# CEN

# WORKSHOP

# AGREEMENT

# CWA 15748-11

July 2008

**ICS** 35.240.50

English version

# Extensions for Financial Services (XFS) interface specification - Release 3.10 - Part 11: Vendor Dependent Mode Device Class Interface - Programmer's Reference



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

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



Ref. No.:CWA 15748-11:2008 E

# Table of Contents

# Foreword

This CWA is revision 3.10 of the XFS interface specification.

The CEN/ISSS XFS Workshop gathers suppliers as well as banks and other financial service companies. A list of companies participating in this Workshop and in support of this CWA is available from the CEN/ISSS Secretariat.

This CWA was formally approved by the XFS Workshop meeting on 2007-11-29. The specification is continuously reviewed and commented in the CEN/ISSS Workshop on XFS. It is therefore expected that an update of the specification will be published in due time as a CWA, superseding this revision 3.10.

The CWA is published as a multi-part document, consisting of:

Part 1: Application Programming Interface (API) - Service Provider Interface (SPI) - Programmer's Reference

Part 2: Service Classes Definition - Programmer's Reference

Part 3: Printer and Scanning Device Class Interface - Programmer's Reference

Part 4: Identification Card Device Class Interface - Programmer's Reference

Part 5: Cash Dispenser Device Class Interface - Programmer's Reference

Part 6: PIN Keypad Device Class Interface - Programmer's Reference

Part 7: Check Reader/Scanner Device Class Interface - Programmer's Reference

Part 8: Depository Device Class Interface - Programmer's Reference

Part 9: Text Terminal Unit Device Class Interface - Programmer's Reference

Part 10: Sensors and Indicators Unit Device Class Interface - Programmer's Reference

Part 11: Vendor Dependent Mode Device Class Interface - Programmer's Reference

Part 12: Camera Device Class Interface - Programmer's Reference

Part 13: Alarm Device Class Interface - Programmer's Reference

Part 14: Card Embossing Unit Device Class Interface - Programmer's Reference

Part 15: Cash-In Module Device Class Interface - Programmer's Reference

Part 16: Card Dispenser Device Class Interface - Programmer's Reference

Part 17: Barcode Reader Device Class Interface - Programmer's Reference

Part 18: Item Processing Module Device Class Interface- Programmer's Reference

Parts 19 - 28: Reserved for future use.

Parts 29 through 47 constitute an optional addendum to this CWA. They define the integration between the SNMP standard and the set of status and statistical information exported by the Service Providers.

Part 29: XFS MIB Architecture and SNMP Extensions - Programmer’s Reference

Part 30: XFS MIB Device Specific Definitions - Printer Device Class

Part 31: XFS MIB Device Specific Definitions - Identification Card Device Class

Part 32: XFS MIB Device Specific Definitions - Cash Dispenser Device Class

Part 33: XFS MIB Device Specific Definitions - PIN Keypad Device Class

Part 34: XFS MIB Device Specific Definitions - Check Reader/Scanner Device Class

Part 35: XFS MIB Device Specific Definitions - Depository Device Class

Part 36: XFS MIB Device Specific Definitions - Text Terminal Unit Device Class

Part 37: XFS MIB Device Specific Definitions - Sensors and Indicators Unit Device Class

Part 38: XFS MIB Device Specific Definitions - Camera Device Class

Part 39: XFS MIB Device Specific Definitions - Alarm Device Class

Part 40: XFS MIB Device Specific Definitions - Card Embossing Unit Class

Part 41: XFS MIB Device Specific Definitions - Cash-In Module Device Class

Part 42: Reserved for future use.

Part 43: XFS MIB Device Specific Definitions - Vendor Dependent Mode Device Class

Part 44: XFS MIB Application Management

Part 45: XFS MIB Device Specific Definitions - Card Dispenser Device Class

Part 46: XFS MIB Device Specific Definitions - Barcode Reader Device Class

Part 47: XFS MIB Device Specific Definitions - Item Processing Module Device Class

Parts 48 - 60 are reserved for future use.

Part 61: Application Programming Interface (API) - Service Provider Interface (SPI) - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 62: Printer Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 63: Identification Card Device Class Interface - Migration from Version 3.02 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 64: Cash Dispenser Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 65: PIN Keypad Device Class Interface - Migration from Version 3.03 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 66: Check Reader/Scanner Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 67: Depository Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 68: Text Terminal Unit Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 69: Sensors and Indicators Unit Device Class Interface - Migration from Version 3.01 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 70: Vendor Dependent Mode Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 71: Camera Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 72: Alarm Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 73: Card Embossing Unit Device Class Interface - Migration from Version 3.0 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

Part 74: Cash-In Module Device Class Interface - Migration from Version 3.02 (CWA 14050) to Version 3.10 (this CWA) - Programmer's Reference

In addition to these Programmer's Reference specifications, the reader of this CWA is also referred to a complementary document, called Release Notes. The Release Notes contain clarifications and explanations on the CWA specifications, which are not requiring functional changes. The current version of the Release Notes is available online from http://www.cen.eu/isss/Workshop/XFS.

The information in this document represents the Workshop's current views on the issues discussed as of the date of publication. It is furnished for informational purposes only and is subject to change without notice. CEN/ISSS makes no warranty, express or implied, with respect to this document.

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

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

Revision History:

| | | |
|---|---|---|
| 2.0 | November 11, 1996 | Initial Release. |
| 3.0 | October 18, 2000 | Updated for XFS version 3.0. |
| 3.10 | November 29, 2007 | For a description of changes see CWA 15748-70:2007 VDM Migration from Version 3.0 to Version 3.10. |

# 1. Introduction

## 1.1 Background to Release 3.10

The CEN/ISSS XFS Workshop aims to promote a clear and unambiguous specification defining a multi-vendor software interface to financial peripheral devices. The XFS (eXtensions for Financial Services) specifications are developed within the CEN/ISSS (European Committee for Standardization/Information Society Standardization System) Workshop environment. CEN/ISSS Workshops aim to arrive at a European consensus on an issue that can be published as a CEN Workshop Agreement (CWA).

The CEN/ISSS XFS Workshop encourages the participation of both banks and vendors in the deliberations required to create an industry standard. The CEN/ISSS XFS Workshop achieves its goals by focused sub-groups working electronically and meeting quarterly.

Release 3.10 of the XFS specification is based on a C API and is delivered with the continued promise for the protection of technical investment for existing applications. This release of the XFS specification has been prompted by a series of factors.

There has been a technical imperative to extend the scope of the existing specification to include new devices, such as the Barcode Reader, Card Dispenser and Item Processing Module.

Similarly, there has also been pressure, through implementation experience and additional requirements, to extend the functionality and capabilities of the existing devices covered by the specification.

## 1.2 XFS Service-Specific Programming

The service classes are defined by their service-specific commands and the associated data structures, error codes, messages, etc. These commands are used to request functions that are specific to one or more classes of Service Providers, but not all of them, and therefore are not included in the common API for basic or administration functions.

When a service-specific command is common among two or more classes of Service Providers, the syntax of the command is as similar as possible across all services, since a major objective of XFS is to standardize function codes and structures for the broadest variety of services. For example, using the **WFSExecute** function, the commands to read data from various services are as similar as possible to each other in their syntax and data structures.

In general, the specific command set for a service class is defined as a superset of the specific capabilities likely to be provided by the developers of the services of that class; thus any particular device will normally support only a subset of the defined command set.

There are three cases in which a Service Provider may receive a service-specific command that it does not support:

The requested capability is defined for the class of Service Providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability is ***not*** considered to be fundamental to the service. In this case, the Service Provider returns a successful completion, but does no operation. An example would be a request from an application to turn on a control indicator on a passbook printer; the Service Provider recognizes the command, but since the passbook printer it is managing does not include that indicator, the Service Provider does no operation and returns a successful completion to the application.

The requested capability is defined for the class of Service Providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability ***is*** considered to be fundamental to the service. In this case, a WFS_ERR_UNSUPP_COMMAND error is returned to the calling application. An example would be a request from an application to a cash dispenser to dispense coins; the Service Provider recognizes the command but, since the cash dispenser it is managing dispenses only notes, returns this error.

The requested capability is ***not*** defined for the class of Service Providers by the XFS specification. In this case, a WFS_ERR_INVALID_COMMAND error is returned to the calling application.

This design allows implementation of applications that can be used with a range of services that provide differing subsets of the functionalities that are defined for their service class. Applications may use the **WFSGetInfo** and **WFSAsyncGetInfo** commands to inquire about the capabilities of the service they are about to use, and modify their behavior accordingly, or they may use functions and then deal with WFS_ERR_UNSUPP_COMMAND error returns to make decisions as to how to use the service.

# 2. Vendor Dependent Mode

This specification describes the functionality of the services provided by the Vendor Dependent Mode (VDM) Service Provider under XFS, by defining the service-specific commands that can be issued, using the **WFSGetInfo, WFSAsyncGetInfo**, **WFSExecute** and **WFSAsyncExecute** functions.

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 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) XFS Service Providers 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 Service Provider can be seen as the central point through which all Enter and Exit VDM requests are synchronized.

Entry into, or exit from, Vendor Dependent Mode can be initiated either by an application or by the VDM Service Provider itself. If initiated by an application, then this application needs to issue the appropriate command to request entry or exit. If initiated by the VDM Service Provider i.e. some vendor dependent switch, then these request commands are in-appropriate and not issued.

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 sessions with XFS Service Providers as soon as possible and then issue an acknowledgement command to the VDM Service Provider when ready. Once all applications have acknowledged, the VDM Service Provider will issue event messages to these applications to indicate that the System is in Vendor Dependent Mode.

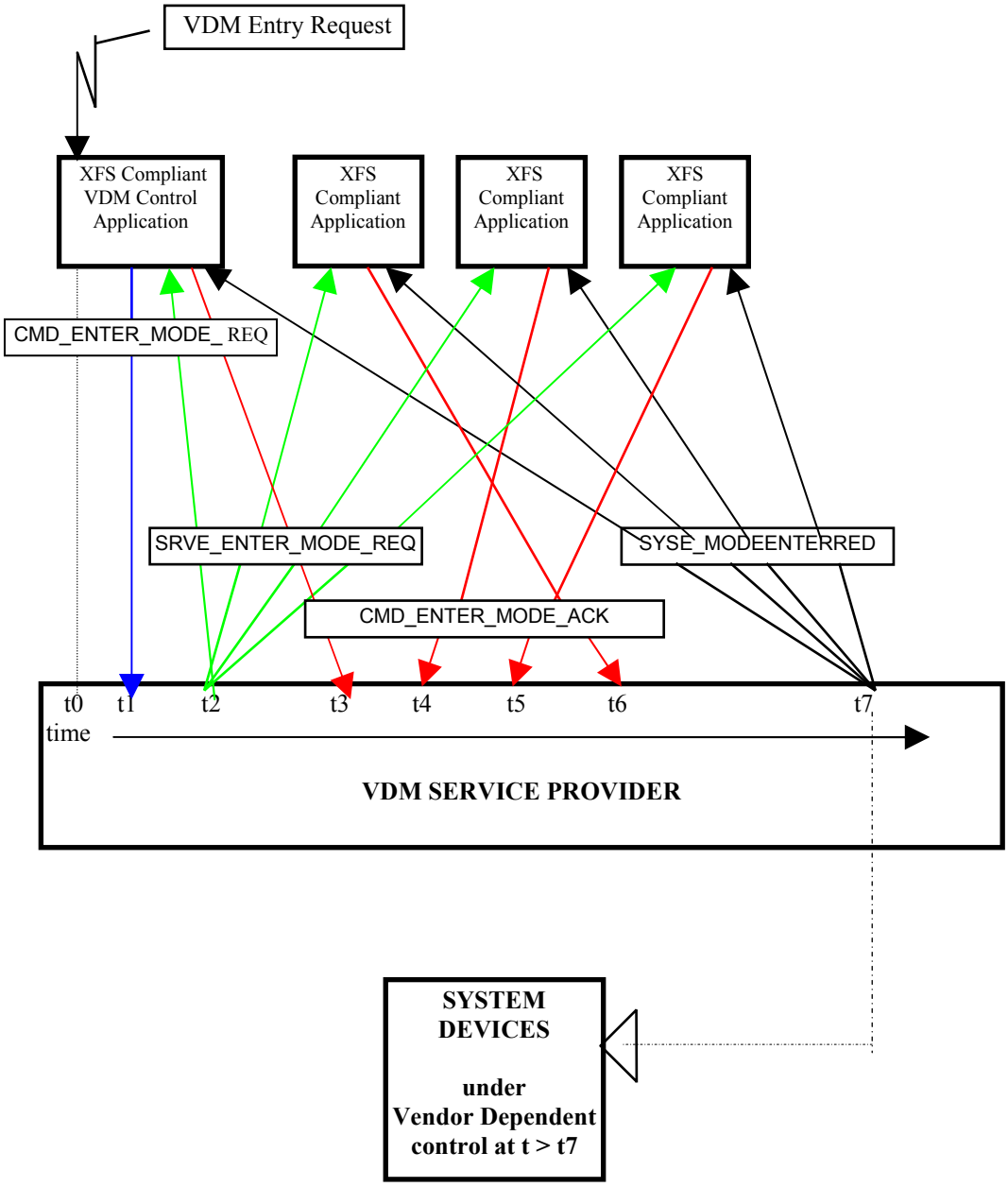
Similarly, once the exit request has been made all registered applications will be notified of the exit request by an event message. These applications must then issue an acknowledgement command to the VDM Service Provider immediately. Once all applications have acknowledged, the VDM Service Provider will issue event messages to these applications to indicate that the system has exited from Vendor Dependent Mode.

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

- Every XFS application should open a session with the VDM Service Provider passing a valid ApplId and then register for all VDM entry and exit notices.
- Before opening a session with any other XFS Service Provider, check the status of the VDM Service Provider. If Vendor Dependent Mode is not “Inactive”, do not open a session.
- When getting a VDM entry notice, close all open sessions with all XFS Service Providers 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 sessions with other XFS Service Providers.

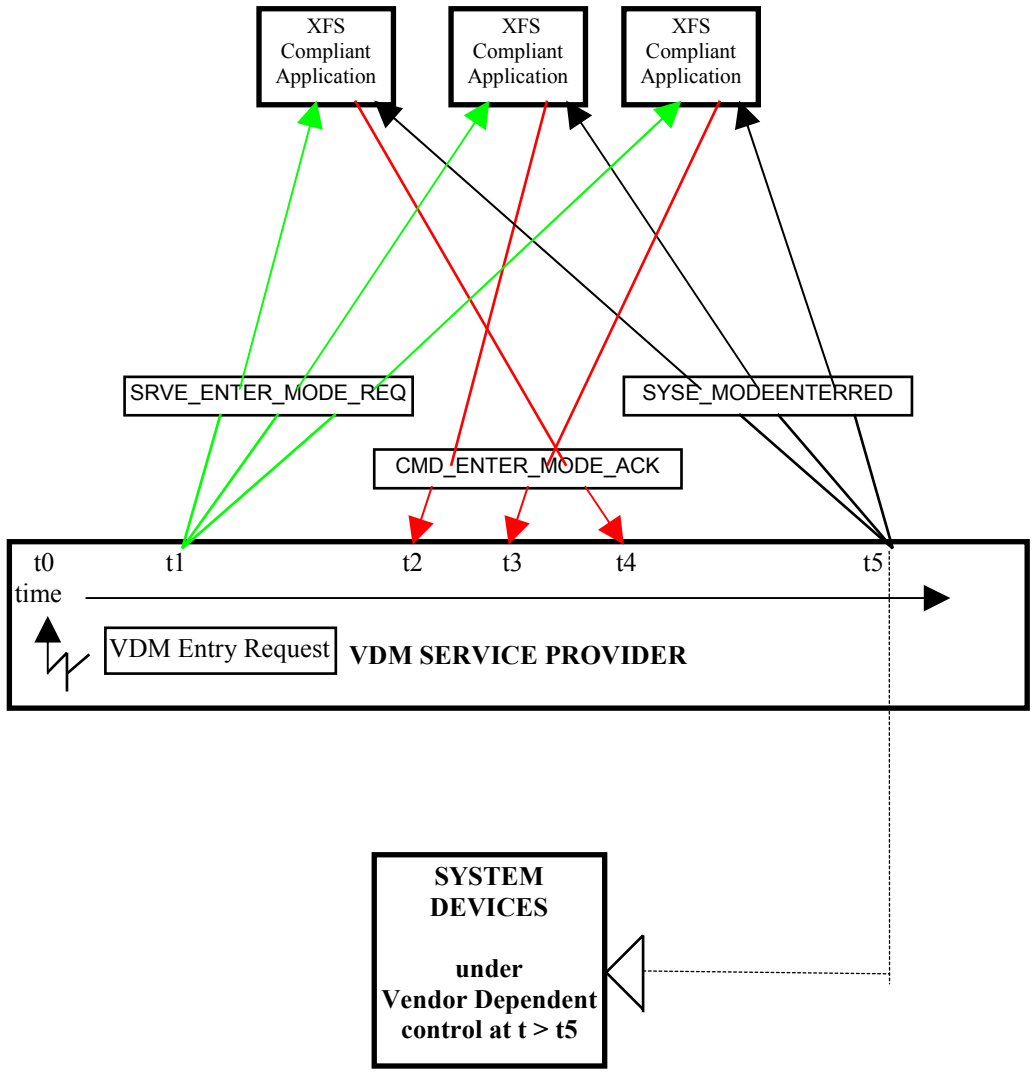
This is mandatory for self-service but optional for branch.

## 2.1 VDM Entry triggered by XFS Application



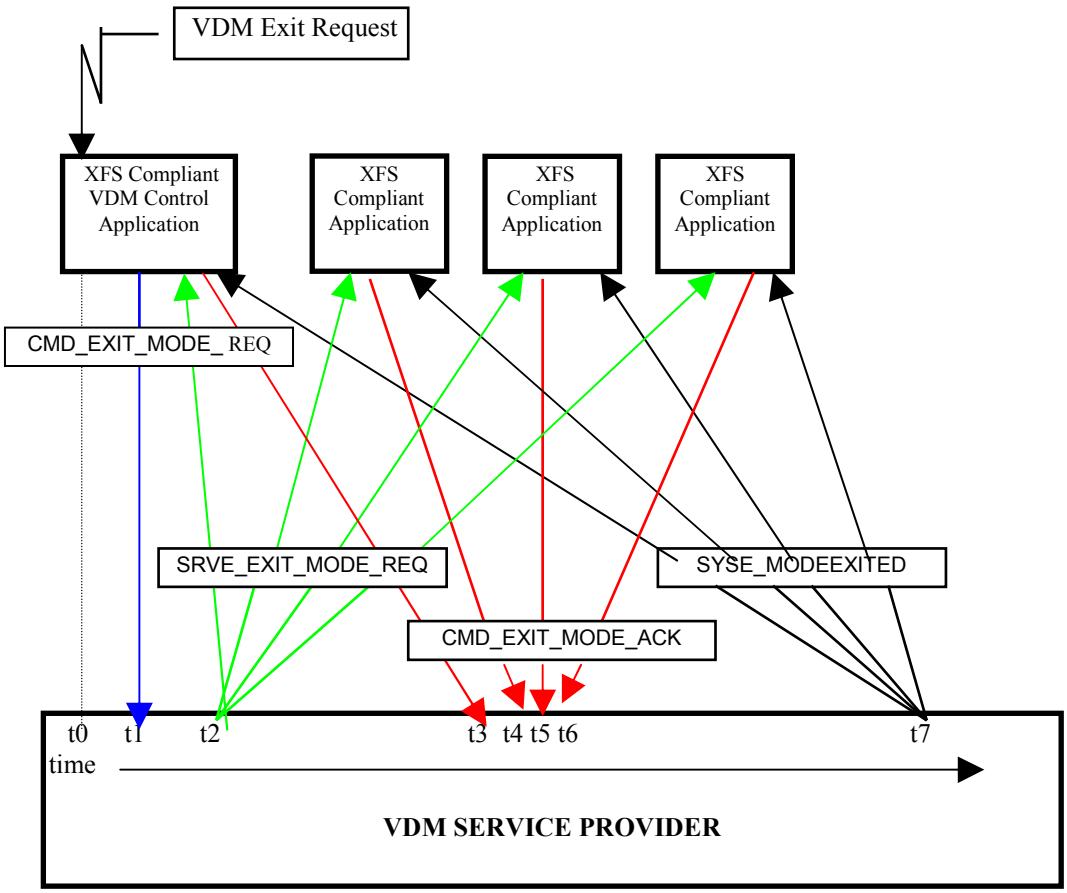

At time t0, status is “Inactive” and a request to Enter VDM arises from within the Application system.
At time t1, an Application Process/Thread/Function issues the CMD_ENTER_MODE_REQ Execute cmd. Status then becomes “Enter Pending”.
At time t2, the VDM Service Provider issues the SRVE_ENTER_MODE_REQ Event to all registered applications.
At time t3, the VDM Service Provider receives a CMD_ENTER_MODE_ACK Execute command from a XFS Compliant Application.
At time t4, the VDM Service Provider receives a CMD_ENTER_MODE_ACK Execute command from a XFS Compliant Application.
At time t5, the VDM Service Provider receives a CMD_ENTER_MODE_ACK Execute command from another XFS Compliant Application.
At time t6, the VDM Service Provider receives a CMD_ENTER_MODE_ACK Execute command from the last XFS Compliant Application.
At time t7, the VDM Service Provider issues the SYSE_MODEENTERED Event to all registered applications Status then becomes “Active”.
The system is now in Vendor Dependent Mode and a Vendor Dependent Application can exclusively use the system devices in a Vendor Dependent manner.

## 2.2 VDM Entry triggered by Vendor Dependent Switch




At time t0, status is “Inactive” and a request to Enter VDM arises from within the Vendor System.
Status then becomes “Enter Pending”.
At time t1, the VDM Service Provider issues the SRVE_ENTER_MODE_REQ Event to all registered applications.
At time t2, the VDM Service Provider receives a CMD_ENTER_MODE_ACK Execute command from a XFS Compliant Application.
At time t3, the VDM Service Provider receives a CMD_ENTER_MODE_ACK Execute command from another XFS Compliant Application.
At time t4, the VDM Service Provider receives a CMD_ENTER_MODE_ACK Execute command from the last XFS Compliant Application.
At time t5, the VDM Service Provider issues the SYSE_MODEENTERED Event to all registered applications
Status then becomes “Active”.
The system is now in Vendor Dependent Mode and a Vendor Dependent Application can exclusively use the system devices in a Vendor Dependent manner.

## 2.3 VDM Exit triggered by XFS Application

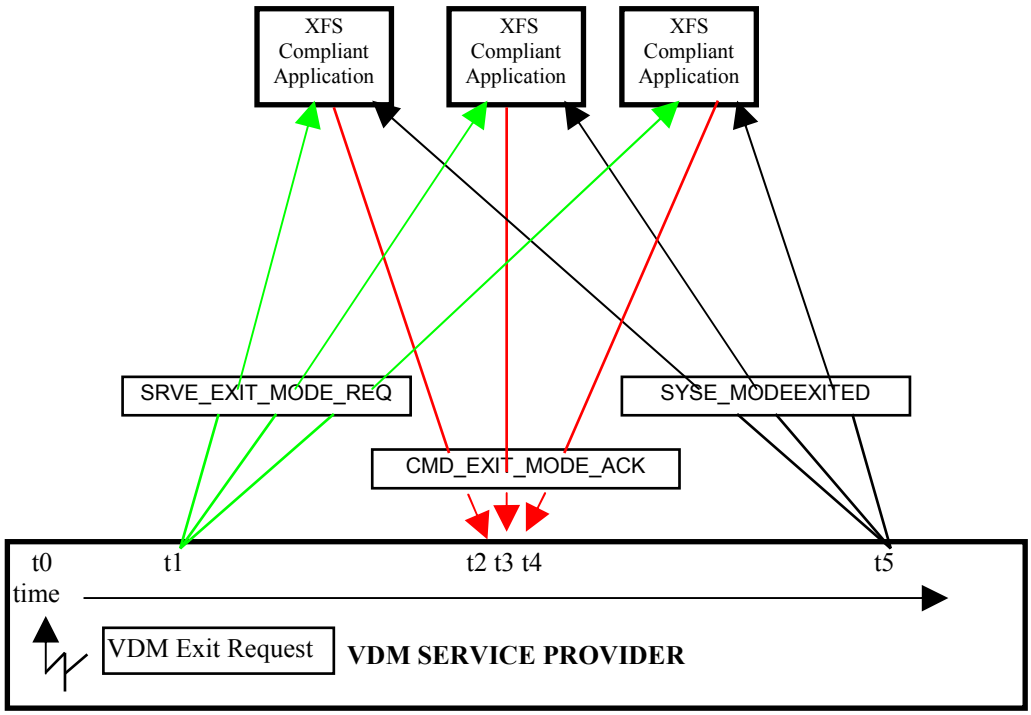



**SYSTEM DEVICES**

**under CEN/XFS control at t > t7**

At time t0, status is “Active” and a request to Exit VDM arises from within the Application system.
At time t1, an Application Process/Thread/Function issues the CMD_EXIT_MODE_REQ Execute cmd.
Status then becomes “Exit Pending”.
At time t2, the VDM Service Provider issues the SRVE_EXIT_MODE_REQ Event to all registered applications.
At time t3, the VDM Service Provider receives a CMD_EXIT_MODE_ACK Execute command from a XFS Compliant Application.
At time t4, the VDM Service Provider receives a CMD_EXIT_MODE_ACK Execute command from a XFS Compliant Application.
At time t5, the VDM Service Provider receives a CMD_EXIT_MODE_ACK Execute command from another XFS Compliant Application.
At time t6, the VDM Service Provider receives a CMD_EXIT_MODE_ACK Execute command from the last XFS Compliant Application.
At time t7, the VDM Service Provider issues the SYSE_MODEEXITED Event to all registered applications
Status then becomes “Inactive”.
The system is now no longer in Vendor Dependent Mode and the XFS Compliant Applications can re-open any required services with other XFS Service Providers.

## 2.4 VDM Exit triggered by Vendor Dependent Switch




**SYSTEM DEVICES**

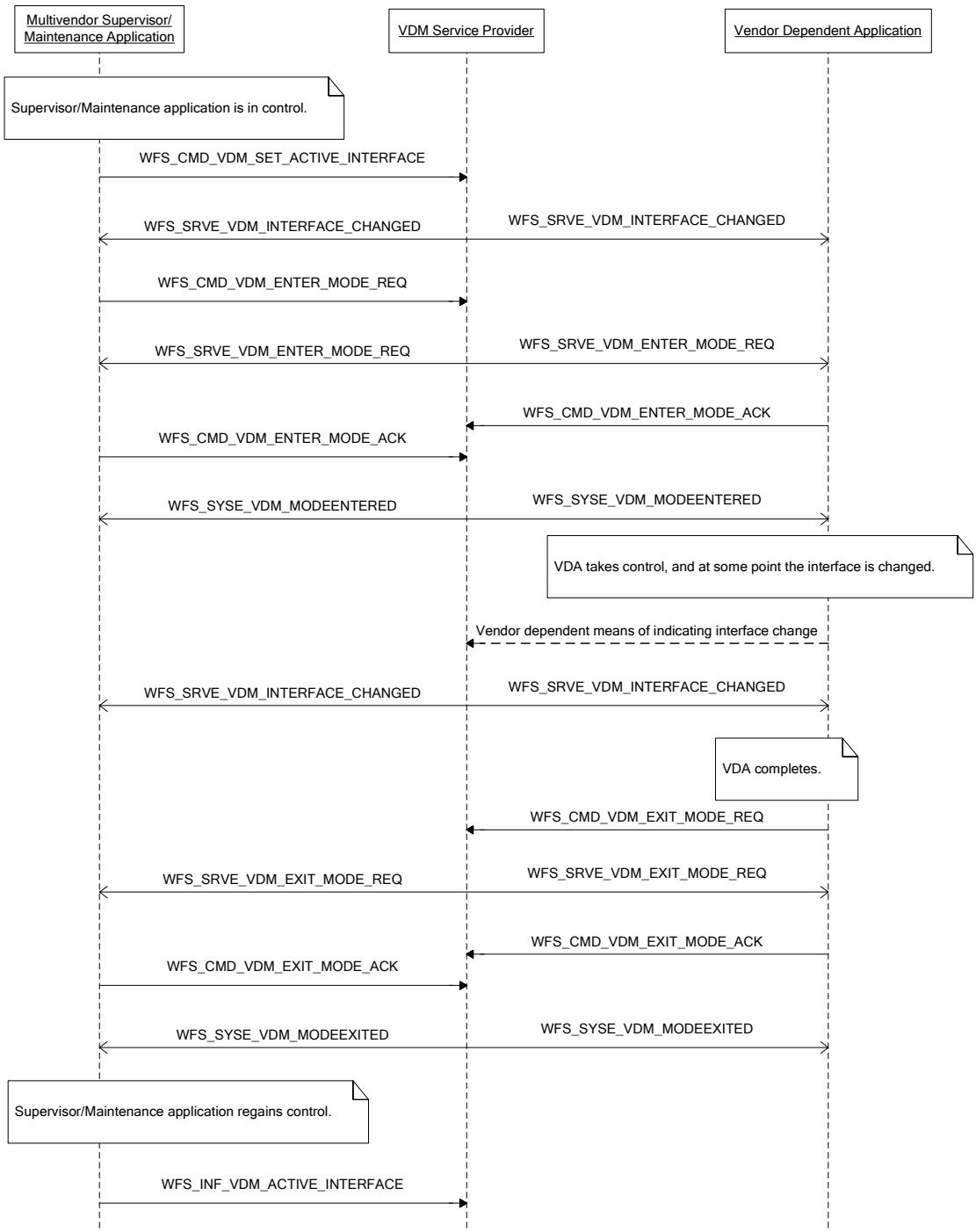
**under CEN/XFS control at t > t5**

At time t0, status is “Active” and a request to Exit VDM arises from within the Vendor System.
Status then becomes “Exit Pending”.
At time t1, the VDM Service Provider issues the SRVE_EXIT_MODE_REQ Event to all registered applications.
At time t2, the VDM Service Provider receives a CMD_EXIT_MODE_ACK Execute command from a XFS Compliant Application.
At time t3, the VDM Service Provider receives a CMD_EXIT_MODE_ACK Execute command from another XFS Compliant Application.
At time t4, the VDM Service Provider receives a CMD_EXIT_MODE_ACK Execute command from the last XFS Compliant Application.
At time t5, the VDM Service Provider issues the SYSE_MODEEXITED Event to all registered applications.
Status then becomes “Inactive”.
The system is now no longer in Vendor Dependent Mode and the XFS Compliant Applications can re-open any required services with other XFS Service Providers.

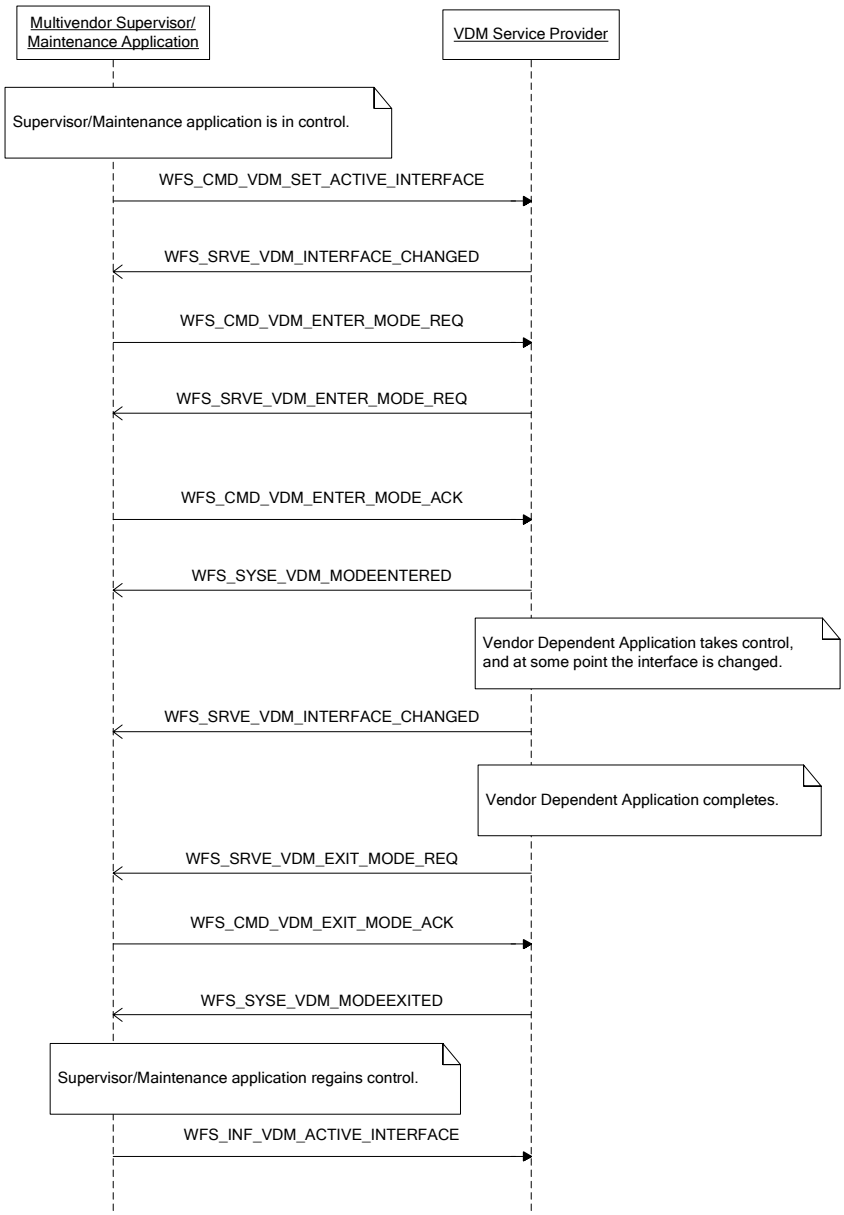
## 2.5 Controlling / Determining the Active Interface

While in a supervisor/maintenance application or Vendor Dependent Mode, it is possible to transfer from the consumer interface to the operator interface and vice-versa. The active interface can be determined and controlled, as described here.

### 2.5.1 Vendor Dependent Application independent of the VDM Service Provider



### 2.5.2 Vendor Dependent Application under Control of the VDM Service Provider



# 3. References

1. XFS Application Programming Interface (API)/Service Provider Interface (SPI), Programmer's Reference Revision 3.10

# 4. Info Commands

## 4.1 WFS_INF_VDM_STATUS

**Description** This command indicates whether or not the system is in Vendor Dependent Mode. It will also indicate which applications have not responded to the WFS_SRVE_ENTER_MODE_REQ event or WFS_SRVE_EXIT_MODE_REQ event if the current service status is WFS_VDM_ENTERPENDING or WFS_VDM_EXITPENDING respectively.

**Input Param** None.

**Output Param** LPWFSVDMSTATUS lpStatus;

```
typedef struct _wfs_vdm_status
     {
     WORD                 wDevice;
     WORD                 wService;
     LPWFSVDMAPPSTATUS    *lppAppStatus;
     LPSTR                lpszExtra;
     } WFSVDMSTATUS, *LPWFSVDMSTATUS;
```

*wDevice*
Specifies the status of the Vendor Dependent Mode Service Provider. Status will be one of the following flags:

| Value | Meaning |
|---|---|
| WFS_VDM_DEVONLINE | Vendor Dependent Mode service available. |
| WFS_VDM_DEVOFFLINE | Vendor Dependent Mode service unavailable. |

*wService*
Specifies the Service state as one of the following flags:

| Value | Meaning |
|---|---|
| WFS_VDM_ENTERPENDING | Vendor Dependent Mode enter request pending. |
| WFS_VDM_ACTIVE | Vendor Dependent Mode active. |
| WFS_VDM_EXITPENDING | Vendor Dependent Mode exit request pending. |
| WFS_VDM_INACTIVE | Vendor Dependent Mode inactive. |

*lppAppStatus*
Pointer to a NULL-terminated array of pointers to WFSVDMAPPSTATUS structures:

```
typedef struct _wfs_vdm_appstatus
     {
     LPSTR               lpszAppID;
     WORD                wAppStatus;
     } WFSVDMAPPSTATUS, *LPWFSVDMAPPSTATUS;
```

*lpszAppID*
Application ID string.

*wAppStatus*
Specifies whether the particular application is ready for the system to enter or exit Vendor Dependent Mode. Values can be one of the following:

| Value | Meaning |
|---|---|
| WFS_VDM_ENTERPENDING | Application not yet ready to enter VDM. |
| WFS_VDM_ACTIVE | Application ready to enter VDM. |
| WFS_VDM_EXITPENDING | Application not yet ready to exit VDM. |
| WFS_VDM_INACTIVE | Application ready to exit VDM. |

*lpszExtra*
Pointer to a list of vendor-specific, or any other extended, information. The information is returned as a series of *"key=value"* strings so that it is easily extensible by Service Providers. Each string is null-terminated, with the final string terminating with two null characters. An empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters

**Error Codes** Only the generic error codes defined in [Ref. 1] can be generated by this command.

**Comments** Applications which require or expect specific information to be present in the *lpszExtra* parameter may not be device or vendor-independent.

## 4.2 WFS_INF_VDM_CAPABILITIES

**Description** This command is used to retrieve the capabilities of the VDM Service Provider.

**Input Param** None.

**Output Param** LPWFSVDMCAPS lpCaps;

```
typedef struct _wfs_vdm_caps
     {
     WORD                wClass;
     LPSTR               lpszExtra;
     } WFSVDMCAPS, *LPWFSVDMCAPS;
```

*wClass*
Specifies the logical service class as SERVICE_CLASS_VDM.

*lpszExtra*
Pointer to a list of vendor-specific, or any other extended, information. The information is returned as a series of *"key=value"* strings so that it is easily extensible by Service Providers. Each string is null-terminated, with the final string terminating with two null characters. An empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters

**Error Codes** Only the generic error codes defined in [Ref. 1] can be generated by this command.

**Comments** Applications which require or expect specific information to be present in the *lpszExtra* parameter may not be device or vendor-independent.

## 4.3 WFS_INF_VDM_ACTIVE_INTERFACE

**Description** This command is used to retrieve the interface that should be used by supervisor/maintenance mode applications.

**Input Param** None.

**Output Param** LPWFSVDMACTIVEINTERFACE lpActiveInterface;

```
typedef struct _wfs_vdm_active_interface
     {
     WORD                 wActiveInterface;
     } WFSVDMACTIVEINTERFACE, *LPWFSVDMACTIVEINTERFACE;
```

*wActiveInterface*
Specifies the interface as one of the following values:

| Value | Meaning |
|---|---|
| WFS_VDM_CONSUMER_INTERFACE | The consumer interface. |
| WFS_VDM_OPERATOR_INTERFACE | The operator interface. |

**Error Codes** Only the generic error codes defined in [Ref. 1] can be generated by this command.

**Comments** None.

# 5. Execute Commands

## 5.1 WFS_CMD_VDM_ENTER_MODE_REQ

**Description** This command is issued by an application to indicate a logical request to enter Vendor Dependent Mode. The VDM Service Provider will then indicate the request to all registered applications by sending a WFS_SRVE_VDM_ENTER_MODE_REQ event and then wait for an acknowledgement back from each registered application before putting the system into Vendor Dependent Mode. The Service Provider status will change to WFS_VDM_ENTERPENDING on receipt of this command and will prevail until all applications have acknowledged, at which time the status will change to WFS_VDM_ACTIVE and the WFS_CMD_VDM_ENTER_MODE_REQ completes.

If the command fails when the status is WFS_VDM_ENTERPENDING, the status is changed to WFS_VDM_INACTIVE and a WFS_SYSE_VDM_MODEEXITED event is sent to all registered applications.

**Input Param** None.

**Output Param** None.

**Error Codes** Only the generic error codes defined in [Ref. 1] can be generated by this command.

**Events** In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

| Value | Meaning |
|---|---|
| WFS_SRVE_VDM_ENTER_MODE_REQ | Request to enter VDM Mode. |
| WFS_SYSE_VDM_MODEENTERED | The system has entered VDM. |
| WFS_SYSE_VDM_MODEEXITED | The system has exited VDM. |

**Comments** None.

## 5.2 WFS_CMD_VDM_ENTER_MODE_ACK

| | |
|---|---|
| **Description** | This command is issued by a registered application as an acknowledgement to the WFS_SRVE_VDM_ENTER_MODE_REQ event and it indicates that the application is ready for the system to enter Vendor Dependent Mode. All registered applications (including the application that issued the request to enter Vendor Dependent Mode) must respond before Vendor Dependent Mode will be entered. Completion of this command is immediate. |
| **Input Param** | None. |
| **Output Param** | None. |
| **Error Codes** | Only the generic error codes defined in [Ref. 1] can be generated by this command. |
| **Events** | Only the generic events defined in [Ref. 1] can be generated by this command. |
| **Comments** | None. |

## 5.3 WFS_CMD_VDM_EXIT_MODE_REQ

**Description** This command is issued by an application to indicate a logical request to exit Vendor Dependent Mode. The VDM Service Provider will then indicate the request to all registered applications by sending a WFS_SRVE_VDM_EXIT_MODE_REQ event and then wait for an acknowledgement back from each registered application before removing the system from Vendor Dependent Mode. The Service Class status will change to WFS_VDM_EXITPENDING on receipt of this command and will prevail until all applications have acknowledged, at which time the status will change to WFS_VDM_INACTIVE and the WFS_CMD_VDM_EXIT_MODE_REQ completes.

If the command fails when the status is WFS_VDM_EXITPENDING, the status is changed to WFS_VDM_ACTIVE and a WFS_SYSE_VDM_MODEENTERED event is sent to all registered applications.

**Input Param** None.

**Output Param** None.

**Error Codes** Only the generic error codes defined in [Ref. 1] can be generated by this command.

**Events** In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

| Value | Meaning |
|---|---|
| WFS_SRVE_VDM_EXIT_MODE_REQ | Request to exit VDM. |
| WFS_SYSE_VDM_MODEENTERED | The system has entered VDM. |
| WFS_SYSE_VDM_MODEEXITED | The system has exited VDM. |

**Comments** None.

## 5.4 WFS_CMD_VDM_EXIT_MODE_ACK

| | |
|---|---|
| **Description** | This command is issued by a registered application as an acknowledgement to the WFS_SRVE_VDM_EXIT_MODE_REQ event and it indicates that the application is ready for the system to exit Vendor Dependent Mode. All registered applications (including the application that issued the request to exit Vendor Dependent Mode) must respond before Vendor Dependent Mode will be exited. Completion of this command is immediate. |
| **Input Param** | None. |
| **Output Param** | None. |
| **Error Codes** | Only the generic error codes defined in [Ref. 1] can be generated by this command. |
| **Events** | Only the generic events defined in [Ref. 1] can be generated by this command. |
| **Comments** | None. |

## 5.5 WFS_CMD_VDM_SET_ACTIVE_INTERFACE

**Description** This command is used to indicate which interface should be used by supervisor/maintenance mode applications. A supervisor/maintenance mode application can issue this command before entry to VDM to ensure that a Vendor Dependent Application (VDA) starts on the correct interface.

**Input Param** LPWFSVDMACTIVEINTERFACE lpActiveInterface;

*lpActiveInterface*
Pointer to a WFSVDMACTIVEINTERFACE structure is specified in the documentation of the WFS_INF_VDM_ACTIVE_INTERFACE command.

**Output Param** None.

**Error Codes** Only the generic error codes defined in [Ref. 1] can be generated by this command.

**Events** In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

| Value | Meaning |
|---|---|
| WFS_SRVE_VDM_INTERFACECHANGED | The active VDM interface has changed. |

**Comments** None.

# 6. Events

## 6.1 WFS_SRVE_VDM_ENTER_MODE_REQ

**Description** This service event is used to indicate the request to enter Vendor Dependent Mode.

**Event Param** None.

**Comments** None.

## 6.2 WFS_SRVE_VDM_EXIT_MODE_REQ

**Description** This service event is used to indicate the request to exit Vendor Dependent Mode.

**Event Param** None.

**Comments** None.

## 6.3 WFS_SYSE_VDM_MODEENTERED

**Description** This system event is used to indicate that the system has entered Vendor Dependent Mode.

**Event Param** None.

**Comments** None.

## 6.4 WFS_SYSE_VDM_MODEEXITED

**Description** This system event is used to indicate that the system has exited Vendor Dependent Mode.

**Event Param** None.

**Comments** None.

## 6.5 WFS_SRVE_VDM_INTERFACECHANGED

**Description** This service event is used to indicate that the required interface has changed. This can be as a result of a WFS_CMD_VDM_SET_ACTIVE_INTERFACE command, or when the active interface is changed through vendor dependent means while in VDM. The *wActiveInterface* field of the WFSVDMACTIVEINTERFACE structure indicates which interface has been selected.

**Event Param** LPWFSVDMACTIVEINTERFACE lpActiveInterface;

*lpActiveInterface*
Pointer to a WFSVDMACTIVEINTERFACE structure. For a description of the WFSVDMACTIVEINTERFACE structure refer to the WFS_INF_VDM_ACTIVE_INTERFACE command.

**Comments** None.

# 7. C-Header file

```
/******************************************************************************
*                                                                             *
* xfsvdm.h    XFS - Vendor Dependent Mode (VDM) definitions                   *
*                                                                             *
*             Version 3.10  (29/11/2007)                                      *
*                                                                             *
******************************************************************************/

#ifndef __INC_XFSVDM__H
#define __INC_XFSVDM__H

#ifdef __cplusplus
extern "C" {
#endif

#include <xfsapi.h>

/*   be aware of alignment   */
#pragma pack(push,1)

/* values of WFSVDMCAPS.wClass */

#define WFS_SERVICE_CLASS_VDM                   (9)
#define WFS_SERVICE_CLASS_VERSION_VDM           (0x0A03) /* Version 3.10 */
#define WFS_SERVICE_CLASS_NAME_VDM              "VDM"

#define VDM_SERVICE_OFFSET                      (WFS_SERVICE_CLASS_VDM * 100)

/* VDM Info Commands */

#define WFS_INF_VDM_STATUS                      (VDM_SERVICE_OFFSET + 1)
#define WFS_INF_VDM_CAPABILITIES                (VDM_SERVICE_OFFSET + 2)
#define WFS_INF_VDM_ACTIVE_INTERFACE            (VDM_SERVICE_OFFSET + 3)

/* VDM Execute Commands */

#define WFS_CMD_VDM_ENTER_MODE_REQ              (VDM_SERVICE_OFFSET + 1)
#define WFS_CMD_VDM_ENTER_MODE_ACK              (VDM_SERVICE_OFFSET + 2)
#define WFS_CMD_VDM_EXIT_MODE_REQ               (VDM_SERVICE_OFFSET + 3)
#define WFS_CMD_VDM_EXIT_MODE_ACK               (VDM_SERVICE_OFFSET + 4)
#define WFS_CMD_VDM_SET_ACTIVE_INTERFACE        (VDM_SERVICE_OFFSET + 5)

/* VDM Messages */

#define WFS_SRVE_VDM_ENTER_MODE_REQ             (VDM_SERVICE_OFFSET + 1)
#define WFS_SRVE_VDM_EXIT_MODE_REQ              (VDM_SERVICE_OFFSET + 2)
#define WFS_SYSE_VDM_MODEENTERED                (VDM_SERVICE_OFFSET + 3)
#define WFS_SYSE_VDM_MODEEXITED                 (VDM_SERVICE_OFFSET + 4)
#define WFS_SRVE_VDM_INTERFACE_CHANGED          (VDM_SERVICE_OFFSET + 5)

/* values of WFSVDMSTATUS.wDevice */

#define WFS_VDM_DEVONLINE                       WFS_STAT_DEVONLINE
#define WFS_VDM_DEVOFFLINE                      WFS_STAT_DEVOFFLINE

/* values of WFSVDMSTATUS.wService */

#define WFS_VDM_ENTERPENDING                    (0)
#define WFS_VDM_ACTIVE                          (1)
#define WFS_VDM_EXITPENDING                     (2)
#define WFS_VDM_INACTIVE                        (3)

/* values of WFSVDMACTIVEINTERFACE.wActiveInterface */

#define WFS_VDM_OPERATOR_INTERFACE              (0)
#define WFS_VDM_CONSUMER_INTERFACE              (1)
```

```
/*================================================================*/
/* VDM Info Command Structures and variables */
/*================================================================*/

typedef struct _wfs_vdm_appstatus
{
    LPSTR                  lpszAppID;
    WORD                   wAppStatus;
} WFSVDMAPPSTATUS, *LPWFSVDMAPPSTATUS;

typedef struct _wfs_vdm_status
{
     WORD                  wDevice;
     WORD                  wService;
     LPWFSVDMAPPSTATUS    *lppAppStatus;
     LPSTR              lpszExtra;
} WFSVDMSTATUS, *LPWFSVDMSTATUS;

typedef struct _wfs_vdm_caps
{
     WORD                  wClass;
     LPSTR                 lpszExtra;
} WFSVDMCAPS, *LPWFSVDMCAPS;

typedef struct _wfs_vdm_active_interface
{
     WORD                  wActiveInterface;
} WFSVDMACTIVEINTERFACE, *LPWFSVDMACTIVEINTERFACE;

/*================================================================*/
/* VDM Execute Command Structures */
/*================================================================*/

/*================================================================*/
/* VDM Message Structures */
/*================================================================*/

/*   restore alignment   */
#pragma pack(pop)

#ifdef __cplusplus
}       /*extern "C"*/
#endif
#endif    /* __INC_XFSVDM__H */
```