CEN WORKSHOP AGREEMENT

ICS 35.200; 35.240.15; 35.240.40

English version

Extensions for Financial Services (XFS) interface specification
Release 3.30 - Part 4: Identification Card Device Class Interface
- Programmer's Reference

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

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

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

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

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

European foreword ............................................................................................................ 4

1. Introduction .................................................................................................................. 7
   1.1 Background to Release 3.30 .................................................................................... 7
   1.2 XFS Service-Specific Programming ....................................................................... 7

2. Identification Card Readers and Writers .................................................................... 9
   2.1 Support for EMV Intelligent Contactless Card Readers ....................................... 10

3. References .................................................................................................................... 11

4. Info Commands .......................................................................................................... 12
   4.1 WFS_INF_IDC_STATUS .......................................................................................... 12
   4.2 WFS_INF_IDC_CAPABILITIES ............................................................................. 18
   4.3 WFS_INF_IDC_FORM_LIST .................................................................................. 23
   4.4 WFS_INF_IDC_QUERY_FORM .............................................................................. 24
   4.5 WFS_INF_IDC_QUERY_IFM_IDENTIFIER ................................................................ 26
   4.6 WFS_INF_IDC_EMVCLESS_QUERY_APPLICATIONS ............................................. 27

5. Execute Commands ..................................................................................................... 28
   5.1 WFS_CMD_IDC_READ_TRACK .............................................................................. 28
   5.2 WFS_CMD_IDC_WRITE_TRACK ............................................................................ 30
   5.3 WFS_CMD_IDC_EJECT_CARD ............................................................................. 32
   5.4 WFS_CMD_IDC_RETAIN_CARD .......................................................................... 34
   5.5 WFS_CMD_IDC_RESET_COUNT .......................................................................... 35
   5.6 WFS_CMD_IDC_SETKEY .................................................................................... 36
   5.7 WFS_CMD_IDC_READ_RAW_DATA ....................................................................... 37
   5.8 WFS_CMD_IDC_WRITE_RAW_DATA ..................................................................... 41
   5.9 WFS_CMD_IDC_CHIP_IO .................................................................................... 43
   5.10 WFS_CMD_IDC_RESET ..................................................................................... 45
   5.11 WFS_CMD_IDC_CHIP_POWER .......................................................................... 46
   5.12 WFS_CMD_IDC_PARSE_DATA .......................................................................... 47
   5.13 WFS_CMD_IDC_SET_GUIDANCE_LIGHT ............................................................ 48
   5.14 WFS_CMD_IDC_POWER_SAVE_CONTROL ......................................................... 50
   5.15 WFS_CMD_IDC_PARK_CARD ............................................................................ 51
   5.16 WFS_CMD_IDC_EMVCLESS_CONFIGURE .......................................................... 52
   5.17 WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION ...................................... 54
   5.18 WFS_CMD_IDC_EMVCLESS_ISSUERUPDATE .................................................... 59
   5.19 WFS_CMD_IDC_SYNCHRONIZE_COMMAND ..................................................... 61

6. Events .......................................................................................................................... 62
   6.1 WFS_EXEE_IDC_INVALIDTRACKDATA ................................................................ 62
6.2 WFS_EXEE_IDC_MEDIAINSERTED ................................................................. 63
6.3 WFS_SRVE_IDC_MEDIAREMOVED ............................................................... 64
6.4 WFS_EXEE_IDC_MEDIARETAINED ............................................................... 65
6.5 WFS_EXEE_IDC_INVALIDMEDIA ........................................................................ 66
6.6 WFS_SRVE_IDC_CARDACTION ................................................................. 67
6.7 WFS_USRE_IDC_RETAINBINTHRESHOLD ...................................................... 68
6.8 WFS_SRVE_IDC_MEDIADETECTED .............................................................. 69
6.9 WFS_SRVE_IDC_RETAINBINREMOVED ......................................................... 70
6.10 WFS_SRVE_IDC_RETAINBININSERTED ......................................................... 71
6.11 WFS_EXEE_IDC_INSERTCARD ................................................................. 72
6.12 WFS_SRVE_IDC_DEVICEPOSITION ............................................................ 73
6.13 WFS_SRVE_IDC_POWER_SAVE_CHANGE .................................................... 74
6.14 WFS_EXEE_IDC_TRACKDETECTED ............................................................. 75
6.15 WFS_EXEE_IDC_EMVCLESSREADSTATUS ............................................... 76

7. Form Description ........................................................................................... 77

8. C-Header file .................................................................................................. 80

9. Intelligent Contactless Card Sequence Diagrams ......................................... 91
   9.1 Single Tap Transaction Without Issuer Update Processing ......................... 92
   9.2 Double Tap Transaction With Issuer Update Processing ............................. 93
This CWA is revision 3.30 of the XFS interface specification.

This CEN Workshop Agreement has been drafted and approved by a Workshop of representatives of interested parties on March 19th 2015, the constitution of which was supported by CEN following the public call for participation made on 1998-06-24. 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 3.30.

A list of the individuals and organizations which supported the technical consensus represented by the CEN Workshop Agreement is available from the CEN/XFS Secretariat. The CEN XFS Workshop gathered suppliers as well as banks and other financial service companies.

The CWA is published as a multi-part document, consisting of:
Part 1: Application Programming Interface (API) - Service Provider Interface (SPI) - Programmer's Reference
Part 2: Service Classes Definition - Programmer's Reference
Part 3: Printer and Scanning Device Class Interface - Programmer's Reference
Part 4: Identification Card Device Class Interface - Programmer's Reference
Part 5: Cash Dispenser Device Class Interface - Programmer's Reference
Part 6: PIN Keypad Device Class Interface - Programmer's Reference
Part 7: Check Reader/Scanner Device Class Interface - Programmer's Reference
Part 8: Depository Device Class Interface - Programmer's Reference
Part 9: Text Terminal Unit Device Class Interface - Programmer's Reference
Part 10: Sensors and Indicators Unit Device Class Interface - Programmer's Reference
Part 11: Vendor Dependent Mode Device Class Interface - Programmer's Reference
Part 12: Camera Device Class Interface - Programmer's Reference
Part 13: Alarm Device Class Interface - Programmer's Reference
Part 14: Card Embossing Unit Device Class Interface - Programmer's Reference
Part 15: Cash-In Module Device Class Interface - Programmer's Reference
Part 16: Card Dispenser Device Class Interface - Programmer's Reference
Part 17: Barcode Reader Device Class Interface - Programmer's Reference
Part 18: Item Processing Module Device Class Interface - Programmer's Reference
Parts 19 - 28: Reserved for future use.

Parts 29 through 47 constitute an optional addendum to this CWA. They define the integration between the SNMP standard and the set of status and statistical information exported by the Service Providers.
Part 29: XFS MIB Architecture and SNMP Extensions - Programmer’s Reference
Part 30: XFS MIB Device Specific Definitions - Printer Device Class
Part 31: XFS MIB Device Specific Definitions - Identification Card Device Class
Part 32: XFS MIB Device Specific Definitions - Cash Dispenser Device Class
Part 33: XFS MIB Device Specific Definitions - PIN Keypad Device Class
Part 34: XFS MIB Device Specific Definitions - Check Reader/Scanner Device Class
Part 35: XFS MIB Device Specific Definitions - Depository Device Class
Part 36: XFS MIB Device Specific Definitions - Text Terminal Unit Device Class
Part 37: XFS MIB Device Specific Definitions - Sensors and Indicators Unit Device Class
Part 38: XFS MIB Device Specific Definitions - Camera Device Class
Part 39: XFS MIB Device Specific Definitions - Alarm Device Class
Part 40: XFS MIB Device Specific Definitions - Card Embossing Unit Class
Part 41: XFS MIB Device Specific Definitions - Cash-In Module Device Class
Part 42: Reserved for future use.
Part 43: XFS MIB Device Specific Definitions - Vendor Dependent Mode Device Class
Part 44: XFS MIB Application Management
Part 45: XFS MIB Device Specific Definitions - Card Dispenser Device Class
Part 46: XFS MIB Device Specific Definitions - Barcode Reader Device Class
Part 47: XFS MIB Device Specific Definitions - Item Processing Module Device Class
Parts 48 - 60 are reserved for future use.
Part 61: Application Programming Interface (API) - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Service Provider Interface (SPI) - Programmer's Reference
Part 62: Printer and Scanning Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 63: Identification Card Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 64: Cash Dispenser Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 65: PIN Keypad Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 66: Check Reader/Scanner Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 67: Depository Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 68: Text Terminal Unit Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 69: Sensors and Indicators Unit Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 70: Vendor Dependent Mode Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 71: Camera Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 72: Alarm Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 73: Card Embossing Unit Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 74: Cash-In Module Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 75: Card Dispenser Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 76: Barcode Reader Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference
Part 77: Item Processing Module Device Class Interface - Migration from Version 3.20 (CWA 16374) to Version 3.30 (this CWA) - Programmer's Reference

In addition to these Programmer's Reference specifications, the reader of this CWA is also referred to a complementary document, called Release Notes. The Release Notes contain clarifications and explanations on the CWA specifications, which are not requiring functional changes. The current version of the Release Notes is available online from http://www.cen.eu/work/areas/ict/ebusiness/pages/ws-xfs.aspx.
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 makes no warranty, express or implied, with respect to this document.

The formal process followed by the Workshop in the development of the CEN Workshop Agreement has been endorsed by the National Members of CEN but neither the National Members of CEN nor the CEN-CENELEC Management Centre can be held accountable for the technical content of the CEN Workshop Agreement or possible conflict with standards or legislation. This CEN Workshop Agreement can in no way be held as being an official standard developed by CEN and its members.

The final review/endorsement round for this CWA was started on 2015-01-16 and was successfully closed on 2015-03-19. The final text of this CWA was submitted to CEN for publication on 2015-06-19. The specification is continuously reviewed and commented in the CEN 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 3.30.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

According to the CEN-CENELEC Internal Regulations, the national standards organizations of the following countries are bound to implement this European Standard: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

Comments or suggestions from the users of the CEN Workshop Agreement are welcome and should be addressed to the CEN-CENELEC Management Centre.

Revision History:

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.00</td>
<td>October 18, 2000</td>
<td>Initial release.</td>
</tr>
<tr>
<td>3.02</td>
<td>May 21, 2003</td>
<td>For a description of changes from version 3.00 to version 3.02 see the IDC 3.02 Migration document.</td>
</tr>
<tr>
<td>3.10</td>
<td>November 29, 2007</td>
<td>For a description of changes from version 3.00 to version 3.10 see the IDC 3.10 Migration document.</td>
</tr>
<tr>
<td>3.20</td>
<td>March 2, 2011</td>
<td>For a description of changes from version 3.10 to version 3.20 see the IDC 3.20 Migration document.</td>
</tr>
<tr>
<td>3.30</td>
<td>March 19, 2015</td>
<td>For a description of changes from version 3.20 to version 3.30 see the IDC 3.30 Migration document.</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Background to Release 3.30

The CEN/XFS Workshop aims to promote a clear and unambiguous specification defining a multi-vendor software interface to financial peripheral devices. The XFS (eXtensions for Financial Services) specifications are developed within the CEN (European Committee for Standardization/Information Society Standardization System) Workshop environment. CEN Workshops aim to arrive at a European consensus on an issue that can be published as a CEN Workshop Agreement (CWA).

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

Release 3.30 of the XFS specification is based on a C API and is delivered with the continued promise for the protection of technical investment for existing applications. This release of the specification extends the functionality and capabilities of the existing devices covered by the specification, but it does not include any new device classes. Notable enhancements include:

- Enhanced reporting of Shutter Jammed Status and a new Shutter Status event for CDM, CIM and IPM.
- Addition of a Synchronize command for all device classes, in order to allow synchronized action where necessary.
- Directional Guidance Light support.
- Addition of a CIM Deplete Command.
- Support for EMV Intelligent Contactless Readers.
- Support in PIN for Encrypting Touch Screen.
- PIN Authentication functionality.
- PIN TR34 standard supported.

1.2 XFS Service-Specific Programming

The service classes are defined by their service-specific commands and the associated data structures, error codes, messages, etc. These commands are used to request functions that are specific to one or more classes of Service Providers, but not all of them, and therefore are not included in the common API for basic or administration functions.

When a service-specific command is common among two or more classes of Service Providers, the syntax of the command is as similar as possible across all services, since a major objective of XFS is to standardize function codes and structures for the broadest variety of services. For example, using the `WFSEexecute` function, the commands to read data from various services are as similar as possible to each other in their syntax and data structures.

In general, the specific command set for a service class is defined as a superset of the specific capabilities likely to be provided by the developers of the services of that class; thus any particular device will normally support only a subset of the defined command set.

There are three cases in which a Service Provider may receive a service-specific command that it does not support:

The requested capability is defined for the class of Service Providers by the XFS specification, the particular vendor implementation of that service does not support it, and the unsupported capability is **not** considered to be fundamental to the service. In this case, the Service Provider returns a successful completion, but does no operation. An example would be a request from an application to turn on a control indicator on a passbook printer; the Service Provider recognizes the command, but since the passbook printer it is managing does not include that indicator, the Service Provider does no operation and returns a successful completion to the application.

The requested capability is defined for the class of Service Providers by the XFS specification, the particular vendor
implementation of that service does not support it, and the unsupported capability is considered to be fundamental to the service. In this case, a WFS_ERR_UNSUPP_COMMAND error is returned to the calling application. An example would be a request from an application to a cash dispenser to dispense coins; the Service Provider recognizes the command but, since the cash dispenser it is managing dispenses only notes, returns this error.

The requested capability is not defined for the class of Service Providers by the XFS specification. In this case, a WFS_ERR_INVALID_COMMAND error is returned to the calling application.

This design allows implementation of applications that can be used with a range of services that provide differing subsets of the functionalities that are defined for their service class. Applications may use the WFSGetInfo and WFSAsyncGetInfo commands to inquire about the capabilities of the service they are about to use, and modify their behavior accordingly, or they may use functions and then deal with WFS_ERR_UNSUPP_COMMAND error returns to make decisions as to how to use the service.
2. Identification Card Readers and Writers

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

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

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

Some motor driven card reader/writers have parking stations inside and can place identification cards there. Once a card is in its parking station another card can be accepted by the card reader. Cards may only be moved out of a parking station if there is no other card present in the media read/write position, the chip I/O position, the transport, or the entry/exit slot.

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
- Cash Transfer Card Track 1 (JIS I: 8 bits/char) Japan
- Cash Transfer Card Track 3 (JIS I: 8 bits/char) Japan
- Front Track 1 (JIS II) Japan
- Watermark  Sweden
- Chip (contacted)  ISO 7816
- Chip (contactless)  ISO 10536, ISO 14443 and ISO 18092

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_CHIP_IO and WFS_CMD_IDC_RESET.

The following defines the roles and responsibilities of an application within EMV: A distinction needs to be made between EMV Contact support and EMV Contactless support.

When defining an EMV Contact implementation

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

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

EMVCo grants EMV Level 1 Approvals to contact IFMs and EMVCo Level 2 Approvals to Application Kernels.

When defining an EMV Contactless implementation

The responsibilities will depend on the type of EMV Contactless Product being implemented.
There are different EMVCo defined product types, they can be found in the EMVCo Type Approval – Contactless Product – Administrative Process document.

- In this specification when referring to the Contactless Product Type – Intelligent Card Reader:

The following must be included and handled below the XFS API:

- An EMVCo Approved Level 1 Contactless PCD
- Entry Point and POS System Architecture according to Book A and B
- EMV Kernels according to Book C1 to C7 (minimum one kernel needs to be supported)

The Network Interface & the Consumer, Merchant Interfaces will be managed above the XFS API.

2.1 Support for EMV Intelligent Contactless Card Readers

In relation to contactless transactions, the terminology used in this document is based on the EMV Contactless Specifications for Payment Systems, see the References section.

There are a number of types of payment systems (or EMV) compliant contactless card readers, from the intelligent reader device; where the reader device handles most of the transaction processing and only returns the result, to a transparent card reader; where the contactless card reader device provides a generic communication channel to the card without having any in-built transaction processing capabilities.

A contactless payment system transaction can be performed in two different ways, magnetic stripe emulation; where the data returned from the chip is formatted as if it was read from the magnetic stripe, and EMV-like; where, in a similar way to a contact EMV transaction, the chip returns a full set of BER-TLV (Basic Encoding Rules-Tag Length Value) data. Each payment system defines when each type, or profile, is used for a transaction, but it is usually dependent on both the configuration of the terminal and contactless card being tapped.

This document will use “magnetic stripe emulation” and “EMV-like” to identify the two profiles of contactless transactions.

Support for a generic contactless communication channel to the card is provided via the WFS_CMD_IDC_CHIP_IO command. This is suitable for use with a transparent contactless card reader or with an intelligent contactless card reader device operating in a pass through mode.

The WFS_CMD_IDC_READ_RAW_DATA command can be used with an intelligent contactless card reader device to provide magnetic track emulation transactions. Only magnetic track emulation transactions can be supported using this command.

When using an intelligent contactless card reader to support both EMV-like and magnetic track emulation transactions a number of commands are required. The WFS_CMD_IDC_EMVCLESS_CONFIGURE command allows the exchange of data to configure the reader for card acceptance and the WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION command enables the reader and performs the transaction with the card when it is tapped. In most cases all the transaction steps involving the card are completed within the initial card tap. Section 9, Appendix provides a sequence diagram showing the expected IDC command sequences, as well as the cardholder and application actions when performing a contactless card based transaction.

Some contactless payment systems allow a 2nd tap of the contactless card. For example a 2nd tap can be used to process authorization data received from the host. In the case of issuer update data this second tap is performed via the WFS_CMD_IDC_EMVCLESS_ISSUERUPDATE command. Section 9, Appendix provides a sequence diagram showing the expected IDC command sequences, as well as the cardholder and application actions. The WFS_INF_IDC_EMVCLESS_QUERY_APPLICATIONS and WFS_CMD_IDC_EMVCLESS_CONFIGURE commands specified later in this document refer to the EMV terminology “Application Identifier (AID) - Kernel Combinations”. A detailed explanation can be found in Reference [2] and Reference [3] documents.

This document refers to BER-TLV tags. These are defined by each individual payment systems and contain the data exchanged between the application, contactless card and an intelligent contactless card reader. They are used to configure and prepare the intelligent contactless card reader for a transaction and are also part of the data that is returned by the reader on completion of the cards tap.

Based on the applicable payment system the application is expected to know which tags are required to be configured, what values to use for the tags and how to interpret the tags returned. Intelligent readers are expected to know the BER-TLV tag definitions supported per payment system application. The tags provided in this document are examples of the types of tags applicable to each command. They are not intended to be a definite list.
3. References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. XFS Application Programming Interface (API)/Service Provider Interface (SPI), Programmer’s Reference Revision 3.30.</td>
<td></td>
</tr>
<tr>
<td>2. EMVCo Integrated Circuit Card Specifications for Payment Systems Version 4.3</td>
<td></td>
</tr>
<tr>
<td>3. EMVCo Contactless Specifications for Payment Systems, Version 2.4</td>
<td></td>
</tr>
<tr>
<td>4. EMVCo Contactless Type Approval Administrative Process Version 2.4</td>
<td></td>
</tr>
</tbody>
</table>
4. Info Commands

4.1 WFS_INF_IDC_STATUS

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

Input Param
None.

Output Param
LPWFSIDCSTATUS lpStatus;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_DEVONLINE</td>
<td>The device is present, powered on and online (i.e. operational, not busy processing a request and not in an error state).</td>
</tr>
<tr>
<td>WFS_IDC_DEVOFFLINE</td>
<td>The device is offline (e.g. the operator has taken the device offline by turning a switch).</td>
</tr>
<tr>
<td>WFS_IDC_DEVPOWEROFF</td>
<td>The device is powered off or physically not connected.</td>
</tr>
<tr>
<td>WFS_IDC_NODEVICE</td>
<td>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.</td>
</tr>
<tr>
<td>WFS_IDC_DEVHWERROR</td>
<td>The device is present but inoperable due to a hardware fault that prevents it from being used.</td>
</tr>
<tr>
<td>WFS_IDC_DEVUSERERROR</td>
<td>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).</td>
</tr>
<tr>
<td>WFS_IDC_DEVBUSY</td>
<td>The device is busy and unable to process an Execute command at this time.</td>
</tr>
</tbody>
</table>
WFS_IDC_DEVFRAUDATTEMPT The device is present but is inoperable because it has detected a fraud attempt.

WFS_IDC_DEVPOTENTIALFRAUD The device has detected a potential fraud attempt and is capable of remaining in service. In this case the application should make the decision as to whether to take the device offline.

\textit{fwMedia}

Specifies the state of the ID card unit as one of the following values. This status is independent of any media in the parking stations.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_MEDIAPRESENT</td>
<td>Media is present in the device, not in the entering position and not jammed. A card in a parking station is not considered to be present. On the latched dip device, this indicates that the card is present in the device and the card is unlatched.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIANOTPRESENT</td>
<td>Media is not present in the device and not at the entering position.</td>
</tr>
<tr>
<td>WFS_IDC.MEDIAJAMMED</td>
<td>Media is jammed in the device; operator intervention is required.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIANOTSUPP</td>
<td>Capability to report media position is not supported by the device (e.g. a typical swipe reader or contactless chip card reader).</td>
</tr>
<tr>
<td>WFS_IDC.MEDIAUNKNOWN</td>
<td>The media state cannot be determined with the device in its current state (e.g. the value of \textit{fwDevice} is WFS_IDC_DEVNODEVICE, WFS_IDC_DEVPOWEROFF, WFS_IDC_DEVOFFLINE, or WFS_IDC_DEVHWERROR).</td>
</tr>
<tr>
<td>WFS_IDC.MEDIAENTERING</td>
<td>Media is at the entry/exit slot of a motorized device.</td>
</tr>
<tr>
<td>WFS_IDC.MEDIALATCHED</td>
<td>Media is present &amp; latched in a latched dip card unit. This means the card can be used for chip card dialog.</td>
</tr>
</tbody>
</table>

\textit{fwRetainBin}

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC.RetainBinOK</td>
<td>The retain bin of the ID card unit is in a good state.</td>
</tr>
<tr>
<td>WFS_IDC.RETAINNOTSUPP</td>
<td>The ID card unit does not support retain capability.</td>
</tr>
<tr>
<td>WFS_IDC.RETAINBINFULL</td>
<td>The retain bin of the ID card unit is full.</td>
</tr>
<tr>
<td>WFS_IDC.RETAINBINHIGH</td>
<td>The retain bin of the ID card unit is nearly full.</td>
</tr>
<tr>
<td>WFS_IDC.RETAINBINMISSING</td>
<td>The retain bin of the ID card unit is missing.</td>
</tr>
</tbody>
</table>

\textit{fwSecurity}

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_SECNOTSUPP</td>
<td>No security module is available.</td>
</tr>
<tr>
<td>WFS_IDC_SECNOTREADY</td>
<td>The security module is not ready to process cards or is inoperable.</td>
</tr>
<tr>
<td>WFS_IDC_SECOPEN</td>
<td>The security module is open and ready to process cards.</td>
</tr>
</tbody>
</table>
usCards
The number of cards retained; applicable only to motor driven ID card units for non-motorized
card units this value is zero. This value is persistent it is reset to zero by the
WFS_CMD_IDC_RESET_COUNT command.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CHIPONLINE</td>
<td>The chip is present, powered on and online (i.e. operational, not busy processing a request and not in an error state)</td>
</tr>
<tr>
<td>WFS_IDC_CHIPPOWEREDOFF</td>
<td>The chip is present, but powered off (i.e. not contacted)</td>
</tr>
<tr>
<td>WFS_IDC_CHIPBUSY</td>
<td>The chip is present, powered on, and busy (unable to process an Execute command at this time)</td>
</tr>
<tr>
<td>WFS_IDC_CHIPNODEVICE</td>
<td>A card is currently present in the device, but has no chip.</td>
</tr>
<tr>
<td>WFS_IDC_CHIPHWERROR</td>
<td>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).</td>
</tr>
<tr>
<td>WFS_IDC_CHIPNOCARD</td>
<td>There is no card in the device.</td>
</tr>
<tr>
<td>WFS_IDC_CHIPNOTSUPP</td>
<td>Capability to report the state of the chip is not supported by the ID card unit device. This value is returned for contactless chip card readers.</td>
</tr>
<tr>
<td>WFS_IDC_CHIPUNKNOWN</td>
<td>The state of the chip cannot be determined with the device in its current state.</td>
</tr>
</tbody>
</table>

lpszExtra
Pointer to a list of vendor-specific, or any other extended, information. The information is
returned as a series of "key=value" strings so that it is easily extensible by Service Providers.
Each string is null-terminated, with the final string terminating with two null characters. An
empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters.

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_GUIDANCE_NOT_AVAILABLE</td>
<td>The status is not available.</td>
<td>A</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_OFF</td>
<td>The light is turned off.</td>
<td>A</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_SLOW_FLASH</td>
<td>The light is blinking slowly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_MEDIUM_FLASH</td>
<td>The light is blinking medium frequency.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_QUICK_FLASH</td>
<td>The light is blinking quickly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_CONTINUOUS</td>
<td>The light is turned on continuous (steady).</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_RED</td>
<td>The light is red.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_GREEN</td>
<td>The light is green.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_YELLOW</td>
<td>The light is yellow.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_BLUE</td>
<td>The light is blue.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_CYAN</td>
<td>The light is cyan.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_MAGENTA</td>
<td>The light is magenta.</td>
<td>C</td>
</tr>
</tbody>
</table>
WFS_IDC_GUIDANCE_WHITE  The light is white.  C
WFS_IDC_GUIDANCE_ENTRY  The light is in the entry state.  D
WFS_IDC_GUIDANCE_EXIT  The light is in the exit state.  D

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CHIPMODOK</td>
<td>The chip card module is in a good state.</td>
</tr>
<tr>
<td>WFS_IDC_CHIPMODINOP</td>
<td>The chip card module is inoperable.</td>
</tr>
<tr>
<td>WFS_IDC_CHIPMODUNKNOWN</td>
<td>The state of the chip card module cannot be determined.</td>
</tr>
<tr>
<td>WFS_IDC_CHIPMODNOTSUPP</td>
<td>Reporting the chip card module status is not supported.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_MAGMODOK</td>
<td>The magnetic card reading module is in a good state.</td>
</tr>
<tr>
<td>WFS_IDC_MAGMODINOP</td>
<td>The magnetic card reading module is inoperable.</td>
</tr>
<tr>
<td>WFS_IDC_MAGMODUNKNOWN</td>
<td>The state of the magnetic reading module cannot be determined.</td>
</tr>
<tr>
<td>WFS_IDC_MAGMODNOTSUPP</td>
<td>Reporting the magnetic reading module status is not supported.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_MAGMODOK</td>
<td>The magnetic card writing module is in a good state.</td>
</tr>
<tr>
<td>WFS_IDC_MAGMODINOP</td>
<td>The magnetic card writing module is inoperable.</td>
</tr>
<tr>
<td>WFS_IDC_MAGMODUNKNOWN</td>
<td>The state of the magnetic card writing module cannot be determined.</td>
</tr>
<tr>
<td>WFS_IDC_MAGMODNOTSUPP</td>
<td>Reporting the magnetic writing module status is not supported.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_IMGMODOK</td>
<td>The front image reading module is in a good state.</td>
</tr>
<tr>
<td>WFS_IDC_IMGMODINOP</td>
<td>The front image reading module is inoperable.</td>
</tr>
<tr>
<td>WFS_IDC_IMGMODUNKNOWN</td>
<td>The state of the front image reading module cannot be determined.</td>
</tr>
<tr>
<td>WFS_IDC_IMGMODNOTSUPP</td>
<td>Reporting the front image reading module status is not supported.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_IMGMODOK</td>
<td>The back image reading module is in a good state.</td>
</tr>
<tr>
<td>WFS_IDC_IMGMODINOP</td>
<td>The back image reading module is inoperable.</td>
</tr>
<tr>
<td>WFS_IDC_IMGMODUNKNOWN</td>
<td>The state of the back image reading module cannot be determined.</td>
</tr>
</tbody>
</table>
WFS_IDC_IMGMODNOTSUPP Reporting the back image reading module status is not supported.

**wDevicePosition**

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

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

**usPowerSaveRecoveryTime**

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

**lpwParkingStationMedia**

Pointer to a zero terminated array of WORDs which contains the states of the parking stations. The lpwParkingStationMedia is NULL if no parking station is supported. The value is specified as one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_MEDIAPRESENT</td>
<td>Media is present in the parking station, and not jammed.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIANOTPRESENT</td>
<td>Media is not present in the parking station.</td>
</tr>
<tr>
<td>WFS_IDC.MEDIAJAMMED</td>
<td>The parking station is jammed; operator intervention is required.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIANOTSUPP</td>
<td>Reporting the media status in a parking station is not supported by the device.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIAUNKNOWN</td>
<td>The media state cannot be determined.</td>
</tr>
</tbody>
</table>

**wAntiFraudModule**

Specifies the state of the anti-fraud module as one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_AFMNOTSUPP</td>
<td>No anti-fraud module is available.</td>
</tr>
<tr>
<td>WFS_IDC_AFMOK</td>
<td>Anti-fraud module is in a good state and no foreign device is detected.</td>
</tr>
<tr>
<td>WFS_IDC_AFMINOP</td>
<td>Anti-fraud module is inoperable.</td>
</tr>
<tr>
<td>WFS_IDC_AFMDEVICEDETECTED</td>
<td>Anti-fraud module detected the presence of a foreign device.</td>
</tr>
<tr>
<td>WFS_IDC_AFMUNKNOWN</td>
<td>The state of the anti-fraud module cannot be determined.</td>
</tr>
</tbody>
</table>

**Error Codes**

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

**Comments**

Applications which require or expect specific information to be present in the lpszExtra parameter may not be device or vendor-independent.

The fwDevice field can indicate that the device is still available (i.e. WFS_IDC_DEVONLINE) even if one of the detailed device status fields (fwSecurity, fwChipModule, fwMagReadModule, fwMagWriteModule or wAntiFraudModule) indicates that there is a problem with one or more modules. In this case, only the functionality provided by modules that do not have a fault should be used.
In the case where communications with the device has been lost, the `fwDevice` field will report `WFS_IDC_DEVPOWEROFF` when the device has been removed or `WFS_IDC_DEVHWERROR` if the communications are unexpectedly lost. All other fields should contain a value based on the following rules and priority:

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

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

Input Param
None.

Output Param
LPWFSIDCCAPS lpCaps;

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

wClass
Specifies the logical service class as WFS_SERVICE_CLASS_IDC.

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

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

The ID card unit is an intelligent contactless card unit, i.e. no insertion of the card is required and the card unit has built-in EMV or smart card application functionality that adheres to the EMVCo Contactless Specifications or individual payment system's specifications. The ID card unit is capable of performing both magnetic stripe emulation and EMV-like transactions.

\textit{bCompound}

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

\textit{fwReadTracks}

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_NOTSUPP</td>
<td>The ID card unit cannot access any track.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK1</td>
<td>The ID card unit can access track 1.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK2</td>
<td>The ID card unit can access track 2.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK3</td>
<td>The ID card unit can access track 3.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK_WM</td>
<td>The ID card unit can access the Swedish Watermark track.</td>
</tr>
<tr>
<td>WFS_IDC_FRONT_TRACK_1</td>
<td>The ID card unit can access the front track 1. In some countries this track is known as JIS II track.</td>
</tr>
<tr>
<td>WFS_IDC_FRONTIMAGE</td>
<td>The ID card unit can read the front image of a card.</td>
</tr>
<tr>
<td>WFS_IDC_BACKIMAGE</td>
<td>The ID card unit can read the back image of a card.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK1_JIS1</td>
<td>The ID card unit can access JIS I track 1.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK3_JIS1</td>
<td>The ID card unit can access JIS I track 3.</td>
</tr>
<tr>
<td>WFS_IDC_DDI</td>
<td>The ID card unit can provide dynamic digital identification of the magnetic strip.</td>
</tr>
</tbody>
</table>

\textit{fwWriteTracks}

Specifies the tracks that can be written by the ID card unit (as a combination of the flags specified in the description of \textit{fwReadTracks} except WFS_IDC_TRACK_WM, WFS_IDC_FRONTIMAGE, WFS_IDC_BACKIMAGE and WFS_IDC_DDI).

\textit{fwChipProtocols}

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_NOTSUPP</td>
<td>The ID card unit cannot handle chip cards.</td>
</tr>
<tr>
<td>WFS_IDC_CHIP0</td>
<td>The ID card unit can handle the T=0 protocol.</td>
</tr>
<tr>
<td>WFS_IDC_CHIP1</td>
<td>The ID card unit can handle the T=1 protocol.</td>
</tr>
<tr>
<td>WFS_IDC_CHIP_PROTOCOL_NOT_REQUIRED</td>
<td>The ID card unit is capable of communicating with a chip card without requiring the application to specify any protocol.</td>
</tr>
<tr>
<td>WFS_IDC_CHIP_TYPEA_PART3</td>
<td>The ID card unit can handle the ISO 14443 (Part3) Type A contactless chip card protocol.</td>
</tr>
<tr>
<td>WFS_IDC_CHIP_TYPEA_PART4</td>
<td>The ID card unit can handle the ISO 14443 (Part4) Type A contactless chip card protocol.</td>
</tr>
<tr>
<td>WFS_IDC_CHIP_TYPEB</td>
<td>The ID card unit can handle the ISO 14443 Type B contactless chip card protocol.</td>
</tr>
</tbody>
</table>
The ID card unit can handle the ISO 18092 (106/212/424kbps) contactless chip card protocol.

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_SECNOTSUPP</td>
<td>Device has no security module.</td>
</tr>
<tr>
<td>WFS_IDC_SECMMBOX</td>
<td>Security module of device is MMBox.</td>
</tr>
<tr>
<td>WFS_IDC_SECCIM86</td>
<td>Security module of device is CIM86.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_NOACTION</td>
<td>No power on actions are supported by the device.</td>
</tr>
<tr>
<td>WFS_IDC_EJECT</td>
<td>The card will be ejected on power-on (or off, see <code>fwPowerOffOption</code> below).</td>
</tr>
<tr>
<td>WFS_IDC_RETAIN</td>
<td>The card will be retained on power-on (off).</td>
</tr>
<tr>
<td>WFS_IDC_EJECTTHENRETAIN</td>
<td>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.</td>
</tr>
<tr>
<td>WFS_IDC_READPOSITION</td>
<td>The card will be moved into the read position on power-on (off).</td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_NOTSUPP</td>
<td>Does not support writing of magnetic stripes.</td>
</tr>
<tr>
<td>WFS_IDC_LOCO</td>
<td>Supports writing of loco magnetic stripes.</td>
</tr>
<tr>
<td>WFS_IDC_HICO</td>
<td>Supports writing of hico magnetic stripes.</td>
</tr>
<tr>
<td>WFS_IDC_AUTO</td>
<td>Service Provider is capable of automatically determining whether loco or hico magnetic stripes should be written.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_NOTSUPP</td>
<td>The ID card unit cannot handle chip power management.</td>
</tr>
<tr>
<td>WFS_IDC_CHIPPOWERCOLD</td>
<td>The ID card unit can power on the chip and reset it (Cold Reset).</td>
</tr>
<tr>
<td>WFS_IDC_CHIPPOWERWARM</td>
<td>The ID card unit can reset the chip (Warm Reset).</td>
</tr>
</tbody>
</table>
WFS_IDC_CHIPPOWEROFF

The ID card unit can power off the chip.

lpszExtra

Pointer to a list of vendor-specific, or any other extended, information. The information is returned as a series of “key=value” strings so that it is easily extensible by Service Providers. Each string is null-terminated, with the final string terminating with two null characters. An empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters.

fwDIPMode

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_NOTSUPP</td>
<td>The ID card unit is not a dip type.</td>
</tr>
<tr>
<td>WFS_IDC_DIP_EXIT</td>
<td>The dip ID card unit reads card track data on exit only.</td>
</tr>
<tr>
<td>WFS_IDC_DIP_ENTRY</td>
<td>The dip ID card unit reads card track data on entry only.</td>
</tr>
<tr>
<td>WFS_IDC_DIP_ENTRY_EXIT</td>
<td>The dip ID card unit reads card track data both on entry and exit.</td>
</tr>
<tr>
<td>WFS_IDC_DIP_UNKNOWN</td>
<td>Unknown whether track data is read on entry or exit.</td>
</tr>
</tbody>
</table>

lpwMemoryChipProtocols

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_MEM_SIEMENS4442</td>
<td>The device supports the Siemens 4442 Card Protocol (also supported by the Gemplus GPM2K card).</td>
</tr>
<tr>
<td>WFS_IDC_MEM_GPM896</td>
<td>The device supports the Gemplus GPM 896 Card Protocol.</td>
</tr>
</tbody>
</table>

dwGuidLights [...]

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

In addition to supporting specific flash rates and colors, some guidance lights also have the capability to show directional movement representing “entry” and “exit”. The “entry” state gives the impression of leading a user to place a card into the device. The “exit” state gives the impression of ejection from a device to a user and would be used for retrieving a card from the device.

The elements of this array are specified as a combination of the following flags and indicate all of the possible flash rates (type B), colors (type C) and directions (type D) that the guidance light indicator is capable of handling. If the guidance light indicator only supports one color then no value of type C is returned. If the guidance light indicator does not support direction then no value of type D is returned. A value of WFS_IDC_GUIDANCE_NOT_AVAILABLE indicates that the device has no guidance light indicator or the device controls the light directly with no application control possible.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_GUIDANCE_NOT_AVAILABLE</td>
<td>There is no guidance light control available at this position.</td>
<td>A</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_OFF</td>
<td>The light can be off.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_SLOW_FLASH</td>
<td>The light can blink slowly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_MEDIUM_FLASH</td>
<td>The light can blink medium frequency.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_QUICK_FLASH</td>
<td>The light can blink quickly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_CONTINUOUS</td>
<td>The light can be continuous (steady).</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_RED</td>
<td>The light can be red.</td>
<td>C</td>
</tr>
</tbody>
</table>
WFS_IDC_GUIDANCE_GREEN The light can be green. C
WFS_IDC_GUIDANCE_YELLOW The light can be yellow. C
WFS_IDC_GUIDANCE_BLUE The light can be blue. C
WFS_IDC_GUIDANCE_CYAN The light can be cyan. C
WFS_IDC_GUIDANCE_MAGENTA The light can be magenta. C
WFS_IDC_GUIDANCE_WHITE The light can be white. C
WFS_IDC_GUIDANCE_ENTRY The light can be in the entry state. D
WFS_IDC_GUIDANCE_EXIT The light can be in the exit state. D

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_EXITPOSITION</td>
<td>The device can eject a card to the exit position, from which the user can remove it.</td>
</tr>
<tr>
<td>WFS_IDC_TRANSPORTPOSITION</td>
<td>The device can eject a card to the transport just behind the exit position, from which the user cannot remove it. The device which supports this flag must also support the WFS_IDC_EXITPOSITION flag.</td>
</tr>
</tbody>
</table>

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

usParkingStations
Specifies the number of supported parking stations. If a zero value is specified there is no parking station supported.

bAntiFraudModule
Specifies whether the anti-fraud module is available. This can either be TRUE if available or FALSE if not available.

lpdwSynchronizableCommands
Pointer to a zero-terminated list of DWORDs which contains the execute command IDs that can be synchronized. If no execute command can be synchronized then this parameter will be NULL.

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

Comments
Applications which require or expect specific information to be present in the lpszExtra parameter may not be device or vendor-independent.
### 4.3 WFS_INF_IDC_FORM_LIST

<table>
<thead>
<tr>
<th>Description</th>
<th>This command is used to retrieve the list of forms available on the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Param</td>
<td>None.</td>
</tr>
<tr>
<td>Output Param</td>
<td>LPSTR lpszFormList;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Only the generic error codes defined in [Ref. 1] can be generated by this command.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td>None.</td>
</tr>
</tbody>
</table>
4.4 WFS_INF_IDC_QUERY_FORM

**Description**

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

**Input Param**

LPSTR lpszFormName;

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

**Output Param**

LPWFSIDCFORM lpForm;

```c
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;
    LPSTR    lpszFrontTrack1Fields;
    char    cFieldSeparatorFrontTrack1;
    LPSTR    lpszJIS1Track1Fields;
    LPSTR    lpszJIS1Track3Fields;
    CHAR    cFieldSeparatorJIS1Track1;
    CHAR    cFieldSeparatorJIS1Track3;
} WFSIDCFORM, *LPWFSIDCFORM;
```

- **lpszFormName**
  - Specifies the null-terminated name of the form.

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

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_ACTIONREAD</td>
<td>The form reads the card.</td>
</tr>
<tr>
<td>WFS_IDC_ACTIONWRITE</td>
<td>The form writes the card.</td>
</tr>
</tbody>
</table>

- **lpszTracks**
  - Specified the read algorithm or the track to write.

- **bSecure**
  - Specifies whether or not to do a security check.

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

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

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

- **lpszFrontTrack1Fields**
  - Pointer to a list of null-terminated field names of Front Track 1, with the final name terminating with two null characters.
cFieldSeparatorFrontTrack1
Specifies the value of the field separator of Front Track 1.

lpszJIS1Track1Fields
Pointer to a list of null-terminated field names of JIS I Track 1, with the final name terminating with two null characters.

lpszJIS1Track3Fields
Pointer to a list of null-terminated field names of JIS I Track 3, with the final name terminating with two null characters.

cFieldSeparatorJIS1Track1
Specifies the value of the field separator of JIS I Track 1.

cFieldSeparatorJIS1Track3
Specifies the value of the field separator of JIS I Track 3.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_FORMNOTFOUND</td>
<td>The specified form cannot be found.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_FORMINVALID</td>
<td>The specified form is invalid.</td>
</tr>
</tbody>
</table>

**Comments**
None.
4.5 WFS_INF_IDC_QUERY_IFM_IDENTIFIER

**Description**

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

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

**Input Param**

None.

**Output Param**

LPWFSIDCIFMIDENTIFIER *lppIFMIdentifier;

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

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

**wIFMAuthority**

Specifies the IFM authority that issued the IFM identifier:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_IFMEMV</td>
<td>The Level 1 Type Approval IFM identifier assigned by EMVCo.</td>
</tr>
<tr>
<td>WFS_IDC_IFMEUROPAY</td>
<td>The Level 1 Type Approval IFM identifier assigned by Europay.</td>
</tr>
<tr>
<td>WFS_IDC_IFMVISA</td>
<td>The Level 1 Type Approval IFM identifier assigned by VISA.</td>
</tr>
<tr>
<td>WFS_IDC_IFMGIECB</td>
<td>The IFM identifier assigned by GIE Cartes Bancaires.</td>
</tr>
</tbody>
</table>

**lpszIFMIdentifier**

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

**Error Codes**

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

**Comments**

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

Description
This command is used to retrieve the supported payment system applications available within an intelligent contactless card unit. The payment system application can either be identified by an AID or by the AID in combination with a Kernel Identifier. The Kernel Identifier has been introduced by the EMVCo specifications; see Reference [3].

Input Param
None.

Output Param
LPWFSIDCAPPPDATA *lppAppData;

lppAppData
Pointer to a NULL terminated array of pointers to the following data structure, each of which specifies a supported application identifier (AID) and the associated Kernel Identifier.

typedef struct wfs_idc_app_data
{
   LPWFSIDCHEXDATA   lpAID;
   LPWFSIDCHEXDATA   lpKernelIdentifier;
} WFSIDCAPPDATA, *LPWFSIDCAPPDATA;

lpAID
Contains the payment system application identifier (AID) supported by the intelligent contactless card unit.

lpKernelIdentifier
Contains the Kernel Identifier associated with the lpAID. This data may return NULL if the reader does not support Kernel Identifiers for example in the case of legacy approved contactless readers.

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

Comments
None.
5. Execute Commands

5.1 WFS_CMD_IDC_READ_TRACK

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_MEDIAJAM</td>
<td>The card is jammed. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_SHUTTERFAIL</td>
<td>The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDDATA</td>
<td>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’ (&amp;) operation cannot be read. lpstrTrackData points to data from the successfully read tracks (if any). One WFS_EXEE_IDC_INVALIDTRACKDAT A execute event is generated for each specified track which could not be read successfully. See the form description for the rules defining how tracks are specified.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_NOMEDIA</td>
<td>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.</td>
</tr>
</tbody>
</table>
CWA 16926-4:2015 (E)

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 cannot be found.

WFS_ERR_IDC_FORMINVALID

The specified form definition is invalid (e.g. syntax error).

WFS_ERR_IDC_SECURITYFAIL

The security module failed reading the cards security sign.

WFS_ERR_IDC_CARDTOOSHORT

The card that was inserted is too short. When this error occurs the card remains at the exit slot.

WFS_ERR_IDC_CARDTOOLONG

The card that was inserted is too long. When this error occurs the card remains at the exit slot.

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_EXEE_IDC_INVALIDTRACKDATA</td>
<td>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).</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_MEDIAINERSED</td>
<td>This event is generated when a card is detected in the device, giving early warning of card entry to an application, allowing it to remove a user prompt and/or do other processing while the card is being read.</td>
</tr>
<tr>
<td>WFS_SRVE_IDC_MEDIASUREMOVED</td>
<td>This event is generated when a card is removed before completion of a read operation.</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_INVALIDMEDIA</td>
<td>The user is attempting to insert the media in the wrong orientation. The card has not been accepted into the device. The device is still ready to accept a card inserted in the correct orientation.</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_INSERTCARD</td>
<td>Device is ready to accept a card from the user.</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_TRACKDETECTED</td>
<td>Track data has been detected during the insertion of the card.</td>
</tr>
</tbody>
</table>

**Comments**

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

**Example of lpstrTrackData:**

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

Description

For motor-driven card readers, the ID card unit checks whether a card has been inserted. If so, the data is written to the track as described in the form specified by the \textit{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 \textbf{WFSExecute} call for a card to be either inserted or pulled through. The next step is writing the data defined by the form and the parameters to the respective track (see Section 7, Form Definition, for a more detailed description of the forms mechanism).

This procedure is followed by data verification.

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

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

Input Param

\begin{verbatim}
LPWFSIDCWRITETRACK lpWriteTrack;
typedef struct _wfs_idc_write_track
{    
    LPSTR    lpstrFormName;
    LPSTR    lpstrTrackData;
    WORD    fwWriteMethod;
} WFSIDCWRITETRACK, *LPWFSIDCWRITETRACK;
\end{verbatim}

\textit{lpstrFormName}

Points to the name of the form to be used.

\textit{lpstrTrackData}

Points to the data to be used in the form.

\textit{fwWriteMethod}

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

\begin{tabular}{ll}
\textbf{Value} & \textbf{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.
\end{tabular}

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:

\begin{tabular}{ll}
\textbf{Value} & \textbf{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 \textbf{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. The syntax of the data pointed to by \textit{lpstrTrackData} is in error, or does not conform to the form definition.
\end{tabular}
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_INVALIDMEDIA</td>
<td>No track found; card may have been inserted or pulled through the wrong way.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_FORMNOTFOUND</td>
<td>The specified form cannot be found.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_FORMINVALID</td>
<td>The specified form definition is invalid (e.g. syntax error).</td>
</tr>
<tr>
<td>WFS_ERR_IDC_WRITE_METHOD</td>
<td>The \texttt{fwWriteMethod} value is inconsistent with device capabilities.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_CARDTOOSHORT</td>
<td>The card that was inserted is too short. When this error occurs the card remains at the exit slot.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_CARDTOOLONG</td>
<td>The card that was inserted is too long. When this error occurs the card remains at the exit slot.</td>
</tr>
</tbody>
</table>

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_EXEE_IDC_INVALIDTRACKDATA</td>
<td>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).</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_MEDIAINSERTED</td>
<td>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.</td>
</tr>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>This event is generated when a card is removed before completion of a write operation.</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_INVALIDMEDIA</td>
<td>The user is attempting to insert the media in the wrong orientation. The card has not been accepted into the device. The device is still ready to accept a card inserted in the correct orientation.</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_INSERTCARD</td>
<td>Device is ready to accept a card from the user.</td>
</tr>
</tbody>
</table>

**Comments**

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

Example of \texttt{lpstrTrackData}:

```
RETRYCOUNT=3
DATE=3132
```

...
5.3 WFS_CMD_IDC_EJECT_CARD

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

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

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

**Input Param**
LPWFSIDCEJECTCARD lpEjectCard;

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

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

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

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

**Output Param**
None.

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

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

**Events**
In addition to the generic events defined in [Ref.1], the following events can be generated by this command:
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>The card has been taken by the user.</td>
</tr>
<tr>
<td>WFS_USRE_IDC_RETAINBINTHRESHOLD</td>
<td>The retain bin reached a threshold value.</td>
</tr>
</tbody>
</table>

**Comments**

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

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

Input Param
None.

Output Param
LPWFSIDCRETAINCARD lpRetainCard;

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_MEDIAUNKNOWN</td>
<td>The position of the card cannot be determined with the device in its current state.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIAPRESENT</td>
<td>The card is present in the reader.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIAENTERING</td>
<td>The card is in the entering position (shutter).</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_USRE_IDC_RETAINBINTHRESHOLD</td>
<td>The retain bin reached a threshold value.</td>
</tr>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>The card has been taken by the user.</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_MEDIARETAINED</td>
<td>The card has been retained. This event is only fired if the command completes successfully (with WFS_SUCCESS).</td>
</tr>
</tbody>
</table>

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

Description
This function resets the present value for number of cards retained to zero. The function is possible for motor-driven card readers only. The number of cards retained is controlled by the service and can be requested before resetting via the WFS_INF_IDC_STATUS.

Input Param
None.

Output Param
None.

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_USRE_IDC_RETAINBINTHRESHOLD</td>
<td>The retain bin was emptied.</td>
</tr>
</tbody>
</table>

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

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

Input Param
LPWFSIDCSETKEY lpSetkey;

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

usKeyLen
Specifies the length of the following key value.

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

Output Param
None.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_INVALIDKEY</td>
<td>The key does not fit to the security module.</td>
</tr>
</tbody>
</table>

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

Comments
None.
5.7  WFS_CMD_IDC_READ_RAW_DATA

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

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

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

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

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

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

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

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

Input Param
LPWORD lpwReadData;

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_TRACK1</td>
<td>Track 1 of the magnetic stripe will be read.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK2</td>
<td>Track 2 of the magnetic stripe will be read.</td>
</tr>
<tr>
<td>WFS_IDC.Track3</td>
<td>Track 3 of the magnetic stripe will be read.</td>
</tr>
<tr>
<td>WFS_IDC_CHIP</td>
<td>The chip will be read.</td>
</tr>
<tr>
<td>WFS_IDC_SECURITY</td>
<td>A security check will be performed.</td>
</tr>
<tr>
<td>WFS_IDC_FLUXINACTIVE</td>
<td>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.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK_WM</td>
<td>The Swedish Watermark track will be read.</td>
</tr>
<tr>
<td>WFS_IDC_MEMORY_CHIP</td>
<td>The memory chip will be read.</td>
</tr>
<tr>
<td>WFS_IDC_FRONT_TRACK_1</td>
<td>Track 1 data is read from the magnetic stripe located on the front of the card. In some countries this track is known as JIS II track.</td>
</tr>
<tr>
<td>WFS_IDC_FRONTIMAGE</td>
<td>The front image of the card will be read in BMP format.</td>
</tr>
<tr>
<td>WFS_IDC_BACKIMAGE</td>
<td>The back image of the card will be read in BMP format.</td>
</tr>
</tbody>
</table>
CWA 16926-4:2015 (E)

WFS_IDC_TRACK1_JIS1
Track 1 of Japanese cash transfer card will be read. In some countries this track is known as JIS I track 1 (8bits/char).

WFS_IDC_TRACK3_JIS1
Track 3 of Japanese cash transfer card will be read. In some countries this track is known as JIS I track 3 (8bits/char).

WFS_IDC_DDI
Dynamic Digital Identification data of the magnetic stripe will be read.

Output Param
LPWFSIDCCARDDATA *lppCardData;

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_TRACK1</td>
<td>lpbData contains data read from track 1.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK2</td>
<td>lpbData contains data read from track 2.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK3</td>
<td>lpbData contains data read from track 3.</td>
</tr>
<tr>
<td>WFS_IDC_CHIP</td>
<td>lpbData contains ATR data read from the chip.</td>
</tr>
<tr>
<td></td>
<td>For contactless chip card readers, multiple identification information can be returned if the card reader detects more than one chip. Each chip identification information is returned as an individual lppCardData array element.</td>
</tr>
<tr>
<td>WFS_IDC_SECURITY</td>
<td>lpbData contains the value returned by the security module.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK_WM</td>
<td>lpbData contains data read from the Swedish Watermark track.</td>
</tr>
<tr>
<td>WFS_IDC_MEMORY_CHIP</td>
<td>lpbData contains Memory Card Identification data read from the memory chip.</td>
</tr>
<tr>
<td>WFS_IDC_FRONT_TRACK_1</td>
<td>lpbData contains data read from the front track 1. In some countries this track is known as JIS II track.</td>
</tr>
<tr>
<td>WFS_IDC_FRONTIMAGE</td>
<td>lpbData contains a null-terminated string containing the full path and file name of the BMP image file for the front of the card.</td>
</tr>
<tr>
<td>WFS_IDC_BACKIMAGE</td>
<td>lpbData contains a null-terminated string containing the full path and file name of the BMP image file for the back of the card.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK1_JIS1</td>
<td>lpbData contains data read from JIS I track 1 (8bits/char).</td>
</tr>
<tr>
<td>WFS_IDC_TRACK3_JIS1</td>
<td>lpbData contains data read from JIS I track 3 (8bits/char).</td>
</tr>
<tr>
<td>WFS_IDC_DDI</td>
<td>lpbData contains dynamic digital identification data read from magnetic stripe.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_DATAOK</td>
<td>The data is OK.</td>
</tr>
</tbody>
</table>
The track/chip/memory chip is blank.

The data contained on the track/chip/memory chip is invalid. This will typically be returned when `lpbData` reports `WFS_IDC_SEC_BADREADLEVEL` or `WFS_IDC_SEC_DATAINVAL`.

The data contained on the track/chip/memory chip is too long.

The data contained on the track/chip/memory chip is too short.

The data source to read from is not supported by the Service Provider.

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

Specifies the length of the following field `lpbData`.

Points to the data read from the track/chip, the value returned by the security module or a null-terminated string containing the full path and file name of the BMP image file. This value is terminated with a single null character and cannot contain UNICODE characters.

The security module can return one of the following values:

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

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

Ignored for this command.

In addition to the generic error codes defined in [Ref. 1], the following error codes can be generated by this command:
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_MEDIAJAM</td>
<td>The card is jammed. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_SHUTTERFAIL</td>
<td>The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_NOMEDIA</td>
<td>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.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDMEDIA</td>
<td>No track or chip found; card may have been inserted or pulled through the wrong way.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_CARDTOOSHORT</td>
<td>The card that was inserted is too short. When this error occurs the card remains at the exit slot.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_CARDTOOLONG</td>
<td>The card that was inserted is too long. When this error occurs the card remains at the exit slot.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_SECURITYFAIL</td>
<td>The security module failed reading the cards security sign.</td>
</tr>
</tbody>
</table>

**Events**

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

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

**Comments**

None.
5.8 **WFS_CMD_IDC_WRITE_RAW_DATA**

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

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

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

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

This procedure is followed by data verification.

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

**Input Param**
LPWFSIDCCARDDATA *lppCardData;

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_TRACK1</td>
<td><code>lpbData</code> contains the data to be written to track 1.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK2</td>
<td><code>lpbData</code> contains the data to be written to track 2.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK3</td>
<td><code>lpbData</code> contains the data to be written to track 3.</td>
</tr>
<tr>
<td>WFS_IDC_FRONT_TRACK_1</td>
<td><code>lpbData</code> contains the data to be written to the front track 1. In some countries this track is known as JIS II track.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK1_JIS1</td>
<td><code>lpbData</code> contains the data to be written to JIS I track 1 (8bits/char).</td>
</tr>
<tr>
<td>WFS_IDC_TRACK3_JIS1</td>
<td><code>lpbData</code> contains the data to be written to JIS I track 3 (8bits/char).</td>
</tr>
</tbody>
</table>

`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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_LOCO</td>
<td>Low coercivity magnetic stripe is being written.</td>
</tr>
<tr>
<td>WFS_IDC_HICO</td>
<td>High coercivity magnetic stripe is being written.</td>
</tr>
</tbody>
</table>
WFS_IDC_AUTO

Service Provider will determine whether low or high coercivity stripe is to be written.

Output Param
None.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_MEDIAJAM</td>
<td>The card is jammed. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_SHUTTERFAIL</td>
<td>The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_NOMEDIA</td>
<td>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.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDMEDIA</td>
<td>No track found; card may have been inserted or pulled through the wrong way.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_WRITE_METHOD</td>
<td>The fwWriteMethod value is inconsistent with device capabilities.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_CARDTOOSHORT</td>
<td>The card that was inserted is too short. When this error occurs the card remains at the exit slot.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_CARDTOOLONG</td>
<td>The card that was inserted is too long. When this error occurs the card remains at the exit slot.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_EXEE_IDC_MEDIAINSERTED</td>
<td>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.</td>
</tr>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>This event is generated when a card is removed before completion of a write operation.</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_INVALIDMEDIA</td>
<td>The user is attempting to insert the media in the wrong orientation. The card has not been accepted into the device. The device is still ready to accept a card inserted in the correct orientation.</td>
</tr>
<tr>
<td>WFS_EXEE_IDC_INSERTCARD</td>
<td>Device is ready to accept a card from the user.</td>
</tr>
</tbody>
</table>

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

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

The identification information e.g. ATR of the chip must be obtained before issuing this command. The identification information for a user card or the Memory Card Identification (when available) must initially be obtained through WFS_CMD_IDC_READ_RAW_DATA. The identification information 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.

For contactless chip card readers, applications need to specify which chip to contact with, as part of lpChipData, if more than one chip has been detected and multiple identification data has been returned by the WFS_CMD_IDC_READ_RAW_DATA command.

Input Param
LPWFSIDCCHIPIO lpChipIoIn;

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

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

ulChipDataLength
Specifies the length of the following field lpbChipData.

lpbChipData
Points to the data sent to the chip.

Output Param
LPWFSIDCCHIPIO lpChipIoOut;

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

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

ulChipDataLength
Specifies the length of the following field lpbChipData.

lpbChipData
Points to the data responded from the chip.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_MEDIAJAM</td>
<td>The card is jammed. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_NOMEDIA</td>
<td>There is no card inside the device.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDMEDIA</td>
<td>No chip found; card may have been inserted the wrong way.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDDATA</td>
<td>An error occurred while communicating with the chip.</td>
</tr>
</tbody>
</table>
In addition to the generic events defined in [Ref. 1], the following events can be generated by this command:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>This event is generated when a card is removed before completion of an operation.</td>
</tr>
</tbody>
</table>

None.
5.10 WFS_CMD_IDC_RESET

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

If the device is a user ID card unit, the device will attempt to either retain, eject or will perform no action on any user cards found in the IDC as specified in the lpwResetln 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.

For devices with parking station capability there will be one WFS_SRVE_IDC_MEDIADETECTED event for each card found.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_EJECT</td>
<td>Eject any card found.</td>
</tr>
<tr>
<td>WFS_IDC_RETAIN</td>
<td>Retain any card found.</td>
</tr>
<tr>
<td>WFS_IDC_NOACTION</td>
<td>No action should be performed on any card found.</td>
</tr>
</tbody>
</table>

If lpwResetln 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_MEDIAJAM</td>
<td>The card is jammed. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_SHUTTERFAIL</td>
<td>The device is unable to open and close its shutter.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_RETAINBINFULL</td>
<td>The retain bin is full; no more cards can be retained. The current card is still in the device.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_IDC_MEDIADETECTED</td>
<td>This event is generated when a media is detected during a reset. For devices with parking station capability there will be one event for each card found.</td>
</tr>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>The card has been taken by the user.</td>
</tr>
<tr>
<td>WFS_USRE_IDC_RETAINBINTHRESHOLD</td>
<td>The retain bin reached a threshold value.</td>
</tr>
</tbody>
</table>

Comments
None.
5.11 WFS_CMD_IDC_CHIP_POWER

**Description**

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

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

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

**Input Param**

LPWORD lpwChipPower;

*lpwChipPower*

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CHIPPOWERCOLD</td>
<td>The chip is powered on and reset (Cold Reset).</td>
</tr>
<tr>
<td>WFS_IDC_CHIPPOWERWARM</td>
<td>The chip is reset (Warm Reset).</td>
</tr>
<tr>
<td>WFS_IDC_CHIPPOWEROFF</td>
<td>The chip is powered off.</td>
</tr>
</tbody>
</table>

**Output Param**

NULL or LPWFSIDCCHIPPOWEROUT lpChipPowerOut;

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

*ulChipDataLength*

Specifies the length of the following field *lpbChipData*.

*lpbChipData*

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

**Error Codes**

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

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

**Events**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>This event is generated when a card is removed before completion of the operation.</td>
</tr>
</tbody>
</table>

**Comments**

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

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

**Input Param**
LPWFSIDCPARSEDATA lpParseData;

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

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

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

**Output Param**
LPSTR lpstrTrackData;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_INVALIDDATA</td>
<td>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' (</td>
</tr>
<tr>
<td>WFS_ERR_IDC_FORMNOTFOUND</td>
<td>The specified form cannot be found.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_FORMINVALID</td>
<td>The specified form definition is invalid (e.g. syntax error).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_EXEE_IDC_INVALIDTRACKDATA</td>
<td>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).</td>
</tr>
</tbody>
</table>

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

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

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

Input Param
LPWFSIDCSETGUIDLIGHT lpSetGuidLight;

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_GUIDANCE_OFF</td>
<td>The light indicator is turned off.</td>
<td>A</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_SLOW_FLASH</td>
<td>The light indicator is set to flash slowly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_MEDIUM_FLASH</td>
<td>The light indicator is set to flash medium frequency.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_QUICK_FLASH</td>
<td>The light indicator is set to flash quickly.</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_CONTINUOUS</td>
<td>The light indicator is turned on continuously (steady).</td>
<td>B</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_RED</td>
<td>The light indicator color is set to red.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_GREEN</td>
<td>The light indicator color is set to green.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_YELLOW</td>
<td>The light indicator color is set to yellow.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_BLUE</td>
<td>The light indicator color is set to blue.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_CYAN</td>
<td>The light indicator color is set to cyan.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_MAGENTA</td>
<td>The light indicator color is set to magenta.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_WHITE</td>
<td>The light indicator color is set to white.</td>
<td>C</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_ENTRY</td>
<td>The light indicator is set to the entry state.</td>
<td>D</td>
</tr>
<tr>
<td>WFS_IDC_GUIDANCE_EXIT</td>
<td>The light indicator is set to the exit state.</td>
<td>D</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_INVALID_PORT</td>
<td>An attempt to set a guidance light to a new value was invalid because the guidance light does not exist.</td>
</tr>
</tbody>
</table>

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

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

The slow and medium flash rates must not be greater than 2.0 Hz. It should be noted that in order to comply with American Disabilities Act guidelines only a slow or medium flash rate must be used.
5.14 WFS_CMD_IDC_POWER_SAVE_CONTROL

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

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

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

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

**Output Param**
None.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_POWERSAVETOOSHORT</td>
<td>The power saving mode has not been activated because the device is not able to resume from the power saving mode within the specified <code>usMaxPowerSaveRecoveryTime</code> value.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_POWERSAVEMEDIAPRESENT</td>
<td>The power saving mode has not been activated because media is present inside the device.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_IDC_POWER_SAVE_CHANGE</td>
<td>The power save recovery time has changed.</td>
</tr>
</tbody>
</table>

**Comments**
None.
5.15 WFS_CMD_IDC_PARK_CARD

Description
This command is used to move a card that is present in the reader to a parking station. A parking
station is defined as an area in the IDC, which can be used to temporarily store the card while the
device performs operations on another card. This command is also used to move a card from the
parking station to the read/write, chip I/O or transport position. When a card is moved from the
parking station to the read/write, chip I/O or transport position (WFSIDCPARKCARD.wDirection = PARK_OUT), the read/write, chip I/O or transport position
must not be occupied with another card, otherwise the error WFS_ERR_IDC_CARDPRESENT
will be returned.

After moving a card to a parking station, another card can be inserted and read by calling e.g. the
WFS_CMD_IDC_READ_RAW_DATA or WFS_CMD_IDC_READ_TRACK command.

Cards in parking stations will not be affected by any IDC commands until they are removed from
the parking station using this command, except for the WFS_CMD_IDC_RESET command. The
WFS_CMD_IDC_RESET command will move the cards in the parking stations as specified in its
lpwResetIn parameter as part of the reset action if possible.

Input Param
LPWFSIDCPARKCARD lpParkCard;
typedef struct _wfs_idc_park_card
{
    WORD    wDirection;
    USHORT   usParkingStation;
} WFSIDCPARKCARD, *LPWFSIDCPARKCARD;

wDirection
Specifies which way to move the card as one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_PARK_IN</td>
<td>The card is moved to the parking station from the read/write, chip I/O or transport position.</td>
</tr>
<tr>
<td>WFS_IDC_PARK_OUT</td>
<td>The card is moved from the parking station to the read/write, chip I/O or transport position. Once the card has been moved any command can be executed e.g. WFS_CMD_IDC_EJECT_CARD or WFS_CMD_IDC_READ_RAW_DATA.</td>
</tr>
</tbody>
</table>

usParkingStation
Specifies which parking station should be used for this command. This value is the same index as is identified in the lpwParkingStationMedia array of the WFS_INF_IDC_STATUS query.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_MEDIAJAM</td>
<td>The card is jammed. Operator intervention is required.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_NOMEDIA</td>
<td>No card is present at the read/write, chip I/O or transport position and the WFS_IDC_PARK_IN option has been selected. Or no card is in the parking station and the WFS_IDC_PARK_OUT option has been selected.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_CARDPRESENT</td>
<td>Another card is present and is preventing successful movement of the card specified by usParkingStation.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_POSITIONINVALID</td>
<td>The specified parking station is invalid.</td>
</tr>
</tbody>
</table>

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

Comments
None.
5.16 WFS_CMD_IDC_EMVCLESS_CONFIGURE

Description
This command is used to configure an intelligent contactless card reader before performing a contactless transaction. This command sets terminal related data elements, the list of terminal acceptable applications with associated application specific data and any encryption key data required for offline data authentication.

This command should be used prior to the WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION command. It may be called once on application start up or when any of the configuration parameters require to be changed. The configuration set by this command is persistent.

This command should be called with a complete list of acceptable payment system applications as any previous configurations will be replaced.

Input Param
LPWFSIDCEMVCLESSCONFIGDATA lpClessConfigData;

typedef struct _wfs_idc_emvcless_config_data
{
    LPWFSIDCHEXDATA lpTerminalData;
    LPWFSIDCAIDDATA *lppAIDData;
    LPWFSIDKEYDATA *lppKeyData;
} WFSIDCEMVCLESSCONFIGDATA, *LPWFSIDCEMVCLESSCONFIGDATA;

lpTerminalData
Specifies the BER-TLV formatted data for the terminal e.g. Terminal Type, Transaction Category Code, Merchant Name & Location etc. Any terminal based data elements referenced in the Payment Systems Specifications or EMVCo Contactless Payment Systems Specifications Books may be included (see References [2] to [14] section for more details).

lppAIDData
Pointer to a NULL terminated array of pointers to data structures.

This data structure specifies the list of acceptable payment system applications. For EMVCo approved contactless card readers each AID is associated with a Kernel Identifier and a Transaction Type. Legacy approved contactless readers may use only the AID.

Each AID-Transaction Type or each AID-Kernel-Transaction Type combination will have its own unique set of configuration data. See References [2] and [3] for more details.

typedef struct _wfs_idc_aid_data
{
    LPWFSIDCHEXDATA lpAID;
    BOOL bPartialSelection;
    ULONG ulTransactionType;
    LPWFSIDCHEXDATA lpKernelIdentifier;
    LPWFSIDCHEXDATA lpConfigData;
} WFSIDCAIDDATA, *LPWFSIDCAIDDATA;

lpAID
Specifies the application identifier to be accepted by the contactless chip card reader. The WFS_INF_IDC_EMVCLESS_QUERY_APPLICATIONS command will return the list of supported application identifiers.

bPartialSelection
If bPartialSelection is TRUE, partial name selection of the specified AID is enabled. If bPartialSelection is FALSE, partial name selection is disabled. A detailed explanation for partial name selection is given in EMV 4.3 Book 1, Section 11.3.5.

ulTransactionType
Specifies the transaction type supported by the AID. This indicates the type of financial transaction represented by the first two digits of the ISO 8583:1987 Processing Code.

lpKernelIdentifier
 Specifies the EMVCo defined kernel identifier associated with the lpAID. This field will be ignored if the reader does not support kernel identifiers.
lpConfigData
Contains the list of BER-TLV formatted configuration data, applicable to the specific AID-
Kernel ID-Transaction Type combination. The appropriate payment systems specifications
define the BER-TLV tags to be configured.

lpKeyData
A pointer to a NULL terminated array of pointers to data structures, each includes encryption key
information required by an intelligent contactless chip card reader for offline data authentication.

typedef struct _wfs_idc_key_data
{
    LPWFSIDCHEXDATA lpRID;
    WORD wCAPublicKeyIndex;
    WORD wCAPublicKeyAlgorithmIndicator;
    LPWFSIDCHEXDATA lpCAPublicKeyExponent;
    LPWFSIDCHEXDATA lpCAPublicKeyModulus;
    LPBYTE lpbCAPublicKeyChecksum;
} WFSIDCKEYDATA, *LPWFSIDCKEYDATA;

lpRID
Specifies the payment system's Registered Identifier (RID). RID is the first 5 bytes of the AID
and identifies the payments system.

wCAPublicKeyIndex
Specifies the CA Public Key Index for the specific RID.

wCAPublicKeyAlgorithmIndicator
Specifies the algorithm used in the calculation of the CA Public Key checksum.

lpCAPublicKeyExponent
Specifies the CA Public Key Exponent for the specific RID.

lpCAPublicKeyModulus
Specifies the CA Public Key Modulus for the specific RID.

lpbCAPublicKeyChecksum
Specifies the 20 byte checksum value for the CA Public Key.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_INVALIDTERMINALDATA</td>
<td>Input data lpTerminalData was invalid. Contactless chip card reader could not be configured successfully.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDAIDDATA</td>
<td>Input data lppAIDData was invalid. Contactless chip card reader could not be configured successfully.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDKEYDATA</td>
<td>Input data lpKeyData was invalid. Contactless chip card reader could not be configured successfully.</td>
</tr>
</tbody>
</table>

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

Comments None.
5.17 WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION

Description
This command is used to enable an intelligent contactless card reader. The transaction will start as soon as the card tap is detected.

Based on the configuration of the contactless chip card and the reader device, this command could return data formatted either as magnetic stripe information or as a set of BER-TLV encoded EMV tags.

This command supports magnetic stripe emulation cards and EMV-like contactless cards but cannot be used on storage contactless cards. The latter must be managed using the WFS_CMD_IDC_READ_RAW_DATA and WFS_CMD_IDC_CHIP_IO commands.

For specific payment system's card profiles an intelligent card reader could return a set of EMV tags along with magnetic stripe formatted data. In this case, two contactless card data structures will be returned, one containing the magnetic stripe like data and one containing BER-TLV encoded tags.

If no card has been tapped, the contactless chip card reader waits for the period of time specified in the WFSExecute call for a card to be tapped.

For intelligent contactless card readers, any in-built audio/visual feedback such as Beep/LEDs, need to be controlled directly by the reader. These indications should be implemented based on the EMVCo and payment system's specifications.

Input Param
LPWFSIDCEMVCLESSTXDATA lpClessTxData;

typedef struct _wfs_idc_emvcless_tx_data

LPWFSIDCHEXDATA lpData;

WFSIDCEMVCLESSTXDATA, *LPWFSIDCEMVCLESSTXDATA;

lpData
Supplies EMV data elements in a BER-TLV format required to perform a transaction. The types of object that could be listed in the lpData are:

- Transaction Type (9C)
- Amount Authorized (9F02)
- Transaction Date (9A)*
- Transaction Time (9F21)*
- Transaction Currency Code (5F2A)

Individual payment systems could define further data elements.
*Tags 9A and 9F21 are not mandatory as they could be managed internally by the reader especially when the card is tapped to initiate the transaction.

Output Param
LPWFSIDCEMVCLESSTXDATAOUTPUT *lppClessTxDataOutput;

typedef struct _wfs_idc_emvcless_tx_data_output

WORD wDataSource;
WORD wTxOutcome;
WORD wCardholderAction;
LPWFSIDCHEXDATA lpDataRead;
LPWFSIDCEMVCLESSEXTCOMMUNICATION lpClessOutcome;

WFSIDCEMVCLESSTXDATAOUTPUT, *LPWFSIDCEMVCLESSTXDATAOUTPUT;

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_TRACK1</td>
<td>lpDataRead contains the chip returned data formatted in as track 1.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK2</td>
<td>lpDataRead contains the chip returned data formatted in as track 2.</td>
</tr>
</tbody>
</table>
WFS_IDC_TRACK3

lpDataRead contains the chip returned data formatted in as track 3.

WFS_IDC_CHIP

lpDataRead contains the BER-TLV formatted data read from the chip.

**wTxOutcome**

Specifies the contactless transaction outcome as one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CLESS_MULTIPLECARDS</td>
<td>Transaction could not be completed as more than one contactless card was tapped.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_APPROVE</td>
<td>Transaction was approved offline.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_DECLINE</td>
<td>Transaction was declined offline.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_ONLINEREQUEST</td>
<td>Transaction was requested for online authorization.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_ONLINEREQUESTCOMPLETIONREQUIRED</td>
<td>Transaction requested online authorization and will be completed after a re-tap of the card. Transaction should be completed by issuing the WFS_CMD_IDC_EMVCLESS_ISSUERUPDATE command.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_TRYAGAIN</td>
<td>Transaction could not be completed due to a card read error. The contactless card could be tapped again to re-attempt the transaction.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_TRYANOTHERINTERFACE</td>
<td>Transaction could not be completed over the contactless interface. Another interface may be suitable for this transaction (for example contact).</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_ENDAPPLICATION</td>
<td>Transaction cannot be completed on the contactless card due to an irrecoverable error.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_CONFIRMATIONREQUIRED</td>
<td>Transaction was not completed as a result of a requirement to allow entry of confirmation code on a mobile device. Transaction should be completed by issuing the WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION after a card removal and a re-tap of the card.</td>
</tr>
</tbody>
</table>

NOTE: The values for **wTxOutcome** have been mapped against the EMV Entry Point Outcome structure values defined in the EMVCo Specifications for Contactless Payment Systems (Book A and B).

**wCardholderAction**

Specifies the cardholder action as one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CLESS_NOACTION</td>
<td>Transaction was completed. No further action is required.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_RETAP</td>
<td>The contactless card should be re-tapped to complete the transaction. This value can be returned when <strong>wTxOutcome</strong> is WFS_IDC_CLESS_ONLINEREQUESTCOMPLETIONREQUIRED or WFS_IDC_CLESS_CONFIRMATIONREQUIRED.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_HOLDCARD</td>
<td>The contactless card should not be removed from the field until the transaction is completed.</td>
</tr>
</tbody>
</table>
**lpDataRead**
Points to the data read from the chip after a contactless transaction has been completed successfully.

**lpClessOutcome**
Pointer to a structure that represents the Entry Point Outcome structure specified in EMVCo Specifications for Contactless Payment Systems (Book A and B). The `lpClessOutcome` can be NULL for contactless chip card readers that do not follow EMVCo Entry Point Specifications.

```c
typedef struct _wfs_idc_emvcless_outcome {
    WORD    wCVM;
    WORD    wAlternateInterface;
    BOOL    bReceipt;
    LPWFSIDCEMVCLESSUI lpClessUIOutcome;
    LPWFSIDCEMVCLESSUI lpClessUIRestart;
    WORD    ulClessFieldOffHoldTime;
    ULONG    ulCardRemovalTimeoutValue;
    LPWFSIDCHEXDATA  lpDiscretionaryData;
} WFSIDCEMVCLESSOUTCOME, *LPWFSIDCEMVCLESSOUTCOME;
```

**wCVM**
Specifies the cardholder verification method (CVM) to be performed as one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CLESS_ONLINEPIN</td>
<td>Online PIN should be entered by the cardholder.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_CONFIRMATIONCODEVERIFIED</td>
<td>A confirmation code entry has been successfully done on a mobile device.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_SIGN</td>
<td>Application should obtain cardholder signature.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_NOCVM</td>
<td>No CVM is required for this transaction.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_NOCVMPREFERENCE</td>
<td>There is no CVM preference, but application can follow the payment system’s rules to process the transaction.</td>
</tr>
</tbody>
</table>

**wAlternateInterface**
Specifies the alternative interface to be used to complete a transaction as one of the following flags, in cases where the `wTxOutcome` is `WFS_IDC_CLESS_TRYANOTHERINTERFACE`:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CLESS_CONTACT</td>
<td>Contact chip interface should be used to complete a transaction.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_MAGNETIC_STRIPE</td>
<td>Magnetic stripe interface should be used to complete a transaction.</td>
</tr>
</tbody>
</table>

**bReceipt**
Specifies whether a receipt should be printed. TRUE indicates that a receipt is required.

**lpClessUIOutcome**
Pointer to a structure representing the user interface details required to be displayed to the cardholder after processing the outcome of a contactless transaction. Please refer to EMVCo Contactless Specifications for Payment Systems Book A, Section 6.2 for details of the data within this structure:

```c
typedef struct _wfs_idc_emvcless_ui {
    WORD    wMessageId;
    WORD    wStatus;
    ULONG    ulHoldTime;
    WORD    wValueQualifier;
    LPSTR    lpszValue;
    LPSTR    lpszCurrencyCode;
    LPSTR    lpszLanguagePreferenceData;
} WFSIDCEMVCLESSUI, *LPWFSIDCEMVCLESSUI;
```
wMessageId
Represents the EMVCo defined message identifier that indicates the text string to be displayed.

wStatus
Represents the EMVCo defined transaction status value to be indicated through the Beep/LEDs as one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CLESS_NOTREADY</td>
<td>Contactless card reader is not able to communicate with a card. This status occurs towards the end of a contactless transaction or if the reader is not powered on.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_IDLE</td>
<td>Contactless card reader is powered on, but the reader field is not yet active for communication with a card.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_READYTOREAD</td>
<td>Contactless card reader is powered on and attempting to initiate communication with a card.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_PROCESSING</td>
<td>Contactless card reader is in the process of reading the card.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_CARDREADOK</td>
<td>Contactless card reader was able to read a card successfully.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_PROCESSINGERROR</td>
<td>Contactless card reader was not able to process the card successfully.</td>
</tr>
</tbody>
</table>

ulHoldTime
Represents the hold time in units of 100 milliseconds for which the application should display the message before processing the next user interface data.

wValueQualifier
This data is defined by EMVCo as either “Amount” or “Balance” as one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CLESS_AMOUNT</td>
<td>Value Qualifier is Amount.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_BALANCE</td>
<td>Value Qualifier is Balance.</td>
</tr>
</tbody>
</table>

lpszValue
Represents the value of the amount or balance to be displayed when wValueQualifier is present.

lpszCurrencyCode
Represents the numeric value of currency code as per ISO 4217.

lpszLanguagePreferenceData
Represents the language preference (EMV Tag ‘5F2D’) if returned by the card. The application should use this data to display all messages in the specified language until the transaction concludes.

lpClessUIRestart
Pointer to a structure representing the user interface details required to be displayed to the cardholder when a transaction needs to be completed with a re-tap. For structure description see the lpClessUIOutcome field description.

ulClessFieldOffHoldTime
The application should wait for this specific hold time in units of 100 milliseconds, before re-enabling the contactless card reader by issuing either the WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION command or the WFS_CMD_IDC_EMVCLESS_ISSUERUPDATE command depending on the value of wTxOutcome. For intelligent contactless card readers, the completion of this command ensures that the contactless chip card reader field is automatically turned off, so there is no need for the application to disable the field.
ulCardRemovalTimeoutValue
Specifies a timeout value in units of 100 milliseconds for prompting the user to remove the card.

lpDiscretionaryData
Points to the payment system's specific discretionary data read from the chip, in a BER-TLV format, after a contactless transaction has been completed.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_NOMEDIA</td>
<td>The card was removed before completion of the read action.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDMEDIA</td>
<td>No track or chip was found or the card tapped cannot be used with this command (e.g. contactless storage cards).</td>
</tr>
<tr>
<td>WFS_ERR_IDC_READERNOTCONFIGURED</td>
<td>This command was issued before calling WFS_CMD_IDC_EMVCLESS_CONFIGURE command.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_EXEE_IDC_EMVCLESSREADSTATUS</td>
<td>This event is generated to notify the application the status of a contactless card tap.</td>
</tr>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>This event is generated when a card is removed before completion of a read operation.</td>
</tr>
</tbody>
</table>

Comments
None.
5.18 WFS_CMD_IDC_EMVCLESS_ISSUERUPDATE

**Description**
This command performs the post authorization processing on payment systems contactless cards. Before an online authorized transaction is considered complete, further chip processing may be requested by the issuer. This is only required when the authorization response includes issuer update data; either issuer scripts or issuer authentication data.

The command enables the contactless card reader and waits for the customer to re-tap their card.

The contactless chip card reader waits for the period of time specified in the **WFSEexecute** call for a card to be tapped.

**Input Param**

```c
typedef struct _wfs_idc_emvcless_tx_data {
    LPWFSIDCHEXDATA lpData;
} WFSIDCEMVCLESSTXDATA, *LPWFSIDCEMVCLESSTXDATA;
```

**lpData**
Supplies BER-TLV formatted EMV data elements received from the authorization response that are required to complete the transaction processing.

The types of object that could be listed in **lpData** are:
- Authorization Code (if present)
- Issuer Authentication Data (if present)
- Issuer Scripts or proprietary payment system's data elements (if present) and any other data elements if required.

**Output Param**

```c
typedef struct _wfs_idc_emvcless_tx_data_output {
    WORD     wDataSource;
    WORD     wTxOutcome;
    WORD     wCardholderAction;
    LPWFSIDCHEXDATA lpDataRead;
    LPWFSIDCEMVCLESSOUTCOME lpClessOutcome;
} WFSIDCEMVCLESSTXDATAOUTPUT, *LPWFSIDCEMVCLESSTXDATAOUTPUT;
```

**wDataSource**
Specifies the source of the card data as the following flag:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CHIP</td>
<td><strong>lpData</strong> contains the BER-TLV formatted data read from the chip.</td>
</tr>
</tbody>
</table>

**wTxOutcome**
Specifies the contactless transaction outcome as one of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CLESS_MULTIPLECARDS</td>
<td>Transaction could not be completed as more than one contactless card was tapped.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_ENDAPPLICATION</td>
<td>Post authorization processing has been completed on the contactless card.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_APPROVE</td>
<td>Transaction was approved offline.</td>
</tr>
<tr>
<td>WFS_IDC_CLESSDECLINE</td>
<td>Transaction was declined offline.</td>
</tr>
<tr>
<td>WFS_IDC_CLESS_TRYAGAIN</td>
<td>Transaction could not be completed due to a card read error. The contactless card could be tapped again to re-attempt the transaction.</td>
</tr>
</tbody>
</table>
WFS_IDC_CLESS_TRYANOTHERINTERFACE
Transaction could not be completed over the contactless interface. Another interface may be suitable for this transaction (for example contact).

\textit{wCardholderAction}
Specifies the cardholder action as the following flag:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CLESS_NOACTION</td>
<td>Transaction was completed. No further action required.</td>
</tr>
</tbody>
</table>

\textit{lpDataRead}
Points to the data read from the chip or issuer script results after a contactless transaction has been completed successfully.

\textit{lpClessOutcome}
Pointer to a structure that represents the Entry Point Outcome structure specified in EMVCo Specifications for Contactless Payment Systems (Book A and B). The \textit{lpClessOutcome} can be NULL for contactless chip card readers that do not follow EMVCo Entry Point Specifications. See the outcome parameter of WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION command for details of \textit{lpClessOutcome} data structure.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_NOMEDIA</td>
<td>The card was removed before completion of the read action.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_INVALIDMEDIA</td>
<td>No track or chip found or card tapped cannot be used with this command (e.g. contactless storage cards or a different card than what was used to complete the WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION command).</td>
</tr>
<tr>
<td>WFS_ERR_IDC_TRANSACTIONNOTINITIATED</td>
<td>This command was issued before calling the WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION command.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_SRVE_IDC_MEDIAREMOVED</td>
<td>This event is generated when a card is removed before completion of a read operation.</td>
</tr>
</tbody>
</table>

\textbf{Comments}
None.
5.19  WFS_CMD_IDC_SYNCHRONIZE_COMMAND

Description

This command is used to reduce response time of a command (e.g. for synchronization with display) as well as to synchronize actions of the different device classes. This command is intended to be used only on hardware which is capable of synchronizing functionality within a single device class or with other device classes.

The list of execute commands which this command supports for synchronization is retrieved in the `lpdwSynchronizableCommands` parameter of the WFS_INF_IDC_CAPABILITIES.

This command is optional, i.e, any other command can be called without having to call it in advance. Any preparation that occurs by calling this command will not affect any other subsequent command. However, any subsequent execute command other than the one that was specified in the `dwCommand` input parameter will execute normally and may invalidate the pending synchronization. In this case the application should call the WFS_CMD_IDC_SYNCHRONIZE_COMMAND again in order to start a synchronization.

Input Param

LPWFSIDCSYNCHRONIZECOMMAND lpSynchronizeCommand;

typedef struct _wfs_bcr_synchronize_command
{
    DWORD     dwCommand;
    LPVOID    lpCmdData;
} WFSIDCSYNCHRONIZECOMMAND, *LPWFSIDCSYNCHRONIZECOMMAND;

dwCommand
The command ID of the command to be synchronized and executed next.

lpCmdData
Pointer to data or a data structure that represents the parameter that is normally associated with the command that is specified in `dwCommand`. For example, if `dwCommand` is WFS_CMD_CIP_IO then `lpCmdData` will point to a WFSIDCCHIPIO structure. This parameter can be NULL if no command input parameter is needed or if this detail is not needed to synchronize for the command.

It will be device-dependent whether the synchronization is effective or not in the case where the application synchronizes for a command with this command specifying a parameter but subsequently executes the synchronized command with a different parameter. This case should not result in an error; however, the preparation effect could be different from what the application expects. The application should, therefore, make sure to use the same parameter between `lpCmdData` of this command and the subsequent corresponding execute command.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_ERR_IDC_COMMANDUNSUPP</td>
<td>The command specified in the <code>dwCommand</code> field is not supported by the Service Provider.</td>
</tr>
<tr>
<td>WFS_ERR_IDC_SYNCHRONIZEUNSUPP</td>
<td>The preparation for the command specified in the <code>dwCommand</code> with the parameter specified in the <code>lpCmdData</code> is not supported by the Service Provider.</td>
</tr>
</tbody>
</table>

Events

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

Comments

For sample flows of this synchronization see the [Ref 1] Appendix C.
6. Events

6.1 WFS_EXEE_IDC_INVALIDTRACKDATA

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

Event Param
LPWFSIDCTRACKEVENT lpTrackEvent;

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

fwStatus
Status of reading the track. Possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_DATAMISSING</td>
<td>The track is blank.</td>
</tr>
<tr>
<td>WFS_IDC_DATAINVALID</td>
<td>The data contained on the track is invalid.</td>
</tr>
<tr>
<td>WFS_IDC_DATATOOLONG</td>
<td>The data contained on the track is too long.</td>
</tr>
<tr>
<td>WFS_IDC_DATATOOSHORT</td>
<td>The data contained on the track is too short.</td>
</tr>
</tbody>
</table>

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

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

Comments
None.
6.2 WFS_EXEE_IDC_MEDIAINSERTED

<table>
<thead>
<tr>
<th>Description</th>
<th>This execute event specifies that a card was inserted into the device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Param</td>
<td>None.</td>
</tr>
<tr>
<td>Comments</td>
<td>None.</td>
</tr>
</tbody>
</table>
### 6.3 WFS_SRVE_IDC_MEDIAREMOVED

| **Description** | This service event specifies that the inserted card was manually removed by the user during the processing of a read/write command, during the processing of a chip_io/power command, during or after a retain/reset operation, after an eject operation or after the card is removed by the user in a latched dip card unit. |
| **Event Param** | None. |
| **Comments** | None. |
### 6.4 WFS_EXEE_IDC_MEDIARETAINED

<table>
<thead>
<tr>
<th>Description</th>
<th>This execute event specifies that the card was retained.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Param</td>
<td>None.</td>
</tr>
<tr>
<td>Comments</td>
<td>None.</td>
</tr>
</tbody>
</table>
6.5 WFS_EXEE_IDC_INVALIDMEDIA

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

Event Param
None.

Comments
None.
### 6.6 WFS_SRVE_IDC_CARDACTION

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

**Event Param**  
LPWFSIDCCARDACT lpCardAct;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CARDRETAINED</td>
<td>The card has been retained.</td>
</tr>
<tr>
<td>WFS_IDC_CARDEJECTED</td>
<td>The card has been ejected.</td>
</tr>
<tr>
<td>WFS_IDC_CARDREADPOSITION</td>
<td>The card has been moved to the read position.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_MEDIAUNKNOWN</td>
<td>The position of the card cannot be determined.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIAPRESENT</td>
<td>The card was present in the reader.</td>
</tr>
<tr>
<td>WFS_IDC_MEDIAENTERING</td>
<td>The card was entering the reader.</td>
</tr>
</tbody>
</table>

**Comments**  
None.
6.7 WFS_USRE_IDC_RETAINBINTHRESHOLD

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

Event Param
LPWORD lpfwRetainBin;

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_RETAINBINOK</td>
<td>The retain bin of the ID card unit was emptied.</td>
</tr>
<tr>
<td>WFS_IDC_RETAINBINFULL</td>
<td>The retain bin of the ID card unit is full.</td>
</tr>
<tr>
<td>WFS_IDC_RETAINBINHIGH</td>
<td>The retain bin of the ID card unit is nearly full.</td>
</tr>
</tbody>
</table>

Comments
None.
6.8 WFS_SRVE_IDC.MEDIADETECTED

**Description**
This service event is generated if media is detected during a reset (WFS_CMD_IDC_RESET). The parameter on the event informs the application of the position of the card on the completion of the reset. For devices with parking station capability there will be one event for each card found.

**Event Param**
LPWORD lpwResetOut;

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_CARDEJECTED</td>
<td>The card was ejected.</td>
</tr>
<tr>
<td>WFS_IDC_CARDRETAINED</td>
<td>The card was retained.</td>
</tr>
<tr>
<td>WFS_IDC_CARDREADPOSITION</td>
<td>The card is in read position.</td>
</tr>
<tr>
<td>WFS_IDC_CARDJAMMED</td>
<td>The card is jammed in the device.</td>
</tr>
</tbody>
</table>

**Comments**
None.
6.9  WFS_SRVE_IDC_RETAINBINREMOVED

<table>
<thead>
<tr>
<th>Description</th>
<th>This event specifies that the retain bin has been removed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Param</td>
<td>None.</td>
</tr>
<tr>
<td>Comments</td>
<td>None.</td>
</tr>
</tbody>
</table>
### 6.10 WFS_SRVE_IDC_RETAINBININSERTED

<table>
<thead>
<tr>
<th>Description</th>
<th>This event specifies that the retain bin has been inserted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Param</td>
<td>None.</td>
</tr>
<tr>
<td>Comments</td>
<td>None.</td>
</tr>
</tbody>
</table>
### 6.11 WFS_EXEE_IDC_INSERTCARD

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>This mandatory event notifies the application when the device is ready for the user to insert a card.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event Param</strong></td>
<td>None.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>None.</td>
</tr>
</tbody>
</table>
6.12 WFS_SRVE_IDCDEVICEPOSITION

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

**Event Param**
LPWFSIDCDEVICEPOSITION lpDevicePosition;

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_DEVICEINPOSITION</td>
<td>The device is in its normal operating position.</td>
</tr>
<tr>
<td>WFS_IDC_DEVICENOTINPOSITION</td>
<td>The device has been removed from its normal operating position.</td>
</tr>
<tr>
<td>WFS_IDC_DEVICEPOSUNKNOWN</td>
<td>The position of the device cannot be determined.</td>
</tr>
</tbody>
</table>

**Comments**
None.
6.13 WFS_SRVE_IDC_POWER_SAVE_CHANGE

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

Event Param
LPWFSIDCPowersavechange IpPowerSaveChange;
typedef struct _wfs_idc_power_save_change
{
    USHORT usPowerSaveRecoveryTime;
} WFSIDCPowersavechange, *LPWFSIDCPowersavechange;

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

Comments
If another device class compound with this device enters into a power saving mode this device will automatically enter into the same power saving mode and this event will be generated.
6.14 WFS_EXEE_IDC_TRACKDETECTED

**Description**
This execute event notifies the application what track data the inserted card has, before the reading of the data has completed. This event will be posted once when tracks are detected during card insertion.

**Event Param**
LPWFSIDCTrackDETECTED lpTrackDetected;

typedef struct _wfs_idc_track_detected
{
    WORD     fwTracks;
} WFSIDCTRACKDETECTED, *LPWFSIDCTRACKDETECTED;

fwTracks
Specifies which tracks are on the card, as a combination of the following flags (zero if there is no track on the inserted card):

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_IDC_TRACK1</td>
<td>The card has track 1.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK2</td>
<td>The card has track 2.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK3</td>
<td>The card has track 3.</td>
</tr>
<tr>
<td>WFS_IDC_TRACK_WM</td>
<td>The card has the Swedish Watermark track.</td>
</tr>
<tr>
<td>WFS_IDC_FRONT_TRACK_1</td>
<td>The card has the front track 1.</td>
</tr>
</tbody>
</table>

**Comments**
None.
6.15 WFS_EXEE_IDC_EMVCLESSREADSTATUS

Description
This execute event notifies that the communication (i.e. the commands exchanged linked to the tap) between the card and the intelligent contactless card reader are complete. The application can use this event to display intermediate messages, progress of card read, audio signals or anything else that might be required. The intelligent contactless card reader will continue the processing and the result of the processing will be returned in the out parameters of the WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION command.

Event Param
LPWFSIDCMVCLESSREADSTATUS lpReadStatus;

define struct _wfs_idc_emv_cless_read_status
{
    LPWFSIDCEMVLESSUI lpClessUI;
} WFSIDCMVCLESSREADSTATUS, *LPWFSIDCMVCLESSREADSTATUS;

lpClessUI
For details of this structure, see definition in the WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION command.

Comments
None.
7. Form Description

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

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

As an example the following registry information can be used:

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

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

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

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

<table>
<thead>
<tr>
<th>Reserved Keywords/Operands</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Form name delimiters.</td>
</tr>
<tr>
<td>TRACK1</td>
<td>Keyword to identify track 1.</td>
</tr>
<tr>
<td>TRACK2</td>
<td>Keyword to identify track 2.</td>
</tr>
<tr>
<td>TRACK3</td>
<td>Keyword to identify track 3.</td>
</tr>
<tr>
<td>TRACK1_JIS1</td>
<td>Keyword to identify JIS I track 1.</td>
</tr>
<tr>
<td>TRACK3_JIS1</td>
<td>Keyword to identify JIS I track 3.</td>
</tr>
<tr>
<td>FRONTTRACK1</td>
<td>Keyword to identify front track 1 (in some countries this track is known as JIS II track).</td>
</tr>
<tr>
<td>FIELDSEPT1</td>
<td>Value of field separator of track 1.</td>
</tr>
<tr>
<td>FIELDSEPT2</td>
<td>Value of field separator of track 2.</td>
</tr>
<tr>
<td>FIELDSEPT3</td>
<td>Value of field separator of track 3.</td>
</tr>
<tr>
<td>FIELDSEPTFRONT1</td>
<td>Value of field separator of front track 1.</td>
</tr>
<tr>
<td>FIELDSEPT1_JIS1</td>
<td>Value of field separator of JIS I track 1.</td>
</tr>
<tr>
<td>FIELDSEPT3_JIS1</td>
<td>Value of field separator of JIS I track 3.</td>
</tr>
<tr>
<td>READ</td>
<td>Description of read action; the track identifier keywords are processed left to right.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Description of write action.</td>
</tr>
<tr>
<td>ALL</td>
<td>Read or write the complete track.</td>
</tr>
<tr>
<td>SECURE</td>
<td>Do the security check via the security module (CIM86 or MM). This check is done on Track 3 only.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Read/write all tracks specified, abort reading on read failure.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 Attributes are not required in any mandatory order.
FIELDSEPPOSn  Position of the nth occurrence of field separator on track. FIELDSEPPOS0 specifies the beginning of the data. Separator in a list of logical fields.

DEFAULT  String for default substitution of track data to be written, that is not defined explicitly by the form fields. DEFAULT also allows an application to input fewer fields than those defined by the form.

ENDTRACK  Represents the end of the data. It is used to identify fields positioned after the last field separator.

Notes

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

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

The keywords FIELDSEPPOS0 and ENDTRACK are used as follows:

- read the first 2 bytes of a track:
  \[\text{FIRST} = \text{FIELDSEPPOS0} + 1, \text{FIELDSEPPOS0} + 2\]
- read the last 2 bytes of a track:
  \[\text{LAST} = \text{ENDTRACK} – 2, \text{ENDTRACK} – 1\]

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

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

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

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

Example 1  Reading tracks:

<table>
<thead>
<tr>
<th>READTRACK3GERMAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELDSEPT1=     = /* field separator of track 1 */</td>
</tr>
<tr>
<td>FIELDSEPT2=     = /* field separator of track 2 */</td>
</tr>
<tr>
<td>FIELDSEPT3=     = /* field separator of track 3 */</td>
</tr>
<tr>
<td>READ= TRACK3 &amp; TRACK1 &amp; TRACK2 /* all tracks must be read */</td>
</tr>
</tbody>
</table>

/* read logical fields as defined below; also check the security */
TRACK3= MII, COUNTRY, ISSUERID, ACCOUNT, LUHNT3, EXPIRATION, SECURE
MII= FIELDSEPPOS0 + 3, FIELDSEPPOS0 + 4
ISSUERID= FIELDSEPPOS3 + 5, FIELDSEPPOS3 + 1
ACCOUNT= FIELDSEPPOS0 + 1, FIELDSEPPOS0 + 2
LUHNT3= FIELDSEPPOS2 - 1, FIELDSEPPOS2 - 1
TRACK2= ALL /* return track 2 complete, don’t return logical fields*/
TRACK1= ALL /* return track 1 complete, don’t return logical fields*/
All tracks must be read (‘READ’), that is, the read fails if an error occurs on reading any one of the tracks (the ‘&’ operand). The field “major industry identifier” (‘MII’) is located after the first field separator (‘FIELDSEPPOS1’) and its length is two bytes. The “issuer identifier” field (‘ISSUERID’) is located after the MII field, with a length of eight bytes. The next field, “account number” (‘ACCOUNT’) is variable length; it ends before the luhn digit field (‘LUHNT3’) that is the last digit in front of the second field separator (‘FIELDSEPPOS2’).

Example 2

Write a track:

```plaintext
[WRITETRACK3]
FIELDSEP3=  =
DEFAULT= ? /* fields not specified in the write form are to be left unchanged, i.e. read and the same data written back to them */
WRITE= TRACK3
TRACK3= RETRYCOUNT, DATE
RETRYCOUNT= FIELDSEPPOS2 + 22, FIELDSEPPOS2 + 22
DATE= FIELDSEPPOS5 + 1, FIELDSEPPOS5 + 4
```

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

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

```
RETRYCOUNT=3
DATE=3132
```

Example 3

Write a track:

```plaintext
[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
```

Example 4

Reading tracks:

```plaintext
[READTRACK2ANDFRONTTRACK1]
READ= TRACK2&FRONTTRACK1 /* track 2 and front track 1 must be read */
TRACK2= ALL
FRONTTRACK1= ALL
```

Track 2 and Front track 1 are to be read by the WFS_CMD_IDC_READ_TRACK command. By specifying ‘&’, reading is aborted if either track fails to read. By specifying ‘ALL’ the physical track is read to the output data without field level formatting.

A sample of output data produced with this form is as follows:

```
TRACK2:ALL=123456789123
FRONTTRACK1:ALL=123456789123
```
8. C-Header file

/******************************************************************************
 * xfsidc.h    XFS - Identification card unit (IDC) definitions
 *             Version 3.30  (March 19 2015)
 ******************************************************************************/

#ifndef __INC_XFSIDC__H
#define __INC_XFSIDC__H

#ifdef __cplusplus
extern "C" {
#endif

#include <xfsapi.h>

/* values of WFSIDCCAPS.wClass */
#define WFS_SERVICE_CLASS_IDC               (2)
#define WFS_SERVICE_CLASS_NAME_IDC          "IDC"
#define WFS_SERVICE_CLASS_VERSION_IDC       (0x1E03) /* Version 3.30 */
#define IDC_SERVICE_OFFSET                  (WFS_SERVICE_CLASS_IDC * 100)

/* IDC Info Commands */
#define WFS_INF_IDC_STATUS                  (IDC_SERVICE_OFFSET + 1)
#define WFS_INF_IDC_CAPABILITIES            (IDC_SERVICE_OFFSET + 2)
#define WFS_INF_IDC_FORM_LIST               (IDC_SERVICE_OFFSET + 3)
#define WFS_INF_IDC_QUERY_FORM              (IDC_SERVICE_OFFSET + 4)
#define WFS_INF_IDC_QUERY_IFM_IDENTIFIER    (IDC_SERVICE_OFFSET + 5)
#define WFS_INF_IDC_EMVCLESS_QUERY_APPLICATIONS (IDC_SERVICE_OFFSET + 6)

/* 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_RAW_DATA          (IDC_SERVICE_OFFSET + 12)
#define WFS_CMD_IDC_CHIP_SET_GUIDANCE_LIGHT (IDC_SERVICE_OFFSET + 13)
#define WFS_CMD_IDC_CHIP_SAVE_CONTROL       (IDC_SERVICE_OFFSET + 14)
#define WFS_CMD_IDC_CHIP_PARK_CARD          (IDC_SERVICE_OFFSET + 15)
#define WFS_CMD_IDC_EMVCLESS_CONFIGURE      (IDC_SERVICE_OFFSET + 16)
#define WFS_CMD_IDC_EMVCLESS_PERFORM_TRANSACTION (IDC_SERVICE_OFFSET + 17)
#define WFS_CMD_IDC_EMVCLESS_ISSUERUPDATE   (IDC_SERVICE_OFFSET + 18)
#define WFS_CMD_IDC_SYNCHRONISE_COMMAND     (IDC_SERVICE_OFFSET + 19)

/* IDC Messages */
#define WFS_EXEE_IDC_INVALIDTRACKDATA       (IDC_SERVICE_OFFSET + 1)
#define WFS_EXEE_IDC_MEDAINserted           (IDC_SERVICE_OFFSET + 2)
#define WFS_SRVE_IDC_MEDIAREMOVED           (IDC_SERVICE_OFFSET + 3)
#define WFS_SRVE_IDC_CARDACTION             (IDC_SERVICE_OFFSET + 4)
#define WFS_USRE_IDC_RETAINBINTHRESHOLD     (IDC_SERVICE_OFFSET + 5)
#define WFS_EXEE_IDC_INVALIDMEDIA           (IDC_SERVICE_OFFSET + 7)
#define WFS_EXEE_IDC_MEDIARETAINED (IDC_SERVICE_OFFSET + 8)
#define WFS_SRVE_IDC_MEDIADETECTED (IDC_SERVICE_OFFSET + 9)
#define WFS_SRVE_IDC_RETAINBININSERTED (IDC_SERVICE_OFFSET + 10)
#define WFS_SRVE_IDC_RETAINBINREMOVED (IDC_SERVICE_OFFSET + 11)
#define WFS_EXEE_IDC_INSERTCARD (IDC_SERVICE_OFFSET + 12)
#define WFS_SRVE_IDC_DEVICEPOSITION (IDC_SERVICE_OFFSET + 13)
#define WFS_SRVE_IDC_POWER_SAVE_CHANGE (IDC_SERVICE_OFFSET + 14)
#define WFS_EXEE_IDC_TRACKDETECTED (IDC_SERVICE_OFFSET + 15)
#define WFS_EXEE_IDC_EMVCLESSREADSTATUS (IDC_SERVICE_OFFSET + 16)

/* values of WFSIDCSTATUS.fwDevice */
#define WFS_IDC_DEVONLINE WFS_STAT_DEVONLINE
#define WFS_IDC_DEVOFFLINE WFS_STAT_DEVOFFLINE
#define WFS_IDC_DEVPOWEROFF WFS_STAT_DEVPOWEROFF
#define WFS_IDC_DEVNODEVICE WFS_STAT_DEVNODEVICE
#define WFS_IDC_DEVHWERROR WFS_STAT_DEVHWERROR
#define WFS_IDC_DEVUSERERROR WFS_STAT_DEVUSERERROR
#define WFS_IDC_DEVBUSY WFS_STAT_DEVBUSY
#define WFS_IDC_DEVFRAUDATTEMPT WFS_STAT_DEVFRAUDATTEMPT
#define WFS_IDC_DEVPOTENTIALFRAUD WFS_STAT_DEVPOTENTIALFRAUD

/* values of WFSIDCSTATUS.fwMedia, WFSIDCRETAINCARD.fwPosition, WFSIDCCARDACT.wPosition, WFSIDCSTATUS.lpwParkingStationMedia */
#define WFS_IDC_MEDIAPRESENT (1)
#define WFS_IDC_MEDIANOTPRESENT (2)
#define WFS_IDC_MEDIAJAMMED (3)
#define WFS_IDC_MEDIANOTSUPP (4)
#define WFS_IDC_MEDIAUNKNOWN (5)
#define WFS_IDC_MEDIAENTERING (6)
#define WFS_IDC_MEDIALATCHED (7)

/* values of WFSIDCSTATUS.fwRetainBin */
#define WFS_IDC_RETAINBINOK (1)
#define WFS_IDC_RETAINNOTSUPP (2)
#define WFS_IDC_RETAINBINFULL (3)
#define WFS_IDC_RETAINBINHIGH (4)
#define WFS_IDC_RETAINBINMISSING (5)

/* values of WFSIDCSTATUS.fwSecurity */
#define WFS_IDC_SECNOTSUPP (1)
#define WFS_IDC_SECNOTSUPP (2)
#define WFS_IDC_SECOPEN (3)

/* values of WFSIDCSTATUS.fwChipPower */
#define WFS_IDC_CHIPONLINE (0)
#define WFS_IDC_CHIPPOWEREDOFF (1)
#define WFS_IDC_CHIPBUSY (2)
#define WFS_IDC_CHIPNODEVICE (3)
#define WFS_IDC_CHIPHWERROR (4)
#define WFS_IDC_CHIPNOCARD (5)
#define WFS_IDC_CHIPNOTSUPP (6)
#define WFS_IDC_CHIPUNKNOWN (7)

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

/* Indices of WFSIDCSTATUS.dwGuidLights [...] WFSIDCCAPS.dwGuidLights [...] */
#define WFS_IDC_GUIDANCE_CARDUNIT (0)
/* Values of WFSIDCSTATUS.dwGuidLights [...] */
#define WFS_IDC_GUIDANCE_NOT_AVAILABLE (0x00000000)
#define WFS_IDC_GUIDANCE_OFF (0x00000001)
#define WFS_IDC_GUIDANCE_ON (0x00000002)
#define WFS_IDC_GUIDANCE_SLOW_FLASH (0x00000004)
#define WFS_IDC_GUIDANCE_MEDIUM_FLASH (0x00000008)
#define WFS_IDC_GUIDANCE_QUICK_FLASH (0x00000010)
#define WFS_IDC_GUIDANCE_CONTINUOUS (0x00000080)
#define WFS_IDC_GUIDANCE_RED (0x00000100)
#define WFS_IDC_GUIDANCE_GREEN (0x00000200)
#define WFS_IDC_GUIDANCE_YELLOW (0x00000400)
#define WFS_IDC_GUIDANCE_BLUE (0x00000800)
#define WFS_IDC_GUIDANCE_CYAN (0x00001000)
#define WFS_IDC_GUIDANCE_MAGENTA (0x00002000)
#define WFS_IDC_GUIDANCE_WHITE (0x00004000)
#define WFS_IDC_GUIDANCE_ENTRY (0x00100000)
#define WFS_IDC_GUIDANCE_EXIT (0x00200000)

/* Values of WFSIDCSTATUS.fwChipModule */
#define WFS_IDC_CHIPMODOK (1)
#define WFS_IDC_CHIPMODINOP (2)
#define WFS_IDC_CHIPMODUNKNOWN (3)
#define WFS_IDC_CHIPMODNOTSUPP (4)

/* Values of WFSIDCSTATUS.fwMagReadModule and */
#define WFS_IDC_MAGMODOK (1)
#define WFS_IDC_MAGMODINOP (2)
#define WFS_IDC_MAGMODUNKNOWN (3)
#define WFS_IDC_MAGMODNOTSUPP (4)

/* Values of WFSIDCSTATUS.fwFrontImageModule and */
#define WFS_IDC_IMGMODOK (1)
#define WFS_IDC_IMGMODINOP (2)
#define WFS_IDC_IMGMODUNKNOWN (3)
#define WFS_IDC_IMGMODNOTSUPP (4)

/* Values of WFSIDCSTATUS.wDevicePosition */
#define WFS_IDC_DEVICEINPOSITION (0)
#define WFS_IDC_DEVICENOTINPOSITION (1)
#define WFS_IDCDEVICEPOSUNKNOWN (2)
#define WFS_IDCDEVICEPOSNOTSUPP (3)

/* Values of WFSIDCCAPS.fwType */
#define WFS_IDC_TYPEMOTOR (1)
#define WFS_IDC_TYPESWIPE (2)
#define WFS_IDC_TYPEDIP (3)
#define WFS_IDC_TYPECONTACTLESS (4)
#define WFS_IDC_TYPETALLENGITEDDIP (5)
#define WFS_IDC_TYPEPERMANENT (6)
#define WFS_IDC_TYPEINTELLIGENTCONTACTLESS (7)

/* Values of WFSIDCCAPS.fwReadTracks, */
#define WFS_IDC_NOTSUPP 0x0000
/* values of WFSIDCCAPS.fwReadTracks, WFSIDCCAPS.fwWriteTracks, WFSIDCCARDDATA.wDataSource, WFS_CMD_IDC_READ_RAW_DATA */

#define WFS_IDC_TRACK1          0x0001
#define WFS_IDC_TRACK2          0x0002
#define WFS_IDC_TRACK3          0x0004
#define WFS_IDC_FRONT_TRACK_L   0x0080
#define WFS_IDC_TRACK1_JIS1     0x0400
#define WFS_IDC_TRACK3_JIS1     0x0800

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

#define WFS_IDC_CHIP            0x0008
#define WFS_IDC_SECURITY        0x0010
#define WFS_IDC_FLUXINACTIVE    0x0020
#define WFS_IDC_TRACK_WM        0x8000
#define WFS_IDC_MEMORY_CHIP     0x0040
#define WFS_IDC_FRONTIMAGE      0x0100
#define WFS_IDC_BACKIMAGE       0x0200
#define WFS_IDC_DDI             0x4000

/* values of WFSIDCCAPS.fwChipProtocols */

#define WFS_IDC_CHIP0           0x0001
#define WFS_IDC_CHIP1           0x0002
#define WFS_IDC_CHIP_PROTOCOL_NOT_REQUIRED 0x0004
#define WFS_IDC_CHIPTYPEA_PART3 0x0008
#define WFS_IDC_CHIPTYPEA_PART4 0x0010
#define WFS_IDC_CHIPTYPEB       0x0020
#define WFS_IDC_CHIPNFC         0x0040

/* values of WFSIDCCAPS.fwSecType */

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

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

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

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

/* Note: WFS_IDC_UNKNOWN was removed as it was an invalid value */

#define WFS_IDC_LOCO           0x0002
#define WFS_IDC_HICO           0x0004
#define WFS_IDC_AUTO           0x0008

/* values of WFSIDCCAPS.fwChipPower */

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

/* values of WFSIDCCAPS.fwDIPMode */

#define WFS_IDC_DIP_UNKNOWN     0x0001
#define WFS_IDC_DIP_EXIT        0x0002
#define WFS_IDC_DIP_ENTRY       0x0004
#define WFS_IDC_DIP_ENTRY_EXIT  0x0008
/* values of WFSIDCCAPS.lpwMemoryChipProtocols */
#define WFS_IDC_MEM_SIEMENS4442 0x0001
#define WFS_IDC_MEM_GPM896 0x0002

/* values of WFSIDCFORM.fwAction */
#define WFS_IDC_ACTIONREAD 0x0001
#define WFS_IDC_ACTIONWRITE 0x0002

/* values of WFSIDCTRACKEVENT.fwStatus, WFSIDCCARDDATA.wStatus */
#define WFS_IDC_DATAOK (0)
#define WFS_IDC_DATAMISSING (1)
#define WFS_IDC_DATAINVALID (2)
#define WFS_IDC_DATATOOLONG (3)
#define WFS_IDC_DATATOOSHORT (4)
#define WFS_IDC_DATASRCNOTSUPP (5)
#define WFS_IDC_DATASRCMISSING (6)

/* values of WFSIDCCARDACT.wAction */
#define WFS_IDC_CARDRETAINED (1)
#define WFS_IDC_CARDEJECTED (2)
#define WFS_IDC_CARDREADPOSITION (3)
#define WFS_IDC_CARDJAMMED (4)

/* values of WFSIDCCARDDATA.lpbData if security is read */
#define WFS_IDC_SEC_READLEVEL1 '1'
#define WFS_IDC_SEC_READLEVEL2 '2'
#define WFS_IDC_SEC_READLEVEL3 '3'
#define WFS_IDC_SEC_READLEVEL4 '4'
#define WFS_IDC_SEC_READLEVEL5 '5'
#define WFS_IDC_SEC_BADREADLEVEL '6'
#define WFS_IDC_SEC_NODATA '7'
#define WFS_IDC_SEC_DATAINVAL '8'
#define WFS_IDC_SEC_HWERROR '9'
#define WFS_IDC_SEC_NOINIT 'A'

/* values of WFSIDCIFMIDENTIFIER.wIFMAuthority */
#define WFS_IDC_IFMEMV (1)
#define WFS_IDC_IFMEUROPAY (2)
#define WFS_IDC_IFMVISA (3)
#define WFS_IDC_IFMGIECB (4)

/* values of WFSIDCCAPS.fwEjectPosition, WFSIDCEJECTCARD.wEjectPosition */
#define WFS_IDC_EXITPOSITION (0x0001)
#define WFS_IDC_TRANSPORTPOSITION (0x0002)

/* values of WFSIDCPARKCARD.wDirection */
#define WFS_IDC_PARK_IN 0x0001
#define WFS_IDC_PARK_OUT 0x0002

/* values of WFSIDCSTATUS.wAntiFraudModule */
#define WFS_IDC_AFMNOTSUPP (0)
#define WFS_IDC_AFMOK (1)
#define WFS_IDC_AFMINOP (2)
#define WFS_IDC_AFMDEVICEDETECTED (3)
#define WFS_IDC_AFMUNKNOWN (4)

/* values of WFSIDCEMVCLESSDATA.wTxOutcome */
#define WFS_IDC_CLESS_MULTIPLECARDS (0)
#define WFS_IDC_CLESS_APPROVE (1)
#define WFS_IDC_CLESSDECLINE (2)
#define WFS_IDC_CLESS_ONLINEREQUEST (3)
#define WFS_IDC_CLESS_ONLINEREQUESTCOMPLETIONREQUIRED (4)
#define WFS_IDC_CLESS_TRYAGAIN (5)
#define WFS_IDC_CLESS_TRYANOTHERINTERFACE (6)
#define WFS_IDC_CLESS_ENDAPPLICATION (7)
#define WFS_IDC_CLESS_CONFIRMATIONREQUIRED (8)

/* values of WFSIDCEMVCLESSOUTCOME.wCardholderAction */
#define WFS_IDC_CLESS_NOACTION (0)
#define WFS_IDC_CLESS_RETAP (1)
#define WFS_IDC_CLESS_HOLDCARD (2)

/* values of WFSIDCEMVCLESSOUTCOME.wCVM */
#define WFS_IDC_CLESS_ONLINEPIN (0)
#define WFS_IDC_CLESS_CONFIRMATIONCODEVERIFIED (1)
#define WFS_IDC_CLESS_SIGN (2)
#define WFS_IDC_CLESS_NOCVM (3)
#define WFS_IDC_CLESS_NOCVMPREFERENCE (4)

/* values of WFSIDCEMVCLESSOUTCOME.wAlternateInterface */
#define WFS_IDC_CLESS_CONTACT (0)
#define WFS_IDC_CLESS_MAGNETICSTRIPE (1)

/* values of WFSIDCEMVCLESSUI.wStatus */
#define WFS_IDC_CLESS_NOT_READY (0)
#define WFS_IDC_CLESS_IDLE (1)
#define WFS_IDC_CLESSREADYTOREAD (2)
#define WFS_IDC_CLESS_PROCESSING (3)
#define WFS_IDC_CLESS_CARDREADOK (4)
#define WFS_IDC_CLESS_PROCESSINGERROR (5)

/* values of WFSIDCEMVCLESSUI.wValueQualifier */
#define WFS_IDC_CLESS_AMOUNT (0)
#define WFS_IDC_CLESS_BALANCE (1)

/* 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_FORMATNOTFOUND (- (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_ATRNTOBTAINED (- (IDC_SERVICE_OFFSET + 12))
#define WFS_ERR_IDC_INVALIDKEY (- (IDC_SERVICE_OFFSET + 13))
#define WFS_ERR_IDC_CHIPPOWERNOTSUPP (- (IDC_SERVICE_OFFSET + 14))
#define WFS_ERR_IDC_CARDTOO_SHORT (- (IDC_SERVICE_OFFSET + 16))
#define WFS_ERR_IDC_CARDTOO_LONG (- (IDC_SERVICE_OFFSET + 17))
#define WFS_ERR_IDC_INVALID_PORT (- (IDC_SERVICE_OFFSET + 18))
#define WFS_ERR_IDC_POWERSAVE_VETO_SHORT (- (IDC_SERVICE_OFFSET + 19))
#define WFS_ERR_IDC_POWERSAVE_MEDIA_PRESENT (- (IDC_SERVICE_OFFSET + 20))
#define WFS_ERR_IDC_CARD_PRESENT (- (IDC_SERVICE_OFFSET + 21))
#define WFS_ERR_IDC_POSITION_INVALID (- (IDC_SERVICE_OFFSET + 22))
#define WFS_ERR_IDC_INVALIDTERMINALDATA (- (IDC_SERVICE_OFFSET + 23))
```c
#define WFS_ERR_IDC_INVALIDAIDDATA             (-(IDC_SERVICE_OFFSET + 24))
#define WFS_ERR_IDC_INVALIDKEYDATA             (-(IDC_SERVICE_OFFSET + 25))
#define WFS_ERR_IDC_READERNOTCONFIGURED        (-(IDC_SERVICE_OFFSET + 26))
#define WFS_ERR_IDC_TRANSACTIONNOTINITIATED    (-(IDC_SERVICE_OFFSET + 27))
#define WFS_ERR_IDC_COMMANDUNSUPP              (-(IDC_SERVICE_OFFSET + 28))
#define WFS_ERR_IDC_SYNCHRONIZEUNSUPP         (-(IDC_SERVICE_OFFSET + 29))

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

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

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

typedef struct _wfs_idc_form
{
    LPSTR                    lpszFormName;
    CHAR                     cFieldSeparatorTrack1;
    CHAR                     cFieldSeparatorTrack2;
    CHAR                     cFieldSeparatorTrack3;
    WORD                     fwAction;
    LPSTR                    lpszTracks;
    BOOL                     bSecure;
```
typedef struct _wfs_idc_ifm_identifier
{
    WORD            wIFMAuthority;
    LPSTR           lpszIFMIdentifier;
} WFSIDCIFMIDENTIFIER, *LPWFSIDCIFMIDENTIFIER;

typedef struct _wfs_idc_hex_data
{
    ULONG            ulLength;
    LPBYTE           lpbData;
} WFSIDCHEXDATA, *LPWFSIDCHEXDATA;

typedef struct wfs_idc_app_data
{
    LPWFSIDCHEXDATA  lpAID;
    LPWFSIDCHEXDATA  lpKernelIdentifier;
} WFSIDCAPPDATA, *LPWFSIDCAPPDATA;

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

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

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

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

typedef struct _wfs_idc_park_card
{
    WORD wDirection;
    USHORT usParkingStation;
} WFSIDCPARKCARD, *LPWFSIDCPARKCARD;

typedef struct _wfs_idc_aid_data
{
    LPWFSIDCHEXDATA lpAID;
    BOOL bPartialSelection;
    ULONG ulTransactionType;
    LPWFSIDCHEXDATA lpKernelIdentifier;
    LPWFSIDCHEXDATA lpConfigData;
} WFSIDCAIDDATA, *LPWFSIDCAIDDATA;

typedef struct _wfs_idc_key_data
{
    LPWFSIDCHEXDATA lpRID;
    WORD wCAPublicKeyIndex;
    WORD wAPublicKeyAlgorithmIndicator;
    LPWFSIDCHEXDATA lpCAPublicKeyExponent;
    LPWFSIDCHEXDATA lpCAPublicKeyModulus;
    LPBYTE lpbCAPublicKeyCheckSum;
} WFSIDCKEYDATA, *LPWFSIDCKEYDATA;

typedef struct _wfs_idc_emvcless_config_data
{
    LPWFSIDCHEXDATA lpTerminalData;
    LPWFSIDCAIDDATA *lppAIDData;
    LPWFSIDCKEYDATA *lppKeyData;
} WFSIDCEMVCLESSCONFIGDATA, *LPWFSIDCEMVCLESSCONFIGDATA;

typedef struct _wfs_idc_emvcless_tx_data
{
    LPWFSIDCHEXDATA lpData;
} WFSIDCEMVCLESTXDATA, *LPWFSIDCEMVCLESTXDATA;

typedef struct _wfs_idc_emvcless_ui
{
    WORD wMessageId;
    WORD wStatus;
    ULONG ulHoldTime;
    WORD wValueQualifier;
}
typedef struct _wfs_idc_emvcless_outcome
{
    WORD wCVM;
    WORD wAlternateInterface;
    BOOL bReceipt;
    LPWFSIDCEMVCLESSUI lpClessUIOutcome;
    LPWFSIDCEMVCLESSUI lpClessUIRestart;
    ULONG ulClessFieldOffHoldTime;
    ULONG ulCardRemovalTimeoutValue;
    LPWFSIDCHEXDATA lpDiscretionaryData;
} WFSIDCEMVCLESSOUTCOME, *LPWFSIDCEMVCLESSOUTCOME;

typedef struct _wfs_idc_emvcless_tx_data_output
{
    WORD wDataSource;
    WORD wTxOutcome;
    WORD wCardholderAction;
    LPWFSIDCHEXDATA lpDataRead;
    LPWFSIDCEMVCLESSOUTCOME lpClessOutcome;
} WFSIDCEMVCLESSTXDATAOUTPUT, *LPWFSIDCEMVCLESSTXDATAOUTPUT;

typedef struct _wfs_idc_synchronize_command
{
    DWORD dwCommand;
    LPVOID lpCmdData;
} WFSIDCSYNCHRONIZECOMMAND, *LPWFSIDCSYNCHRONIZECOMMAND;

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

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

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

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

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

typedef struct _wfs_idc_track_detected
{
    WORD fwTracks;
} WFSIDCTRACKDETECTED, *LPWFSIDCTRACKDETECTED;

typedef struct _wfs_idc_emv_cless_read_status
{
    LPWFSIDCEMVCLESSUI lpClessUI;
} WFSIDCMVCLESSREADSTATUS, *LPWFSIDCMVCLESSREADSTATUS;
/* restore alignment */
#pragma pack(pop)
#endif /* __cplusplus */
#endif /* __INC_XFSIDC_H */
9. Intelligent Contactless Card Sequence Diagrams

This section illustrates the sequence diagrams of EMV-like intelligent contactless transactions.
9.1 Single Tap Transaction Without Issuer Update Processing

Figure 1 - Single tap transaction with no issuer update data received from host
9.2 Double Tap Transaction With Issuer Update Processing

Figure 2 - Double tap transaction with issuer update data received from host