CEN

WORKSHOP

AGREEMENT

CWA 14050-26

October 2003

ICS 35.200; 35.240.15

English version

Extensions for Financial Services (XFS) interface specification - Release 3.02 - Part 26: Identification Card Unit Device Class Interface - Migration from Version 3.00 to Version 3.02 - 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 14050-26:2003 D/E/F

Table of Contents

Foreword

This CWA is revision 3.02 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 2003-05-21. 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 3.02.

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 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 Class Interface - Programmer's Reference

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

Part 16: Application Programming Interface (API) - Service Provider Interface (SPI) - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 17: Printer Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 18: Identification Card Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (see CWA 14050-4:2000; superseded) - Programmer's Reference

Part 19: Cash Dispenser Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 20: PIN Keypad Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (see CWA 14050-6:2000; superseded) - Programmer's Reference

Part 21: Depository Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 22: Text Terminal Unit Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 23: Sensors and Indicators Unit Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.01 (this CWA) - Programmer's Reference

Part 24: Camera Device Class Interface - Migration from Version 2.00 (see CWA 13449) to Version 3.00 (this CWA) - Programmer's Reference

Part 25: Identification Card Device Class Interface - PC/SC Integration Guidelines

Part 26: Identification Card Device Class Interface - Migration from Version 3.00 (see CWA 14050-4:2000;

superseded) to Version 3.02 (this CWA) - Programmer's Reference

Part 27: PIN Keypad Device Class Interface - Migration from Version 3.00 (see CWA 14050-6:2000; superseded) to Version 3.02 (this CWA) - Programmer's Reference

Part 28: Cash In Module Device Class Interface - Migration from Version 3.00 (see CWA 14050-15:2000; superseded) to Version 3.02 (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.cenorm.be/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.

1. Introduction

1.1. Background to Release 3.02

The CEN XFS Workshop is a continuation of the Banking Solution Vendors Council workshop and maintains a technical commitment to the Win 32 API. However, the XFS Workshop has extended the franchise of multi vendor software by encouraging the participation of both banks and vendors to take part in the deliberations of the creation of an industry standard. This move towards opening the participation beyond the BSVC's original membership has been very successful with a current membership level of more than 20 companies.

The fundamental aims of the XFS Workshop are to promote a clear and unambiguous specification for both service providers and application developers. This has been achieved to date by sub groups working electronically and quarterly meetings.

The move from an XFS 3.00 specification to a 3.02 specification has been prompted by a series of factors. There has been pressure from the market to fully support Smart/DIP card readers and card readers where there are chip cards which are permanently connected.

The clear direction of the XFS Workshop, therefore, is the delivery of a new Release 3.02 specification based on a C API. It will be delivered with the promise of the protection of technical investment for existing applications and the design to safeguard future developments. All XFS 3.00 IDC clarifications still apply to this document.

1.2. 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
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.

2. Changes to Existing Info Commands

2.1. WFS_INF_IDC_STATUS

...

Output Param `WFSIDCSTATUS lpStatus;`

```
typedef struct _wfs_idc_status
   {
   WORD          fwDevice;
   WORD          fwMedia;
   WORD          fwRetainBin;
   WORD          fwSecurity;
   USHORT        usCards;
   WORD          fwChipPower;
   LPSTR         lpszExtra;
   } 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.

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

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 flags:

Value	Meaning
WFS_IDC_RETAINBINOK	The retain bin of the ID card unit is not full.
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_RETAINNOTSUPP	The ID card unit does not support retain capability.

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

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

usCards
The number of cards retained; applicable only to motor driven ID card units for non-motorized card units this value is 0. 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
Points 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.

2.2. WFS_INF_IDC_CAPABILITIES

...

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;
   } WFSIDCCAPS, * LPWFSIDCCAPS;
```

wClass
Specifies the logical service class; value is WFS_SERVICE_CLASS_IDC

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

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). 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 and is either TRUE or FALSE.

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.

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_CHIPT2	The ID card unit can handle the T=2 protocol.
WFS_IDC_CHIPT3	The ID card unit can handle the T=3 protocol.
WFS_IDC_CHIPT4	The ID card unit can handle the T=4 protocol.
WFS_IDC_CHIPT5	The ID card unit can handle the T=5 protocol.
WFS_IDC_CHIPT6	The ID card unit can handle the T=6 protocol.
WFS_IDC_CHIPT7	The ID card unit can handle the T=7 protocol.
WFS_IDC_CHIPT8	The ID card unit can handle the T=8 protocol.
WFS_IDC_CHIPT9	The ID card unit can handle the T=9 protocol.
WFS_IDC_CHIPT10	The ID card unit can handle the T=10 protocol.
WFS_IDC_CHIPT11	The ID card unit can handle the T=11 protocol.
WFS_IDC_CHIPT12	The ID card unit can handle the T=12 protocol.
WFS_IDC_CHIPT13	The ID card unit can handle the T=13 protocol.
WFS_IDC_CHIPT14	The ID card unit can handle the T=14 protocol.
WFS_IDC_CHIPT15	The ID card unit can handle the T=15 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 flags:

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 flags; 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. This value is either TRUE or FALSE. It is only TRUE if the capabilities of the device are not affected by one of these sequences of commands.

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
Points 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.

2.3. 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 card is driven to the exit slot from where the user can remove it. 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.

2.4. 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.

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. In this situation the error WFS_ERR_IDC_SECURITYFAIL will be returned if the command specifies only security data to be read, in all other cases WFS_SUCCESS will be returned with the lpbData field of the output parameter set to WFS_IDC_SEC_HWERROR.

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
Specifies which data 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_TRACK_WM	The Swedish Watermark track 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.

Output Param

```
LPWFSIDCCARDDATA    *lppCardData;
```

lppCardData
Pointer to a null-terminated array of pointers to card data structures:

```
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.

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 is blank.
WFS_IDC_DATAINVALID	The data contained on the track/chip is invalid.
WFS_IDC_DATATOOLONG	The data contained on the track/chip is too long.
WFS_IDC_DATATOOSHORT	The data contained on the track/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.

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

lpbData

Points to the data read from the track/chip or the value returned by the security module. 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).

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.

2.5. 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 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.

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 was not obtained before by issuing a Read Command.

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

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

2.6. 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 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 card found within the IDC 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 this value 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 it's shutter

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.

Comments None

2.7. WFS_CMD_IDC_CHIP_POWER

Description This command handles the power actions that can be done on the chip. This command is only used for user chips after the chip has been contacted for the first time using the

WFS_CMD_IDC_READ_RAW_DATA command. This command is the only way to control the chip power for permanently connected chip cards.

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;`

```
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.

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.

3. Changes to existing Events

3.1. 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, after an eject operation, or after the card is removed by the user in a latched DIP card unit.

Event Param None.

4. Changes to C-Header file

```
/******************************************************************************
*                                                                             *
* xfsidc.h    XFS - Identification card reader UNIT (IDC) definitions          *
*                                                                             *
*             Version 3.02  (09/05/03)                                        *
*                                                                             *
******************************************************************************/

#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       0x0203

#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)

/* 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)


/* 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_EXEE_IDC_MEDIADETECTED          (IDC_SERVICE_OFFSET + 9)

/* 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
```

```
/* 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)

/* 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)


/* 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 */

#define     WFS_IDC_NOTSUPP                     0x0000
#define     WFS_IDC_TRACK1                      0x0001
#define     WFS_IDC_TRACK2                      0x0002
#define     WFS_IDC_TRACK3                      0x0004

/* further values of WFSIDCCARDDATA.wDataSource */

#define     WFS_IDC_CHIP                        0x0008
#define     WFS_IDC_SECURITY                    0x0010
#define     WFS_IDC_FLUXINACTIVE                0x0020
#define     WFS_IDC_TRACK_WM                    0x8000

/* values of WFSIDCCAPS.fwChipProtocols */

#define     WFS_IDC_CHIPT0                      0x0001
#define     WFS_IDC_CHIPT1                      0x0002
#define     WFS_IDC_CHIPT2                      0x0004
#define     WFS_IDC_CHIPT3                      0x0008
#define     WFS_IDC_CHIPT4                      0x0010
#define     WFS_IDC_CHIPT5                      0x0020
#define     WFS_IDC_CHIPT6                      0x0040
#define     WFS_IDC_CHIPT7                      0x0080
#define     WFS_IDC_CHIPT8                      0x0100
#define     WFS_IDC_CHIPT9                      0x0200
#define     WFS_IDC_CHIPT10                     0x0400
```

```
#define     WFS_IDC_CHIPT11                     0x0800
#define     WFS_IDC_CHIPT12                     0x1000
#define     WFS_IDC_CHIPT13                     0x2000
#define     WFS_IDC_CHIPT14                     0x4000
#define     WFS_IDC_CHIPT15                     0x8000

/* 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 */

#define     WFS_IDC_UNKNOWN                     0x0001
#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 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)


/* 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'

/* 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))

/*=================================================================*/
/* 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;
} 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;
} 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;

/*=================================================================*/
/* 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;


/*=================================================================*/
/* 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;


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

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

#endif  /* __INC_XFSIDC__H */
```