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

WORKSHOP AGREEMENT

CWA 14050-18

November 2000

ICS 35.200; 35.240.15

Extensions for Financial Services (XFS) interface specification - Release 3.0 - Part 18: Identification Card Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

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

Ref. No CWA 14050-18:2000 E

Rue de Stassart, 36 • B-1050 Bruxelles
Tel : +32 2 550 08 11 • Fax : +32 2 550 08 19

Table of Contents

Foreword

This CWA is revision 3.0 of the XFS interface specification.

The move from an XFS 2.0 specification (CWA 13449) to a 3.0 specification has been prompted by a series of factors.

Initially, there has been a technical imperative to extend the scope of the existing specification of the XFS Manager to include new devices, such as the Card Embossing Unit.

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

Finally, it is also clear that our customers and the market are asking for an update to a specification, which is now over 2 years old. Increasing market acceptance and the need to meet this demand is driving the Workshop towards this release.

The clear direction of the CEN/ISSS XFS Workshop, therefore, is the delivery of a new Release 3.0 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.

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 2000-10-18. 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 3.0.

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 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 17: Printer Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 18: Identification Card Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 19: Cash Dispenser Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 20: PIN Keypad Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 21: Depository Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 22: Text Terminal Unit Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 23: Sensors and Indicators Unit Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Part 24: Camera Device Class Interface - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

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

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. New Chapters

1.1. Relation with PC/SC

The PC/SC (Personal Computer / Smart Card) Workgroup was formed in May 1996 in partnership with major PC and smart card companies. The main focus of the workgroup has been to develop specifications that ensure interoperability among smart cards, smart card readers, and computers made by different manufacturers:

Interoperability Specification for Integrated Circuit Cards (ICC) and Personal Computer Systems

Version 1.0 of these specifications were released in December 1997. There are available on the Web at:
http://www.pcscworkgroup.com

The related document *PC/SC Integration Guidelines* describes the relation between XFS and PC/SC and provides guidelines to manage PC/SC compliant readers from the XFS subsystem.

In order to make integration of PC/SC compliant smart cards easier, the following principles have been defined to add new chip capabilities to the IDC Device Class Interface:

- A new set of chip capabilities is made of new queries and commands which should be consistent.
- An associated COM-based interface definition reflects these new queries and commands.
- This COM-based interface definition and its associated GUID are published part of this specification, to allow its implementation in PC/SC ICC service providers.

These principles allow the IDC service provider for a PC/SC compliant reader to be a wrapper for ICC commands, which are handled in the PC/SC subsystem by the corresponding PC/SC ICC service provider.

The following international standard was also taken into account in the IDC 3.0 document :-
Watermark (Sweden)

1.2. References

1. XFS Application Programming Interface (API)/Service Provider Interface (SPI), Programmer's Reference Revision 3.0, October 18, 2000

2. New Info Commands

There are no new Info Commands.

3. Changes to Existing Info Commands

3.1. *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.
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.

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 on the currently inserted card in the device as 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.

3.2. 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.
WFS_IDC_TYPECONTACTLESS	The ID card unit is a contactless card unit, i.e. no insertion of the card is required.

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, 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.

4. New Execute Commands

4.1. 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.

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 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 card will not be moved even if this means that the IDC cannot be recovered.

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

4.2. WFS_CMD_IDC_CHIP_POWER

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

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 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_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 None.

4.3. 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 behaviour for the reading of tracks (see Section 6, 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 6, Form Definition.

Example of *lpstrTrackData*:

```
TRACK2:ALL=47..\0\0TRACK3:MII=59\0PAN=500..\0\0\0
```

5. Changes to existing 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). In addition to that, the results of a security check via a security module (i.e., MM, CIM86) are specified and added to the track data.

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 form specifies only security data to be read, in all other cases WFS_SUCCESS will be returned with the security field of the output parameter set to WFS_IDC_SEC_HWERROR.

. . .

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 execute event (WFS_EXEE_IDC_INVALIDTRACKDATA) 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 get in card again
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.

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.

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;
```

```
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.

. . .

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	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 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.

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 6, 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 The card is driven to the exit slot from where the user can remove it; applicable only to motor driven card readers. 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.

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.

~~If the execution of this command is performed without errors, the total number of cards retained includes the current card. If, however, an error occurs during the execution, the total number of cards retained does not include the current card.~~

. . .

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.

5.5. WFS_CMD_IDC_SETKEY

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.

5.6. 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.

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.

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.

5.7. 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 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:

```
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 to be written to track 1.
WFS_IDC_TRACK2	*lpbData* contains data to be written to track 2.
WFS_IDC_TRACK3	*lpbData* contains data to be written to track 3.

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.

. . .

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	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.

5.8. WFS_CMD_IDC_CHIP_IO

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 ~~write~~ operation.

6. New Events

6.1. WFS_EXEE_IDC_MEDIARETAINED

Description This service event specifies that the card was retained.

Event Param None.

6.2. WFS_EXEE_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;`

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.

7. Changes to existing Events

7.1. 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.

8. Changes to Form Description Section

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).

. . .

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

9. Changes to C-Header file

```
/******************************************************************************
*                                                                             *
* xfsidc.h    XFS - Identification card reader UNIT (IDC) definitions         *
*                                                                             *
*             Version 3.00  (18/10/00)                                        *
*                                                                             *
******************************************************************************/

#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        0x0003

#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)

/* 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)

/* 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_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 */
```