issues an asynchronous *exitVDM*() command to the VDM device service.

This changes the status of the VDM device service from JXFS_VDM_ACTIVE to JXFS_VDM_EXIT_PENDING. All applications that have opened the VDM device service are notified of this change by a JXFS_S_VDM_EXIT_REQUEST event.

Now all applications have to call the synchronous *exitVDMAcknowledge()* method of the VDM device control.

After all applications have notified the VDM device service that they are aware of the exit out of the VDM mode, the internal state of the VDM device service changes to JXFS_ VDM_INACTIVE. All applications will be notified of this change by a JXFS_S_VDM_EXITED event.

Finally the application that originally requested the exit of the VDM will be notified by an operation complete event JXFS_O_VDM_EXIT_VDM that all applications are aware that the VDM has been left.

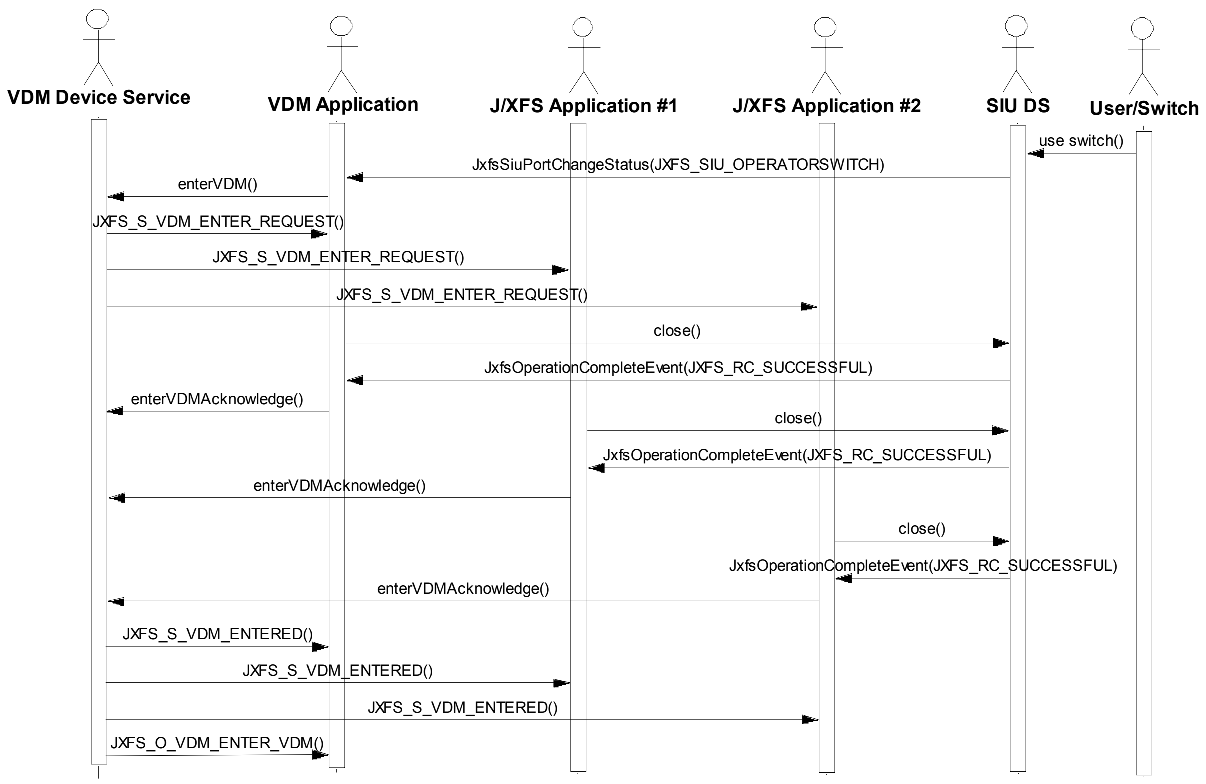
2.1.3 VDM Entry and Hardware Triggers

The VDM is not triggered directly by a hardware switch (as in the XFS Standard), but such a functionality can be replicated by the following mechanism.

In the following example all applications have opened the VDM device service and the SIU device service and registered for status and operation complete events. At the beginning, the VDM is not active.

In this scenario a hardware switch exists that may be triggered by an external user. This hardware switch is mapped to the operatorswitch port of the SIU device.

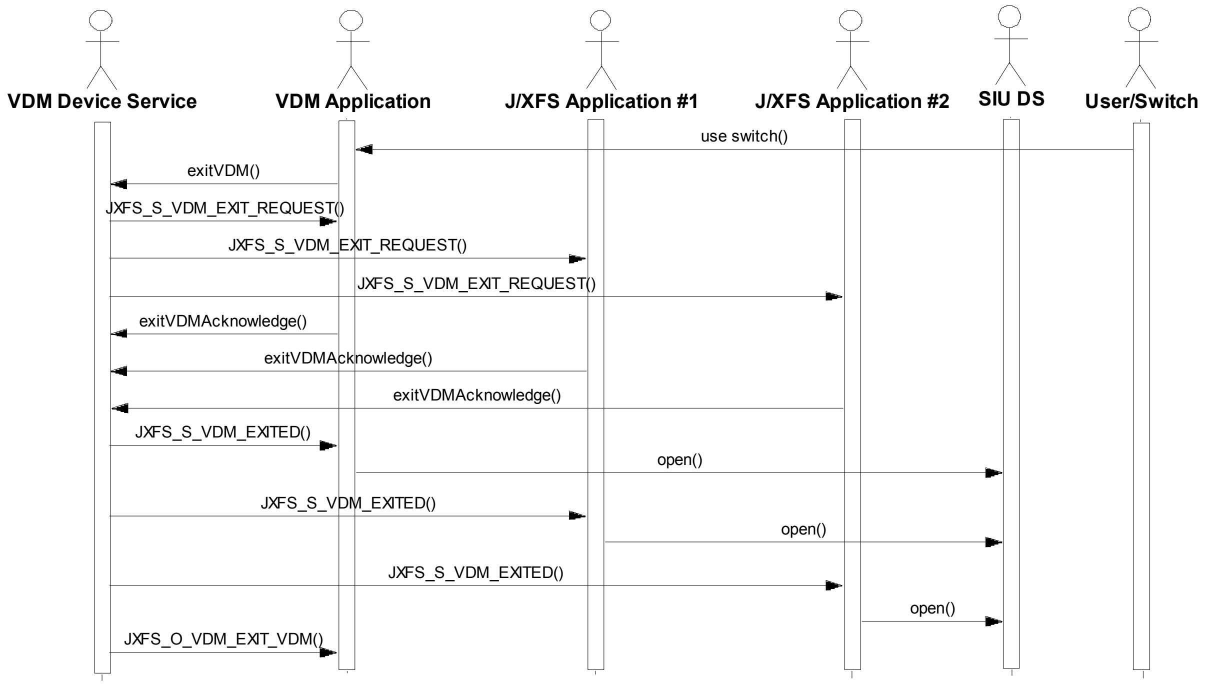
The sequence is similar to the sequence in the chapter "VDM Entry triggered by J/XFS Application". The difference is that the reason for the VDM application to apply for the VDM mode is the switch that is triggered by the user and reported to the VDM application with a SIU status event. All connections to the SIU device service are also closed on the VDM enter request.



2.1.4 VDM Exit and Hardware Triggers

In the following example all applications have opened the VDM device service and registered for status and operation complete events. At the beginning, the VDM is active and no application has opened the SIU device service.

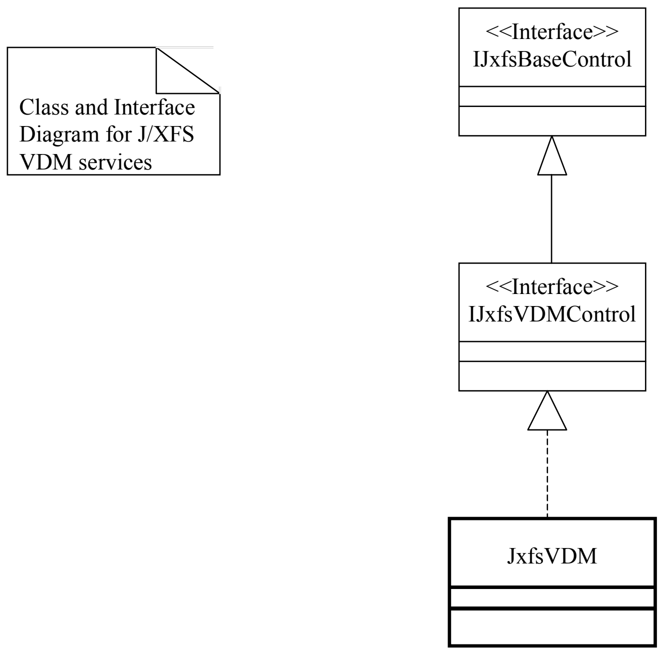
In this scenario a hardware switch exists that may be triggered by an external user. This hardware switch is mapped to the operatorswitch port of the SIU device. But because the system is in the VDM, the SIU device service is not able to check the state of that switch.



Therefore the VDM application must query the switch by proprietary means that are outside the scope of the J/XFS standard. Once the VDM application is aware that the switch is triggered, it requests the exit out of the VDM mode.

Afterwards every application reopens the SIU device service after having been notified about the exit of the VDM mode.

3 Class Hierarchy


Class and Interface Diagram for J/XFS VDM services
<<Interface>> IJxfsBaseControl
<<Interface>> IJxfsVDMControl
JxfsVDM

3.1 Class and Interface Summary

The following classes and interfaces are used by the J/XFS VDM device controls.

Class or Inter-face	**Name**	**Description**	**Extends / Implements**
Inter-face	**IJxfsBaseControl**	Base interface for all device controls. Contains methods specific to all the device controls.	--
Class	**JxfsBaseControl**	Base class for all device controls. Implements the methods defined in the **IJxfsBaseControl** Interface. Contains the properties specific to all device controls.	Implements: **IJxfsBaseControl**
Inter-face	**IJxfsVDMControl**	Base interface for all vendor dependent mode controls. Contains the methods specific to all the device controls for the vendor dependent mode category.	Extends: **IJxfsBaseControl**
Class	**JxfsVDM**	Class for the Vendor Dependent Mode control	**IJxfsVDMControl**
Inter-face	**IJxfsEventNotification**	Includes one callback method per event type. The Device Service calls these methods to cause events to be delivered to the application.	--

4 Device behavior

This device service does not access any device as it is a logical device service.

4.1 Unsupported General Methods of base Control

The following base methods are not supported and will throw a JXFS_E_NO_HARDWARE exception when called:

getPhysicalDeviceDescription()
getPhysicalDeviceName()
getFirmwareStatus()

The updateFirmware() method will throw a JXFS_E_NOHARDWARE exception when called.

A JXFS_E_NOT_SUPPORTED exception is returned by the following methods:

getDeviceFirmwareVersion()
getRepositoryFirmwareVersion()
wakeUpFromPowerSave()

The isPowerSaveModeSupported() method will return the value true and not throw an exception.

5 Classes and Interfaces

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

After processing has taken place, a JxfsOperationComplete – Event is generated which contains detailed information about the status of the operation, i.e. if it failed or succeeded, and eventually additional data as a result.

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

5.1 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.

5.1.1.1 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

5.1.1.2 set*Property*

Syntax	**void *set*Property*(Property) throws JxfsException;***
Description	Sets the requested property.
Parameter	**Single parameter of property type.**
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

5.2 IJxfsVDMControl

The J/XFS Vendor Dependent Mode Device Control Subclass is defined in JxfsVDMControl and is a subclass of JxfsBaseControl. Its interface is defined in IJxfsVDMControl which is a subclass of IJxfsBaseControl. The intent of the J/XFS Vendor Dependent Mode Device Control object is to allow data and control to pass between the application and the Vendor Dependent Mode device service code so that the required functions can be performed.

5.2.1 Summary

Property	Type	Access	Initialized after
openAcknowledges	**int**	R	*open(),enterVDM(), exitVDM()*
vdmStatus	**JxfsVDMStatus**	R	*open()*
deviceList	**java.lang.String[]**	R	*open()*

Method	Return	May be used after
get*Property*	*Property*	*open()*
set*Property*	*Property*	*open()*
enterVDM	identificationID	*open(),exitVDM()*
enterVDMAcknowledge	void	*enterVDM()*
exitVDM	identificationID	*enterVDM()*
exitVDMAcknowledge	void	*exitVDM()*

Event	May occur after
Status Event JXFS_S_VDM JXFS_S_VDM_ENTER_REQUEST JXFS_S_VDM_ENTERED JXFS_S_VDM_EXIT_REQUEST JXFS_S_VDM_EXITED	 *open(), close(), claim(), release()* *enterVDM (), enterVDMAcknowledge (), exitVDM(), exitVDMAcknowledge()*
JxfsOperationCompleteEvent JXFS_O_VDM_ENTER_VDM JXFS_O_VDM_EXIT_VDM	 *enterVDMAcknowledge ()* *exitVDMAcknowledge ()*

5.2.2 Properties

5.2.2.1 openAcknowledges

Type	***int***
Initial Value	*0*
Description	Number of pending application acknowledges.
Event	none

5.2.2.2 vdmStatus

Type	***JxfsVDMStatus***
Initial Value	*a JxfsVDMStatus* (for initial values see *JxfsVDMStatus*)
Description	see *JxfsVDMStatus*
Event	If the value of this property changes, the Device Service will send all registered StatusListeners a JxfsStatusEvent with the following status value :
Events	JXFS_S_VDM_ENTER_REQUEST JXFS_S_VDM_ENTERED JXFS_S_VDM_EXIT_REQUEST JXFS_S_VDM_EXITED

5.2.2.3 deviceList

Type	***java.lang.String[]***
Initial Value	null
Description	This array contains all logical device names of the devices that are under the control of this VDM instance. All elements of the array have to be valid and not empty strings. A value of null is not allowed after the first successfull open of the VDM. To be able to provide an unambiguous list of device names, all device names for device services instantiated on one workstation must be unambiguous.
Event	none

5.2.3 Methods

5.2.3.1 enterVDM

Syntax ***identificationID enterVDM(java.lang.String logicalDevices[]) throws JxfsException;***

Description This command is issued by an application to indicate a logical request to enter Vendor Dependent Mode. The VDM Device Service will then indicate the request to all registered applications by sending a JXFS_S_VDM_ENTER_REQUEST event message and then wait for an acknowledgement back from each registered application before putting the system into Vendor Dependent Mode. The VDM prevail until all applications have acknowledged, at which time the status will change to JXFS_S_VDM_ENTERED and the *enterVDM*() completes. If another application has also requested the VDM, the *JxfsOperationCompleteEvent* might be sent after the other application has been granted VDM and afterwards released the VDM. That means that VDM request will be queued.

Parameter

Type	Name	Meaning
java.lang.String[]	logicalDevices	Array with names of logical devices that shall be exclusively used in the VDM. An empty array with no Strings is not allowed. A null value for this method means that all devices under the control of the VDM device service shall be requested for the VDM mode.

Exceptions No additional exceptions generated.

Events Additional Events can be generated :

JxfsOperationCompleteEvent

When a *enterVDM ()* operation is completed a *JxfsOperationCompleteEvent* will be sent by J/XFS VDM device service to all registered *IJxfsOperationCompleteListeners* with the following data:

Field	Value
operationID	JXFS_O_VDM_ENTER_VDM
identificationID	The corresponding ID
result	Common or device dependent error code. (See section on *Error Codes*).
data	none

5.2.3.2 enterVDMAcknowledge

Syntax	***void enterVDMAcknowledge() throws JxfsException;***
Description	This command is issued by a registered application as an acknowledgement to the JXFS_S_VDM_ENTER_REQUEST event message and it indicates that the application is ready for the system to enter Vendor Dependent Mode.
Parameter	none
Exceptions	No additional exceptions generated.

5.2.3.3 exitVDM

Syntax	***identificationID exitVDM() throws JxfsException;***
Description	This command is issued by an application to indicate a logical request to exit Vendor Dependent Mode. The VDM device service will then indicate the request to all registered applications by sending a JXFS_S_VDM_EXIT_REQUEST event message and then wait for an acknowledgement back from each registered application before removing the system from Vendor Dependent Mode.
Parameter	none
Exceptions	Additional Exception:

Value	**Meaning**
JXFS_E_VDM_NOT_IN_VDM	The VDM cannot be exited as it is not engaged by this application.

Events Additional Events can be generated :

JxfsOperationCompleteEvent

When a *exitVDM()* operation is completed a *JxfsOperationCompleteEvent* will be sent by J/XFS VDM Device Control to all registered IJxfsOperationCompleteListeners with the following data:

Field	**Value**
operationID	JXFS_O_VDM_EXIT_VDM
identificationID	The corresponding ID
result	Common or device dependent error code. (See section on *Error Codes*).
data	none

5.2.3.4 exitVDMAcknowledge

Syntax	***void exitVDMAcknowledge() throws JxfsException;***
Description	This command is issued by a registered application as an acknowledgement to the JXFS_S_VDM_EXIT_REQUEST event message and it indicates that the application is ready for the system to exit Vendor Dependent Mode.
Parameter	none
Exceptions	No additional exceptions generated.

6 Support Classes

No support classes are used by this service.

7 Status Event Classes

If the service status or the value of the *vdmStatus* property changes the appropriate class is returned as details via a *JxfsStatusEvent*.

7.1 JxfsVDMStatus

This class specifies the status of the current VDM service mode.

7.1.1 Summary

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

Property	Type	Acces s	Initialized after
vdmStatus	int	R	
logicalDevices	java.lang.String[]	R	

Constructor	Parameter	Parameter-Type
JxfsVDMStatus	vdmStatus	int
	logicalDevices	java.lang.String[]

Method	Return	Meaning
get*Property*	*Property*	
isVDMActive	*boolean*	returns *true i*f the *vdmStatus* is JXFS_ VDM_ACTIVE
isVDMEnterPending	*boolean*	returns *true* if the *vdmStatus* is JXFS_ VDM_ENTER_PENDING
isVDMExitPending	*boolean*	returns *true* if the *vdmStatus* is JXFS_ VDM_EXIT_PENDING
isVDMInactive	*boolean*	returns *true* if the *vdmStatus* is JXFS_VDM_INACTIVE

Event	May occur after
Status Event	
JXFS_S_VDM_ENTER_REQUEST	*enterVDM(),*
JXFS_S_VDM_ENTERED	*enterVDMAcknowledge(),*
JXFS_S_VDM_EXIT_REQUEST	*exitVDM(),*
JXFS_S_VDM_EXITED	*exitVDMAcknowledge()*

7.1.2 Properties

7.1.2.1 vdmStatus

Type	***int***
Initial Value	JXFS_VDM_INACTIVE
Description	Specifies the service state as one of the following flags: JXFS_VDM_ACTIVE JXFS_VDM_ENTER_PENDING JXFS_VDM_EXIT_PENDING JXFS_VDM_INACTIVE

7.1.2.2 logicalDevices

Type	***java.lang.String[]***
Initial Value	null
Description	This array contains a list of the logical devices names that are relevant for this VDM. If the VDM is requested, all applications have to release and close all the devices in this list. If this property is equal to null, all devices under control of the VDM device service have to be released and closed. An array with no initialized string is not allowed. After a change to the JXFS_VDM_INACTIVE state, all *JxfsVDMStatus* instances received from the VDM device service will deliver a null as the logicalDevices property.

7.1.3 Methods

7.1.3.1 isVDMActive

Syntax	***boolean isVDMActive() throws JxfsException;***
Description	Returns *true* if the status of the service is JXFS_ VDM_ACTIVE
Parameter	**None**

7.1.3.2 isVDMEnterPending

Syntax	***boolean isVDMEnterPending() throws JxfsException;***
Description	Returns *true* if the status of the service is JXFS_ VDM_ENTER_PENDING
Parameter	**None**

7.1.3.3 isVDMExitPending

Syntax	***boolean isVDMExitPending() throws JxfsException;***
Description	Returns *true* if the status of the service is JXFS_VDM_EXIT_PENDING
Parameter	**None**

7.1.3.4 isVDMInactive

Syntax	***boolean isVDMInactive() throws JxfsException;***
Description	Returns *true* if the status of the service is JXFS_VDM_INACTIVE
Parameter	**None**

8 Enum Classes

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

8.1 JxfsVDMStatusSelectorEnum

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

Extends	Implements
JxfsStatusSelectorEnum	

Field	Returned Type	Description
vdmStatus	*JxfsVDMStatus*	Information about the current state of the VDM.
deviceList	*java.lang.String[]*	Array of logical devices under VDM control

9 Codes

9.1 Error Codes

Value	Meaning
JXFS_E_VDM_NOT_IN_VDM	The operation cannot be executed because the VDM is not active.

9.2 Status Codes

9.2.1.1 General Status Codes

General Status Codes that specify a value change.

Value	Meaning
JXFS_S_VDM	The general JxfsStatus of the device service changed.
JXFS_S_VDM_ENTER_REQUEST	An application has called the *enterVDM()* method. The details property contains a reference to a current instance of the *vdmStatus* property.
JXFS_S_VDM_ENTERED	The system has entered the vendor dependent mode. The details property contains a reference to a current instance of the *vdmStatus* property.
JXFS_S_VDM_EXIT_REQUEST	An application has called the *exitVDM()* method. The details property contains a reference to a current instance of the *vdmStatus* property.
JXFS_S_VDM_EXITED	The system has exited the vendor dependent mode. The details property contains a reference to a current instance of the *vdmStatus* property.

9.3 Operation ID Codes

Following codes specify the operation which generated the *JxfsOperationCompleteEvent*.

Value	Method
JXFS_O_VDM_ENTER_REQUEST	*enterVDM()*
JXFS_O_VDM_EXIT_REQUEST	*exitVDM()*

9.4 Numerical values

Value	Meaning
13000	JXFS_S_VDM
13001	JXFS_S_VDM_ENTER_REQUEST
13002	JXFS_S_VDM_ENTERED
13003	JXFS_S_VDM_EXIT_REQUEST
13004	JXFS_S_VDM_EXITED
13005	JXFS_O_VDM_ENTER_VDM
13006	JXFS_O_VDM_EXIT_VDM
13007	JXFS_E_VDM_NOT_IN_VDM
13008	JXFS_VDM_ACTIVE
13009	JXFS_VDM_ENTER_PENDING
13010	JXFS_VDM_EXIT_PENDING
13011	JXFS_VDM_INACTIVE

10 Device Service Interface Methods

The Device Service interface is common to all device services of this device type. It is used by the Device Controls to access the functionality of the device. This interface has to be implemented by any J/XFS Device Service.

The device type specific Device Service interface is similar to the Device Control interface. All device specific method calls are extended by an additional parameter (int control_id). This is always added as the last parameter in every operation.

The name of the device service interface is IJxfsVdmService. It is extended from IJxfsBaseService. The constants are defined in the IJxfsVDMConst interface.