CEN

WORKSHOP

AGREEMENT

CWA 15748-4

July 2008

ICS 35.240.50

English version

Extensions for Financial Services (XFS) interface specification - Release 3.10 - Part 4: Identification Card 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 15748-4:2008 D/E/F

Table of Contents

Foreword 4

1. Introduction 7

1.1 Background to Release 3.10 7

1.2 XFS Service-Specific Programming 7

2. Identification Card Readers and Writers 8

3. References 9

4. Info Commands 10

4.1 WFS_INF_IDC_STATUS 10

4.2 WFS_INF_IDC_CAPABILITIES 15

4.3 WFS_INF_IDC_FORM_LIST 19

4.4 WFS_INF_IDC_QUERY_FORM 20

4.5 WFS_INF_IDC_QUERY_IFM_IDENTIFIER 21

5. Execute Commands 22

5.1 WFS_CMD_IDC_READ_TRACK 22

5.2 WFS_CMD_IDC_WRITE_TRACK 24

5.3 WFS_CMD_IDC_EJECT_CARD 26

5.4 WFS_CMD_IDC_RETAIN_CARD 28

5.5 WFS_CMD_IDC_RESET_COUNT 29

5.6 WFS_CMD_IDC_SETKEY 30

5.7 WFS_CMD_IDC_READ_RAW_DATA 31

5.8 WFS_CMD_IDC_WRITE_RAW_DATA 35

5.9 WFS_CMD_IDC_CHIP_IO 37

5.10 WFS_CMD_IDC_RESET 39

5.11 WFS_CMD_IDC_CHIP_POWER 40

5.12 WFS_CMD_IDC_PARSE_DATA 41

5.13 WFS_CMD_IDC_SET_GUIDANCE_LIGHT 42

5.14 WFS_CMD_IDC_POWER_SAVE_CONTROL 43

6. Events 44

6.1 WFS_EXEE_IDC_INVALIDTRACKDATA 44

6.2 WFS_EXEE_IDC_MEDIAINSERTED 45

6.3 WFS_SRVE_IDC_MEDIAREMOVED 46

6.4 WFS_EXEE_IDC_MEDIARETAINED 47

6.5 WFS_EXEE_IDC_INVALIDMEDIA 48

6.6 WFS_SRVE_IDC_CARDACTION 49

6.7 WFS_USRE_IDC_RETAINBINTHRESHOLD 50

6.8 WFS_SRVE_IDC_MEDIADETECTED 51

6.9 WFS_SRVE_IDC_RETAINBINREMOVED 52
6.10 WFS_SRVE_IDC_RETAINBININSERTED 53
6.11 WFS_EXEE_IDC_INSERTCARD 54
6.12 WFS_SRVE_IDC_DEVICEPOSITION 55
6.13 WFS_SRVE_IDC_POWER_SAVE_CHANGE 56

7. Form Description 57

8. C-Header file 60

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:

1.0	May 24, 1993	Initial release of API and SPI specification
1.11	February 3, 1995	Separation of specification into separate documents for API/SPI and service class definitions
2.0	November 11, 1996	Updated release encompassing self-service environment Chip Card handling inserted.
3.0	October 18, 2000	Eliminate reference to Registry as a form of implementation for threshold value in WFS_USRE_IDC_RETAINBINTHRESHOLD command. Clarify that Form Definition attributes are not required in any mandatory order. Clarify WFS_IDC_DEVBUSY meaning. Add WFS_CMD_IDC_RESET command. High Coercivity enhancements For a detailed description see CWA 14050-18:2000 IDC migration from version 2.0 to version 3.0.
3.02	May 21, 2003	Updated to handle latching smart cards within a Smart/Dip device and devices with permanently connected chip cards. For a detailed description see CWA 14050-26:2003 IDC migration from version 3.0 to version 3.02.
3.10	November 29, 2007	For a description of changes see CWA 15748-63:2007 IDC Migration from Version 3.02 (see CWA 14050) 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. Identification Card Readers and Writers

This section describes the functions provided by a generic identification card reader/writer service (IDC). These descriptions include definitions of the service-specific commands that can be issued, using the **WFSAsyncExecute**, **WFSExecute**, **WFSGetInfo** and **WFSAsyncGetInfo** functions.

This service allows for the operation of the following categories of units:

- motor driven card reader/writer
- pull through card reader (writing facilities only partially included)
- dip reader
- contactless chip card readers
- permanent chip card readers (each chip is accessed through a unique logical service)

The following tracks/chips and the corresponding international standards are taken into account in this document:

- Track 1 ISO 7811
- Track 2 ISO 7811
- Track 3 ISO 7811 / ISO 4909
- Front Track 1 (JIS II) Japan
- Watermark Sweden
- Chip (contacted) ISO 7816
- Chip (contactless) ISO 10536.

National standards like Transac for France are not considered, but can be easily included via the forms mechanism (see Section 7, Form Definition).

In addition to the pure reading of the tracks mentioned above, security boxes can be used via this service to check the data of writable tracks for manipulation. These boxes (such as CIM or MM) are sensor-equipped devices that are able to check some other information on the card and compare it with the track data.

Persistent values are maintained through power failures, open sessions, close session and system resets.

When the service controls a permanently connected chip card, WFS_ERR_UNSUPP_COMMAND will be returned to all commands except WFS_INF_IDC_STATUS, WFS_INF_IDC_CAPABILITIES, WFS_CMD_IDC_CHIP_POWER, WFS_CMD_IDC_CHP_IO and WFS_CMD_IDC_RESET.

The following defines the roles and responsibilities of an application within EMV:

- EMV Level 2 interaction is handled above the XFS API
- EMV Level 1 interaction is handled below the XFS API

All EMV status information that is defined as a Level 1 responsibility in the EMV specification should be handled below the XFS API.

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_IDC_STATUS

Description This command reports the full range of information available, including the information that is provided either by the Service Provider or, if present, by any of the security modules. In addition to that, the number of cards retained is transmitted for motor driven card reader/writer (for devices of the other categories this number is always set to zero).

Input Param None.

Output Param LPWFSIDCSTATUS lpStatus;

```
typedef struct _wfs_idc_status
     {
     WORD                 fwDevice;
     WORD                 fwMedia;
     WORD                 fwRetainBin;
     WORD                 fwSecurity;
     USHORT               usCards;
     WORD                 fwChipPower;
     LPSTR                lpszExtra;
     DWORD                dwGuidLights[WFS_IDC_GUIDLIGHTS_SIZE];
     WORD                 fwChipModule;
     WORD                 fwMagReadModule;
     WORD                 fwMagWriteModule;
     WORD                 fwFrontImageModule;
     WORD                 fwBackImageModule;
     WORD                 wDevicePosition;
     USHORT               usPowerSaveRecoveryTime;
     } WFSIDCSTATUS, *LPWFSIDCSTATUS;
```

fwDevice
Specifies the state of the ID card device as one of the following flags:

Value	Meaning
WFS_IDC_DEVONLINE	The device is present, powered on and online (i.e. operational, not busy processing a request and not in an error state).
WFS_IDC_DEVOFFLINE	The device is offline (e.g. the operator has taken the device offline by turning a switch or pulling out the device).
WFS_IDC_DEVPOWEROFF	The device is powered off or physically not connected.
WFS_IDC_DEVNODEVICE	There is no device intended to be there; e.g. this type of self service machine does not contain such a device or it is internally not configured.
WFS_IDC_DEVHWERROR	The device is present but inoperable due to a hardware fault that prevents it from being used.
WFS_IDC_DEVUSERERROR	The device is present but a person is preventing proper device operation. The application should suspend the device operation or remove the device from service until the Service Provider generates a device state change event indicating the condition of the device has changed e.g. the error is removed (WFS_IDC_DEVONLINE) or a permanent error condition has occurred (WFS_IDC_DEVHWERROR).
WFS_IDC_DEVBUSY	The device is busy and unable to process an Execute command at this time.
WFS_IDC_DEVFRAUDATTEMPT	The device is present but has detected a fraud attempt.

fwMedia
Specifies the state of the ID card unit as one of the following values:

Value	Meaning
WFS_IDC_MEDIAPRESENT	Media is present in the device, not in the entering position and not jammed. On the latched dip device, this indicates that the card is present in the device and the card is unlatched.
WFS_IDC_MEDIANOTPRESENT	Media is not present in the device and not at the entering position.
WFS_IDC_MEDIAJAMMED	Media is jammed in the device; operator intervention is required.
WFS_IDC_MEDIANOTSUPP	Capability to report media position is not supported by the device (e.g. a typical swipe reader).
WFS_IDC_MEDIAUNKNOWN	The media state cannot be determined with the device in its current state (e.g. the value of *fwDevice* is WFS_IDC_DEVNODEVICE, WFS_IDC_DEVPOWEROFF, WFS_IDC_DEVOFFLINE, or WFS_IDC_DEVHWERROR).
WFS_IDC_MEDIAENTERING	Media is at the entry/exit slot of a motorized device.
WFS_IDC_MEDIALATCHED	Media is present & latched in a latched dip card unit. This means the card can be used for chip card dialog.

fwRetainBin
Specifies the state of the ID card unit retain bin as one of the following values:

Value	Meaning
WFS_IDC_RETAINBINOK	The retain bin of the ID card unit is not full.
WFS_IDC_RETAINNOTSUPP	The ID card unit does not support retain capability.
WFS_IDC_RETAINBINFULL	The retain bin of the ID card unit is full.
WFS_IDC_RETAINBINHIGH	The retain bin of the ID card unit is nearly full.
WFS_IDC_RETAINBINMISSING	The retain bin of the ID card unit is missing.

fwSecurity
Specifies the state of the security unit as one of the following values:

Value	Meaning
WFS_IDC_SECNOTSUPP	No security module is available.
WFS_IDC_SECNOTREADY	The security module is not ready to process cards or is inoperable.
WFS_IDC_SECOPEN	The security module is open and ready to process cards.

usCards
The number of cards retained; applicable only to motor driven ID card units for non-motorized card units this value is zero. This value is persistent it is reset to zero by the WFS_CMD_IDC_RESET_COUNT command.

fwChipPower
Specifies the state of the chip controlled by this service. Depending on the value of *fwType* within the WFS_INF_IDC_CAPABILITIES structure, this can either be the chip on the currently inserted user card or the chip on a permanently connected chip card. The state of the chip is one of the following flags:

Value	Meaning
WFS_IDC_CHIPONLINE	The chip is present, powered on and online (i.e. operational, not busy processing a request and not in an error state).
WFS_IDC_CHIPPOWEREDOFF	The chip is present, but powered off (i.e. not contacted).
WFS_IDC_CHIPBUSY	The chip is present, powered on, and busy (unable to process an Execute command at this time).
WFS_IDC_CHIPNODEVICE	A card is currently present in the device, but has no chip.
WFS_IDC_CHIPHWERROR	The chip is present, but inoperable due to a hardware error that prevents it from being used (e.g. MUTE, if there is an unresponsive card in the reader).
WFS_IDC_CHIPNOCARD	There is no card in the device.
WFS_IDC_CHIPNOTSUPP	Capability to report the state of the chip is not supported by the ID card unit device.
WFS_IDC_CHIPUNKNOWN	The state of the chip cannot be determined with the device in its current state.

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.

dwGuidLights [...]
Specifies the state of the guidance light indicators. A number of guidance light types are defined below. Vendor specific guidance lights are defined starting from the end of the array. The maximum guidance light index is WFS_IDC_GUIDLIGHTS_MAX.

Specifies the state of the guidance light indicator as WFS_IDC_GUIDANCE_NOT_AVAILABLE, WFS_IDC_GUIDANCE_OFF or a combination of the following flags consisting of one type B, and optionally one type C.

Value	Meaning	Type
WFS_IDC_GUIDANCE_NOT_AVAILABLE	The status is not available.	A
WFS_IDC_GUIDANCE_OFF	The light is turned off.	A
WFS_IDC_GUIDANCE_SLOW_FLASH	The light is blinking slowly.	B
WFS_IDC_GUIDANCE_MEDIUM_FLASH	The light is blinking medium frequency.	B
WFS_IDC_GUIDANCE_QUICK_FLASH	The light is blinking quickly.	B
WFS_IDC_GUIDANCE_CONTINUOUS	The light is turned on continuous (steady).	B
WFS_IDC_GUIDANCE_RED	The light is red.	C
WFS_IDC_GUIDANCE_GREEN	The light is green.	C
WFS_IDC_GUIDANCE_YELLOW	The light is yellow.	C
WFS_IDC_GUIDANCE_BLUE	The light is blue.	C
WFS_IDC_GUIDANCE_CYAN	The light is cyan.	C
WFS_IDC_GUIDANCE_MAGENTA	The light is magenta.	C
WFS_IDC_GUIDANCE_WHITE	The light is white.	C

dwGuidLights [WFS_IDC_GUIDANCE_CARDUNIT]
Specifies the state of the guidance light indicator on the card unit.

fwChipModule
Specifies the state of the chip card module reader as one of the following values:

Value	Meaning
WFS_IDC_CHIPMODOK	The chip card module is in a good state.
WFS_IDC_CHIPMODINOP	The chip card module is inoperable.
WFS_IDC_CHIPMODUNKNOWN	The state of the chip card module cannot be determined.

WFS_IDC_CHIPMODNOTSUPP	Reporting the chip card module status is not supported.

fwMagReadModule
Specifies the state of the magnetic card reader as one of the following values:

Value	Meaning
WFS_IDC_MAGMODOK	The magnetic card reading module is in a good state.
WFS_IDC_MAGMODINOP	The magnetic card reading module is inoperable.
WFS_IDC_MAGMODUNKNOWN	The state of the magnetic reading module cannot be determined.
WFS_IDC_MAGMODNOTSUPP	Reporting the magnetic reading module status is not supported.

fwMagWriteModule
Specifies the state of the magnetic card writer as one of the following values:

Value	Meaning
WFS_IDC_MAGMODOK	The magnetic card writing module is in a good state.
WFS_IDC_MAGMODINOP	The magnetic card writing module is inoperable.
WFS_IDC_MAGMODUNKNOWN	The state of the magnetic card writing module cannot be determined.
WFS_IDC_MAGMODNOTSUPP	Reporting the magnetic writing module status is not supported.

fwFrontImageModule
Specifies the state of the front image reader as one of the following values:

Value	Meaning
WFS_IDC_IMGMODOK	The front image reading module is in a good state.
WFS_IDC_IMGMODINOP	The front image reading module is inoperable.
WFS_IDC_IMGMODUNKNOWN	The state of the front image reading module cannot be determined.
WFS_IDC_IMGMODNOTSUPP	Reporting the front image reading module status is not supported.

fwBackImageModule
Specifies the state of the back image reader as one of the following values:

Value	Meaning
WFS_IDC_IMGMODOK	The back image reading module is in a good state.
WFS_IDC_IMGMODINOP	The back image reading module is inoperable.
WFS_IDC_IMGMODUNKNOWN	The state of the back image reading module cannot be determined.
WFS_IDC_IMGMODNOTSUPP	Reporting the back image reading module status is not supported.

wDevicePosition
Specifies the device position. The device position value is independent of the *fwDevice* value, e.g. when the device position is reported as WFS_IDC_DEVICENOTINPOSITION, *fwDevice* can have any of the values defined above (including WFS_IDC_DEVONLINE or WFS_IDC_DEVOFFLINE). If the device is not in its normal operating position (i.e. WFS_IDC_DEVICEINPOSITION) then media may not be presented through the normal customer interface. This value is one of the following values:

Value	Meaning
WFS_IDC_DEVICEINPOSITION	The device is in its normal operating position, or is fixed in place and cannot be moved.
WFS_IDC_DEVICENOTINPOSITION	The device has been removed from its normal operating position.
WFS_IDC_DEVICEPOSUNKNOWN	Due to a hardware error or other condition, the position of the device cannot be determined.
WFS_IDC_DEVICEPOSNOTSUPP	The physical device does not have the capability of detecting the position.

usPowerSaveRecoveryTime
Specifies the actual number of seconds required by the device to resume its normal operational state from the current power saving mode. This value is zero if either the power saving mode has not been activated or no power save control is supported.

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.

The *fwDevice* field can indicate that the device is still available (i.e. WFS_IDC_DEVONLINE) even if one of the detailed device status fields (*fwSecurity*, *fwChipModule*, *fwMagReadModule* or *fwMagWriteModule*) indicates that there is a problem with one or more modules. In this case, only the functionality provided by modules that do not have a fault should be used.

In the case where communications with the device has been lost, the *fwDevice* field will report WFS_IDC_DEVPOWEROFF when the device has been removed or WFS_IDC_DEVHWERROR if the communications are unexpectedly lost. All other fields should contain a value based on the following rules and priority:

1. Report the value as unknown.
2. Report the value as a general h/w error.
3. Report the value as the last known value.

4.2 WFS_INF_IDC_CAPABILITIES

Description This command is used to retrieve the capabilities of the ID card unit.

Input Param None.

Output Param LPWFSIDCCAPS lpCaps;

```
typedef struct _wfs_idc_caps
     {
     WORD                 wClass;
     WORD                 fwType;
     BOOL                 bCompound;
     WORD                 fwReadTracks;
     WORD                 fwWriteTracks;
     WORD                 fwChipProtocols;
     USHORT               usCards;
     WORD                 fwSecType;
     WORD                 fwPowerOnOption;
     WORD                 fwPowerOffOption;
     BOOL                 bFluxSensorProgrammable;
     BOOL                 bReadWriteAccessFollowingEject;
     WORD                 fwWriteMode;
     WORD                 fwChipPower;
     LPSTR                lpszExtra;
     WORD                 fwDIPMode;
     LPWORD               lpwMemoryChipProtocols;
     DWORD                dwGuidLights[WFS_IDC_GUIDLIGHTS_SIZE];
     WORD                 fwEjectPosition;
     BOOL                 bPowerSaveControl;
     } WFSIDCCAPS, *LPWFSIDCCAPS;
```

wClass
Specifies the logical service class as WFS_SERVICE_CLASS_IDC.

fwType
Specifies the type of the ID card unit as one of the following values:

Value	Meaning
WFS_IDC_TYPEMOTOR	The ID card unit is a motor driven card unit.
WFS_IDC_TYPESWIPE	The ID card unit is a swipe (pull-through) card unit.
WFS_IDC_TYPEDIP	The ID card unit is a dip card unit. This dip type is not capable of latching cards entered.
WFS_IDC_TYPECONTACTLESS	The ID card unit is a contactless card unit, i.e. no insertion of the card is required.
WFS_IDC_TYPELATCHEDDIP	The ID card unit is a latched dip card unit. This device type is used when a dip IDC device supports chip communication. The latch ensures the consumer cannot remove the card during chip communication. Any card entered will automatically latch when a request to initiate a chip dialog is made (via the WFS_CMD_IDC_READ_RAW_DATA command). The WFS_CMD_IDC_EJECT_CARD command is used to unlatch the card.
WFS_IDC_TYPEPERMANENT	The ID card unit is dedicated to a permanently housed chip card (no user interaction is available with this type of card).

bCompound
Specifies whether the logical device is part of a compound physical device.

fwReadTracks
Specifies the tracks that can be read by the ID card unit as a combination of the following flags:

Value	Meaning
WFS_IDC_NOTSUPP	The ID card unit can not access any track.
WFS_IDC_TRACK1	The ID card unit can access track 1.
WFS_IDC_TRACK2	The ID card unit can access track 2.
WFS_IDC_TRACK3	The ID card unit can access track 3.
WFS_IDC_TRACK_WM	The ID card unit can access the Swedish Watermark track.
WFS_IDC_FRONT_TRACK_1	The ID card unit can access the front track 1. In some countries this track is known as JIS II track.
WFS_IDC_FRONTIMAGE	The ID card unit can read the front image of a card.
WFS_IDC_BACKIMAGE	The ID card unit can read the back image of a card.

fwWriteTracks
Specifies the tracks that can be written by the ID card unit (as a combination of the flags specified in the description of *fwReadTracks* except WFS_IDC_TRACK_WM).

fwChipProtocols
Specifies the chip card protocols that are supported by the Service Provider as a combination of the following flags:

Value	Meaning
WFS_IDC_NOTSUPP	The ID card unit can not handle chip cards.
WFS_IDC_CHIPT0	The ID card unit can handle the T=0 protocol.
WFS_IDC_CHIPT1	The ID card unit can handle the T=1 protocol.
WFS_IDC_CHIP_PROTOCOL_NOT_REQUIRED	The ID card unit is capable of communicating with a chip card without requiring the application to specify any protocol.

usCards
Specifies the maximum numbers of cards that the retain bin can hold (zero if not available).

fwSecType
Specifies the type of security module used as one of the following values:

Value	Meaning
WFS_IDC_SECNOTSUPP	Device has no security module.
WFS_IDC_SECMMBOX	Security module of device is MMBox.
WFS_IDC_SECCIM86	Security module of device is CIM86.

fwPowerOnOption
Specifies the power-on capabilities of the device hardware as one of the following values (applicable only to motor driven ID card units):

Value	Meaning
WFS_IDC_NOACTION	No power on actions are supported by the device.
WFS_IDC_EJECT	The card will be ejected on power-on (or off, see *fwPowerOffOption* below).
WFS_IDC_RETAIN	The card will be retained on power-on (off).
WFS_IDC_EJECTTHENRETAIN	The card will be ejected for a specified time on power-on (off), then retained if not taken. The time for which the card is ejected is vendor dependent.
WFS_IDC_READPOSITION	The card will be moved into the read position on power-on (off).

fwPowerOffOption
Specifies the power-off capabilities of the device hardware, as one of the flags specified for *fwPowerOnOption*; applicable only to motor driven ID card units.

bFluxSensorProgrammable
Specifies whether the Flux Sensor on the card unit is programmable, this can either be TRUE or FALSE.

bReadWriteAccessFollowingEject
Specifies whether a card may be read or written after having been pushed to the exit slot with an eject command. The card will be retracted back into the IDC.

fwWriteMode
A combination of the following flags specify the write capabilities, with respect to whether the device can write low coercivity (loco) and/or high coercivity (hico) magnetic stripes:

Value	Meaning
WFS_IDC_NOTSUPP	Does not support writing of magnetic stripes.
WFS_IDC_LOCO	Supports writing of loco magnetic stripes.
WFS_IDC_HICO	Supports writing of hico magnetic stripes.
WFS_IDC_AUTO	Service Provider is capable of automatically determining whether loco or hico magnetic stripes should be written.

fwChipPower
Specifies the capabilities of the ID card unit (in relation to the user or permanent chip controlled by the service), for chip power management as a combination of the following flags*:*

Value	Meaning
WFS_IDC_NOTSUPP	The ID card unit can not handle chip power management.
WFS_IDC_CHIPPOWERCOLD	The ID card unit can power on the chip and reset it (Cold Reset).
WFS_IDC_CHIPPOWERWARM	The ID card unit can reset the chip (Warm Reset).
WFS_IDC_CHIPPOWEROFF	The ID card unit can power off the chip.

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.

fwDIPMode
Specifies whether data track data is read on entry or exit from the dip card unit as one of the following flags:

Value	Meaning
WFS_IDC_NOTSUPP	The ID card unit is not a dip type.
WFS_IDC_DIP_EXIT	The dip ID card unit reads card track data on exit only.
WFS_IDC_DIP_ENTRY	The dip ID card unit reads card track data on entry only.
WFS_IDC_DIP_ENTRY_EXIT	The dip ID card unit reads card track data both on entry and exit.
WFS_IDC_DIP_UNKNOWN	Unknown whether track data is read on entry or exit.

lpwMemoryChipProtocols
Pointer to a zero terminated array that specifies the memory card protocols that are supported by the Service Provider as an array of constants. If this parameter is NULL then the Service Provider does not support any memory card protocols. Valid Memory Card Identifiers are:

Value	Meaning
WFS_IDC_MEM_SIEMENS4442	The device supports the Siemens 4442 Card Protocol (also supported by the Gemplus GPM2K card).
WFS_IDC_MEM_GPM896	The device supports the Gemplus GPM 896 Card Protocol.

dwGuidLights [...]
Specifies which guidance lights are available. A number of guidance light types are defined below. Vendor specific guidance lights are defined starting from the end of the array. The maximum guidance light index is WFS_IDC_GUIDLIGHTS_MAX.

The elements of this array are specified as a combination of the following flags and indicate all of the possible flash rates (type B) and colors (type C) that the guidance light indicator is capable of handling. If the guidance light indicator only supports one color then no value of type C is returned. A value of WFS_IDC_GUIDANCE_NOT_AVAILABLE indicates that the device has no guidance light indicator or the device controls the light directly with no application control possible.

Value	Meaning	Type
WFS_IDC_GUIDANCE_NOT_AVAILABLE	There is no guidance light control available at this position.	A
WFS_IDC_GUIDANCE_OFF	The light can be off.	B
WFS_IDC_GUIDANCE_SLOW_FLASH	The light can blink slowly.	B
WFS_IDC_GUIDANCE_MEDIUM_FLASH	The light can blink medium frequency.	B
WFS_IDC_GUIDANCE_QUICK_FLASH	The light can blink quickly.	B
WFS_IDC_GUIDANCE_CONTINUOUS	The light can be continuous (steady).	B
WFS_IDC_GUIDANCE_RED	The light can be red.	C
WFS_IDC_GUIDANCE_GREEN	The light can be green.	C
WFS_IDC_GUIDANCE_YELLOW	The light can be yellow.	C
WFS_IDC_GUIDANCE_BLUE	The light can be blue.	C
WFS_IDC_GUIDANCE_CYAN	The light can be cyan.	C
WFS_IDC_GUIDANCE_MAGENTA	The light can be magenta.	C
WFS_IDC_GUIDANCE_WHITE	The light can be white.	C

dwGuidLights [WFS_IDC_GUIDANCE_CARDUNIT]
Specifies whether the guidance light indicator on the card unit is available.

fwEjectPosition
Specifies the target position that is supported for the eject operation, as a combination of the following flags:

Value	Meaning
WFS_IDC_EXITPOSITION	The device can eject a card to the exit position, from which the user can remove it.
WFS_IDC_TRANSPORTPOSITION	The device can eject a card to the transport just behind the exit position, from which the user can not remove it. The device which supports this flag must also support the WFS_IDC_EXITPOSITION flag.

bPowerSaveControl
Specifies whether power saving control is available. This can either be TRUE if available or FALSE if not available.

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_IDC_FORM_LIST

Description This command is used to retrieve the list of forms available on the device.

Input Param None.

Output Param LPSTR lpszFormList;

lpszFormList
Pointer to a list of null-terminated form names, with the final name terminating with two null characters.

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

Comments None.

4.4 WFS_INF_IDC_QUERY_FORM

Description This command is used to retrieve details of the definition of a specified form.

Input Param LPSTR lpszFormName;

lpszFormName
Points to the null-terminated form name on which to retrieve details.

Output Param LPWFSIDCFORM lpForm;

```
typedef struct _wfs_idc_form
     {
     LPSTR               lpszFormName;
     char                cFieldSeparatorTrack1;
     char                cFieldSeparatorTrack2;
     char                cFieldSeparatorTrack3;
     WORD                fwAction;
     LPSTR               lpszTracks;
     BOOL                bSecure;
     LPSTR               lpszTrack1Fields;
     LPSTR               lpszTrack2Fields;
     LPSTR               lpszTrack3Fields;
     } WFSIDCFORM, *LPWFSIDCFORM;
```

lpszFormName
Specifies the null-terminated name of the form.

cFieldSeparatorTrack1
Specifies the value of the field separator of Track 1.

cFieldSeparatorTrack2
Specifies the value of the field separator of Track 2.

cFieldSeparatorTrack3
Specifies the value of the field separator of Track 3.

fwAction
Specifies the form action; can be one of the following flags:

Value	Meaning
WFS_IDC_ACTIONREAD	The form reads the card.
WFS_IDC_ACTIONWRITE	The form writes the card.

lpszTracks
Specifies the read algorithm or the track to write.

bSecure
Specifies whether or not to do a security check.

lpszTrack1Fields
Pointer to a list of null-terminated field names of Track 1, with the final name terminating with two null characters.

lpszTrack2Fields
Pointer to a list of null-terminated field names of Track 2, with the final name terminating with two null characters.

lpszTrack3Fields
Pointer to a list of null-terminated field names of Track 3, with the final name terminating with two null characters.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_FORMNOTFOUND	The specified form cannot be found.
WFS_ERR_IDC_FORMINVALID	The specified form is invalid.

Comments None.

4.5 WFS_INF_IDC_QUERY_IFM_IDENTIFIER

Description This command is used to retrieve the complete list of registration authority Interface Module (IFM) identifiers. The primary registration authority is EMVCo but other organizations are also supported for historical or local country requirements.

New registration authorities may be added in the future so applications should be able to handle the return of new (as yet undefined) IFM identifiers.

Input Param None**.**

Output Param LPWFSIDCIFMIDENTIFIER *lppIFMIdentifier;

Pointer to a NULL terminated array of pointers to data structures. There is one array element for each IFM identifier supported by the Service Provider (in no particular order). If there is no IFM identifier available for one of the defined IFM authorities then no element is returned in the array for that authority. If there are no IFM identifiers for the device then the output parameter *lppIFMIdentifier* will be NULL.

```
typedef struct _wfs_idc_ifm_identifier
     {
     WORD                wIFMAuthority;
     LPSTR               lpszIFMIdentifier;
     } WFSIDCIFMIDENTIFIER, *LPWFSIDCIFMIDENTIFIER;
```

wIFMAuthority
Specifies the IFM authority that issued the IFM identifier:

Value	Meaning
WFS_IDC_IFMEMV	The Level 1 Type Approval IFM identifier assigned by EMVCo.
WFS_IDC_IFMEUROPAY	The Level 1 Type Approval IFM identifier assigned by Europay.
WFS_IDC_IFMVISA	The Level 1 Type Approval IFM identifier assigned by VISA.
WFS_IDC_IFMGIECB	The IFM identifier assigned by GIE Cartes Bancaires.

lpszIFMIdentifier
Returns an ASCII string containing the IFM Identifier of the chip card reader (or IFM) as assigned by the specified authority.

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

Comments If this command is not supported then this does not necessarily mean that the device is not certified by one or more certification authorities.

5. Execute Commands

5.1 WFS_CMD_IDC_READ_TRACK

Description For motor driven card readers, the ID card unit checks whether a card has been inserted. If so, the tracks are read immediately as described in the form specified by the *lpstrFormsName* parameter.

If no card has been inserted, and for all other categories of card readers, the ID card unit waits for the period of time specified in the **WFSExecute** call for a card to be either inserted or pulled through. Again the next step is reading the tracks specified in the form (see Section 7, Form Definition, for a more detailed description of the forms mechanism). When the SECURE tag is specified in the associated form, the results of a security check via a security module (i.e. MM, CIM86) are specified and added to the track data.

The WFS_EXEE_IDC_INSERTCARD event will be generated when there is no card in the card reader and the device is ready to accept a card.

If the security check fails however this should not stop valid data being returned. The error WFS_ERR_IDC_SECURITYFAIL will be returned if the form specifies only security data to be read and the security check could not be executed, in all other cases WFS_SUCCESS will be returned with the security field of the output parameter set to the relevant value including WFS_IDC_SEC_HWERROR.

For non-motorized Card Readers which read track data on card exit, the WFS_ERR_INVALID_DATA error code is returned when a call to WFS_CMD_IDC_READ_RAW_DATA is made to read both track data and chip data.

Input Param LPSTR lpstrFormName;

lpstrFormName
Points to the name of the form that defines the behavior for the reading of tracks (see Section 7, Form Definition).

Output Param LPSTR lpstrTrackData;

lpstrTrackData
Points to the data read successfully from the selected tracks (and value of security module if available).

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required.
WFS_ERR_IDC_SHUTTERFAIL	The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.
WFS_ERR_IDC_INVALIDDATA	The read operation specified by the forms definition could not be completed successfully due to invalid track data. This is returned if all tracks in an ‘or’ (|) operation cannot be read or if any track in an ‘and’ (&) operation cannot be read. lpstrTrackData points to data from the successfully read tracks (if any). One WFS_EXEE_IDC_INVALIDTRACKDAT A execute event is generated for each specified track which could not be read successfully. See the form description for the rules defining how tracks are specified.

WFS_ERR_IDC_NOMEDIA	The card was removed before completion of the read action (the event WFS_EXEE_IDC_MEDIAINSERTED has been generated). For motor driven devices, the read is disabled; i.e. another command has to be issued to enable the reader for card entry.
WFS_ERR_IDC_INVALIDMEDIA	No track found; card may have been inserted or pulled through the wrong way.
WFS_ERR_IDC_FORMNOTFOUND	The specified form can not be found.
WFS_ERR_IDC_FORMINVALID	The specified form definition is invalid (e.g. syntax error).
WFS_ERR_IDC_SECURITYFAIL	The security module failed reading the cards security sign.
WFS_ERR_IDC_CARDTOOSHORT	The card that was inserted is too short. When this error occurs the card remains at the exit slot.
WFS_ERR_IDC_CARDTOOLONG	The card that was inserted is too long. When this error occurs the card remains at the exit slot.

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

Value	Meaning
WFS_EXEE_IDC_INVALIDTRACKDATA	One event is generated for each blank track (no data) or invalid track (either data error reading the track or the data does not conform to the specified form definition).
WFS_EXEE_IDC_MEDIAINSERTED	This event is generated when a card is detected in the device, giving early warning of card entry to an application, allowing it to remove a user prompt and/or do other processing while the card is being read.
WFS_SRVE_IDC_MEDIAREMOVED	This event is generated when a card is removed before completion of a read operation.
WFS_EXEE_IDC_INVALIDMEDIA	The user is attempting to insert the media in the wrong orientation. The card has not been accepted into the device. The device is still ready to accept a card inserted in the correct orientation.
WFS_EXEE_IDC_INSERTCARD	Device is ready to accept a card from the user.

Comments The track data is preceded by the keyword for the track, separated by a ':'. The field data is always preceded by the corresponding keyword, separated by a '='. The fields are separated by 0x00. The data of the different tracks is separated by an additional 0x00. The end of the buffer is marked by another additional 0x00 (see example below). Data encoding is defined in Section 7, Form Definition.

Example of *lpstrTrackData*:
TRACK2:ALL=47..\0\0TRACK3:MII=59\0PAN=500..\0\0\0

5.2 WFS_CMD_IDC_WRITE_TRACK

Description For motor-driven card readers, the ID card unit checks whether a card has been inserted. If so, the data is written to the track as described in the form specified by the *lpstrFormName* parameter, and the other parameters.

If no card has been inserted, and for all other categories of devices, the ID card unit waits for the period of time specified in the **WFSExecute** call for a card to be either inserted or pulled through. The next step is writing the data defined by the form and the parameters to the respective track (see Section 7, Form Definition, for a more detailed description of the forms mechanism).

This procedure is followed by data verification.

The WFS_EXEE_IDC_INSERTCARD event will be generated when there is no card in the card reader and the device is ready to accept a card.

If power fails during a write the outcome of the operation will be vendor specific, there is no guarantee that the write will have succeeded.

Input Param LPWFSIDCWRITETRACK lpWriteTrack;

```
typedef struct _wfs_idc_write_track
     {
     LPSTR               lpstrFormName;
     LPSTR               lpstrTrackData;
     WORD                fwWriteMethod;
     } WFSIDCWRITETRACK, *LPWFSIDCWRITETRACK;
```

lpstrFormName
Points to the name of the form to be used.

lpstrTrackData
Points to the data to be used in the form.

fwWriteMethod
Indicates whether a low coercivity or high coercivity magnetic stripe is being written.

Value	Meaning
WFS_IDC_LOCO	Low coercivity magnetic stripe is being written.
WFS_IDC_HICO	High coercivity magnetic stripe is being written.
WFS_IDC_AUTO	Service Provider will determine whether low or high coercivity stripe is to be written.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required.
WFS_ERR_IDC_SHUTTERFAIL	The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.
WFS_ERR_IDC_NOMEDIA	The card was removed before completion of the write action (the event WFS_EXEE_IDC_MEDIAINSERTED has been generated). For motor driven devices, the write is disabled; i.e. another command has to be issued to enable the reader for card entry.
WFS_ERR_IDC_INVALIDDATA	An error occurred while writing the track.
WFS_ERR_IDC_DATASYNTAX	The syntax of the data pointed to by *lpstrTrackData* is in error, or does not conform to the form definition.

WFS_ERR_IDC_INVALIDMEDIA	No track found; card may have been inserted or pulled through the wrong way.
WFS_ERR_IDC_FORMNOTFOUND	The specified form can not be found.
WFS_ERR_IDC_FORMINVALID	The specified form definition is invalid (e.g. syntax error).
WFS_ERR_IDC_WRITE_METHOD	The *fwWriteMethod* value is inconsistent with device capabilities.
WFS_ERR_IDC_CARDTOOSHORT	The card that was inserted is too short. When this error occurs the card remains at the exit slot.
WFS_ERR_IDC_CARDTOOLONG	The card that was inserted is too long. When this error occurs the card remains at the exit slot.

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

Value	Meaning
WFS_EXEE_IDC_INVALIDTRACKDATA	One event is generated for each blank track (no data) or invalid track (either data error reading the track or the data does not conform to the specified form definition).
WFS_EXEE_IDC_MEDIAINSERTED	This event is generated when a card is detected in the device, giving early warning of card entry to an application, allowing it to remove a user prompt and/or do other processing while the card is being written.
WFS_SRVE_IDC_MEDIAREMOVED	This event is generated when a card is removed before completion of a write operation.
WFS_EXEE_IDC_INVALIDMEDIA	The user is attempting to insert the media in the wrong orientation. The card has not been accepted into the device. The device is still ready to accept a card inserted in the correct orientation.
WFS_EXEE_IDC_INSERTCARD	Device is ready to accept a card from the user.

Comments The field data is always preceded by the corresponding keyword, separated by an '='. This keyword could be one of the fields defined in the form or the predefined keyword 'ALL'. Fields are separated by 0x00. The end of the buffer is marked with an additional 0x00. (See the example below and Section 7, Form Definition.). This specification means that only one track can be written in the same command. This is a fundamental capability of an ID card unit; thus if a write request is received by a device with no write capability, the WFS_ERR_UNSUPP_COMMAND error is returned.

Example of *lpstrTrackData*:
RETRYCOUNT=3\0DATE=3132\0\0

5.3 WFS_CMD_IDC_EJECT_CARD

Description This command is only applicable to motor driven card readers and latched dip card readers. For motorized card readers the default operation is that the card is driven to the exit slot from where the user can remove it. After successful completion of this command, a service event message is generated to inform the application when the card is taken. The card remains in position for withdrawal until either it is taken or another command is issued that moves the card.

For latched dip readers, this command causes the card to be unlatched (if not already unlatched), enabling removal.

After successful completion of this command, a WFS_SRVE_IDC_MEDIAREMOVED event is generated to inform the application when the card is taken.

Input Param LPWFSIDCEJECTCARD lpEjectCard;

```
typedef struct _wfs_idc_eject_card
     {
     WORD                  wEjectPosition;
     } WFSIDCEJECTCARD, *LPWFSIDCEJECTCARD;
```

wEjectPosition
Specifies the destination of the card ejection for motorized card readers. Possible values are one of the following:

Value	Meaning
WFS_IDC_EXITPOSITION	The card will be transferred to the exit slot from where the user can remove it. In the case of a latched dip the card will be unlatched, enabling removal.
WFS_IDC_TRANSPORTPOSITION	The card will be transferred to the transport just behind the exit slot. If a card is already at this position then WFS_SUCCESS will be returned. Another WFS_CMD_IDC_EJECT_CARD command is required with the *wEjectPosition* set to WFS_IDC_EXITPOSITION in order to present the card to the user for removal.

If *lpEjectCard* is a NULL pointer, the card will be transferred to the exit slot from where the user can remove it. In the case of a latched dip the card will be unlatched, enabling removal. This action is the same as when WFS_IDC_EXITPOSITION is specified for *wEjectPosition*.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required. A possible scenario is also when an attempt to retain the card was made during attempts to eject it. The retain bin is full; no more cards can be retained. The current card is still in the device.
WFS_ERR_IDC_SHUTTERFAIL	The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.
WFS_ERR_IDC_NOMEDIA	No card is present.
WFS_ERR_IDC_MEDIARETAINED	The card has been retained during attempts to eject it. The device is clear and can be used.

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

Value	Meaning
WFS_SRVE_IDC_MEDIAREMOVED	The card has been taken by the user.
WFS_USRE_IDC_RETAINBINTHRESHOLD	The retain bin reached a threshold value.

Comments This is a fundamental capability of an ID card unit; thus if an eject request is received by a device with no eject capability, the WFS_ERR_UNSUPP_COMMAND error is returned.

5.4 WFS_CMD_IDC_RETAIN_CARD

Description The card is removed from its present position (card inserted into device, card entering, unknown position) and stored in the retain bin; applicable to motor-driven card readers only. The ID card unit sends an event, if the storage capacity of the retain bin is reached. If the storage capacity has already been reached, and the command cannot be executed, an error is returned and the card remains in its present position.

Input Param None.

Output Param LPWFSIDCRETAINCARD lpRetainCard;

```
typedef struct _wfs_idc_retain_card
     {
     USHORT              usCount;
     WORD                fwPosition;
     } WFSIDCRETAINCARD, *LPWFSIDCRETAINCARD;
```

usCount
Total number of ID cards retained up to and including this operation, since the last WFS_CMD_IDC_RESET_COUNT command was executed.

fwPosition
Position of card; only relevant if card could not be retained. Possible positions:

Value	Meaning
WFS_IDC_MEDIAUNKNOWN	The position of the card can not be determined with the device in its current state.
WFS_IDC_MEDIAPRESENT	The card is present in the reader.
WFS_IDC_MEDIAENTERING	The card is in the entering position (shutter).

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required.
WFS_ERR_IDC_NOMEDIA	No card has been inserted. The *fwPosition* parameter has the value WFS_IDC_MEDIAUNKNOWN.
WFS_ERR_IDC_RETAINBINFULL	The retain bin is full; no more cards can be retained. The current card is still in the device.
WFS_ERR_IDC_SHUTTERFAIL	The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.

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

Value	Meaning
WFS_USRE_IDC_RETAINBINTHRESHOLD	The retain bin reached a threshold value.
WFS_SRVE_IDC_MEDIAREMOVED	The card has been taken by the user.
WFS_EXEE_IDC_MEDIARETAINED	The card has been retained.

Comments This is a fundamental capability of an ID card unit; thus if a retain request is received by a device with no retain capability, the WFS_ERR_UNSUPP_COMMAND error is returned.

5.5 WFS_CMD_IDC_RESET_COUNT

Description This function resets the present value for number of cards retained to zero. The function is possible for motor-driven card readers only.

The number of cards retained is controlled by the service and can be requested before resetting via the WFS_INF_IDC_STATUS.

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_USRE_IDC_RETAINBINTHRESHOLD	The retain bin was emptied.

Comments This is a fundamental capability of an ID card unit; thus if this request is received by a device with no retain capability, the WFS_ERR_UNSUPP_COMMAND error is returned.

5.6 WFS_CMD_IDC_SETKEY

Description This command is used for setting the DES key that is necessary for operating a CIM86 module. The command must be executed before the first read command is issued to the card reader.

Input Param LPWFSIDCSETKEY lpSetkey;

```
typedef struct _wfs_idc_setkey
     {
     USHORT              usKeyLen;
     LPBYTE              lpbKeyValue;
     } WFSIDCSETKEY, *LPWFSIDCSETKEY;
```

usKeyLen
Specifies the length of the following key value.

lpbKeyValue
Pointer to a byte array containing the CIM86 DES key. This key is supplied by the vendor of the CIM86 module.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_INVALIDKEY	The key does not fit to the security module.

Events Only the generic events defined in [Ref. 1] can be generated by this command.

Comments None.

5.7 WFS_CMD_IDC_READ_RAW_DATA

Description For motor driven card readers, the ID card unit checks whether a card has been inserted. If so, all specified tracks are read immediately. If reading the chip is requested, the chip will be contacted and reset and the ATR (Answer To Reset) data will be read. When this command completes the chip will be in contacted position. This command can also be used for an explicit cold reset of a previously contacted chip.

This command should only be used for user cards and should not be used for permanently connected chips.

If no card has been inserted, and for all other categories of card readers, the ID card unit waits for the period of time specified in the **WFSExecute** call for a card to be either inserted or pulled through. The next step is trying to read all tracks specified.

The WFS_EXEE_IDC_INSERTCARD event will be generated when there is no card in the card reader and the device is ready to accept a card.

Magnetic stripe track data is converted from its 5 or 7 bit character form to 8 bit ASCII form. The parity bit from each 5 or 7 bit magnetic stripe character is discarded. Start and end sentinel characters are not returned to the application. Field separator characters are returned to the application, and are also converted to 8 bit ASCII form.

In addition to that, a security check via a security module (i.e. MM, CIM86) can be requested. If the security check fails however this should not stop valid data being returned. The error WFS_ERR_IDC_SECURITYFAIL will be returned if the command specifies only security data to be read and the security check could not be executed, in all other cases WFS_SUCCESS will be returned with the *lpbData* field of the output parameter set to the relevant value including WFS_IDC_SEC_HWERROR.

For non-motorized Card Readers which read track data on card exit, the WFS_ERR_INVALID_DATA error code is returned when a call to WFS_CMD_IDC_READ_RAW_DATA is made to read both track data and chip data.

If the card unit is a latched dip unit then the device will latch the card when the chip card will be read, i.e. WFS_IDC_CHIP is specified (see below). The card will remain latched until a call to WFS_CMD_IDC_EJECT_CARD is made.

Input Param LPWORD lpwReadData;

lpwReadData
If *lpwReadData* points to a zero value any previously ejected card will be moved back inside the device and no data will be returned. Otherwise, *lpwReadData* specifies the data that should be read as a combination of the following flags:

Value	Meaning
WFS_IDC_TRACK1	Track 1 of the magnetic stripe will be read.
WFS_IDC_TRACK2	Track 2 of the magnetic stripe will be read.
WFS_IDC_TRACK3	Track 3 of the magnetic stripe will be read.
WFS_IDC_CHIP	The chip will be read.
WFS_IDC_SECURITY	A security check will be performed.
WFS_IDC_FLUXINACTIVE	If the IDC Flux Sensor is programmable it will be disabled in order to allow chip data to be read on cards which have no magnetic stripes.
WFS_IDC_TRACK_WM	The Swedish Watermark track will be read.
WFS_IDC_MEMORY_CHIP	The memory chip will be read.
WFS_IDC_FRONT_TRACK_1	Track 1 data is read from the magnetic stripe located on the front of the card. In some countries this track is known as JIS II track.
WFS_IDC_FRONTIMAGE	The front image of the card will be read in BMP format.
WFS_IDC_BACKIMAGE	The back image of the card will be read in BMP format.

Output Param LPWFSIDCCARDDATA *lppCardData;

lppCardData
Pointer to a NULL terminated array of pointers to card data structures or if no data has been requested *lppCardData* will be NULL:

```
typedef struct _wfs_idc_card_data
     {
     WORD                wDataSource;
     WORD                wStatus;
     ULONG               ulDataLength;
     LPBYTE              lpbData;
     WORD                fwWriteMethod;
     } WFSIDCCARDDATA, *LPWFSIDCCARDDATA;
```

wDataSource
Specifies the source of the card data as one of the following flags:

Value	Meaning
WFS_IDC_TRACK1	*lpbData* contains data read from track 1.
WFS_IDC_TRACK2	*lpbData* contains data read from track 2.
WFS_IDC_TRACK3	*lpbData* contains data read from track 3.
WFS_IDC_CHIP	*lpbData* contains ATR data read from the chip.
WFS_IDC_SECURITY	*lpbData* contains the value returned by the security module.
WFS_IDC_TRACK_WM	*lpbData* contains data read from the Swedish Watermark track.
WFS_IDC_MEMORY_CHIP	*lpbData* contains Memory Card Identification data read from the memory chip.
WFS_IDC_FRONT_TRACK_1	*lpbData* contains data read from the front track 1. In some countries this track is known as JIS II track.
WFS_IDC_FRONTIMAGE	*lpbData* contains a null-terminated string containing the full path and file name of the BMP image file for the front of the card.
WFS_IDC_BACKIMAGE	*lpbData* contains a null-terminated string containing the full path and file name of the BMP image file for the back of the card.

wStatus
Status of reading the card data. Possible values are:

Value	Meaning
WFS_IDC_DATAOK	The data is OK.
WFS_IDC_DATAMISSING	The track/chip/memory chip is blank.
WFS_IDC_DATAINVALID	The data contained on the track/chip/memory chip is invalid. This will typically be returned when *lpbData* reports WFS_IDC_SEC_BADREADLEVEL or WFS_IDC_SEC_DATAINVAL.
WFS_IDC_DATATOOLONG	The data contained on the track/chip/memory chip is too long.
WFS_IDC_DATATOOSHORT	The data contained on the track/chip/memory chip is too short.
WFS_IDC_DATASRCNOTSUPP	The data source to read from is not supported by the Service Provider.

WFS_IDC_DATASRCMISSING	The data source to read from is missing on the card, or is unable to be read due to a hardware problem, or the module has not been initialized. For example, this will be returned on a request to read a Memory Card and the customer has entered a magnetic card without associated memory chip. This will also be reported when *lpbData* reports WFS_IDC_SEC_NODATA, WFS_IDC_SEC_NOINIT or WFS_IDC_SEC_HWERROR. This will also be reported when the image reader could not create a BMP file due to the state of the image reader or due to a failure.

ulDataLength
Specifies the length of the following field *lpbData*.

lpbData
Points to the data read from the track/chip, the value returned by the security module or a null-terminated string containing the full path and file name of the BMP image file.

The security module can return one of the following values:

Value	Meaning
WFS_IDC_SEC_READLEVEL1	The security data readability level is 1.
WFS_IDC_SEC_READLEVEL2	The security data readability level is 2.
WFS_IDC_SEC_READLEVEL3	The security data readability level is 3.
WFS_IDC_SEC_READLEVEL4	The security data readability level is 4.
WFS_IDC_SEC_READLEVEL5	The security data readability level is 5.
WFS_IDC_SEC_BADREADLEVEL	The security data reading quality is not acceptable.
WFS_IDC_SEC_NODATA	There are no security data on the card.
WFS_IDC_SEC_DATAINVAL	The validation of the security data with the specific data on the magnetic stripe was not successful.
WFS_IDC_SEC_HWERROR	The security module could not be used, because of a hardware error.
WFS_IDC_SEC_NOINIT	The security module could not be used, because it was not initialized (e.g. CIM key is not loaded).

The memory card returns the memory card protocol used to communicate with the card in the first WORD of the buffer, with the actual data following the protocol WORD. See *lpwMemoryChipProtocols* from WFS_INF_IDC_CAPABILITIES for a description of possible memory card protocols.

fwWriteMethod
Ignored for this command.

Error Codes

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

Value	Meaning
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required.
WFS_ERR_IDC_SHUTTERFAIL	The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.
WFS_ERR_IDC_NOMEDIA	The card was removed before completion of the read action (the event WFS_EXEE_IDC_MEDIAINSERTED has been generated). For motor driven devices, the read is disabled; i.e. another command has to be issued to enable the reader for card entry.

WFS_ERR_IDC_INVALIDMEDIA	No track or chip found; card may have been inserted or pulled through the wrong way.
WFS_ERR_IDC_CARDTOOSHORT	The card that was inserted is too short. When this error occurs the card remains at the exit slot.
WFS_ERR_IDC_CARDTOOLONG	The card that was inserted is too long. When this error occurs the card remains at the exit slot.
WFS_ERR_IDC_SECURITYFAIL	The security module failed reading the cards security sign.

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

Value	Meaning
WFS_EXEE_IDC_MEDIAINSERTED	This event is generated when a card is detected in the device, giving early warning of card entry to an application, allowing it to remove a user prompt and/or do other processing while the card is being read.
WFS_SRVE_IDC_MEDIAREMOVED	This event is generated when a card is removed before completion of a read operation.
WFS_EXEE_IDC_INVALIDMEDIA	The user is attempting to insert the media in the wrong orientation. The card has not been accepted into the device. The device is still ready to accept a card inserted in the correct orientation.
WFS_EXEE_IDC_INSERTCARD	Device is ready to accept a card from the user.

Comments None.

5.8 WFS_CMD_IDC_WRITE_RAW_DATA

Description For motor-driven card readers, the ID card unit checks whether a card has been inserted. If so, the data is written to the tracks.

If no card has been inserted, and for all other categories of devices, the ID card unit waits for the period of time specified in the **WFSExecute** call for a card to be either inserted or pulled through. The next step is writing the data to the respective tracks.

The WFS_EXEE_IDC_INSERTCARD event will be generated when there is no card in the card reader and the device is ready to accept a card.

The application must pass the magnetic stripe data in ASCII without any sentinels. The data will be converted by the Service Provider (ref WFS_CMD_IDC_READ_RAW_DATA). If the data passed in is too long the WFS_ERR_INVALID_DATA error code will be returned.

This procedure is followed by data verification.

If power fails during a write the outcome of the operation will be vendor specific, there is no guarantee that the write will have succeeded.

Input Param LPWFSIDCCARDDATA *lppCardData;

Pointer to a NULL terminated array of pointers to card data structures:

```
typedef struct _wfs_idc_card_data
     {
     WORD                 wDataSource;
     WORD                 wStatus;
     ULONG                ulDataLength;
     LPBYTE               lpbData;
     WORD                 fwWriteMethod;
     } WFSIDCCARDDATA, *LPWFSIDCCARDDATA;
```

wDataSource
Specifies the source of the card data as one of the following flags:

Value	Meaning
WFS_IDC_TRACK1	*lpbData* contains the data to be written to track 1.
WFS_IDC_TRACK2	*lpbData* contains the data to be written to track 2.
WFS_IDC_TRACK3	*lpbData* contains the data to be written to track 3.
WFS_IDC_FRONT_TRACK_1	*lpbData* contains the data to be written to the front track 1. In some countries this track is known as JIS II track.

wStatus
This parameter is ignored by this command.

ulDataLength
Specifies the length of the following field *lpbData*.

lpbData
Points to the data to be written to the track.

fwWriteMethod
Indicates whether a loco or hico magnetic stripe is being written.

Value	Meaning
WFS_IDC_LOCO	Low coercivity magnetic stripe is being written.
WFS_IDC_HICO	High coercivity magnetic stripe is being written.
WFS_IDC_AUTO	Service Provider will determine whether low or high coercivity stripe is to be written.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required.
WFS_ERR_IDC_SHUTTERFAIL	The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.
WFS_ERR_IDC_NOMEDIA	The card was removed before completion of the write action (the event WFS_EXEE_IDC_MEDIAINSERTED has been generated). For motor driven devices, the write is disabled; i.e. another command has to be issued to enable the reader for card entry.
WFS_ERR_IDC_INVALIDMEDIA	No track found; card may have been inserted or pulled through the wrong way.
WFS_ERR_IDC_WRITE_METHOD	The *fwWriteMethod* value is inconsistent with device capabilities.
WFS_ERR_IDC_CARDTOOSHORT	The card that was inserted is too short. When this error occurs the card remains at the exit slot.
WFS_ERR_IDC_CARDTOOLONG	The card that was inserted is too long. When this error occurs the card remains at the exit slot.

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

Value	Meaning
WFS_EXEE_IDC_MEDIAINSERTED	This event is generated when a card is detected in the device, giving early warning of card entry to an application, allowing it to remove a user prompt and/or do other processing while the card is being written.
WFS_SRVE_IDC_MEDIAREMOVED	This event is generated when a card is removed before completion of a write operation.
WFS_EXEE_IDC_INVALIDMEDIA	The user is attempting to insert the media in the wrong orientation. The card has not been accepted into the device. The device is still ready to accept a card inserted in the correct orientation.
WFS_EXEE_IDC_INSERTCARD	Device is ready to accept a card from the user.

Comments This is a fundamental capability of an ID card unit; thus if a write request is received by a device with no write capability, the WFS_ERR_UNSUPP_COMMAND error is returned.

5.9 WFS_CMD_IDC_CHIP_IO

Description This command is used to communicate with the chip. Transparent data is sent from the application to the chip and the response of the chip is returned transparently to the application.

The ATR of the chip must be obtained before issuing this command. The ATR for a user card or the Memory Card Identification (when available) must initially be obtained through WFS_CMD_IDC_READ_RAW_DATA. The ATR for subsequent resets of a user card can be obtained either through WFS_CMD_IDC_READ_RAW_DATA command or through WFS_CMD_IDC_CHIP_POWER. The ATR for permanent connected chips is always obtained through WFS_CMD_IDC_CHIP_POWER.

Input Param LPWFSIDCCHIPIO lpChipIoIn;

```
typedef struct _wfs_idc_chip_io
     {
     WORD                wChipProtocol;
     ULONG               ulChipDataLength;
     LPBYTE              lpbChipData;
     } WFSIDCCHIPIO, *LPWFSIDCCHIPIO;
```

wChipProtocol
Identifies the protocol that is used to communicate with the chip. Possible values are those described in WFS_INF_IDC_CAPABILITIES. This field is ignored in communications with Memory Cards. The Service Provider knows which memory card type is currently inserted and therefore there is no need for the application to manage this.

ulChipDataLength
Specifies the length of the following field *lpbChipData*.

lpbChipData
Points to the data sent to the chip.

Output Param LPWFSIDCCHIPIO lpChipIoOut;

```
typedef struct _wfs_idc_chip_io
     {
     WORD                wChipProtocol;
     ULONG               ulChipDataLength;
     LPBYTE              lpbChipData;
     } WFSIDCCHIPIO, *LPWFSIDCCHIPIO;
```

wChipProtocol
Identifies the protocol that is used to communicate with the chip. This field contains the same value as the corresponding field in the input structure. This field should be ignored in Memory Card dialogs and will contain WFS_IDC_NOTSUPP when returned for any Memory Card dialog.

ulChipDataLength
Specifies the length of the following field *lpbChipData*.

lpbChipData
Points to the data responded from the chip.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required.
WFS_ERR_IDC_NOMEDIA	There is no card inside the device.
WFS_ERR_IDC_INVALIDMEDIA	No chip found; card may have been inserted the wrong way.
WFS_ERR_IDC_INVALIDDATA	An error occurred while communicating with the chip.
WFS_ERR_IDC_PROTOCOLNOTSUPP	The protocol used was not supported by the Service Provider.
WFS_ERR_IDC_ATRNOTOBTAINED	The ATR has not been obtained.

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

command:

Value	Meaning
WFS_SRVE_IDC_MEDIAREMOVED	This event is generated when a card is removed before completion of an operation.

Comments None.

5.10 WFS_CMD_IDC_RESET

Description This command is used by the application to perform a hardware reset which will attempt to return the IDC device to a known good state. This command does not over-ride a lock obtained by another application or service handle.

If the device is a user ID card unit, the device will attempt to either retain, eject or will perform no action on any user cards found in the IDC as specified in the *lpwResetIn* parameter. It may not always be possible to retain or eject the items as specified because of hardware problems. If a user card is found inside the device the WFS_SRVE_IDC_MEDIADETECTED event will inform the application where card was actually moved to. If no action is specified the user card will not be moved even if this means that the IDC cannot be recovered.

If the device is a permanent chip card unit, this command will power-off the chip.

Input Param LPWORD lpwResetIn;

Specifies the action to be performed on any user card found within the ID card unit as one of the following values:

Value	Meaning
WFS_IDC_EJECT	Eject any card found.
WFS_IDC_RETAIN	Retain any card found.
WFS_IDC_NOACTION	No action should be performed on any card found.

If *lpwResetIn* is NULL the Service Provider will determine where to move any card found.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required.
WFS_ERR_IDC_SHUTTERFAIL	The device is unable to open and close its shutter.
WFS_ERR_IDC_RETAINBINFULL	The retain bin is full; no more cards can be retained. The current card is still in the device.

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

Value	Meaning
WFS_SRVE_IDC_MEDIADETECTED	This event is generated when a media is detected during a reset.
WFS_SRVE_IDC_MEDIAREMOVED	The card has been taken by the user.
WFS_USRE_IDC_RETAINBINTHRESHOLD	The retain bin reached a threshold value.

Comments None.

5.11 WFS_CMD_IDC_CHIP_POWER

Description This command handles the power actions that can be done on the chip.

For user chips, this command is only used after the chip has been contacted for the first time using the WFS_CMD_IDC_READ_RAW_DATA command.

For permanently connected chip cards, this command is the only way to control the chip power.

Input Param LPWORD lpwChipPower;

lpwChipPower
Specifies the action to perform as one of the following flags:

Value	Meaning
WFS_IDC_CHIPPOWERCOLD	The chip is powered on and reset (Cold Reset).
WFS_IDC_CHIPPOWERWARM	The chip is reset (Warm Reset).
WFS_IDC_CHIPPOWEROFF	The chip is powered off.

Output Param NULL or LPWFSIDCCHIPPOWEROUT lpChipPowerOut;

```
typedef struct _wfs_idc_chip_power_out
     {
     ULONG               ulChipDataLength;
     LPBYTE              lpbChipData;
     } WFSIDCCHIPPOWEROUT, *LPWFSIDCCHIPPOWEROUT;
```

ulChipDataLength
Specifies the length of the following field *lpbChipData*.

lpbChipData
Points to the ATR data responded from the chip. NULL if the action was not a power on.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_CHIPPOWERNOTSUPP	The specified action is not supported by the hardware device.
WFS_ERR_IDC_MEDIAJAM	The card is jammed. Operator intervention is required.
WFS_ERR_IDC_NOMEDIA	There is no card inside the device.
WFS_ERR_IDC_INVALIDMEDIA	No chip found; card may have been inserted or pulled through the wrong way.
WFS_ERR_IDC_INVALIDDATA	An error occurred while communicating with the chip.
WFS_ERR_IDC_ATRNOTOBTAINED	The ATR has not been obtained (only applies to user chips).

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

Value	Meaning
WFS_SRVE_IDC_MEDIAREMOVED	This event is generated when a card is removed before completion of the operation.

Comments The NULL return value for the output parameter is provided for backwards compatibility and is only valid for user cards. Permanent chips must return the ATR in the output parameter. User cards should return the ATR in the output parameter.

5.12 WFS_CMD_IDC_PARSE_DATA

Description This command takes form name and the output of a successful WFS_CMD_IDC_READ_RAW_DATA command and returns the parsed string.

Input Param LPWFSIDCPARSEDATA lpParseData;

```
typedef struct _wfs_idc_parse_data
     {
     LPSTR                lpstrFormName;
     LPWFSIDCCARDDATA     *lppCardData;
     } WFSIDCPARSEDATA, *LPWFSIDCPARSEDATA;
```

lpstrFormName
Points to the name of the form that defines the behavior for the reading of tracks (see Section 7, Form Description).

lppCardData
Points to a NULL terminated array of pointers to card data structures, as returned from the WFS_CMD_IDC_READ_RAW_DATA command.

Output Param LPSTR lpstrTrackData;

lpstrTrackData
Points to the data read successfully from the selected tracks (and value of security module if available).

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_INVALIDDATA	The read operation specified by the forms definition could not be completed successfully due to invalid or incomplete track data being passed in. This is returned if none of the tracks in an 'or' (|) operation is contained in the *lppCardData* array or if any track in an 'and' (&) operation is not found in the input. One execute event (WFS_EXEE_IDC_INVALIDTRACKDATA) is generated for each specified track which could not be parsed successfully. See the form description for the rules defining how tracks are specified.
WFS_ERR_IDC_FORMNOTFOUND	The specified form can not be found.
WFS_ERR_IDC_FORMINVALID	The specified form definition is invalid (e.g. syntax error).

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

Value	Meaning
WFS_EXEE_IDC_INVALIDTRACKDATA	One event is generated for each blank track (no data) or invalid track (either data error reading the track or the data does not conform to the specified form definition).

Comments The track data is preceded by the keyword for the track, separated by a ':'. The field data is always preceded by the corresponding keyword, separated by a '='. The fields are separated by 0x00. The data of the different tracks is separated by an additional 0x00. The end of the buffer is marked by another additional 0x00 (see example below). Data encoding is defined in Section 7, Form Definition.

Example of *lpstrTrackData*:
TRACK2:ALL=47..\0\0TRACK3:MII=59\0PAN=500..\0\0\0

5.13 WFS_CMD_IDC_SET_GUIDANCE_LIGHT

Description This command is used to set the status of the IDC guidance lights. This includes defining the flash rate and the color. When an application tries to use a color that is not supported then the Service Provider will return the generic error WFS_ERR_UNSUPP_DATA.

Input Param LPWFSIDCSETGUIDLIGHT lpSetGuidLight;

```
typedef struct _wfs_idc_set_guidlight
     {
     WORD                wGuidLight;
     DWORD               dwCommand;
     } WFSIDCSETGUIDLIGHT, *LPWFSIDCSETGUIDLIGHT;
```

wGuidLight
Specifies the index of the guidance light to set as one of the values defined within the capabilities section.

dwCommand
Specifies the state of the guidance light indicator as WFS_IDC_GUIDANCE_OFF or a combination of the following flags consisting of one type B, and optionally one type C. If no value of type C is specified then the default color is used. The Service Provider determines which color is used as the default color.

Value	Meaning	Type
WFS_IDC_GUIDANCE_OFF	The light indicator is turned off.	A
WFS_IDC_GUIDANCE_SLOW_FLASH	The light indicator is set to flash slowly.	B
WFS_IDC_GUIDANCE_MEDIUM_FLASH	The light indicator is set to flash medium frequency.	B
WFS_IDC_GUIDANCE_QUICK_FLASH	The light indicator is set to flash quickly.	B
WFS_IDC_GUIDANCE_CONTINUOUS	The light indicator is turned on continuously (steady).	B
WFS_IDC_GUIDANCE_RED	The light indicator color is set to red.	C
WFS_IDC_GUIDANCE_GREEN	The light indicator color is set to green.	C
WFS_IDC_GUIDANCE_YELLOW	The light indicator color is set to yellow.	C
WFS_IDC_GUIDANCE_BLUE	The light indicator color is set to blue.	C
WFS_IDC_GUIDANCE_CYAN	The light indicator color is set to cyan.	C
WFS_IDC_GUIDANCE_MAGENTA	The light indicator color is set to magenta.	C
WFS_IDC_GUIDANCE_WHITE	The light indicator color is set to white.	C

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_INVALID_PORT	An attempt to set a guidance light to a new value was invalid because the guidance light does not exist.

Events Only the generic events defined in [Ref. 1] can be generated by this command:

Comments Guidance light support was added into the IDC primarily to support guidance lights for workstations where more than one instance of an IDC is present. The original SIU guidance light mechanism was not able to manage guidance lights for workstations with multiple IDCs. This command can also be used to set the status of the IDC guidance lights when only one instance of an IDC is present.

5.14 WFS_CMD_IDC_POWER_SAVE_CONTROL

Description This command activates or deactivates the power-saving mode.

If the Service Provider receives another execute command while in power saving mode, the Service Provider automatically exits the power saving mode, and executes the requested command. If the Service Provider receives an information command while in power saving mode, the Service Provider will not exit the power saving mode.

Input Param LPWFSIDCPOWERSAVECONTROL lpPowerSaveControl;

```
typedef struct _wfs_idc_power_save_control
     {
     USHORT               usMaxPowerSaveRecoveryTime;
     } WFSIDCPOWERSAVECONTROL, *LPWFSIDCPOWERSAVECONTROL;
```

usMaxPowerSaveRecoveryTime
Specifies the maximum number of seconds in which the device must be able to return to its normal operating state when exiting power save mode. The device will be set to the highest possible power save mode within this constraint. If *usMaxPowerSaveRecoveryTime* is set to zero then the device will exit the power saving mode.

Output Param None.

Error Codes In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:

Value	Meaning
WFS_ERR_IDC_POWERSAVETOOSHORT	The power saving mode has not been activated because the device is not able to resume from the power saving mode within the specified *usMaxPowerSaveRecoveryTime* value.
WFS_ERR_IDC_POWERSAVEMEDIAPRESENT	The power saving mode has not been activated because media is present inside the device.

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

Value	Meaning
WFS_SRVE_IDC_POWER_SAVE_CHANGE	The power save recovery time has changed.

Comments None.

6. Events

6.1 WFS_EXEE_IDC_INVALIDTRACKDATA

Description This execute event specifies that a track contained invalid or no data.

Event Param LPWFSIDCTRACKEVENT lpTrackEvent;

```
typedef struct _wfs_idc_track_event
     {
     WORD                fwStatus;
     LPSTR               lpstrTrack;
     LPSTR               lpstrData;
     } WFSIDCTRACKEVENT, *LPWFSIDCTRACKEVENT;
```

fwStatus
Status of reading the track. Possible values are:

Value	Meaning
WFS_IDC_DATAMISSING	The track is blank.
WFS_IDC_DATAINVALID	The data contained on the track is invalid.
WFS_IDC_DATATOOLONG	The data contained on the track is too long.
WFS_IDC_DATATOOSHORT	The data contained on the track is too short.

lpstrTrack
Points to the keyword of the track on which the error occurred.

lpstrData
Points to the data that could be read (that may be only a fragment of the track), terminated by a null character. This data is simply a stream of characters; it does not contain keywords.

Comments None.

6.2 WFS_EXEE_IDC_MEDIAINSERTED

Description This execute event specifies that a card was inserted into the device.

Event Param None.

Comments None.

6.3 WFS_SRVE_IDC_MEDIAREMOVED

Description This service event specifies that the inserted card was manually removed by the user during the processing of a read/write command, during the processing of a chip_io/power command, during or after a retain/reset operation, after an eject operation or after the card is removed by the user in a latched dip card unit.

Event Param None.

Comments None.

6.4 WFS_EXEE_IDC_MEDIARETAINED

Description This service event specifies that the card was retained.

Event Param None.

Comments None.

6.5 WFS_EXEE_IDC_INVALIDMEDIA

Description This execute event specifies that the media the user is attempting to insert is not a valid card or it is a card but it is in the wrong orientation.

Event Param None.

Comments None.

6.6 WFS_SRVE_IDC_CARDACTION

Description This service event specifies that a card has been retained or ejected by either the automatic power on or power off action of the device.

Event Param LPWFSIDCCARDACT lpCardAct;

```
typedef struct _wfs_idc_card_act
     {
     WORD                wAction;
     WORD                wPosition;
     } WFSIDCCARDACT, *LPWFSIDCCARDACT;
```

wAction
Specifies which action has been performed with the card. Possible values are:

Value	Meaning
WFS_IDC_CARDRETAINED	The card has been retained.
WFS_IDC_CARDEJECTED	The card has been ejected.
WFS_IDC_CARDREADPOSITION	The card has been moved to the read position.

wPosition
Position of card before being retained or ejected. Possible values are:

Value	Meaning
WFS_IDC_MEDIAUNKNOWN	The position of the card can not be determined.
WFS_IDC_MEDIAPRESENT	The card was present in the reader.
WFS_IDC_MEDIAENTERING	The card was entering the reader.

Comments None.

6.7 WFS_USRE_IDC_RETAINBINTHRESHOLD

Description This user event specifies that the retain bin holding the retained cards has reached a threshold condition or the threshold condition is removed.

Event Param LPWORD lpfwRetainBin;

lpfwRetainBin
Specifies the state of the ID card unit retain bin as one of the following values:

Value	Meaning
WFS_IDC_RETAINBINOK	The retain bin of the ID card unit was emptied.
WFS_IDC_RETAINBINFULL	The retain bin of the ID card unit is full.
WFS_IDC_RETAINBINHIGH	The retain bin of the ID card unit is nearly full.

Comments None.

6.8 WFS_SRVE_IDC_MEDIADETECTED

Description This service event is generated if media is detected during a reset (WFS_CMD_IDC_RESET). The parameter on the event informs the application of the position of the card on the completion of the reset.

Event Param LPWORD lpwResetOut;

lpwResetOut
Specifies the action that was performed on any card found within the IDC as one of the following values:

Value	Meaning
WFS_IDC_CARDEJECTED	The card was ejected.
WFS_IDC_CARDRETAINED	The card was retained.
WFS_IDC_CARDREADPOSITION	The card is in read position.
WFS_IDC_CARDJAMMED	The card is jammed in the device.

Comments None.

6.9 WFS_SRVE_IDC_RETAINBINREMOVED

Description This event specifies that the retain bin has been removed.

Event Param None.

Comments None.

6.10 WFS_SRVE_IDC_RETAINBININSERTED

Description This event specifies that the retain bin has been inserted.

Event Param None.

Comments None.

6.11 WFS_EXEE_IDC_INSERTCARD

Description This mandatory event notifies the application when the device is ready for the user to insert a card.

Event Param None.

Comments None.

6.12 WFS_SRVE_IDC_DEVICEPOSITION

Description This service event reports that the device has changed its position status.

Event Param LPWFSIDCDEVICEPOSITION lpDevicePosition;

```
typedef struct _wfs_idc_device_position
     {
     WORD                wPosition;
     } WFSIDCDEVICEPOSITION, *LPWFSIDCDEVICEPOSITION;
```

wPosition
Position of the device as one of the following values:

Value	Meaning
WFS_IDC_DEVICEINPOSITION	The device is in its normal operating position.
WFS_IDC_DEVICENOTINPOSITION	The device has been removed from its normal operating position.
WFS_IDC_DEVICEPOSUNKNOWN	The position of the device cannot be determined.

Comments None.

6.13 WFS_SRVE_IDC_POWER_SAVE_CHANGE

Description This service event specifies that the power save recovery time has changed.

Event Param LPWFSIDCPOWERSAVECHANGE lpPowerSaveChange;

```
typedef struct _wfs_idc_power_save_change
     {
     USHORT               usPowerSaveRecoveryTime;
     } WFSIDCPOWERSAVECHANGE, *LPWFSIDCPOWERSAVECHANGE;
```

usPowerSaveRecoveryTime
Specifies the actual number of seconds required by the device to resume its normal operational state. This value is zero if the device exited the power saving mode.

Comments None.

7. Form Description

This section describes the forms mechanism used to define the tracks to be read or written. Forms are contained in a single file, with one section for each defined form. The name of each section is the form name parameter in the WFS_CMD_IDC_READ_TRACK and WFS_CMD_IDC_WRITE_TRACK commands.

The way to specify the location of a form file is vendor dependent.

As an example the following registry information can be used:

```
WOSA/XFS_ROOT
        FORMS
                IDCU
                        formfile=<path><filename>
```

The read form defines which tracks should be read in the WFS_CMD_IDC_READ_TRACK command and what the response should be to a read failure. The read form can also be used to define logical track data, i.e. fields like “account number”, “issuer identifier”, and their position within the physical track data. For example, the output parameter of the WFS_CMD_IDC_READ_TRACK command with input parameter *lpstrFormName* = READTRACK3GERMAN could look like (see example 1 below):

"TRACK3:MII=59\0COUNTRY=280\0ISSUERID=50050500\0ACCOUNT=1234567890\0LUHNT3=1\0EXPIRATION=9912\0SECURE=1\0\0\0"

The write form defines which track is to be written, the logical track data that is handed over in the WFS_CMD_IDC_WRITE_TRACK command, and how the write data is to be converted to the physical data to be written.

Reserved Keywords/Operands [1]	Meaning
[]	Form name delimiters.
TRACK1	Keyword to identify track 1.
TRACK2	Keyword to identify track 2.
TRACK3	Keyword to identify track 3.
FIELDSEPT1	Value of field separator of track 1.
FIELDSEPT2	Value of field separator of track 2.
FIELDSEPT3	Value of field separator of track 3.
READ	Description of read action; the TRACKn keywords are processed left to right.
WRITE	Description of write action.
ALL	Read or write the complete track.
SECURE	Do the security check via the security module (CIM86 or MM). This check is done on Track 3 only.
&	Read/write all tracks specified, abort reading on read failure.
|	Read/write at least one of the tracks specified, continue reading on read failure.
FIELDSEPPOSn	Position of the nth occurrence of field separator on track. FIELDSEPPOS0 specifies the beginning of the data.
,	Separator in a list of logical fields.
DEFAULT	String for default substitution of track data to be written, that is not defined explicitly by the form fields. DEFAULT also allows an application to input fewer fields than those defined by the form.

[1] Attributes are not required in any mandatory order.

?	Reserved value for DEFAULT keyword: substitute track data to write with its value read before.
ENDTRACK	Represents the end of the data. It is used to identify fields positioned after the last field separator.

Notes

The & and | operands may be combined in a single READ statement; for example:

- read track3 or track2, trying track3 first:
  READ= TRACK3 | TRACK2
- read track 3 and at least one of track2 or track1:
  READ= TRACK3 & (TRACK2 | TRACK1)
  or:
  READ= TRACK2 | TRACK1 & TRACK3

The keywords FIELDSEPPOS0 and ENDTRACK are used as follows:

- read the first 2 bytes of a track:
  FIRST= FIELDSEPPOS0 + 1, FIELDSEPPOS0 + 2
- read the last 2 bytes of a track:
  LAST= ENDTRACK – 2, ENDTRACK – 1

Use of field separators in track layouts is to replace optional fields and terminate variable length fields.

Write forms are designed for updating specific fields without altering the position of the field separators.

The application may alter the position of the field separators by rewriting the card tracks (ALL option or DEFAULT option with default track data).

It is valid to define a field that spans another field separator, e.g. FIELDSEPPOS1+1, FIELDSEPPOS3+1 is valid as is FIELDSEPPOS3-4, FIELDSEPPOS3-1 where a field separator (e.g. FIELDSEPPOS2) lies within this range on the data read from the card. During a read track the field separator is returned within the track data. During a write track the application must ensure the correct number of field separators at the correct location with the correct spacing is included in the data, otherwise a WFS_ERR_IDC_DATASYNTAX error will be returned.

Example 1 Reading tracks:

```
[READTRACK3GERMAN]
/* field separator of track 3 */
FIELDSEPT3= =

/* only track 3 must be read */
READ= TRACK3

/* read logical fields as defined below; also check the security */
TRACK3= MII, COUNTRY, ISSUERID, ACCOUNT, LUHNT3, EXPIRATION, SECURE
MII= FIELDSEPPOS0 + 3, FIELDSEPPOS0 + 4
ISSUERID= FIELDSEPPOS0 + 5, FIELDSEPPOS1 - 1
ACCOUNT= FIELDSEPPOS1 + 1, FIELDSEPPOS2 - 2
LUHNT3= FIELDSEPPOS2 - 1, FIELDSEPPOS2 – 1
COUNTRY= FIELDSEPPOS2 + 1, FIELDSEPPOS2 + 3
EXPIRATION= FIELDSEPPOS2 + 36, FIELDSEPPOS2 + 39
```

All tracks must be read (‘READ’), that is, the read fails if an error occurs on reading any one of the tracks (the ‘&’ operand). The field “major industry identifier” (‘MII’) is located after the first field separator (‘FIELDSEPPOS1’) and its length is two bytes. The “issuer identifier” field (‘ISSUERID’) is located after the MII field, with a length of eight bytes. The next field, “account number” (‘ACCOUNT’) is variable length; it ends before the luhn digit field (‘LUHNT3’) that is the last digit in front of the second field separator (‘FIELDSEPPOS2’).

Example 2 Write a track:

```
[WRITETRACK3]
FIELDSEPT3=  =
```

```
DEFAULT= ? /* fields not specified in the write form are to be left
unchanged, i.e. read and the same data written back to them */
WRITE= TRACK3
TRACK3= RETRYCOUNT, DATE
RETRYCOUNT= FIELDSEPPOS2 + 22, FIELDSEPPOS2 + 22
DATE= FIELDSEPPOS5 + 1, FIELDSEPPOS5 + 4
```

Track 3 is to be written. In the example only the retry counter and the date of the last transaction are updated, the other fields are unchanged.

A sample of input data to be used with this form is as follows:

RETRYCOUNT=3\0DATE=3132\00

Example 3 Write a track:

```
[WRITETRACK3ALL]
WRITE= TRACK3
TRACK3= ALL
```

Track 3 is to be written. By specifying ALL, the data passed in the WFS_CMD_IDC_WRITE_TRACK command is written to the physical track without formatting.

A sample of input data to be used with this form is as follows:

ALL=123456789123\0\0

8. C-Header file

```
/******************************************************************************
*                                                                             *
* xfsidc.h    XFS - Identification card unit (IDC) definitions                *
*                                                                             *
*             Version 3.10  (29/11/2007)                                      *
*                                                                             *
******************************************************************************/

#ifndef __INC_XFSIDC__H
#define __INC_XFSIDC__H

#ifdef __cplusplus
extern "C" {
#endif

#include <xfsapi.h>

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

/* values of WFSIDCCAPS.wClass */

#define     WFS_SERVICE_CLASS_IDC               (2)
#define     WFS_SERVICE_CLASS_NAME_IDC          "IDC"
#define     WFS_SERVICE_CLASS_VERSION_IDC       (0x0A03) /* Version 3.10 */

#define     IDC_SERVICE_OFFSET                  (WFS_SERVICE_CLASS_IDC * 100)

/* IDC Info Commands */

#define     WFS_INF_IDC_STATUS                  (IDC_SERVICE_OFFSET + 1)
#define     WFS_INF_IDC_CAPABILITIES            (IDC_SERVICE_OFFSET + 2)
#define     WFS_INF_IDC_FORM_LIST               (IDC_SERVICE_OFFSET + 3)
#define     WFS_INF_IDC_QUERY_FORM              (IDC_SERVICE_OFFSET + 4)
#define     WFS_INF_IDC_QUERY_IFM_IDENTIFIER    (IDC_SERVICE_OFFSET + 5)

/* IDC Execute Commands */

#define     WFS_CMD_IDC_READ_TRACK              (IDC_SERVICE_OFFSET + 1)
#define     WFS_CMD_IDC_WRITE_TRACK             (IDC_SERVICE_OFFSET + 2)
#define     WFS_CMD_IDC_EJECT_CARD              (IDC_SERVICE_OFFSET + 3)
#define     WFS_CMD_IDC_RETAIN_CARD             (IDC_SERVICE_OFFSET + 4)
#define     WFS_CMD_IDC_RESET_COUNT             (IDC_SERVICE_OFFSET + 5)
#define     WFS_CMD_IDC_SETKEY                  (IDC_SERVICE_OFFSET + 6)
#define     WFS_CMD_IDC_READ_RAW_DATA           (IDC_SERVICE_OFFSET + 7)
#define     WFS_CMD_IDC_WRITE_RAW_DATA          (IDC_SERVICE_OFFSET + 8)
#define     WFS_CMD_IDC_CHIP_IO                 (IDC_SERVICE_OFFSET + 9)
#define     WFS_CMD_IDC_RESET                   (IDC_SERVICE_OFFSET + 10)
#define     WFS_CMD_IDC_CHIP_POWER              (IDC_SERVICE_OFFSET + 11)
#define     WFS_CMD_IDC_PARSE_DATA              (IDC_SERVICE_OFFSET + 12)
#define     WFS_CMD_IDC_SET_GUIDANCE_LIGHT      (IDC_SERVICE_OFFSET + 13)
#define     WFS_CMD_IDC_POWER_SAVE_CONTROL      (IDC_SERVICE_OFFSET + 14)

/* IDC Messages */

#define     WFS_EXEE_IDC_INVALIDTRACKDATA       (IDC_SERVICE_OFFSET + 1)
#define     WFS_EXEE_IDC_MEDIAINSERTED          (IDC_SERVICE_OFFSET + 3)
#define     WFS_SRVE_IDC_MEDIAREMOVED           (IDC_SERVICE_OFFSET + 4)
#define     WFS_SRVE_IDC_CARDACTION             (IDC_SERVICE_OFFSET + 5)
#define     WFS_USRE_IDC_RETAINBINTHRESHOLD     (IDC_SERVICE_OFFSET + 6)
#define     WFS_EXEE_IDC_INVALIDMEDIA           (IDC_SERVICE_OFFSET + 7)
#define     WFS_EXEE_IDC_MEDIARETAINED          (IDC_SERVICE_OFFSET + 8)
#define     WFS_SRVE_IDC_MEDIADETECTED          (IDC_SERVICE_OFFSET + 9)
#define     WFS_SRVE_IDC_RETAINBININSERTED      (IDC_SERVICE_OFFSET + 10)
#define     WFS_SRVE_IDC_RETAINBINREMOVED       (IDC_SERVICE_OFFSET + 11)
#define     WFS_EXEE_IDC_INSERTCARD             (IDC_SERVICE_OFFSET + 12)
#define     WFS_SRVE_IDC_DEVICEPOSITION         (IDC_SERVICE_OFFSET + 13)
```

```
#define     WFS_SRVE_IDC_POWER_SAVE_CHANGE      (IDC_SERVICE_OFFSET + 14)

/* values of WFSIDCSTATUS.fwDevice */

#define     WFS_IDC_DEVONLINE                   WFS_STAT_DEVONLINE
#define     WFS_IDC_DEVOFFLINE                  WFS_STAT_DEVOFFLINE
#define     WFS_IDC_DEVPOWEROFF                 WFS_STAT_DEVPOWEROFF
#define     WFS_IDC_DEVNODEVICE                 WFS_STAT_DEVNODEVICE
#define     WFS_IDC_DEVHWERROR                  WFS_STAT_DEVHWERROR
#define     WFS_IDC_DEVUSERERROR                WFS_STAT_DEVUSERERROR
#define     WFS_IDC_DEVBUSY                     WFS_STAT_DEVBUSY
#define     WFS_IDC_DEVFRAUDATTEMPT             WFS_STAT_DEVFRAUDATTEMPT

/* values of WFSIDCSTATUS.fwMedia, WFSIDCRETAINCARD.fwPosition,  */
/* WFSIDCCARDACT.fwPosition  */

#define     WFS_IDC_MEDIAPRESENT                (1)
#define     WFS_IDC_MEDIANOTPRESENT             (2)
#define     WFS_IDC_MEDIAJAMMED                 (3)
#define     WFS_IDC_MEDIANOTSUPP                (4)
#define     WFS_IDC_MEDIAUNKNOWN                (5)
#define     WFS_IDC_MEDIAENTERING               (6)
#define     WFS_IDC_MEDIALATCHED                (7)

/* values of WFSIDCSTATUS.fwRetainBin */

#define     WFS_IDC_RETAINBINOK                 (1)
#define     WFS_IDC_RETAINNOTSUPP               (2)
#define     WFS_IDC_RETAINBINFULL               (3)
#define     WFS_IDC_RETAINBINHIGH               (4)
#define     WFS_IDC_RETAINBINMISSING            (5)

/* values of WFSIDCSTATUS.fwSecurity */

#define     WFS_IDC_SECNOTSUPP                  (1)
#define     WFS_IDC_SECNOTREADY                 (2)
#define     WFS_IDC_SECOPEN                     (3)

/* values of WFSIDCSTATUS.fwChipPower */

#define     WFS_IDC_CHIPONLINE                  (0)
#define     WFS_IDC_CHIPPOWEREDOFF              (1)
#define     WFS_IDC_CHIPBUSY                    (2)
#define     WFS_IDC_CHIPNODEVICE                (3)
#define     WFS_IDC_CHIPHWERROR                 (4)
#define     WFS_IDC_CHIPNOCARD                  (5)
#define     WFS_IDC_CHIPNOTSUPP                 (6)
#define     WFS_IDC_CHIPUNKNOWN                 (7)

/* Size and max index of dwGuidLights array */
#define     WFS_IDC_GUIDLIGHTS_SIZE             (32)
#define     WFS_IDC_GUIDLIGHTS_MAX              (WFS_IDC_GUIDLIGHTS_SIZE - 1)

/* Indices of WFSIDCSTATUS.dwGuidLights [...]
              WFSIDCCAPS.dwGuidLights [...]
*/
#define     WFS_IDC_GUIDANCE_CARDUNIT           (0)

/* Values of WFSIDCSTATUS.dwGuidLights [...]
             WFSIDCCAPS.dwGuidLights [...]
*/
#define     WFS_IDC_GUIDANCE_NOT_AVAILABLE      (0x00000000)
#define     WFS_IDC_GUIDANCE_OFF                (0x00000001)
#define     WFS_IDC_GUIDANCE_ON                 (0x00000002)
#define     WFS_IDC_GUIDANCE_SLOW_FLASH         (0x00000004)
#define     WFS_IDC_GUIDANCE_MEDIUM_FLASH       (0x00000008)
#define     WFS_IDC_GUIDANCE_QUICK_FLASH        (0x00000010)
#define     WFS_IDC_GUIDANCE_CONTINUOUS         (0x00000080)
#define     WFS_IDC_GUIDANCE_RED                (0x00000100)
#define     WFS_IDC_GUIDANCE_GREEN              (0x00000200)
```

```
#define     WFS_IDC_GUIDANCE_YELLOW             (0x00000400)
#define     WFS_IDC_GUIDANCE_BLUE               (0x00000800)
#define     WFS_IDC_GUIDANCE_CYAN               (0x00001000)
#define     WFS_IDC_GUIDANCE_MAGENTA            (0x00002000)
#define     WFS_IDC_GUIDANCE_WHITE              (0x00004000)



/* values of WFSIDCSTATUS.fwChipModule */

#define     WFS_IDC_CHIPMODOK                   (1)
#define     WFS_IDC_CHIPMODINOP                 (2)
#define     WFS_IDC_CHIPMODUNKNOWN              (3)
#define     WFS_IDC_CHIPMODNOTSUPP              (4)

/* values of WFSIDCSTATUS.fwMagReadModule and
             WFSIDCSTATUS.fwMagWriteModule  */

#define     WFS_IDC_MAGMODOK                    (1)
#define     WFS_IDC_MAGMODINOP                  (2)
#define     WFS_IDC_MAGMODUNKNOWN               (3)
#define     WFS_IDC_MAGMODNOTSUPP               (4)

/* values of WFSIDCSTATUS.fwFrontImageModule and
             WFSIDCSTATUS.fwBackImageModule */

#define     WFS_IDC_IMGMODOK                    (1)
#define     WFS_IDC_IMGMODINOP                  (2)
#define     WFS_IDC_IMGMODUNKNOWN               (3)
#define     WFS_IDC_IMGMODNOTSUPP               (4)

/* values of WFSIDCSTATUS.wDevicePosition
             WFSIDCDEVICEPOSITION.wPosition */

#define     WFS_IDC_DEVICEINPOSITION            (0)
#define     WFS_IDC_DEVICENOTINPOSITION         (1)
#define     WFS_IDC_DEVICEPOSUNKNOWN            (2)
#define     WFS_IDC_DEVICEPOSNOTSUPP            (3)

/* values of WFSIDCCAPS.fwType */

#define     WFS_IDC_TYPEMOTOR                   (1)
#define     WFS_IDC_TYPESWIPE                   (2)
#define     WFS_IDC_TYPEDIP                     (3)
#define     WFS_IDC_TYPECONTACTLESS             (4)
#define     WFS_IDC_TYPELATCHEDDIP              (5)
#define     WFS_IDC_TYPEPERMANENT               (6)

/* values of WFSIDCCAPS.fwReadTracks,
             WFSIDCCAPS.fwWriteTracks,
             WFSIDCCARDDATA.wDataSource,
             WFSIDCCAPS.fwChipProtocols,
             WFSIDCCAPSfwWriteMode,
             WFSIDCCAPS.fwChipPower */

#define     WFS_IDC_NOTSUPP                     0x0000

/* values of WFSIDCCAPS.fwReadTracks, WFSIDCCAPS.fwWriteTracks,
             WFSIDCCARDDATA.wDataSource,
             WFS_CMD_IDC_READ_RAW_DATA */

#define     WFS_IDC_TRACK1                      0x0001
#define     WFS_IDC_TRACK2                      0x0002
#define     WFS_IDC_TRACK3                      0x0004
#define     WFS_IDC_FRONT_TRACK_1               0x0080

/* further values of WFSIDCCARDDATA.wDataSource (except
   WFS_IDC_FLUXINACTIVE), WFS_CMD_IDC_READ_RAW_DATA */

#define     WFS_IDC_CHIP                        0x0008
```

```
#define     WFS_IDC_SECURITY                    0x0010
#define     WFS_IDC_FLUXINACTIVE                0x0020
#define     WFS_IDC_TRACK_WM                    0x8000
#define     WFS_IDC_MEMORY_CHIP                 0x0040
#define     WFS_IDC_FRONTIMAGE                  0x0100
#define     WFS_IDC_BACKIMAGE                   0x0200

/* values of WFSIDCCAPS.fwChipProtocols */

#define     WFS_IDC_CHIPT0                      0x0001
#define     WFS_IDC_CHIPT1                      0x0002
#define     WFS_IDC_CHIP_PROTOCOL_NOT_REQUIRED  0x0004

/* values of WFSIDCCAPS.fwSecType */

#define     WFS_IDC_SECNOTSUPP                  (1)
#define     WFS_IDC_SECMMBOX                    (2)
#define     WFS_IDC_SECCIM86                    (3)

/* values of WFSIDCCAPS.fwPowerOnOption, WFSIDCCAPS.fwPowerOffOption*/

#define     WFS_IDC_NOACTION                    (1)
#define     WFS_IDC_EJECT                       (2)
#define     WFS_IDC_RETAIN                      (3)
#define     WFS_IDC_EJECTTHENRETAIN             (4)
#define     WFS_IDC_READPOSITION                (5)

/* values of WFSIDCCAPS.fwWriteMode; WFSIDCWRITETRACK.fwWriteMethod,
WFSIDCCARDDATA.fwWriteMethod */

/* Note: WFS_IDC_UNKNOWN was removed as it was an invalid value */
#define     WFS_IDC_LOCO                        0x0002
#define     WFS_IDC_HICO                        0x0004
#define     WFS_IDC_AUTO                        0x0008

/* values of WFSIDCCAPS.fwChipPower */

#define     WFS_IDC_CHIPPOWERCOLD               0x0002
#define     WFS_IDC_CHIPPOWERWARM               0x0004
#define     WFS_IDC_CHIPPOWEROFF                0x0008

/* values of WFSIDCCAPS.fwDIPMode */

#define     WFS_IDC_DIP_UNKNOWN                 0x0001
#define     WFS_IDC_DIP_EXIT                    0x0002
#define     WFS_IDC_DIP_ENTRY                   0x0004
#define     WFS_IDC_DIP_ENTRY_EXIT              0x0008

/* values of WFSIDCCAPS. lpwMemoryChipProtocols */

#define     WFS_IDC_MEM_SIEMENS4442             0x0001
#define     WFS_IDC_MEM_GPM896                  0x0002

/* values of WFSIDCFORM.fwAction */

#define     WFS_IDC_ACTIONREAD                  0x0001
#define     WFS_IDC_ACTIONWRITE                 0x0002

/* values of WFSIDCTRACKEVENT.fwStatus, WFSIDCCARDDATA.wStatus */

#define     WFS_IDC_DATAOK                      (0)
#define     WFS_IDC_DATAMISSING                 (1)
#define     WFS_IDC_DATAINVALID                 (2)
#define     WFS_IDC_DATATOOLONG                 (3)
#define     WFS_IDC_DATATOOSHORT                (4)
#define     WFS_IDC_DATASRCNOTSUPP              (5)
#define     WFS_IDC_DATASRCMISSING              (6)

/* values WFSIDCCARDACT.wAction */
```

```
#define     WFS_IDC_CARDRETAINED                (1)
#define     WFS_IDC_CARDEJECTED                 (2)
#define     WFS_IDC_CARDREADPOSITION            (3)
#define     WFS_IDC_CARDJAMMED                  (4)

/* values of WFSIDCCARDDATA.lpbData if security is read */

#define     WFS_IDC_SEC_READLEVEL1              '1'
#define     WFS_IDC_SEC_READLEVEL2              '2'
#define     WFS_IDC_SEC_READLEVEL3              '3'
#define     WFS_IDC_SEC_READLEVEL4              '4'
#define     WFS_IDC_SEC_READLEVEL5              '5'
#define     WFS_IDC_SEC_BADREADLEVEL            '6'
#define     WFS_IDC_SEC_NODATA                  '7'
#define     WFS_IDC_SEC_DATAINVAL               '8'
#define     WFS_IDC_SEC_HWERROR                 '9'
#define     WFS_IDC_SEC_NOINIT                  'A'

/* values of WFSIDCIFMIDENTIFIER.wIFMAuthority */

#define     WFS_IDC_IFMEMV                      (1)
#define     WFS_IDC_IFMEUROPAY                  (2)
#define     WFS_IDC_IFMVISA                     (3)
#define     WFS_IDC_IFMGIECB                    (4)

/* values WFSIDCCAPS.fwEjectPosition, WFSIDCEJECTCARD.wEjectPosition */

#define     WFS_IDC_EXITPOSITION                (0x0001)
#define     WFS_IDC_TRANSPORTPOSITION           (0x0002)

/* WOSA/XFS IDC Errors */

#define WFS_ERR_IDC_MEDIAJAM                    (-(IDC_SERVICE_OFFSET + 0))
#define WFS_ERR_IDC_NOMEDIA                     (-(IDC_SERVICE_OFFSET + 1))
#define WFS_ERR_IDC_MEDIARETAINED               (-(IDC_SERVICE_OFFSET + 2))
#define WFS_ERR_IDC_RETAINBINFULL               (-(IDC_SERVICE_OFFSET + 3))
#define WFS_ERR_IDC_INVALIDDATA                 (-(IDC_SERVICE_OFFSET + 4))
#define WFS_ERR_IDC_INVALIDMEDIA                (-(IDC_SERVICE_OFFSET + 5))
#define WFS_ERR_IDC_FORMNOTFOUND                (-(IDC_SERVICE_OFFSET + 6))
#define WFS_ERR_IDC_FORMINVALID                 (-(IDC_SERVICE_OFFSET + 7))
#define WFS_ERR_IDC_DATASYNTAX                  (-(IDC_SERVICE_OFFSET + 8))
#define WFS_ERR_IDC_SHUTTERFAIL                 (-(IDC_SERVICE_OFFSET + 9))
#define WFS_ERR_IDC_SECURITYFAIL                (-(IDC_SERVICE_OFFSET + 10))
#define WFS_ERR_IDC_PROTOCOLNOTSUPP             (-(IDC_SERVICE_OFFSET + 11))
#define WFS_ERR_IDC_ATRNOTOBTAINED              (-(IDC_SERVICE_OFFSET + 12))
#define WFS_ERR_IDC_INVALIDKEY                  (-(IDC_SERVICE_OFFSET + 13))
#define WFS_ERR_IDC_WRITE_METHOD                (-(IDC_SERVICE_OFFSET + 14))
#define WFS_ERR_IDC_CHIPPOWERNOTSUPP            (-(IDC_SERVICE_OFFSET + 15))
#define WFS_ERR_IDC_CARDTOOSHORT                (-(IDC_SERVICE_OFFSET + 16))
#define WFS_ERR_IDC_CARDTOOLONG                 (-(IDC_SERVICE_OFFSET + 17))
#define WFS_ERR_IDC_INVALID_PORT                (-(IDC_SERVICE_OFFSET + 18))
#define WFS_ERR_IDC_POWERSAVETOOSHORT           (-(IDC_SERVICE_OFFSET + 19))
#define WFS_ERR_IDC_POWERSAVEMEDIAPRESENT       (-(IDC_SERVICE_OFFSET + 20))

/*=================================================================*/
/* IDC Info Command Structures and variables */
/*=================================================================*/

typedef struct _wfs_idc_status
{
    WORD                    fwDevice;
    WORD                    fwMedia;
    WORD                    fwRetainBin;
    WORD                    fwSecurity;
    USHORT                  usCards;
    WORD                    fwChipPower;
    LPSTR                   lpszExtra;
    DWORD                   dwGuidLights[WFS_IDC_GUIDLIGHTS_SIZE];
    WORD                    fwChipModule;
    WORD                    fwMagReadModule;
```

```
    WORD                    fwMagWriteModule;
    WORD                    fwFrontImageModule;
    WORD                    fwBackImageModule;
    WORD                    wDevicePosition;
    USHORT                  usPowerSaveRecoveryTime;
} WFSIDCSTATUS, *LPWFSIDCSTATUS;

typedef struct _wfs_idc_caps
{
    WORD                    wClass;
    WORD                    fwType;
    BOOL                    bCompound;
    WORD                    fwReadTracks;
    WORD                    fwWriteTracks;
    WORD                    fwChipProtocols;
    USHORT                  usCards;
    WORD                    fwSecType;
    WORD                    fwPowerOnOption;
    WORD                    fwPowerOffOption;
    BOOL                    bFluxSensorProgrammable;
    BOOL                    bReadWriteAccessFollowingEject;
    WORD                    fwWriteMode;
    WORD                    fwChipPower;
    LPSTR                   lpszExtra;
    WORD                    fwDIPMode;
    LPWORD                  lpwMemoryChipProtocols;
    DWORD                   dwGuidLights[WFS_IDC_GUIDLIGHTS_SIZE];
    WORD                    fwEjectPosition;
    BOOL                    bPowerSaveControl;
} WFSIDCCAPS, *LPWFSIDCCAPS;

typedef struct _wfs_idc_form
{
    LPSTR                   lpszFormName;
    CHAR                    cFieldSeparatorTrack1;
    CHAR                    cFieldSeparatorTrack2;
    CHAR                    cFieldSeparatorTrack3;
    WORD                    fwAction;
    LPSTR                   lpszTracks;
    BOOL                    bSecure;
    LPSTR                   lpszTrack1Fields;
    LPSTR                   lpszTrack2Fields;
    LPSTR                   lpszTrack3Fields;
} WFSIDCFORM, *LPWFSIDCFORM;

typedef struct _wfs_idc_ifm_identifier
{
    WORD                    wIFMAuthority;
    LPSTR                   lpszIFMIdentifier;
} WFSIDCIFMIDENTIFIER, *LPWFSIDCIFMIDENTIFIER;

/*=================================================================*/
/* IDC Execute Command Structures */
/*=================================================================*/

typedef struct _wfs_idc_write_track
{
    LPSTR                   lpstrFormName;
    LPSTR                   lpstrTrackData;
    WORD                    fwWriteMethod;
} WFSIDCWRITETRACK, *LPWFSIDCWRITETRACK;

typedef struct _wfs_idc_retain_card
{
    USHORT                  usCount;
    WORD                    fwPosition;
} WFSIDCRETAINCARD, *LPWFSIDCRETAINCARD;

typedef struct _wfs_idc_setkey
{
```

```
    USHORT                  usKeyLen;
    LPBYTE                  lpbKeyValue;
} WFSIDCSETKEY, *LPWFSIDCSETKEY;

typedef struct _wfs_idc_card_data
{
    WORD                    wDataSource;
    WORD                    wStatus;
    ULONG                   ulDataLength;
    LPBYTE                  lpbData;
    WORD                    fwWriteMethod;
} WFSIDCCARDDATA, *LPWFSIDCCARDDATA;

typedef struct _wfs_idc_chip_io
{
    WORD                    wChipProtocol;
    ULONG                   ulChipDataLength;
    LPBYTE                  lpbChipData;
} WFSIDCCHIPIO, *LPWFSIDCCHIPIO;

typedef struct _wfs_idc_chip_power_out
{
    ULONG                   ulChipDataLength;
    LPBYTE                  lpbChipData;
} WFSIDCCHIPPOWEROUT, *LPWFSIDCCHIPPOWEROUT;

typedef struct _wfs_idc_parse_data
{
    LPSTR                   lpstrFormName;
    LPWFSIDCCARDDATA        *lppCardData;
} WFSIDCPARSEDATA, *LPWFSIDCPARSEDATA;

typedef struct _wfs_idc_set_guidlight
{
    WORD                    wGuidLight;
    DWORD                   dwCommand;
} WFSIDCSETGUIDLIGHT, *LPWFSIDCSETGUIDLIGHT;

typedef struct _wfs_idc_eject_card
{
    WORD                    wEjectPosition;
} WFSIDCEJECTCARD, *LPWFSIDCEJECTCARD;

typedef struct _wfs_idc_power_save_control
{
    USHORT                  usMaxPowerSaveRecoveryTime;
} WFSIDCPOWERSAVECONTROL, *LPWFSIDCPOWERSAVECONTROL;

/*=================================================================*/
/* IDC Message Structures */
/*=================================================================*/

typedef struct _wfs_idc_track_event
{
    WORD                    fwStatus;
    LPSTR                   lpstrTrack;
    LPSTR                   lpstrData;
} WFSIDCTRACKEVENT, *LPWFSIDCTRACKEVENT;

typedef struct _wfs_idc_card_act
{
    WORD                    wAction;
    WORD                    wPosition;
} WFSIDCCARDACT, *LPWFSIDCCARDACT;

typedef struct _wfs_idc_device_position
{
    WORD                    wPosition;
} WFSIDCDEVICEPOSITION, *LPWFSIDCDEVICEPOSITION;
```

```
typedef struct _wfs_idc_power_save_change
{
    USHORT                  usPowerSaveRecoveryTime;
} WFSIDCPOWERSAVECHANGE, *LPWFSIDCPOWERSAVECHANGE;

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

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

#endif  /* __INC_XFSIDC__H */
```