CEN

WORKSHOP

AGREEMENT

CWA 16926-18

February 2020

ICS 35.200; 35.240.15; 35.240.40

English version

Extensions for Financial Services (XFS) interface specification Release 3.40 - Part 18: Item Processing Module 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 16926-18:2020 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 16926-18, but this does not guarantee, either explicitly or implicitly, its correctness. Users of CWA 16926-18 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 16926-18 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. Notable enhancements include:

- 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. Item Processing Module

This specification describes the XFS service class for Item Processing Modules (IPM). The specification of this service class includes definitions of the service-specific commands that can be issued, using the **WFSAsyncExecute**, **WFSExecute, WFSGetInfo** and **WFSAsyncGetInfo** functions.

This service class is currently defined only for self service devices.

In the U.S., checks are always encoded in magnetic ink for reading by Magnetic Ink Character Recognition (MICR), and a single font is always used. In Europe some countries use MICR and some use Optical Character Recognition (OCR) character sets, with different fonts, for their checks.

Item Processing Modules accept one or more media items (Checks, Giros, etc) and process these items according to application requirements. The IPM class supports devices that can handle a single item as well as those devices that can handle bunches of items. The following are the three principle device types:

- Single Item: can accept and process a single item at a time.
- Multi-Item Feed with no stacker (known as an escrow in some environments): can accept a bunch of media from the customer but each item has to be processed fully (i.e. deposited in a bin or returned) before the next item can be processed.
- Multi-Item Feed with a stacker: can accept a bunch of media from the customer and all items can be processed together.

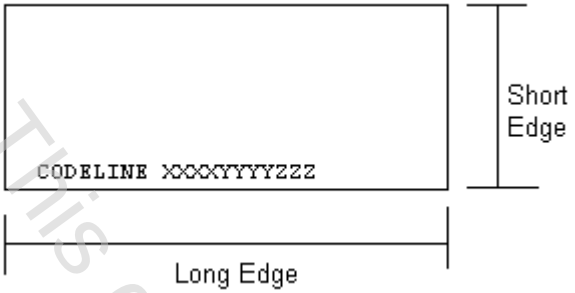
The IPM class provides applications with an interface to control the following functions (depending on the capabilities of the specific underlying device):

- Capture an image of the front of an item in multiple formats and bit depths.
- Capture an image of the back of an item in multiple formats and bit depths.
- Read the code line of an item using MICR reader.
- Read the code line of an item using OCR.
- Endorse (print text) on an item.
- Stamp an item.
- Return an item to the customer.
- Deposit an item in a bin.
- Retract items left by the customer.

The IPM device class uses the concept of a Media-In transaction to track and control a customer's interaction with the device. A Media-In transaction consists of one or more WFS_CMD_IPM_MEDIA_IN commands. The transaction is initiated by the first WFS_CMD_IPM_MEDIA_IN command and remains active until the transaction is either confirmed through WFS_CMD_IPM_MEDIA_IN_END, or terminated by WFS_CMD_IPM_MEDIA_IN_ROLLBACK, WFS_CMD_IPM_RETRACT_MEDIA or WFS_CMD_IPM_RESET. While a transaction is active the WFS_INF_IPM_TRANSACTION_STATUS command reports the status of the current transaction. When a transaction is not active the WFS_INF_IPM_TRANSACTION_STATUS command reports the status of the last transaction as well as some current status values.

There are primarily two types of devices supported by the IPM, those devices with a stacker and those without.

In this the specification the terms "long edge" and "short edge" are used to describe the orientation of a check and length of its edges. The diagram below illustrates these definitions.



2.1 Devices with a Stacker

On devices with a stacker, the IPM device class supports two mechanisms for deciding if physically acceptable items should be accepted onto the stacker or refused:

- The device/Service Provider automatically makes the accept/refuse decision.
- The application controls the accept/refuse decision.

2.1.1 Automatic Accept/Refuse

In summary, the following process is followed (the exact order will depend on application requirements):

1. The application initiates the transaction via the WFS_CMD_IPM_MEDIA_IN command. This command accepts a bunch of media items. The images and code line for every media item accepted is sent to the application before the command completes.
2. The application then asks the customer if they have any more items to process.
3. If the customer has more items to deposit then the WFS_CMD_IPM_MEDIA_IN command is called one or more times to add more items to the stacker.
4. Once the customer has inserted all their bunches of items and they have been added to the stacker the application can process each item and predefine what should happen to each media item during the WFS_CMD_IPM_MEDIA_IN_END command, e.g.:
    a. Define if the item should be stamped and what should be printed on the item (using WFS_CMD_IPM_PRINT_TEXT), set the destination bin (using WFS_CMD_IPM_SET_DESTINATION), and request the item is rescanned after printing (using WFS_CMD_IPM_GET_IMAGE_AFTER_PRINT), or
    b. Define that the item should be returned to the customer (using WFS_CMD_IPM_SET_DESTINATION).
5. When all items have been processed the application calls WFS_CMD_IPM_MEDIA_IN_END to complete the transaction and carry out the predefined actions, e.g. print and deposit some items while returning others.

Note: Before the WFS_CMD_IPM_MEDIA_IN_END command is called, the customer can cancel the transaction at any time and all items are returned to the customer by the application calling WFS_CMD_IPM_ROLLBACK.

2.1.2 Application Controlled Accept/Refuse

In summary, the following process is followed (the exact order will depend on application requirements):

1. The application uses the WFS_CMD_IPM_MEDIA_IN command to accept a bunch of media items (the first use of this command initiates the transaction). The application indicates that it wants to make the accept/refuse decision for each item via an input parameter, and as a result only one item is processed and the code line and images are only produced for a single item.
2. The application processes the item and decides if it should be accepted or refused using the WFS_CMD_IPM_ACCEPT_ITEM command.
3. The application calls WFS_CMD_IPM_GET_NEXT_ITEM to read the next item. If an item is read then the flow continues at step 2. When there are no items left to process the flow continues with the next step.
4. The application can return the refused items to the customer with WFS_CMD_IPM_PRESENT_MEDIA.
5. The application then asks the customer if they have any more items to process or wish to re-insert the refused items after correcting the issue causing the refusal.
6. If the customer has more items to deposit then flow continues at step 1, otherwise the flow continues at the next step.
7. Once the customer has inserted all their bunches of items and they have been added to the stacker the application can process each item and predefine what should happen to each media item during the WFS_CMD_IPM_MEDIA_IN_END command, e.g.:

a. Define if the item should be stamped and what should be printed on the item (using WFS_CMD_IPM_PRINT_TEXT), set the destination bin (using WFS_CMD_IPM_SET_DESTINATION), and request the item is rescanned after printing (using WFS_CMD_IPM_GET_IMAGE_AFTER_PRINT), or

b. Define that the item should be returned to the customer (using WFS_CMD_IPM_SET_DESTINATION).

8. When all items have been processed the application calls WFS_CMD_IPM_MEDIA_IN_END to complete the transaction and carry out the predefined actions, e.g. print and deposit some items while returning others.

Note: Before the WFS_CMD_IPM_MEDIA_IN_END command is called, the customer can cancel the transaction at any time and all items are returned to the customer by the application calling WFS_CMD_IPM_ROLLBACK.

2.2 Device without a Stacker

Devices without a stacker fall into two categories those with a multi-item feed unit and those without. Both of these types of devices can be handled by the same application flow, however they are both documented below for clarity.

2.2.1 Multi-Feed Devices without a Stacker

In summary, the following process is followed (the exact order will depend on application requirements):

1. The application uses the WFS_CMD_IPM_MEDIA_IN command to accept a bunch of media items (the first use of this command initiates the transaction). However as there is no stacker only one item is processed and the code line and images are only produced for a single item.
2. The application processes the item and decides what should be done to the item, e.g.:
    a. Define if the item should be stamped and what should be printed on the item (using WFS_CMD_IPM_PRINT_TEXT), set the destination bin (using WFS_CMD_IPM_SET_DESTINATION), and request the item is rescanned after printing (using WFS_CMD_IPM_GET_IMAGE_AFTER_PRINT), or
    b. Define that the item should be returned to the customer (using WFS_CMD_IPM_SET_DESTINATION).
3. The application calls WFS_CMD_IPM_ACTION_ITEM to have the predefined actions executed.
4. The application calls WFS_CMD_IPM_GET_NEXT_ITEM to read the next item. If an item is read then the flow continues at step 2. When there are not items left to process the flow continues with the next step.
5. The application then asks the customer if they have any more items to process.
6. If the customer has more items to deposit then flow continues at step 1.
7. When the customer is finished the application calls WFS_CMD_IPM_MEDIA_IN_END to terminate the transaction.

2.2.2 Single-Feed Devices

In summary, the following process is followed:

1. The application initiates the transaction via the WFS_CMD_IPM_MEDIA_IN command. This command accepts a single item and produces the image and code line.
2. The application processes the item and decides what should be done to the item, e.g.:
    a. Define if the item should be stamped and what should be printed on the item (using WFS_CMD_IPM_PRINT_TEXT), set the destination bin (using WFS_CMD_IPM_SET_DESTINATION), and request the item is rescanned after printing (using WFS_CMD_IPM_GET_IMAGE_AFTER_PRINT), or
    b. Define that the item should be returned to the customer (using WFS_CMD_IPM_SET_DESTINATION).
3. The application calls WFS_CMD_IPM_ACTION_ITEM to have the predefined actions executed.
4. The application optionally calls WFS_CMD_IPM_GET_NEXT_ITEM to have a single flow for devices with multi-feed and without. The flow continues with the next step.
5. The application then asks the customer if they have any more items to process.
6. If the customer has more items to deposit then flow continues at step 1.
7. When the customer is finished the application calls WFS_CMD_IPM_MEDIA_IN_END to terminate the transaction.

3. References

1. XFS Application Programming Interface (API)/Service Provider Interface (SPI), Programmer's Reference Revision 3.40
2. Extensions for Financial Services (XFS) interface specification, Release 3.40, Part 15: Cash-In Module, Device Class Interface, Programmer's Reference