CEN

WORKSHOP

AGREEMENT

CWA 16926-11

February 2020

ICS 35.200; 35.240.15; 35.240.40

English version

Extensions for Financial Services (XFS) interface specification Release 3.40 - Part 11: Vendor Dependent Mode Device Class Interface - Programmer's Reference

This CEN Workshop Agreement has been drafted and approved by a Workshop of representatives of interested parties, the constitution of which is indicated in the foreword of this Workshop Agreement.

The formal process followed by the Workshop in the development of this Workshop Agreement has been endorsed by the National Members of CEN but neither the National Members of CEN nor the CEN-CENELEC Management Centre can be held accountable for the technical content of this CEN Workshop Agreement or possible conflicts with standards or legislation.

This CEN Workshop Agreement can in no way be held as being an official standard developed by CEN and its Members.

This CEN Workshop Agreement is publicly available as a reference document from the CEN Members National Standard Bodies.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.

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

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Ref. No.:CWA 16926-11:2020 E

Table of Contents

European Foreword

This CEN Workshop Agreement has been developed in accordance with the CEN-CENELEC Guide 29 "CEN/CENELEC Workshop Agreements – The way to rapid consensus" and with the relevant provisions of CEN/CENELEC Internal Regulations - Part 2. It was approved by a Workshop of representatives of interested parties on 2019-10-08, the constitution of which was supported by CEN following several public calls for participation, the first of which was made on 1998-06-24. However, this CEN Workshop Agreement does not necessarily include all relevant stakeholders.
The final text of this CEN Workshop Agreement was provided to CEN for publication on 2019-12-12.
The following organizations and individuals developed and approved this CEN Workshop Agreement:

- ATM Japan LTD
- AURIGA SPA
- BANK OF AMERICA
- CASHWAY TECHNOLOGY
- CHINAL ECTRONIC FINANCIAL EQUIPMENT SYSTEM CO.
- CIMA SPA
- CLEAR2PAY SCOTLAND LIMITED
- DIEBOLD NIXDORF
- EASTERN COMMUNICATIONS CO. LTD – EASTCOM
- FINANZ INFORMATIK
- FUJITSU FRONTECH LIMITED
- FUJITSU TECHNOLOGY
- GLORY LTD
- GRG BANKING EQUIPMENT HK CO LTD
- HESS CASH SYSTEMS GMBH & CO. KG
- HITACHI OMRON TS CORP.
- HYOSUNG TNS INC
- JIANGSU GUOGUANG ELECTRONIC INFORMATION TECHNOLOGY
- KAL
- KEBA AG
- NCR FSG
- NEC CORPORATION
- OKI ELECTRIC INDUSTRY SHENZHEN
- OKI ELECTRONIC INDUSTRY CO
- PERTO S/A

- REINER GMBH & CO KG
- SALZBURGER BANKEN SOFTWARE
- SIGMA SPA
- TEB
- ZIJIN FULCRUM TECHNOLOGY CO

It is possible that some elements of this CEN/CWA may be subject to patent rights. The CEN-CENELEC policy on patent rights is set out in CEN-CENELEC Guide 8 “Guidelines for Implementation of the Common IPR Policy on Patents (and other statutory intellectual property rights based on inventions)”. CEN shall not be held responsible for identifying any or all such patent rights.
The Workshop participants have made every effort to ensure the reliability and accuracy of the technical and non-technical content of CWA 16926-11, but this does not guarantee, either explicitly or implicitly, its correctness. Users of CWA 16926-11 should be aware that neither the Workshop participants, nor CEN can be held liable for damages or losses of any kind whatsoever which may arise from its application. Users of CWA 16926-11 do so on their own responsibility and at their own risk.

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

Part 19: Biometrics Device Class Interface - Programmer's Reference

Parts 20 - 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

Part 48: XFS MIB Device Specific Definitions - Biometrics Device Class

Parts 49 - 60 are reserved for future use.

Part 61: Application Programming Interface (API) - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Service Provider Interface (SPI) - Programmer's Reference

Part 62: Printer and Scanning Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 63: Identification Card Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 64: Cash Dispenser Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 65: PIN Keypad Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 66: Check Reader/Scanner Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 67: Depository Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 68: Text Terminal Unit Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 69: Sensors and Indicators Unit Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 70: Vendor Dependent Mode Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 71: Camera Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 72: Alarm Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 73: Card Embossing Unit Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 74: Cash-In Module Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 75: Card Dispenser Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 76: Barcode Reader Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (this CWA) - Programmer's Reference

Part 77: Item Processing Module Device Class Interface - Migration from Version 3.30 (CWA 16926) to Version 3.40 (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: https://www.cen.eu/work/Sectors/Digital_society/Pages/WSXFS.aspx.

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

1. Introduction

1.1 Background to Release 3.40

The CEN/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 (European Committee for Standardization/Information Society Standardization System) Workshop environment. CEN Workshops aim to arrive at a European consensus on an issue that can be published as a CEN Workshop Agreement (CWA).

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

Release 3.40 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 specification extends the functionality and capabilities of the existing devices covered by the specification:

- Common API level based 'Service Information' command to report Service Provider information, data and versioning.
- Common API level based events to report changes in status and invalid parameters.
- Support for Advanced Encryption Standard (AES) in PIN.
- VDM Entry Without Closing XFS Service Providers.
- Addition of a Biometrics device class.
- CDM/CIM Note Classification List handling.
- Support for Derived Unique Key Per Transaction (DUKPT) in PIN.
- Addition of Transaction Start/End commands.
- Addition of explicit CIM Prepare/Present commands.

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 for Execute commands or WFS_ERR_UNSUPP_CATEGORY error for Info commands is returned to the calling application. An example would be a request from an application to a cash dispenser to retract items where the dispenser hardware does not have that capability; the Service Provider recognizes the command but, since the cash dispenser it is managing is unable to fulfil the request, 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 for Execute commands or WFS_ERR_INVALID_CATEGORY error for Info commands 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 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 (except for specific Service Providers which explicitly allow sessions to remain open) 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.

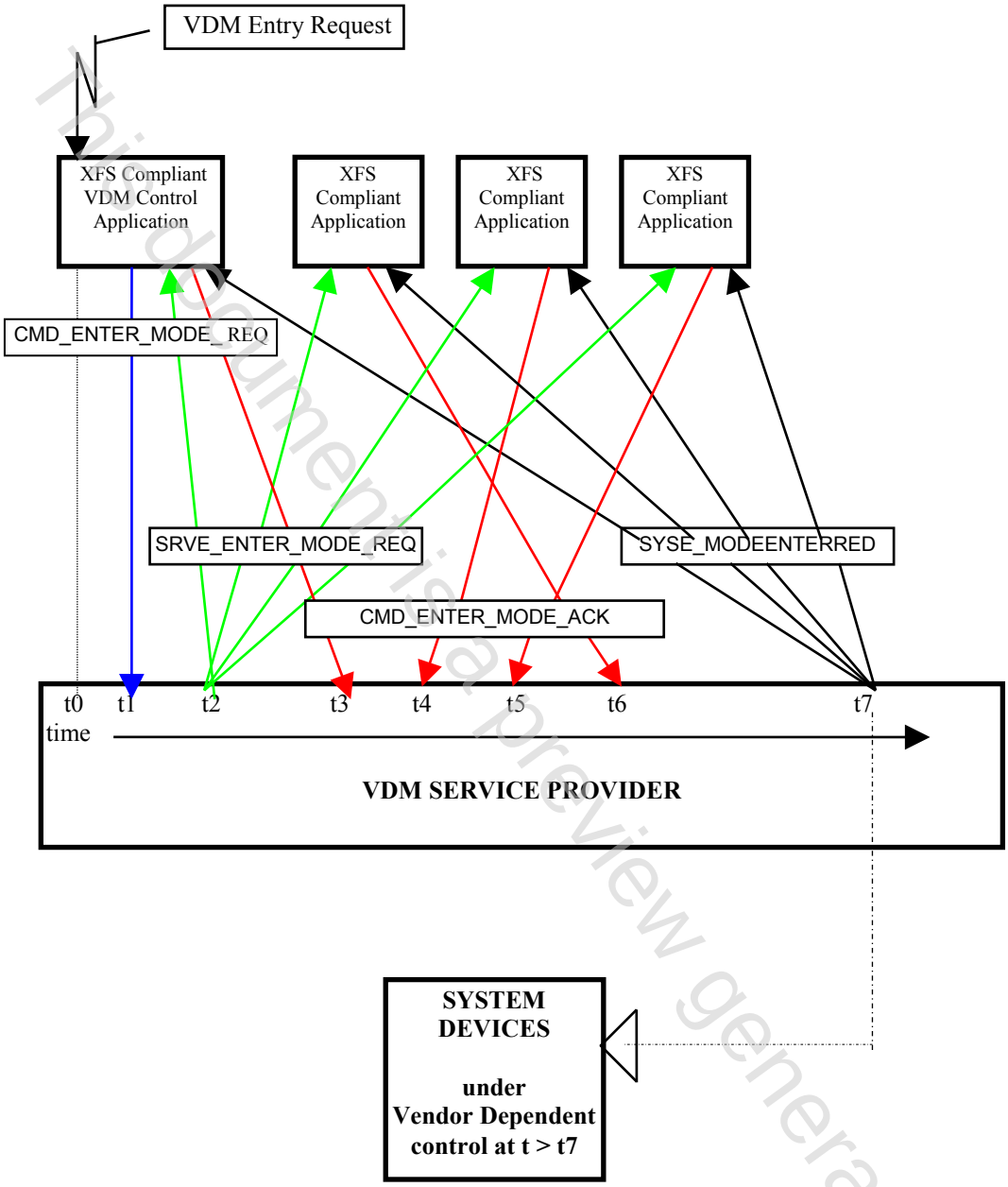
Similarly, once the exit request has been made all registered applications will be notified of the exit request by an event message. These applications must then issue an acknowledgement command to the VDM Service Provider immediately. Once all applications have acknowledged, the VDM Service Provider 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 request entry to 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 which require sessions to be closed 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.

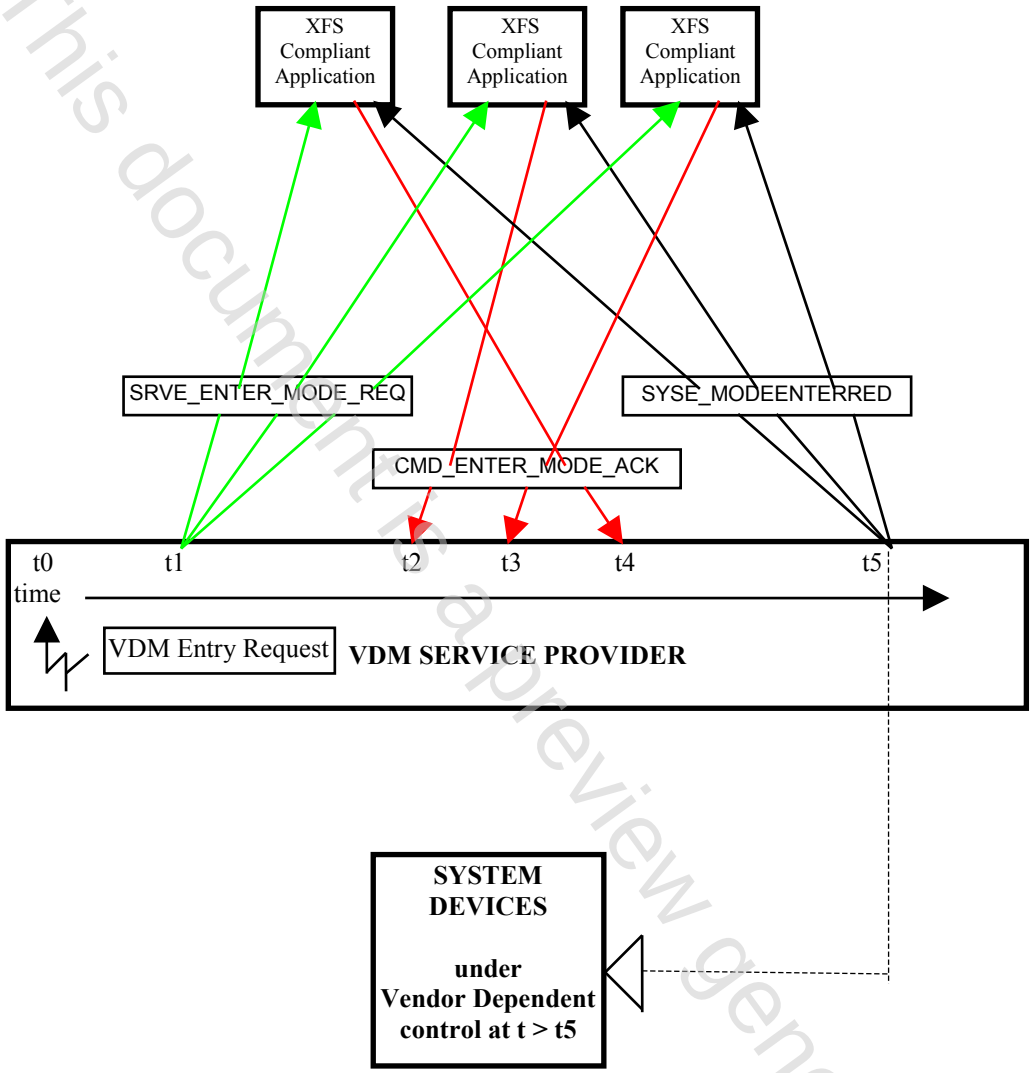
This is mandatory for self-service but optional for branch.

2.1 VDM Entry triggered by XFS Application



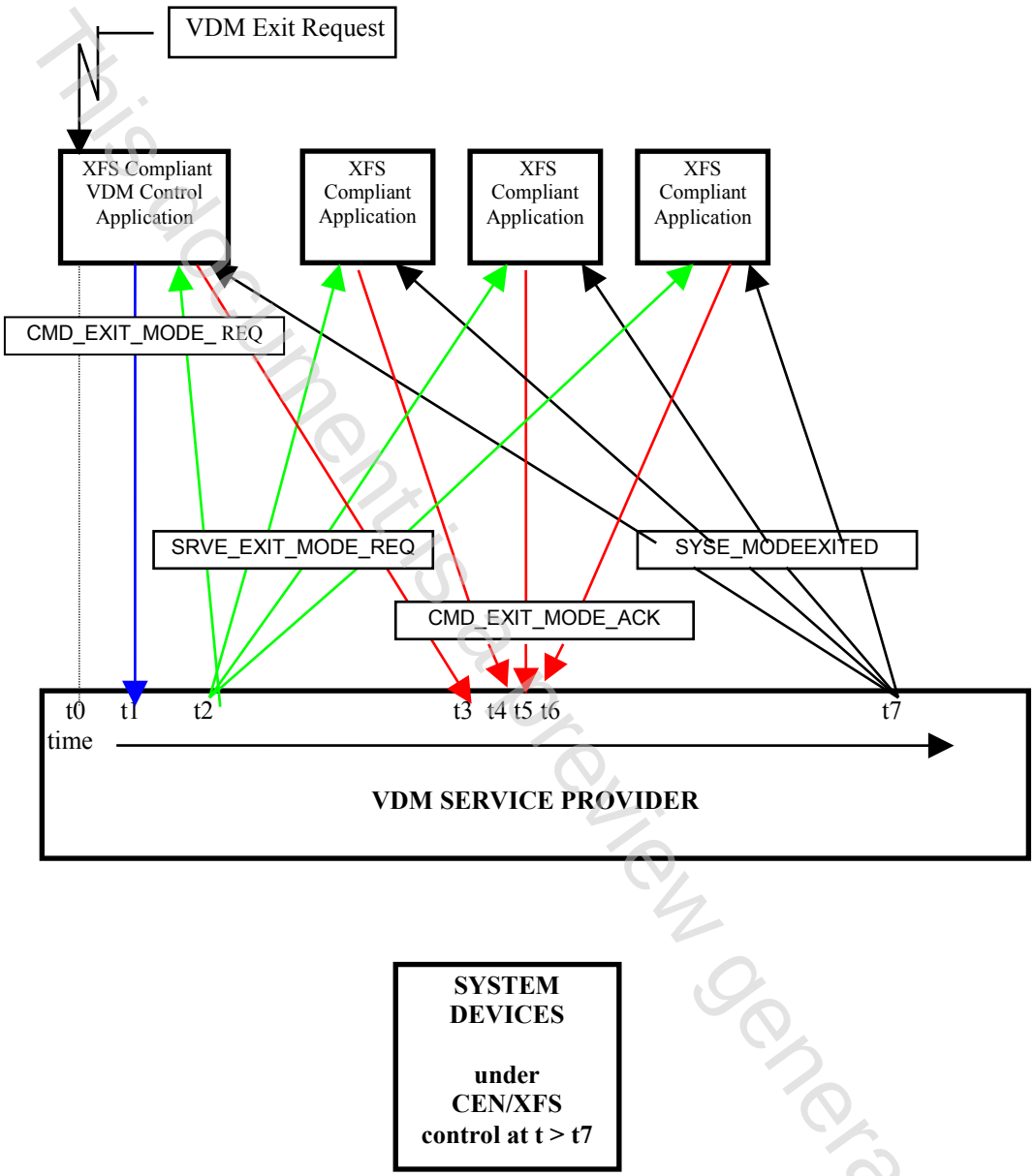
At time t0, status is “Inactive” and a request to Enter VDM arises from within the Application system.
At time t1, an Application Process/Thread/Function issues the CMD_ENTER_MODE_REQ Execute cmd. Status then becomes “Enter Pending”.
At time t2, the VDM Service Provider issues the SRVE_ENTER_MODE_REQ 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



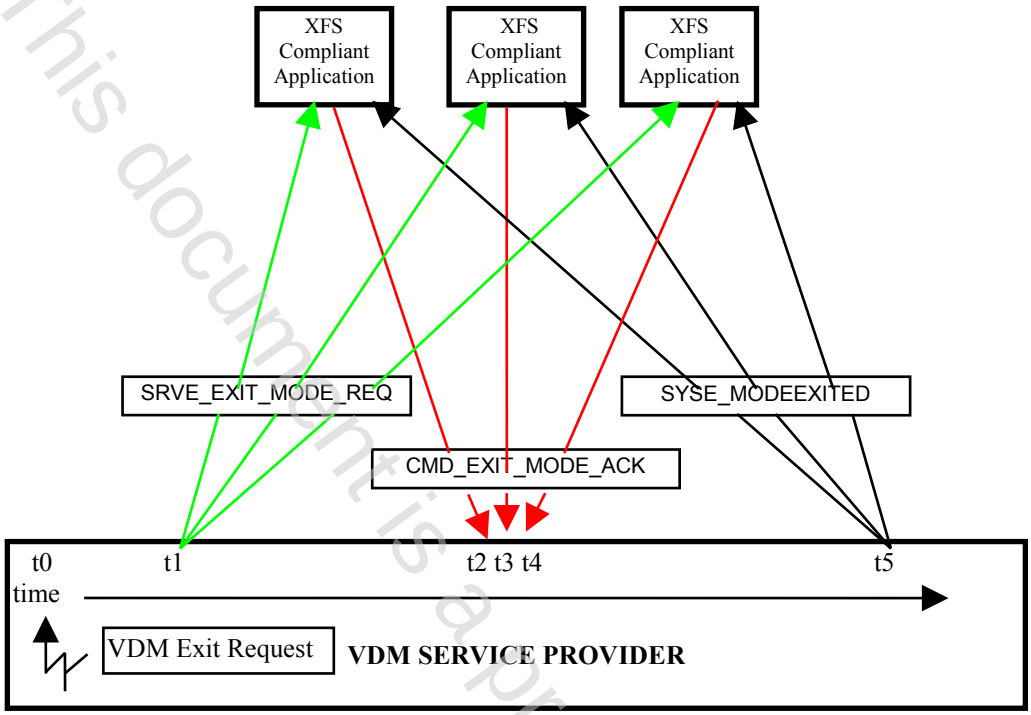
At time t0, status is “Inactive” and a request to Enter VDM arises from within the Vendor System.
Status then becomes “Enter Pending”.
At time t1, the VDM Service Provider issues the SRVE_ENTER_MODE_REQ Event to all registered applications.
At time t2, the VDM Service Provider receives a CMD_ENTER_MODE_ACK 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



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

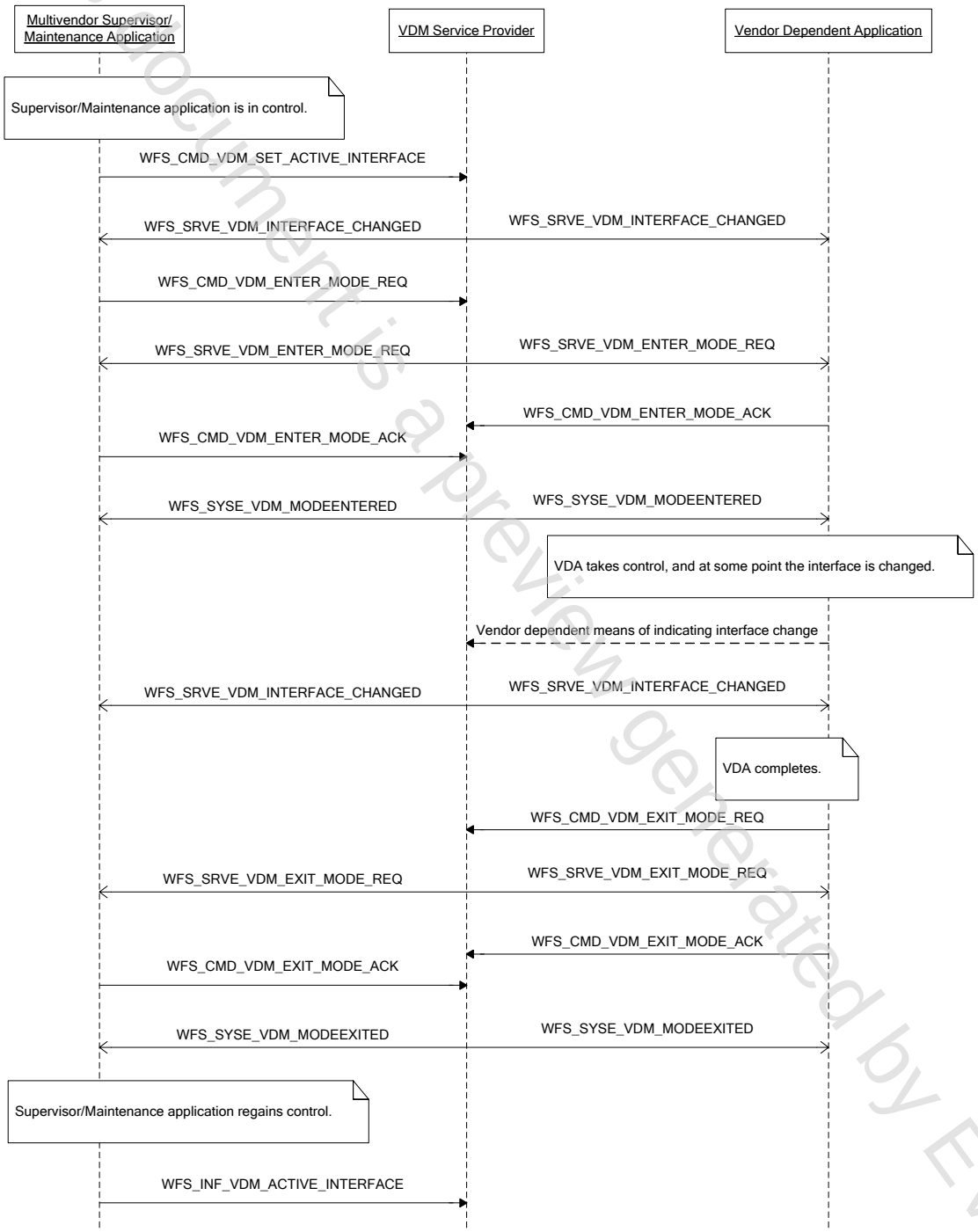
under CEN/XFS control at t > t5

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.

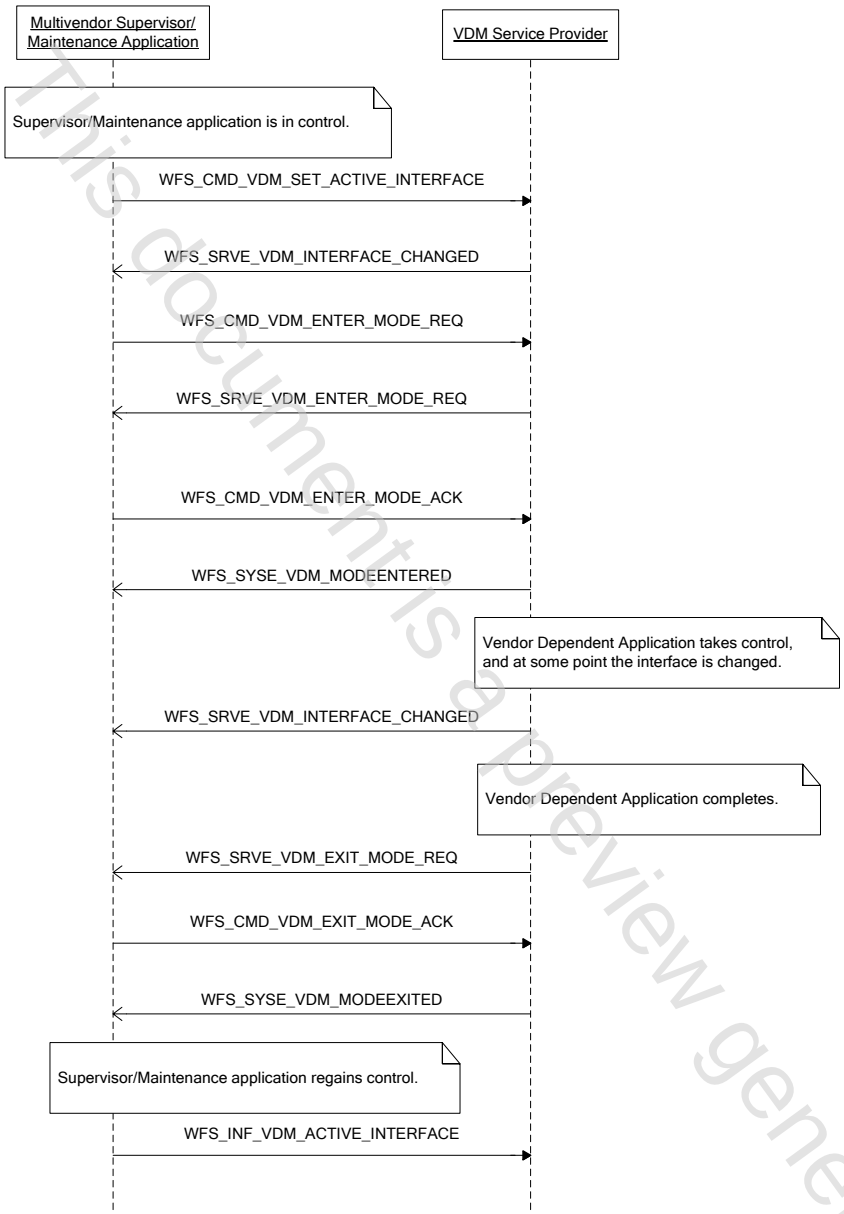
2.5 Controlling / Determining the Active Interface

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 3.40