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

WORKSHOP AGREEMENT

CWA 14050-6

November 2000

ICS 35.200; 35.240.15; 35.240.40

Extensions for Financial Services (XFS) interface specification - Release 3.0 - Part 6: Pin Keypad Device Class Interface

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

Ref. No CWA 14050-6:2000 E

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

Table of Contents

Foreword

This CWA is revision 3.0 of the XFS interface specification.

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

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

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

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

The clear direction of the CEN/ISSS XFS Workshop, therefore, is the delivery of a new Release 3.0 specification based on a C API. It will be delivered with the promise of the protection of technical investment for existing applications and the design to safeguard future developments.

The CEN/ISSS XFS Workshop gathers suppliers as well as banks and other financial service companies. A list of companies participating in this Workshop and in support of this CWA is available from the CEN/ISSS Secretariat.

This CWA was formally approved by the XFS Workshop meeting on 2000-10-18. The specification is continuously reviewed and commented in the CEN/ISSS Workshop on XFS. It is therefore expected that an update of the specification will be published in due time as a CWA, superseding this revision 3.0.

The CWA is published as a multi-part document, consisting of:

Part 1: Application Programming Interface (API) - Service Provider Interface (SPI); Programmer's Reference

Part 2: Service Classes Definition; Programmer's Reference

Part 3: Printer Device Class Interface - Programmer's Reference

Part 4: Identification Card Device Class Interface - Programmer's Reference

Part 5: Cash Dispenser Device Class Interface - Programmer's Reference

Part 6: PIN Keypad Device Class Interface - Programmer's Reference

Part 7: Check Reader/Scanner Device Class Interface - Programmer's Reference

Part 8: Depository Device Class Interface - Programmer's Reference

Part 9: Text Terminal Unit Device Class Interface - Programmer's Reference

Part 10: Sensors and Indicators Unit Device Class Interface - Programmer's Reference

Part 11: Vendor Dependent Mode Device Class Interface - Programmer's Reference

Part 12: Camera Device Class Interface - Programmer's Reference

Part 13: Alarm Device Class Interface - Programmer's Reference

Part 14: Card Embossing Unit Class Interface - Programmer's Reference

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

Part 16: Application Programming Interface (API) - Service Provider Interface (SPI) - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

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

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

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

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

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

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

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

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

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

In addition to these Programmer's Reference specifications, the reader of this CWA is also referred to a complementary document, called Release Notes. The Release Notes contain clarifications and explanations on the CWA specifications, which are not requiring functional changes. The current version of the Release Notes is available online from http://www.cenorm.be/isss/Workshop/XFS.

The information in this document represents the Workshop's current views on the issues discussed as of the date of publication. It is furnished for informational purposes only and is subject to change without notice. CEN/ISSS makes no warranty, express or implied, with respect to this document.

Revision History:

1.0	May 24, 1993	Initial release of API and SPI specification
1.11	February 3, 1995	Separation of specification into separate documents for API/SPI and service class definitions
2.00	November 11, 1996	Update release encompassing the self-service environment
3.00	October 18, 2000	Update release encompassing: - new commands to support the German ZKA chip card standard – support of Banksys Security Control Module - Added clarification note for Pin format 3624 - Added WFS_CMD_PIN_ENC_IO, which is currently used for the swiss proprietary protocol only. - Double and triple zero clarification in WFS_CMD_PIN_GET_DATA - key deletion in WFS_CMD_PIN_IMPORT_KEY inserted. For a detailed description see CWA 14050-20 PIN Migration from Version 2.00 to Version 3.00, Revision 1.00, October 18, 2000.

1. Introduction

1.1 Background to Release 3.0

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

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

The move from an XFS 2.0 specification to a 3.0 specification has been prompted by a series of factors. Initially, there has been a technical imperative to extend the scope of the existing specification of the XFS Manager to include new devices, such as the Card Embossing Unit.

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

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

The clear direction of the XFS Workshop, therefore, is the delivery of a new Release 3.0 specification based on a C API. It will be delivered with the promise of the protection of technical investment for existing applications and the design to safeguard future developments.

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 the Extensions for Financial Services is to standardize command codes and structures for the broadest variety of services. For example, using the **WFSExecute** function, the commands to read data from various services are as similar as possible to each other in their syntax and data structures.

In general, the specific command set for a service class is defined as the union 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. Personal Identification Number (PIN) Keypads

This section describes the application program interface for personal identification number keypads (PIN pads) and other encryption/decryption devices. This description includes definitions of the service-specific commands that can be issued, using the **WFSAsyncExecute**, **WFSExecute**, **WFSGetInfo** and **WFSAsyncGetInfo** functions.

This section describes the general interface for the following functions:

- Administration of encryption devices
- Loading of encryption keys
- Encryption / decryption
- Entering Personal Identification Numbers (PINs)
- PIN verification
- PIN block generation (encrypted PIN)
- Clear text data handling
- Function key handling
- PIN presentation to chipcard
- Read and write safety critical Terminal Data from/to HSM
- HSM and Chipcard Authentication

If the PIN Pad device has local display capability, display handling should be handled using the Text Terminal Unit (TTU) interface.

The adoption of this specification does not imply the adoption of a specific security standard.

Important Notes:

- This revision of this specification does not define key management procedures; key management is vendor-specific.
- Key space management is customer-specific, and is therefore handled by vendor-specific mechanisms.
- Only numeric PIN pads are handled in this specification.

This specification also supports the Hardware Security Module (HSM), which is necessary for the German ZKA Electronic Purse transactions. Furthermore the HSM stores terminal specific data.
This data will be compared against the message data fields (Sent and Received ISO8583 messages) prior to HSM-MAC generation/verification. HSM-MACs are generated/verified only if the message fields match the data stored.

Keys used for cryptographic HSM functions are stored separate from other keys. This must be considered when importing keys.

This version of PinPad complies to the current ZKA specification 3.0. It supports loading and unloading against card account for both card types (Type 0and Type 1) of the ZKA electronic purse. It also covers the necessary functionality for ‘Loading against other legal tender’.

Key values are passed to the API as binary hexadecimal values, for example:

0123456789ABCDEF = 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF

3. References

1. XFS Application Programming Interface (API)/Service Provider Interface (SPI), Programmer’s Reference Revision 3.00, October 18, 2000

4. Info Commands

4.1 WFS_INF_PIN_STATUS

Description The WFS_INF_PIN_STATUS command returns several kinds of status information.

Input Param None.

Output Param

```
LPWFSPINSTATUS      lpStatus;

typedef struct _wfs_pin_status
   {
   WORD            fwDevice;
   WORD            fwEncStat;
   LPSTR           lpszExtra;
   } WFSPINSTATUS, * LPWFSPINSTATUS;
```

fwDevice
Specifies the state of the PIN pad device as one of the following flags:

Value	Meaning
WFS_PIN_DEVONLINE	The device is online (i.e. powered on and operable).
WFS_PIN_DEVOFFLINE	The device is offline (e.g., the operator has taken the device offline by turning a switch or pulling out the device).
WFS_PIN_DEVPOWEROFF	The device is powered off or physically not connected.
WFS_PIN_DEVNODEVICE	There is no device intended to be there; e.g. this type of self service machine does not contain such a device or it is internally not configured.
WFS_PIN_DEVHWERROR	The device is inoperable due to a hardware error.
WFS_PIN_DEVUSERERROR	The device is present but a person is preventing proper device operation.
WFS_PIN_DEVBUSY	The device is busy and unable to process an execute command at this time.

fwEncStat
Specifies the state of the Encryption Module as one of the following flags:

Value	Meaning
WFS_PIN_ENCREADY	The encryption module is initialized and ready (at least one key is imported into the encryption module).
WFS_PIN_ENCNOTREADY	The encryption module is not ready.
WFS_PIN_ENCNOTINITIALIZED	The encryption module is not initialized (no master key loaded).
WFS_PIN_ENCBUSY	The encryption module is busy (implies that the device is busy).
WFS_PIN_ENCUNDEFINED	The encryption module state is undefined.
WFS_PIN_ENCINITIALIZED	The encryption module is initialized and master key (where required) and any other initial keys are loaded; ready to import other keys.

lpszExtra
Specifies 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 extendable by service providers. Each string will be null-terminated, with the final string terminating with two null characters.

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.

4.2 WFS_INF_PIN_CAPABILITIES

Description This command is used to retrieve the capabilities of the PIN pad.

Input Param None.

Output Param `LPWFSPINCAPS lpCaps;`

```
typedef struct _wfs_pin_caps
   {
   WORD         wClass;
   WORD         fwType;
   BOOL         bCompound;
   USHORT       usKeyNum;
   WORD         fwAlgorithms;
   WORD         fwPinFormats;
   WORD         fwDerivationAlgorithms;
   WORD         fwPresentationAlgorithms;
   WORD         fwDisplay;
   BOOL         bIDConnect;
   WORD         fwIDKey;
   WORD         fwValidationAlgorithms;
   WORD         fwKeyCheckModes;
   LPSTR        lpszExtra;
   } WFSPINCAPS, * LPWFSPINCAPS;
```

wClass
Specifies the logical service class, value is:
WFS_SERVICE_CLASS_PIN

fwType
Specifies the type of the PIN pad security module as a combination of the following flags. PIN entry is only possible when at least WFS_PIN_TYPEEPP and WFS_PIN_TYPEEDM are set. In order to use the ZKA-Electronic purse, all flags must be set.

Value	Meaning
WFS_PIN_TYPEEPP	electronic PIN pad (keyboard data entry device)
WFS_PIN_TYPEEDM	encryption/decryption module
WFS_PIN_TYPEHSM	hardware security module (electronic PIN pad and encryption module within the same physical unit)

bCompound
Specifies whether the logical device is part of a compound physical device and is either TRUE or FALSE.

usKeyNum
Number of the keys which can be stored in the encryption/decryption module.

fwAlgorithms
Supported encryption modes; a combination of the following flags:

Value	Meaning
WFS_PIN_CRYPTDESECB	Electronic Code Book
WFS_PIN_CRYPTDESCBC	Cipher Block Chaining
WFS_PIN_CRYPTDESCFB	Cipher Feed Back
WFS_PIN_CRYPTRSA	RSA Encryption
WFS_PIN_CRYPTECMA	ECMA Encryption
WFS_PIN_CRYPTDESMAC	MAC calculation using CBC
WFS_PIN_CRYPTTRIDESECB	Triple DES with Electronic Code Book
WFS_PIN_CRYPTTRIDESCBC	Triple DES with Cipher Block Chaining
WFS_PIN_CRYPTTRIDESCFB	Triple DES with Cipher Feed Back
WFS_PIN_CRYPTTRIDESMAC	Triple DES MAC calculation using CBC

fwPinFormats

Supported PIN formats; a combination of the following flags:

Value	Meaning
WFS_PIN_FORM3624	PIN left justified, filled with padding characters, PIN length 4-16 digits. The Padding Character is a Hexadecimal Digit in the range 0x00 to 0x0F.
WFS_PIN_FORMANSI	PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, minimum 12 digits without check number)
WFS_PIN_FORMISO0	PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, no minimum length specified, missing digits are filled with 0x00)
WFS_PIN_FORMISO1	PIN is preceded by 0x01 and the length of the PIN (0x04 to 0x0C), padding characters are taken from a transaction field (10 digits).
WFS_PIN_FORMECI2	(similar to WFS_PIN_FORM3624), PIN only 4 digits
WFS_PIN_FORMECI3	PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from X’0’ through X’F’.
WFS_PIN_FORMVISA	PIN is preceded by the length (digit), PIN length 4-6 digits. If the PIN length is less than six digits the PIN is filled with X’0’ to the length of six, the padding character can range from X ' 0 ' through X ' 9 ' (This format is also referred to as VISA2).
WFS_PIN_FORMDIEBOLD	PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted.
WFS_PIN_FORMDIEBOLDCO	PIN with the length of 4 to 12 digits, each one with a value of X’0’ to X’9’, is preceded by the one-digit coordination number with a value from X’0’ to X’F’, padded with the padding character with a value from X’0’ to X’F’ and may be not encrypted, single encrypted or double encrypted.
WFS_PIN_FORMVISA3	PIN with the length of 4 to 12 digits, each one with a value of X’0’ to X’9’, is followed by a delimiter with the value of X’F’ and then padded by the padding character with a value between X’0’ to X’F’.
WFS_PIN_FORMBANKSYS	PIN is encrypted and formatted according to the Banksys Pin Block specifications.

fwDerivationAlgorithms

Supported derivation algorithms; a combination of the following flags:

Value	Meaning
WFS_PIN_CHIP_ZKA	Algorithm for the derivation of a chip card individual key as described by the German ZKA.

fwPresentationAlgorithms

Supported presentation algorithms; a combination of the following flags:

Value	Meaning
WFS_PIN_PRESENT_CLEAR	Algorithm for the presentation of a clear text PIN to a chipcard.

fwDisplay

Specifies the type of the display used in the PIN pad module as one of the following flags:

Value	Meaning
WFS_PIN_DISPNONE	no display unit
WFS_PIN_DISPLEDTHROUGH	lights next to text guide user

WFS_PIN_DISPDISPLAY	a real display is available (this doesn't apply for self-service)

bIDConnect
Specifies whether the PIN pad is directly physically connected to the ID card unit. The value of this parameter is either TRUE or FALSE.

fwIDKey
Specifies whether an ID key is supported as a combination of the following flags:

Value	Meaning
WFS_PIN_IDKEYINITIALIZATION	ID key supported in the WFS_CMD_PIN_INITIALIZATION command.
WFS_PIN_IDKEYIMPORT	ID key supported in the WFS_CMD_PIN_IMPORT_KEY command.

fwValidationAlgorithms
Specifies the algorithms for PIN validation supported by the service; combination of the following flags:

Value	Meaning
WFS_PIN_DES	DES algorithm
WFS_PIN_EUROCHEQUE	EUROCHEQUE algorithm
WFS_PIN_VISA	VISA algorithm
WFS_PIN_DES_OFFSET	DES offset generation algorithm
WFS_PIN_BANKSYS	Banksys algorithm.

fwKeyCheckModes
Specifies the key check modes that are supported to check the correctness of an imported key value; can be a combination of the following flags:

Value	Meaning
WFS_PIN_KCVSELF	The key check value is created by an encryption of the key with itself.
WFS_PIN_KCVZERO	The key check value is created by an encryption of the key with a zero value.

lpszExtra
Points to a list of vendor-specific, or any other extended information. The information is returned as a series of "*key=value*" strings so that it is easily extendable by service providers. Each string is null-terminated, with the final string terminating with two null characters.

For German HSMs this parameter will contain the following information:

- HSM=<HSM vendor> (can contain the values KRONE,ASCOM,IBM or NCR)
- JOURNAL=<0/1> (0 means that the HSM does not support journaling by the WFS_CMD_PIN_GET_JOURNAL command, 1 means it supports journaling)

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 *lpsExtra* parameter may not be device or vendor-independent.

4.3 WFS_INF_PIN_KEY_DETAIL

Description This command returns detailed information about the keys in the encryption module.

Input Param `LPSTR  lpsKeyName;`

lpsKeyName
Name of the key for which detailed information is requested.
If NULL, detailed information about all the keys in the encryption module is returned.

Output Param `LPWFSPINKEYDETAIL * lppKeyDetail;`

Pointer to a null-terminated array of pointers to key detail structures.

```
typedef struct _wfs_pin_key_detail
   {
   LPSTR        lpsKeyName;
   WORD         fwUse;
   BOOL         bLoaded;
   } WFSPINKEYDETAIL, * LPWFSPINKEYDETAIL;
```

lpsKeyName
Specifies the name of the key.

fwUse
Specifies the type of access for which the key is used as a combination of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key can be used for encryption/decryption
WFS_PIN_USEFUNCTION	key can be used for PIN functions
WFS_PIN_USEMACING	key can be used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USENODUPLICATE	key can be imported only once
WFS_PIN_USESVENCKEY	key is used as CBC Start Value encryption key

bLoaded
Specifies whether the key has been loaded (imported from Application or locally from Operator) and is either TRUE or FALSE.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key name is not found.

Comments None.

4.4 WFS_INF_PIN_FUNCKEY_DETAIL

Description This command returns information about the names of the Function Keys supported by the device. Location information is also returned for the supported FDKs (Function Descriptor Keys). This includes screen overlay FDKs.

This command should be issued before the first call to WFS_CMD_PIN_GET_PIN or WFS_CMD_PIN_GET_DATA to determine which Function Keys (FKs) and Function Descriptor Keys (FDKs) are available and where the FDKs are located. Then, in these two commands, they can then be specified as Active and Terminate keys and options on the customer screen can be aligned with the active FDKs.

Input Param `LPULONG      lpulFDKMask;`

lpulFDKMask
Mask for the FDKs for which additional information is requested.
If 0x00000000, only information about function keys is returned.
If 0xFFFFFFFF, information about all the supported FDKs is returned.

Output Param `LPWFSPINFUNCKEYDETAIL      lpFuncKeyDetail;`

```
typedef struct _wfs_pin_func_key_detail
   {
   ULONG           ulFuncMask;
   USHORT          usNumberFDKs;
   LPWFSPINFDK     * lppFDKs;
   } WFSPINFUNCKEYDETAIL, * LPWFSPINFUNCKEYDETAIL;
```

ulFuncMask
Specifies the function keys available for this physical device as a combination of the following flags. The defines WFS_PIN_FK_0 through WFS_PIN_FK_9 correspond to numeric digits:

WFS_PIN_FK_0	(numeric digit 0)
WFS_PIN_FK_1	(numeric digit 1)
WFS_PIN_FK_2	(numeric digit 2)
WFS_PIN_FK_3	(numeric digit 3)
WFS_PIN_FK_4	(numeric digit 4)
WFS_PIN_FK_5	(numeric digit 5)
WFS_PIN_FK_6	(numeric digit 6)
WFS_PIN_FK_7	(numeric digit 7)
WFS_PIN_FK_8	(numeric digit 8)
WFS_PIN_FK_9	(numeric digit 9)
WFS_PIN_FK_ENTER	
WFS_PIN_FK_CANCEL	
WFS_PIN_FK_CLEAR	
WFS_PIN_FK_BACKSPACE	
WFS_PIN_FK_HELP	
WFS_PIN_FK_DECPOINT	
WFS_PIN_FK_00	
WFS_PIN_FK_000	
WFS_PIN_FK_RES1	(reserved for future use)
WFS_PIN_FK_RES2	(reserved for future use)
WFS_PIN_FK_RES3	(reserved for future use)
WFS_PIN_FK_RES4	(reserved for future use)
WFS_PIN_FK_RES5	(reserved for future use)
WFS_PIN_FK_RES6	(reserved for future use)
WFS_PIN_FK_RES7	(reserved for future use)
WFS_PIN_FK_RES8	(reserved for future use)

The remaining 6 bit masks may be used as vendor dependent keys.
WFS_PIN_FK_OEM1
WFS_PIN_FK_OEM2
WFS_PIN_FK_OEM3
WFS_PIN_FK_OEM4
WFS_PIN_FK_OEM5
WFS_PIN_FK_OEM6

usNumberFDKs
This value indicates the number of FDK structures returned. This number can be less than the number of keys requested, if any keys are not supported.

lppFDKs
Pointer to an array of pointers to FDK structures. It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

```
typedef struct _wfs_pin_fdk
    {
    ULONG     ulFDK;
    USHORT    usXPosition;
    USHORT    usYPosition;
    } WFSPINFDK, * LPWFSPINFDK;
```

ulFDK
Specifies the code returned by this FDK, defined as one of the following values:
WFS_PIN_FK_FDK01
WFS_PIN_FK_FDK02
WFS_PIN_FK_FDK03
WFS_PIN_FK_FDK04
WFS_PIN_FK_FDK05
WFS_PIN_FK_FDK06
WFS_PIN_FK_FDK07
WFS_PIN_FK_FDK08

WFS_PIN_FK_FDK09
WFS_PIN_FK_FDK10
WFS_PIN_FK_FDK11
WFS_PIN_FK_FDK12
WFS_PIN_FK_FDK13
WFS_PIN_FK_FDK14
WFS_PIN_FK_FDK15
WFS_PIN_FK_FDK16
WFS_PIN_FK_FDK17
WFS_PIN_FK_FDK18
WFS_PIN_FK_FDK19
WFS_PIN_FK_FDK20
WFS_PIN_FK_FDK21
WFS_PIN_FK_FDK22
WFS_PIN_FK_FDK23
WFS_PIN_FK_FDK24
WFS_PIN_FK_FDK25
WFS_PIN_FK_FDK26
WFS_PIN_FK_FDK27
WFS_PIN_FK_FDK28
WFS_PIN_FK_FDK29
WFS_PIN_FK_FDK30
WFS_PIN_FK_FDK31
WFS_PIN_FK_FDK32

usXPosition
For FDKs, specifies the FDK position relative to the Left Hand side of the screen expressed as a percentage of the width of the screen.

usYPosition
For FDKs, specifies the FDK position relative to the top of the screen expressed as a percentage of the height of the screen.

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

Comments None.

4.5 WFS_INF_PIN_HSM_TDATA

Description This function returns the current HSM terminal data. The data is returned as a series of “tag/length/value” items.

Input Param None.

Ouput Param `LPWFSXDATA    lpxTData;`

lpxTData
Contains the parameter settings as a series of “tag/length/value” items with no separators. See command WFS_CMD_PIN_HSM_SET_TDATA for the tags supported.

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

Comments None.

4.6 WFS_INF_PIN_KEY_DETAIL_EX

Description This command returns extended detailed information about the keys in the encryption module. Information like generation, version, activating and expiry date can be returned only for keys which are loaded via the WFS_CMD_PIN_SECURE_MSG_SEND command with WFS_PIN_PROTISOPS or a vendor dependant mechanism.

Input Param

```
LPSTR  lpsKeyName;
```

lpsKeyName
Name of the key for which detailed information is requested.
If NULL, detailed information about all the keys in the encryption module is returned.

Output Param

```
LPWFSPINKEYDETAILEX *      lppKeyDetailEx;
```

Pointer to a null-terminated array of pointers to key detail structures.

```
typedef struct _wfs_pin_key_detail_ex
   {
   LPSTR        lpsKeyName;
   DWORD        dwUse;
   BYTE         bGeneration;
   BYTE         bVersion;
   BYTE         bActivatingDate[4];
   BYTE         bExpiryDate[4];
   BOOL         bLoaded;
   } WFSPINKEYDETAILEX, * LPWFSPINKEYDETAILEX;
```

lpsKeyName
Specifies the name of the key.

dwUse
Specifies the type of access for which the key is used as a combination of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key can be used for encryption/decryption
WFS_PIN_USEFUNCTION	key can be used for PIN functions
WFS_PIN_USEMACING	key can be used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USENODUPLICATE	key can be imported only once
WFS_PIN_USESVENCKEY	key is used as CBC Start Value encryption key
WFS_PIN_USEPINLOCAL	key is used for local PIN check
WFS_PIN_USERSAPUBLIC	key is used as a public key for RSA encryption
WFS_PIN_USERSAPRIVATE	key is used as a private key for RSA encryption
WFS_PIN_USECHIPINFO	key is used as KGK_{INFO} key (only ZKA standard)
WFS_PIN_USECHIPPIN	key is used as KGK_{PIN} key (only ZKA standard)
WFS_PIN_USECHIPPS	key is used as K_{PS} key (only ZKA standard)
WFS_PIN_USECHIPMAC	key is used as K_{MAC} key (only ZKA standard)
WFS_PIN_USECHIPLT	key is used as KGK_{LT} key (only ZKA standard)
WFS_PIN_USECHIPMACLZ	key is used as K_{PACMAC} key (only ZKA standard)
WFS_PIN_USECHIPMACAZ	key is used as K_{MASTER} key (only ZKA standard)

bGeneration
Specifies the generation of the key as BCD value. Will be 0xff if no such information is available for the key.

bVersion
Specifies the version of the key as BCD value. Will be 0xff if no such information is available for the key.

bActivatingDate
Specifies the date when the key is activated as BCD value in the format YYYYMMDD. Will be 0xffffffff if no such information is available for the key.

bExpiryDate
Specifies the date when the key expires as BCD value in the format YYYYMMDD. Will be 0xffffffff if no such information is available for the key.

bLoaded
Specifies whether the key has been loaded (imported from Application or locally from Operator) and is either TRUE or FALSE.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key name is not found.

Comments None.

5. Execute Commands

5.1 WFS_CMD_PIN_CRYPT

Description The input data is either encrypted or decrypted using the specified or selected encryption mode. The available modes are defined in the WFS_INF_PIN_CAPABILITIES command.

This command can also be used for random number generation.

Furthermore it can be used for Message Authentication Code generation (i.e. MACing). For this purpose, it is possible to specify how the data is formatted before the encryption.

The input data can be expanded with a fill-character to the necessary length (mandated by the encryption algorithm being used).

The Start Value (or Initialization Vector) should be able to be passed encrypted like the specified encryption/decryption key. It would therefore need to be decrypted with a loaded key so the name of this key must also be passed. However, both these parameters are optional.

Input Param `LPWFSPINCRYPT lpCrypt;`

```
typedef struct _wfs_pin_crypt
   {
   WORD             wMode;
   LPSTR            lpsKey;
   LPWFSXDATA          lpxKeyEncKey;
   WORD             wAlgorithm;
   LPSTR            lpsStartValueKey;
   LPWFSXDATA          lpxStartValue;
   BYTE             bPadding;
   BYTE             bCompression;
   LPWFSXDATA          lpxCryptData;
   } WFSPINCRYPT, * LPWFSPINCRYPT;
```

wMode
Specifies whether to encrypt or decrypt, values are one of the following:

Value	Meaning
WFS_PIN_MODEENCRYPT	encrypt with key
WFS_PIN_MODEDECRYPT	decrypt with key
WFS_PIN_MODERANDOM	an 8 byte random value shall be returned (in this case all the other input parameters are ignored)

This parameter does not apply to MACing.

lpsKey
Specifies the name of the stored key. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

lpxKeyEncKey
If NULL, *lpsKey* is used directly for encryption/decryption. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used for encryption/decryption. Key is a double length key when used for Triple DES encryption/decryption. Users of this specification must adhere to local regulations when using Triple DES. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

wAlgorithm
Specifies the encryption algorithm. Possible values are those described in WFS_INF_PIN_CAPABILITIES. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

lpsStartValueKey
Specifies the name of the stored key used to decrypt the *lpxStartValue* to obtain the Initialization Vector. If this parameter is NULL, *lpxStartValue* is used as the Initialization Vector. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

lpxStartValue
DES and Triple DES initialization vector for CBC / CFB encryption and MACing. If this parameter is NULL *lpsStartValueKey* is used as the Start Value. If *lpsStartValueKey* is also NULL, the default value for CBC / CFB / MAC is 16 hex digits 0x0. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

bPadding
Specifies the padding character for encryption. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

bCompression
Specifies whether data is to be compressed (blanks removed) before building the MAC. If *bCompression* is 0x00 no compression is selected, otherwise *bCompression* holds the representation of the blank character in the actual code table. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

lpxCryptData
Pointer to the data to be encrypted, decrypted, or MACed. This value is ignored, if *wMode* equals WFS_PIN_MODERANDOM.

Output Param

```
LPWFSXDATA    lpxCryptData;
```

lpxCryptData
Pointer to the encrypted or decrypted data, MAC value or 8 byte random value.

Error Codes

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_MODENOTSUPPORTED	The specified mode is not supported.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* or *lpxStartValue* is not supported.
WFS_ERR_PIN_NOCHIPTRANSACTIVE	A chipcard key is used as encryption key and there is no chip transaction active.
WFS_ERR_PIN_ALGORITHMNOTSUPP	The specified algorithm is not supported.

Events

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments

The data type LPWFSXDATA is used to pass hexadecimal data and is defined as follows:

```
typedef struct _wfs_hex_data
    {
    USHORT    usLength;
    LPBYTE    lpbData;
    } WFSXDATA, *LPWFSXDATA;
```

usLength
Length of the byte stream pointed to by *lpbData.*

lpbData
Pointer to the binary data stream.

5.2 WFS_CMD_PIN_IMPORT_KEY

Description The key passed by the application is loaded in the encryption module. The key can be passed in clear text mode or encrypted with an accompanying “key encryption key”.

Input Param

```
LPWFSPINIMPORT      lpImport;

typedef struct _wfs_pin_import
   {
   LPSTR            lpsKey;
   LPSTR            lpsEncKey;
   LPWFSXDATA          lpxIdent;
   LPWFSXDATA          lpxValue;
   WORD             fwUse;
   } WFSPINIMPORT, * LPWFSPINIMPORT;
```

lpsKey
Specifies the name of key being loaded.

lpsEncKey
If *lpsEncKey* is NULL the key is loaded directly into the encryption module. Otherwise, *lpsEncKey* specifies a key name or a format name which were used to encrypt the key passed in *lpxValue*.

lpxIdent
Specifies the key owner identification. The use of this parameter is vendor dependent.

lpxValue
Specifies the value of key to be loaded.

fwUse
Specifies the type of access for which the key can be used as a combination of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key can be used for encryption/decryption
WFS_PIN_USEFUNCTION	key can be used for PIN functions
WFS_PIN_USEMACING	key can be used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USENODUPLICATE	key can be imported only once
WFS_PIN_USESVENCKEY	key is used as CBC Start Value encryption key

If *fwUse* equals zero the specified key is deleted. In that case all parameters but *lpsKey* are ignored.

Output Param

```
LPWFSXDATA    lpxKVC;
```

lpxKVC
pointer to the key verification code data that can be used for verification of the loaded key, NULL if device does not have that capability.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key encryption key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_INVALIDID	The ID passed was not valid.
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEYNOVALUE	The specified key encryption key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxValue* is not supported.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.3 WFS_CMD_PIN_DERIVE_KEY

Description A key is derived from input data using a key generating key and an initialization vector. The input data can be expanded with a fill-character to the necessary length (mandated by the encryption algorithm being used). The derived key is imported into the encryption module and is used for encryption or decryption operations.

Input Param

```
LPWFSPINDERIVE      lpDerive;

typedef struct _wfs_pin_derive
   {
   WORD             wDerivationAlgorithm;
   LPSTR            lpsKey;
   LPSTR            lpsKeyGenKey;
   LPSTR            lpsStartValueKey;
   LPWFSXDATA          lpxStartValue;
   BYTE             bPadding;
   LPWFSXDATA          lpxInputData;
   LPWFSXDATA          lpxIdent;
   } WFSPINDERIVE, * LPWFSPINDERIVE;
```

wDerivationAlgorithm
Specifies the algorithm that is used for derivation. Possible values are:
(see command WFS_INF_PIN_CAPABILITIES)

lpsKey
Specifies the name where the derived key will be stored.

lpsKeyGenKey
Specifies the name of the key generating key that is used for the derivation.

lpsStartValueKey
Specifies the name of the stored key used to decrypt the *lpxStartValue* to obtain the Initialization Vector. If this parameter is NULL, *lpxStartValue* is used as the Initialization Vector.

lpxStartValue
DES initialization vector for the encryption step within the derivation.

bPadding
Specifies the padding character for the encryption step within the derivation.

lpxInputData
Pointer to the data to be used for key derivation.

lpxIdent
Specifies the key owner identification. The use of this parameter is vendor dependent.

Output Param None.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized (or not ready for some vendor specific reason).
WFS_ERR_PIN_INVALIDID	The ID passed was not valid.

WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxStartValue* is not supported.
WFS_ERR_PIN_ALGORITHMNOTSUPP	The specified algorithm is not supported.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.4 WFS_CMD_PIN_GET_PIN

Description This function stores the PIN entry via the PIN pad. From the point this function is invoked, PIN digit entries are ***not*** passed to the application. For each PIN digit, or any other active key entered, an execute notification event is sent in order to allow an application to perform the appropriate display action (i.e. when the PIN pad has no integrated display). The application is not informed of the value entered, the execute notification only informs that a key has been depressed.

Some PIN pad devices do not inform the application as each PIN digit is entered, but locally process the PIN entry based upon minimum PIN length and maximum PIN length input parameters. These PIN pad devices which provide local PIN entry management and optional display tracking may or may not notify the application of a minimum PIN length violation.

When the maximum number of PIN digits is entered, or a completion key is pressed after the minimum number of PIN digits is entered, a WFS_EXEC_COMPLETE event message is sent to the application. Once this notification is received, the output parameters are then returned to the application from this function call. The depression of the <Cancel> key is also passed to the application via the WFS_EXEC_COMPLETE event message.

If *usMaxLen* is zero, the service provider does not terminate the command unless the application sets *ulTerminateKeys* or *ulTerminateFDKs*. In the event that *ulTerminateKeys* or *ulTerminateFDKs* are not set and *usMaxLen* is zero, the command will not terminate and the application must issue a WFSCancel command.

Terminating keys have to be active keys to operate.

If this command is cancelled by a WFSCancelAsyncRequest or a WFSCancelBlockingCall the PIN buffer is not cleared.

It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

Input Param

```
LPWFSPINGETPIN      lpGetPin;
```

```
typedef struct _wfs_pin_getpin
   {
   USHORT    usMinLen;
   USHORT    usMaxLen;
   BOOL      bAutoEnd;
   CHAR      cEcho;
   ULONG     ulActiveFDKs;
   ULONG     ulActiveKeys;
   ULONG     ulTerminateFDKs;
   ULONG     ulTerminateKeys;
   } WFSPINGETPIN, * LPWFSPINGETPIN;
```

usMinLen
Specifies the minimum number of digits which must be entered for the PIN. A value of zero indicates no minimum PIN length verification.

usMaxLen
Specifies the maximum number of digits which can be entered for the PIN.

bAutoEnd
If *bAutoEnd* is set to true, the service provider terminates the command when the maximum number of digits are entered. Otherwise, the input is terminated by the user using one of the termination keys. When *usMaxLen* is reached, the service provider will disable all numeric keys. *bAutoEnd* is ignored when *usMaxLen* is set to 0.

cEcho
Specifies the replace character to be echoed on a local display for the PIN digit.

ulActiveFDKs
Specifies those FDKs which are active during the execution of the command.

ulActiveKeys
Specifies those (other) Function Keys which are active during the execution of the command.

ulTerminateFDKs
Specifies those FDKs which must terminate the execution of the command.

ulTerminateKeys
Specifies those (other) Function Keys which must terminate the execution of the command.

Output Param `LPWFSPINENTRY lpEntry;`

```
typedef struct _wfs_pin_entry
   {
   USHORT    usDigits;
   WORD      wCompletion;
   } WFSPINENTRY, * LPWFSPINENTRY;
```

usDigits
Specifies the number of PIN digits entered.

wCompletion
Specifies the reason for completion of the entry. Possible values are:

Value	Meaning
WFS_PIN_COMPAUTO	The command terminated automatically, because maximum PIN length was reached.
WFS_PIN_COMPENTER	The ENTER Function Key was pressed as terminating key.
WFS_PIN_COMPCANCEL	The CANCEL Function Key was pressed as terminating key.
WFS_PIN_COMPCONTINUE	Input continues, function key was pressed (this value is only used in the execute event WFS_EXEE_PIN_KEY).
WFS_PIN_COMPCLEAR	The CLEAR Function Key was pressed as terminating key and the previous input is cleared.
WFS_PIN_COMPBACKSPACE	The last input digit was cleared and the key was pressed as terminating key.
WFS_PIN_COMPFDK	Indicates input is terminated only if the FDK pressed was set to be a terminating FDK.
WFS_PIN_COMPHELP	The HELP Function Key was pressed as terminating key.
WFS_PIN_COMPFK	A Function Key (FK) other than ENTER, CLEAR, CANCEL, BACKSPACE, HELP was pressed as terminating key.
WFS_PIN_COMPCONTFDK	Input continues, FDK was pressed (this value is only used in the execute event WFS_EXEE_PIN_KEY).

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

Value	Meaning
WFS_ERR_PIN_KEYINVALID	At least one of the specified function keys or FDKs is invalid.
WFS_ERR_PIN_KEYNOTSUPPORTED	At least one of the specified function keys or FDKs is not supported by the service provider.

WFS_ERR_PIN_NOACTIVEKEYS	There are no active function keys specified.
WFS_ERR_PIN_NOTERMINATEKEYS	There are no terminate keys specified and usMaxLen is not set to 0 and bAutoEnd is FALSE.
WFS_ERR_PIN_MINIMUMLENGTH	The minimum PIN length field is invalid or greater than the maximum PIN length field.

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

Value	Meaning
WFS_EXEE_PIN_KEY	A key has been pressed at the PIN pad.

Comments None.

5.5 WFS_CMD_PIN_LOCAL_DES

Description The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the DES validation algorithm and locally verified for correctness. The local DES verification is based on the IBM 3624 standard. The result of the verification is returned to the application. This command will clear the PIN.

Input Param

```
LPWFSPINLOCALDES    lpLocalDES;

typedef struct _wfs_pin_local_des
   {
   LPSTR        lpsValidationData;
   LPSTR        lpsOffset;
   BYTE         bPadding;
   USHORT       usMaxPIN;
   USHORT       usValDigits;
   BOOL         bNoLeadingZero;
   LPSTR        lpsKey;
   LPWFSXDATA       lpxKeyEncKey;
   LPSTR        lpsDecTable;
   } WFSPINLOCALDES, * LPWFSPINLOCALDES;
```

lpsValidationData
Validation data

lpsOffset
Offset for the PIN block; if NULL then no offset is used.

bPadding
Specifies the padding character for validation data.

usMaxPIN
Maximum number of PIN digits to be used for validation.

usValDigits
Number of Validation digits to be used for validation.

bNoLeadingZero
If set to TRUE and the first digit of result of the modulo 10 addition is a X’0’, it is replaced with X’1’ before performing the verification against the entered PIN. If set to FALSE, a leading zero is allowed in entered PINs.

lpsKey
Name of the validation key

lpxKeyEncKey
If NULL, *lpsKey* is used directly for PIN validation. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used for PIN validation.

lpsDecTable
ASCII decimalization table (16 character string containing characters ‘0’ to ‘9’). Used to

convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

Output Param

```
LPBOOL        pbResult;
```

lpbResult
Pointer to a boolean value which specifies whether the PIN is correct or not.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* is not supported.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.6 WFS_CMD_PIN_CREATE_OFFSET

Description This function is used to generate a PIN Offset that is used to verify PINs using the WFS_CMD_PIN_LOCAL_DES execute command. The PIN offset is computed by combining validation data with the keypad entered PIN. This command will clear the PIN.

Input Param

```
LPWFSPINCREATEOFFSET      lpPINOffset;

typedef struct _wfs_pin_create_offset
   {
   LPSTR        lpsValidationData;
   BYTE         bPadding;
   USHORT       usMaxPIN;
   USHORT       usValDigits;
   LPSTR        lpsKey;
   LPWFSXDATA       lpxKeyEncKey;
   LPSTR        lpsDecTable;
   } WFSPINCREATEOFFSET, * LPWFSPINCREATEOFFSET;
```

lpsValidationData
Validation data

bPadding
Specifies the padding character for validation data.

usMaxPIN
Maximum number of PIN digits to be used for PIN Offset creation.

usValDigits
Number of Validation Data digits to be used for PIN Offset creation.

lpsKey
Name of the validation key

lpxKeyEncKey
If NULL, *lpsKey* is used directly in PIN Offset creation. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used in PIN Offset creation.

lpsDecTable
ASCII decimalization table (16 character string containing characters '0' to '9'). Used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

Output Param `LPSTR  lpsOffset;`

lpsOffset
Computed PIN Offset.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_NOTALLOWED	PIN entered by the user is not allowed.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments The list of 'forbidden' PINs (values that cannot be chosen as a PIN, e.g. 1111) is configured in the device in a vendor dependent way during the configuration of the system.

5.7 WFS_CMD_PIN_LOCAL_EUROCHEQUE

Description The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the Eurocheque validation algorithm and locally verified for correctness. The result of the verification is returned to the application. This command will clear the PIN.

Input Param `LPWFSPINLOCALEUROCHEQUE    lpLocalEurocheque;`

```
typedef struct _wfs_pin_local_eurocheque
    {
    LPSTR         lpsEurochequeData;
    LPSTR         lpsPVV;
    WORD          wFirstEncDigits;
    WORD          wFirstEncOffset;
    WORD          wPVVDigits;
    WORD          wPVVOffset;
    LPSTR         lpsKey;
    LPWFSXDATA         lpxKeyEncKey;
    LPSTR         lpsDecTable;
    } WFSPINLOCALEUROCHEQUE, * LPWFSPINLOCALEUROCHEQUE;
```

lpsEurochequeData
Track-3 Eurocheque data

lpsPVV
PIN Validation Value from track data.

wFirstEncDigits
Number of digits to extract after first encryption.

wFirstEncOffset
Offset of digits to extract after first encryption.

wPVVDigits
Number of digits to extract for PVV.

wPVVOffset
Offset of digits to extract for PVV.

lpsKey
Name of the validation key.

lpxKeyEncKey
If NULL, *lpsKey* is used directly for PIN validation. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used for PIN validation.

lpsDecTable
ASCII decimalization table (16 character string containing characters '0' to '9'). Used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

Output Param

```
LPBOOL        lpbResult;
```

lpbResult
Pointer to a boolean value which specifies whether the PIN is correct or not.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* is not supported.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.8 WFS_CMD_PIN_LOCAL_VISA

Description The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the VISA validation algorithm and locally verified for correctness. The result of the verification is returned to the application. This command will clear the PIN.

Input Param

```
LPWFSPINLOCALVISA   lpLocalVISA;

typedef struct _wfs_pin_local_visa
   {
   LPSTR        lpsPAN;
   LPSTR        lpsPVV;
   WORD         wPVVDigits;
   LPSTR        lpsKey;
   LPWFSXDATA       lpxKeyEncKey;
   } WFSPINLOCALVISA, * LPWFSPINLOCALVISA;
```

lpsPAN
Primary Account Number from track data.

lpsPVV
PIN Validation Value from track data.

wPVVDigits
Number of digits of PVV.

lpsKey
Name of the validation key.

lpxKeyEncKey
If NULL, *lpsKey* is used directly for PIN validation. Otherwise, *lpsKey* is used to decrypt the encrypted key passed in *lpxKeyEncKey* and the result is used for PIN validation.

Output Param

```
LPBOOL        lpbResult;
```

lpbResult
Pointer to a boolean value which specifies whether the PIN is correct or not.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxKeyEncKey* is not supported.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.9 WFS_CMD_PIN_PRESENT_IDC

Description The PIN, which was entered with the WFS_PIN_GET_PIN command, is combined with the requisite data specified by the IDC presentation algorithm and presented to the smartcard contained in the ID Card unit. The result of the presentation is returned to the application. This command will clear the PIN.

Input Param

```
LPWFSPINPRESENTIDC  lpPresentIDC;

typedef struct _wfs_pin_presentidc
   {
   WORD          wPresentAlgorithm;
   WORD          wChipProtocol;
   ULONG         ulChipDataLength;
   LPBYTE        lpbChipData;
   LPVOID        lpAlgorithmData;
   } WFSPINPRESENTIDC, * LPWFSPINPRESENTIDC;
```

wPresentAlgorithm
Specifies the algorithm that is used for presentation. Possible values are: (see command WFS_INF_PIN_CAPABILITIES).

wChipProtocol
Identifies the protocol that is used to communicate with the chip. Possible values are: (see command WFS_INF_IDC_CAPABILITIES in the Identification Card Device Class Interface).

ulChipDataLength
Specifies the length of the byte stream pointed to by *lpbChipData*.

lpbChipData
Points to the data to be sent to the chip.

lpAlgorithmData
Pointer to a structure that contains the data required for the specified presentation algorithm. For the WFS_PIN_PRESENT_CLEAR algorithm, this structure is defined as:

```
typedef struct _wfs_pin_presentclear
  {
  ULONG         ulPINPointer;
  USHORT        usPINOffset;
  } WFSPINPRESENTCLEAR, * LPWFSPINPRESENTCLEAR;
```

ulPINPointer
Describes the byte position where to insert the PIN in the *lpbChipData* buffer. The first byte of the *lpbChipData* buffer is numbered 0.

usPINOffset
Describes the bit position where to insert the PIN in the *lpbChipData* buffer. In each byte, the most-significant bit is numbered 0, the less significant bit is numbered 7.

Output Param

```
LPWFSPINPRESENTRESULT      lpPresentResult;

typedef struct _wfs_pin_present_result
    {
    WORD      wChipProtocol;
    ULONG     ulChipDataLength;
    LPBYTE    lpbChipData;
    } WFSPINPRESENTRESULT, * LPWFSPINPRESENTRESULT;
```

wChipProtocol
Identifies the protocol that was used to communicate with the chip. This field contains the same value as the corresponding field in the input structure.

ulChipDataLength
Specifies the length of the byte stream pointed to by *lpbChipData*.

lpbChipData
Points to the data responded from the chip.

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

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The ID card unit is not ready for PIN presentation or for any vendor specific reason. The ID card service provider, if any, may have generated a service event that further describes the reason for that error code.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared.
WFS_ERR_PIN_PROTOCOLNOTSUPP	The specified protocol is not supported by the service provider.
WFS_ERR_PIN_INVALIDDATA	An error occurred while communicating with the chip.

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

Comments None.

5.10 WFS_CMD_PIN_GET_PINBLOCK

Description This function takes the account information and a PIN entered by the user to build a formatted PIN. Encrypting this formatted PIN once or twice returns a PIN block which can be written on a magnetic card or sent to a host. The PIN block can be calculated using one of the formats specified in the WFS_INF_PIN_CAPABILITIES command. This command clears the PIN.

Input Param LPWFSPINBLOCK lpPinBlock;

```
typedef struct _wfs_pin_block
   {
   LPSTR        lpsCustomerData;
   LPSTR        lpsXORData;
   BYTE         bPadding;
   WORD         wFormat;
   LPSTR        lpsKey;
   LPSTR        lpsKeyEncKey;
   } WFSPINBLOCK, * LPWFSPINBLOCK;
```

lpsCustomerData
Used for ANSI, ISO-0 and ISO-1 algorithm to build the formatted PIN. For ANSI and ISO-0 the PAN (Primary Account Number) is used, for ISO-1 a ten digit transaction field is required. If not used a NULL is required.
Used for DIEBOLD with coordination number, as a two digit coordination number.

lpsXORData
If the formatted PIN is encrypted twice to build the resulting PIN block, this data can be used to modify the result of the first encryption by an XOR-operation.

bPadding
Specifies the padding character.

wFormat
Specifies the format of the PIN block. Possible values are:
(see command WFS_INF_PIN_CAPABILITIES)

lpsKey
Specifies the key used to encrypt the formatted pin for the first time, NULL if no encryption is required. If this specifies a double length key, triple DES encryption will be performed.

lpsKeyEncKey
Specifies the key used to format the once encrypted formatted PIN, NULL if no second encryption required.

Output Param LPWFSXDATA lpxPinBlock;

lpxPinBlock
Pointer to the encrypted/decrypted data.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not found
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.
WFS_ERR_PIN_NOPIN	PIN has been cleared.
WFS_ERR_PIN_FORMATNOTSUPP	The specified format is not supported.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.11 WFS_CMD_PIN_GET_DATA

Description This function is used to return keystrokes entered by the user. It will automatically set the PIN pad to echo characters on the display if there is a display. For each keystroke an execute notification event is sent in order to allow an application to perform the appropriate display action (i.e. when the PIN pad has no integrated display).

If *usMaxLen* is zero, the service provider does not terminate the command unless the application sets *ulTerminateKeys* or *ulTerminateFDKs*. In the event that *ulTerminateKeys* or *ulTerminateFDKs* are not set and *usMaxLen* is zero, the command will not terminate and the application must issue a WFSCancel command.

Terminating keys have to be active keys to operate.

It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

The following keys may effect the contents of the WFSPINDATA output parameter but are not returned in it:

WFS_PIN_FK_ENTER
WFS_PIN_FK_CANCEL
WFS_PIN_FK_CLEAR
WFS_PIN_FK_BACKSPACE

The WFS_PIN_FK_CANCEL and WFS_PIN_FK_CLEAR keys will cause the output buffer to be cleared. The WFS_PIN_FK_BACKSPACE key will cause the last key in the buffer to be removed.

Input Param

```
LPWFSPINGETDATA     lpPinGetData;
```

```
typedef struct _wfs_pin_getdata
    {
    USHORT    usMaxLen;
    BOOL      bAutoEnd;
    ULONG     ulActiveFDKs;
    ULONG     ulActiveKeys;
    ULONG     ulTerminateFDKs;
    ULONG     ulTerminateKeys;
    } WFSPINGETDATA, * LPWFSPINGETDATA;
```

usMaxLen
Specifies the maximum number of digits which can be returned to the application in the output parameter.

bAutoEnd
If *bAutoEnd* is set to true, the service provider terminates the command when the maximum number of digits are entered. Otherwise, the input is terminated by the user using one of the termination keys. When *usMaxLen* is reached, the service provider will disable all numeric keys. *bAutoEnd* is ignored when *usMaxLen* is set to 0.

ulActiveFDKs
Specifies those FDKs which are active during the execution of the command.

ulActiveKeys
Specifies those (other) Function Keys which are active during the execution of the command.

ulTerminateFDKs
Specifies those FDKs which must terminate the execution of the command.

ulTerminateKeys
Specifies those (other) Function Keys which must terminate the execution of the command.

Output Param `LPWFSPINDATA lpPinData;`

```
typedef struct _wfs_pin_data
    {
    USHORT           usKeys;
    LPWFSPINKEY *    lpPinKeys;
    WORD             wCompletion;
    } WFSPINDATA, * LPWFSPINDATA;
```

usKeys
Number of keys entered by the user (i.e. number of following WFSPINKEY structures).

lpPinKeys
Pointer to an array of pointers to WFSPINKEY structures that contain the keys entered by the user (for a description of the WFSPINKEY structure see the definition of the WFS_EXEE_PIN_KEY event).

wCompletion
Specifies the reason for completion of the entry. Possible values are:
(see command WFS_CMD_PIN_GET_PIN)

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

Value	Meaning
WFS_ERR_PIN_KEYINVALID	At least one of the specified function keys or FDKs is invalid.
WFS_ERR_PIN_KEYNOTSUPPORTED	At least one of the specified function keys or FDKs is not supported by the service provider.
WFS_ERR_PIN_NOACTIVEKEYS	There are no active function keys specified.

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

Value	Meaning
WFS_EXEE_PIN_KEY	A key has been pressed at the PIN pad.

Comments If the triple zero key is pressed one WFS_EXEE_PIN_KEY event is sent that contains the WFS_PIN_FK_000 code.

If the triple zero key is pressed when 3 keys are already inserted and usMaxLen equals 4 the key is not accepted and no event is sent to the application.

If the backspace key is pressed after the triple zero key only one zero is deleted out of the buffer.

Double zero is handled similar to this.

5.12 WFS_CMD_PIN_INITIALIZATION

Description The encryption module must be initialized before any encryption function can be used. Every initialization destroys all keys that have been loaded or imported. Usually this command is called by an operator task and not by the application program.

Initialization also involves loading “initial” application keys and local vendor dependent keys. These can be supplied, for example, by an operator through a keyboard, a local configuration file or possibly by means of some secure hardware that can be attached to the device. The application “initial” keys would normally get updated by the application during a WFS_CMD_PIN_IMPORT_KEY command as soon as possible. Local vendor dependent static keys (e.g. storage, firmware and offset keys) would normally be transparent to the application and by definition can not be dynamically changed.

Where initial keys are not available immediately when this command is issued (i.e. when operator intervention is required), the Service Provider returns WFS_ERR_PIN_ACCESS_DENIED and the application must await the WFS_SRVE_PIN_INITIALIZED event.

During initialization an optional encrypted ID key can be stored in the HW module. The ID key and the corresponding encryption key can be passed as parameters; if not, they are generated automatically by the encryption module. The encrypted ID is returned to the application and serves as authorization for the key import function. The WFS_INF_PIN_CAPABILITIES command indicates whether or not the device will support this feature.

This function also resets the HSM terminal data, except session key index and trace number.

Input Param

```
LPWFSPININIT lpInit;
```

```
typedef struct _wfs_pin_init
   {
   LPWFSXDATA       lpxIdent;
   LPWFSXDATA       lpxKey;
   } WFSPININIT, * LPWFSPININIT;
```

lpxIdent
Pointer to the value of the ID key. Null if not required.

lpxKey
Pointer to the value of the encryption key. Null if not required.

Output Param

```
LPWFSXDATA   lpxIdentification;
```

lpxIdentification
Pointer to the value of the ID key encrypted by the encryption key. Can be used as authorization for the WFS_CMD_PIN_IMPORT_KEY command, can be NULL if no authorization required.

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

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized (or not ready for some vendor specific reason).
WFS_ERR_PIN_INVALIDID	The ID passed was not valid.

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

Value	Meaning
WFS_SRVE_PIN_INITIALIZED	The encryption module is now initialized.
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.13 WFS_CMD_PIN_LOCAL_BANKSYS

Description The PIN Block previously built by the WFS_CMD_PIN_GET_PINBLOCK according to the BANKSYS specifications is combined with the ATMVAC code for local validation.

Input Param

```
LPWFSPINLOCALBANKSYS             lpLocalBanksys;
```

```
typedef struct _wfs_pin_local_banksys
   {
   LPWFSXDATA       lpxATMVAC;
   } WFSPINLOCALBANKSYS, * LPWFSPINLOCALBANKSYS;
```

lpxATMVAC
The ATMVAC code calculated by the BANKSYS Security Control Module.

Output Param

```
LPBOOL       lpbResult;
```

lpbResult
Pointer to a boolean value which specifies whether the PIN is correct or not.

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

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_NOPIN	PIN has not been entered or has been cleared without building the Banksys PIN Block.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxATMVAC* is not supported.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.14 WFS_CMD_PIN_BANKSYS_IO

Description This command sends a single command to the Banksys Security Control Module.

Input Param

```
LPWFSPINBANKSYSIO   lpBANKSYSIoIn;

typedef struct _wfs_pin_BANKSYS_io
   {
   ULONG     ulLength;
   LPBYTE    lpbData;
   } WFSPINBANKSYSIO, * LPWFSPINBANKSYSIO;
```

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

lpbData
Points to the data sent to the BANKSYS Security Control Module.

Output Param

```
LPWFSPINBANKSYSIO          lpBANKSYSIoOut;
typedef struct _wfs_pin_BANKSYS_io
   {
   ULONG     ulLength;
   LPBYTE    lpbData;
   } WFSPINBANKSYSIO, * LPWFSPINBANKSYSIO;
```

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

lpbData
Points to the data responded by the BANKSYS Security Control Module.

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

Value	Meaning
WFS_ERR_PIN_INVALIDDATA	An error occurred while communicating with the device.

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

Comments The Banksys command and response message data are defined in the Banksys document “SCM DKH Manual Rel 2.x ”

5.15 WFS_CMD_PIN_RESET

Description Sends a service reset to the service provider.

Input Param None

Output Param None.

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

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

Comments This command is used by an application control program to cause a device to reset itself to a known good condition. It does not delete any keys.

5.16 WFS_CMD_PIN_HSM_SET_TDATA

Description This function allows to set the HSM terminal data except keys, trace number and session key index. The data must be provided as a series of “tag/length/value” items.

Input Param `LPWFSXDATA    lpxTData;`

lpxTData
Specifies which parameter(s) is(are) to be set. lpxTData is a series of “tag/length/value” items where each item consists of

- one byte tag (see the list of tags below),
- one byte specifying the length of the following data as an unsigned binary number
- n bytes data (see the list below for formatting)

with no separators.

The following tags are supported:

tag (hexadecimal)	Format	Length (in bytes)	Meaning
C2	BCD	4	Terminal ID ISO BMP 41
C3	BCD	4	Bank code ISO BMP 42 (rightmost 4 bytes)
C4	BCD	9	Account data for terminal account ISO BMP 60 (load against other card)
C5	BCD	9	Account data for fee account ISO BMP 60 ("Laden vom Kartenkonto")
C6	EBCDIC	40	Terminal location ISO BMP 43
C7	ASCII	3	Terminal currency
C8	BCD	7	Online date and time (YYYYMMDDHHMMSS) ISO BMP 61
C9	BCD	4	Minimum load fee in units of 1/100 of terminal currency, checked against leftmost 4 Bytes of ISO BMP42,
CA	BCD	4	Maximum load fee in units of 1/100 of terminal currency, checked against leftmost 4 Bytes of ISO BMP42,

Output Param None.

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

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this command.

Events None.

Comments None.

5.17 WFS_CMD_PIN_SECURE_MSG_SEND

Description This command handles all messages that should be send through a secure messaging to a authorization system, German "Ladezentrale", personalisation system or the chip. The encryption module adds the security relevant fields to the message and returns the modified message in the output structure. All messages must be presented to the encryptor via this command even if they do not contain security fields in order to keep track of the transaction status in the internal state machine.

Input Param

```
LPWFSPINSECMSG      lpSecMsgIn;
```

```
typedef struct _wfs_pin_secure_message
   {
   WORD      wProtocol;
   ULONG     ulLength;
   LPBYTE    lpbMsg;
   } WFSPINSECMSG, * LPWFSPINSECMSG;
```

wProtocol
Specifies the protocol the message belongs to. Specified as one of the following flags:

Value	Meaning
WFS_PIN_PROTISOAS	ISO 8583 protocol for the authorization system
WFS_PIN_PROTISOLZ	ISO 8583 protocol for the German "Ladezentrale"
WFS_PIN_PROTISOPS	ISO 8583 protocol for the personalisation system
WFS_PIN_PROTCHIPZKA	ZKA chip protocol
WFS_PIN_PROTRAWDATA	raw data protocol

ulLength
Specifies the length in bytes of the message in *lpbMsg*.

lpbMsg
Specifies the message that should be send.

Output Param

```
LPWFSPINSECMSG      lpSecMsgOut;
```

lpSecMsgOut
pointer to a WFSPINSECMSG structure that contains the modified message that can now be send to a authorization system, German "Ladezentrale", personalisation system or the chip.

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

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this message.
WFS_ERR_PIN_PROTINVALID	The specified protocol is invalid.
WFS_ERR_PIN_CONTENTINVALID	The contents of one of the security relevant fields are invalid.

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

Comments None.

5.18 WFS_CMD_PIN_SECURE_MSG_RECEIVE

Description This command handles all messages that are received through a secure messaging from a authorization system, German "Ladezentrale", personalisation system or the chip. The encryption module checks the security relevant fields. All messages must be presented to the encryptor via this command even if they do not contain security relevant fields in order to keep track of the transaction status in the internal state machine.

Input Param

```
LPWFSPINSECMSG      lpSecMsgIn;
```

```
typedef struct _wfs_pin_secure_message
   {
   WORD      wProtocol;
   ULONG     ulLength;
   LPBYTE    lpbMsg;
   } WFSPINSECMSG, * LPWFSPINSECMSG;
```

wProtocol
Specifies the protocol the message belongs to. Specified as one of the following flags:

Value	Meaning
WFS_PIN_PROTISOAS	ISO 8583 protocol for the authorization system
WFS_PIN_PROTISOLZ	ISO 8583 protocol for the German "Ladezentrale"
WFS_PIN_PROTISOPS	ISO 8583 protocol for the personalisation system
WFS_PIN_PROTCHIPZKA	ZKA chip protocol
WFS_PIN_PROTRAWDATA	raw data protocol

ulLength
Specifies the length in bytes of the message in *lpbMsg*.

lpbMsg
Specifies the message that was received. Can be NULL if during a specified time period no response was reveived from the communication partner (necessary to set the internal state machine to the correct state).

Output Param None.

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

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to handle this message.
WFS_ERR_PIN_MACINVALID	The MAC of the message is not correct.
WFS_ERR_PIN_PROTINVALID	The specified protocol is invalid.
WFS_ERR_PIN_FORMATINVALID	The format of the message is invalid.
WFS_ERR_PIN_CONTENTINVALID	The contents of one of the security relevant fields are invalid.

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

Comments None.

5.19 WFS_CMD_PIN_GET_JOURNAL

Description This command is used to get journal data from the encryption module. It retrieves cryptographically secured information about the result of the last transaction that was done with the indicated protocol. When the service provider supports journaling (see Capabilities) then it is impossible to do any WFS_CMD_PIN_SECURE_MSG_SEND/RECEIVE with this protocol, unless the journal data is retrieved. It is possible - especially after restarting a system - to get the same journal data again.

Input Param

```
LPWORD        lpwProtocol;
```

lpwProtocol
Specifies the protocol the journal data belong to. Specified as one of the following flags:

Value	Meaning
WFS_PIN_PROTISOAS	Get authorization system journal data
WFS_PIN_PROTISOLZ	Get German "Ladezentrale" journal data
WFS_PIN_PROTISOPS	Get personalisation system journal data

Output Param

```
LPWFSXDATA    lpxJournalData;
```

lpxJournalData
Pointer to the journal data

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

Value	Meaning
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_HSMSTATEINVALID	The HSM is not in a correct state to return journal data.
WFS_ERR_PIN_PROTINVALID	The specified protocol is invalid.

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

Comments None.

5.20 WFS_CMD_PIN_IMPORT_KEY_EX

Description The key passed by the application is loaded in the encryption module. The key can be passed in clear text mode or encrypted with an accompanying "key encryption key". The dwUse parameter is needed to separate the keys in several parts of the encryption module to avoid the manipulation of a key.

Input Param

```
LPWFSPINIMPORTKEYEX lpImportKeyEx;

typedef struct _wfs_pin_import_key_ex
    {
    LPSTR           lpsKey;
    LPSTR           lpsEncKey;
    LPWFSXDATA      lpxValue;
    LPWFSXDATA      lpxControlVector;
    DWORD           dwUse;
    WORD            wKeyCheckMode;
    LPWFSXDATA      lpxKeyCheckValue;
    } WFSPINIMPORTKEYEX, * LPWFSPINIMPORTKEYEX;
```

lpsKey
Specifies the name of key being loaded.

lpsEncKey
If *lpsEncKey* is NULL the key is loaded directly into the encryption module. Otherwise *lpsEncKey* specifies a key name which was used to encrypt the key string passed in *lpxValue*.

lpxValue
Specifies the value of key to be loaded. If it is an RSA key the first 4 bytes contain the exponent and the following 128 the modulus.

lpxControlVector
Specifies the control vector of the key to be loaded. It contains the attributes of the key. If this parameter is NULL the keys is only specified by its use.

dwUse
Specifies the type of access for which the key can be used. If this parameter equals zero, the key is deleted. Otherwise the parameter can be one of the following flags:

Value	Meaning
WFS_PIN_USECRYPT	key is used for encryption and decryption
WFS_PIN_USEFUNCTION	key is used for PIN block creation
WFS_PIN_USEMACING	key is used for MACing
WFS_PIN_USEKEYENCKEY	key is used as key encryption key
WFS_PIN_USEPINLOCAL	key is used for local PIN check
WFS_PIN_USERSAPUBLIC	key is used as a public key for RSA encryption
WFS_PIN_USERSAPRIVATE	key is used as a private key for RSA encryption

If *dwUse* equals zero the specified key is deleted. In that case all parameters but *lpsKey* are ignored.

wKeyCheckMode
Specifies the mode that is used to create the key check value. It can be one of the following flags:

Value	Meaning
WFS_PIN_KCVNONE	There is no key check value verification required.
WFS_PIN_KCVSELF	The key check value is created by an encryption of the key with itself.
WFS_PIN_KCVZERO	The key check value is created by an encryption of the key with a zero value.

lpxKeyCheckValue
Specifies a check value to verify that the value of the imported key is correct. It can be NULL, if no key check value verification is required and wKeyCheckMode equals WFS_PIN_KCVNONE.

Output Param None.

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

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key encryption key was not found.
WFS_ERR_PIN_ACCESSDENIED	The encryption module is either not initialized or not ready for any vendor specific reason.
WFS_ERR_PIN_DUPLICATEKEY	A key exists with that name and cannot be overwritten.
WFS_ERR_PIN_KEYNOVALUE	The specified key encryption key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use conflicts with a previously for the same key specified one.
WFS_ERR_PIN_INVALIDKEYLENGTH	The length of *lpxValue* is not supported.
WFS_ERR_PIN_KEYINVALID	The key value is invalid. The key check value verification failed.
WFS_ERR_PIN_NOKEYRAM	There is no space left in the key RAM for a key of the specified type.

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

Value	Meaning
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	An error occurred accessing an encryption key.

Comments None.

5.21 WFS_CMD_PIN_ENC_IO

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

Input Param

```
LPWFSPINENCIO       lpEncIoIn;
```

```
typedef struct _wfs_pin_enc_io
   {
   WORD         wProtocol;
   ULONG        ulDataLength;
   LPVOID       lpvData;
   } WFSPINENCIO, *LPWFSPINENCIO;
```

wProtocol
Identifies the protocol that is used to communicate with the encryption module.
The following protocol numbers are defined:

Value	Meaning
WFS_PIN_ENC_PROT_CH	For Swiss specific protocols. The document specification for Swiss specific protocols is "CMD_ENC_IO - CH Protocol.doc". This document is available at the following address: *EUROPAY (Switzerland) SA* *Terminal Management* *Hertistrasse 27* *CH-8304 Wallisellen*

ulDataLength
Specifies the length in bytes of the structure pointed to by the following field *lpvData*.

lpvData
Points to a structure containing the data to be sent to the encryption module.

Output Param

```
LPWFSPINENCIO       lpEncIoOut;
```

```
typedef struct _wfs_pin_enc_io
   {
   WORD         wProtocol;
   ULONG        ulDataLength;
   LPVOID       lpvData;
   } WFSPINENCIO, *LPWFSPINENCIO;
```

wProtocol
Identifies the protocol that is used to communicate with the encryption module. This field contains the same value as the corresponding field in the input structure.

ulDataLength
Specifies the length in bytes of the structure pointed to by the following field *lpvData*.

lpvData
Points to a structure containing the data responded by the encryption module.

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

Value	Meaning
WFS_ERR_PIN_PROTOCOLNOTSUPP	The specified protocol is not supported by the service provider.

Events None.

Comments None.

6. Events

6.1 WFS_EXEE_PIN_KEY

Description This event specifies that any active key has been pressed at the PIN pad. It is used if the device has no internal display unit and the application has to manage the display of the entered digits.

It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

Event Param

```
LPWFSPINKEY   lpKey;
```

```
typedef struct _wfs_pin_key
   {
   WORD      wCompletion;
   ULONG     ulDigit;
   } WFSPINKEY, * LPWFSPINKEY;
```

wCompletion
Specifies the reason for completion or continuation of the entry. Possible values are:
(see command WFS_CMD_PIN_GET_PIN)

ulDigit
Specifies the digit entered by the user. When working in encryption mode (WFS_CMD_PIN_GET_PIN), the value of this field is zero. For each key pressed, the corresponding FK or FDK mask value is stored in this field.

Comments None.

6.2 WFS_SRVE_PIN_INITIALIZED

Description This event specifies that, as a result of a WFS_CMD_PIN_INITIALIZATION, the encryption module is now initialized and the master key (where required) and any other initial keys are loaded; ready to import other keys.

Event Param

```
LPWFSPININIT  lpInit;
```

lpInit
For a definition of WFSPININIT see command WFS_CMD_PIN_INITIALIZATION.

Comments None.

6.3 WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS

Description This event specifies that an error occurred accessing an encryption key. Possible situations for generating this event are the encryption key was not found, had no value, or a use violation.

Event Param

```
LPWFSPINACCESS      lpAccess;
```

```
typedef struct _wfs_pin_access
   {
   LPSTR     lpsKeyName;
   LONG      lErrorCode;
   } WFSPINACCESS, * LPWFSPINACCESS;
```

lpsKeyName
Specifies the name of the key that caused the error.

lErrorCode

Specifies the type of illegal key access that occurred. Possible values are:

Value	Meaning
WFS_ERR_PIN_KEYNOTFOUND	The specified key was not loaded.
WFS_ERR_PIN_KEYNOVALUE	The specified key is not loaded.
WFS_ERR_PIN_USEVIOLATION	The specified use is not supported by this key.

Comments None.

6.4 WFS_SRVE_PIN_OPT_REQUIRED

Description This event indicates that the online date/time stored in a HSM has been reached.

Event Param None.

Comments This event may be triggered by the clock reaching a previously stored online time or by the online time being set to a time that lies in the past.

The online time may be set by the command WFS_CMD_PIN_HSM_SET_TDATA or by a command WFS_CMD_PIN_SECURE_MSG_RECEIVE that contains a message from a host system containing a new online date/time.

The event does not mean that any keys or other data in the HSM is out of date now. It just indicates that the terminal should communicate with a "Personalisierungsstelle" as soon as possible using the commands WFS_CMD_PIN_SECURE_MSG_SEND / _RECEIVE and wProtocol=WFS_PIN_PROTISOPS.

7. C - Header File

```
/******************************************************************************
*                                                                             *
*xfspin.h XFS - Personal Identification Number Keypad (PIN) definitions      *
*                                                                             *
*            Version 3.00  (10/18/00)                                         *
*                                                                             *
******************************************************************************/

#ifndef __INC_XFSPIN__H
#define __INC_XFSPIN__H

#ifdef __cplusplus
extern "C" {
#endif

#include <xfsapi.h>

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


/* values of WFSPINCAPS.wClass */

#define WFS_SERVICE_CLASS_PIN              (4)
#define WFS_SERVICE_CLASS_VERSION_PIN      (0x0003) /* Version 3.00 */
#define WFS_SERVICE_CLASS_NAME_PIN         "PIN"

#define PIN_SERVICE_OFFSET                 (WFS_SERVICE_CLASS_PIN * 100)

/* PIN Info Commands */

#define WFS_INF_PIN_STATUS                 (PIN_SERVICE_OFFSET + 1)
#define WFS_INF_PIN_CAPABILITIES           (PIN_SERVICE_OFFSET + 2)
#define WFS_INF_PIN_KEY_DETAIL             (PIN_SERVICE_OFFSET + 4)
#define WFS_INF_PIN_FUNCKEY_DETAIL         (PIN_SERVICE_OFFSET + 5)
#define WFS_INF_PIN_HSM_TDATA              (PIN_SERVICE_OFFSET + 6)
#define WFS_INF_PIN_KEY_DETAIL_EX          (PIN_SERVICE_OFFSET + 7)

/* PIN Command Verbs */

#define WFS_CMD_PIN_CRYPT                  (PIN_SERVICE_OFFSET + 1)
#define WFS_CMD_PIN_IMPORT_KEY             (PIN_SERVICE_OFFSET + 3)
#define WFS_CMD_PIN_GET_PIN                (PIN_SERVICE_OFFSET + 5)
#define WFS_CMD_PIN_GET_PINBLOCK           (PIN_SERVICE_OFFSET + 7)
#define WFS_CMD_PIN_GET_DATA               (PIN_SERVICE_OFFSET + 8)
#define WFS_CMD_PIN_INITIALIZATION         (PIN_SERVICE_OFFSET + 9)
#define WFS_CMD_PIN_LOCAL_DES              (PIN_SERVICE_OFFSET + 10)
#define WFS_CMD_PIN_LOCAL_EUROCHEQUE       (PIN_SERVICE_OFFSET + 11)
#define WFS_CMD_PIN_LOCAL_VISA             (PIN_SERVICE_OFFSET + 12)
#define WFS_CMD_PIN_CREATE_OFFSET          (PIN_SERVICE_OFFSET + 13)
#define WFS_CMD_PIN_DERIVE_KEY             (PIN_SERVICE_OFFSET + 14)
#define WFS_CMD_PIN_PRESENT_IDC            (PIN_SERVICE_OFFSET + 15)
#define WFS_CMD_PIN_LOCAL_BANKSYS          (PIN_SERVICE_OFFSET + 16)
#define WFS_CMD_PIN_BANKSYS_IO             (PIN_SERVICE_OFFSET + 17)
#define WFS_CMD_PIN_RESET                  (PIN_SERVICE_OFFSET + 18)
#define WFS_CMD_PIN_HSM_SET_TDATA          (PIN_SERVICE_OFFSET + 19)
#define WFS_CMD_PIN_SECURE_MSG_SEND        (PIN_SERVICE_OFFSET + 20)
#define WFS_CMD_PIN_SECURE_MSG_RECEIVE     (PIN_SERVICE_OFFSET + 21)
#define WFS_CMD_PIN_GET_JOURNAL            (PIN_SERVICE_OFFSET + 22)
#define WFS_CMD_PIN_IMPORT_KEY_EX          (PIN_SERVICE_OFFSET + 23)
#define WFS_CMD_PIN_ENC_IO                 (PIN_SERVICE_OFFSET + 24)


/* PIN Messages */

#define WFS_EXEE_PIN_KEY                   (PIN_SERVICE_OFFSET + 1)
#define WFS_SRVE_PIN_INITIALIZED           (PIN_SERVICE_OFFSET + 2)
#define WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS    (PIN_SERVICE_OFFSET + 3)
#define WFS_SRVE_PIN_OPT_REQUIRED          (PIN_SERVICE_OFFSET + 4)

/* values of WFSPINSTATUS.fwDevice */
```

```
#define WFS_PIN_DEVONLINE                  WFS_STAT_DEVONLINE
#define WFS_PIN_DEVOFFLINE                 WFS_STAT_DEVOFFLINE
#define WFS_PIN_DEVPOWEROFF                WFS_STAT_DEVPOWEROFF
#define WFS_PIN_DEVNODEVICE                WFS_STAT_DEVNODEVICE
#define WFS_PIN_DEVHWERROR                 WFS_STAT_DEVHWERROR
#define WFS_PIN_DEVUSERERROR               WFS_STAT_DEVUSERERROR
#define WFS_PIN_DEVBUSY                    WFS_STAT_DEVBUSY

/* values of WFSPINSTATUS.fwEncStat */

#define WFS_PIN_ENCREADY                   (0)
#define WFS_PIN_ENCNOTREADY                (1)
#define WFS_PIN_ENCNOTINITIALIZED          (2)
#define WFS_PIN_ENCBUSY                    (3)
#define WFS_PIN_ENCUNDEFINED               (4)
#define WFS_PIN_ENCINITIALIZED             (5)

/* values of WFSPINCAPS.wType */

#define WFS_PIN_TYPEEPP                    (0x0001)
#define WFS_PIN_TYPEEDM                    (0x0002)
#define WFS_PIN_TYPEHSM                    (0x0004)

/* values of WFSPINCAPS.fwAlgorithms, WFSPINCRYPT.wAlgorithm */

#define WFS_PIN_CRYPTDESECB                (0x0001)
#define WFS_PIN_CRYPTDESCBC                (0x0002)
#define WFS_PIN_CRYPTDESCFB                (0x0004)
#define WFS_PIN_CRYPTRSA                   (0x0008)
#define WFS_PIN_CRYPTECMA                  (0x0010)
#define WFS_PIN_CRYPTDESMAC                (0x0020)
#define WFS_PIN_CRYPTTRIDESECB             (0x0040)
#define WFS_PIN_CRYPTTRIDESCBC             (0x0080)
#define WFS_PIN_CRYPTTRIDESCFB             (0x0100)
#define WFS_PIN_CRYPTTRIDESMAC             (0x0200)

/* values of WFSPINCAPS.fwPinFormats */

#define WFS_PIN_FORM3624                   (0x0001)
#define WFS_PIN_FORMANSI                   (0x0002)
#define WFS_PIN_FORMISO0                   (0x0004)
#define WFS_PIN_FORMISO1                   (0x0008)
#define WFS_PIN_FORMECI2                   (0x0010)
#define WFS_PIN_FORMECI3                   (0x0020)
#define WFS_PIN_FORMVISA                   (0x0040)
#define WFS_PIN_FORMDIEBOLD                (0x0080)
#define WFS_PIN_FORMDIEBOLDCO              (0x0100)
#define WFS_PIN_FORMVISA3                  (0x0200)
#define WFS_PIN_FORMBANKSYS                (0x0400)

/* values of WFSPINCAPS.fwDerivationAlgorithms */

#define WFS_PIN_CHIP_ZKA                   (0x0001)

/* values of WFSPINCAPS.fwPresentationAlgorithms */

#define WFS_PIN_PRESENT_CLEAR              (0x0001)

/* values of WFSPINCAPS.fwDisplay */

#define WFS_PIN_DISPNONE                   (1)
#define WFS_PIN_DISPLEDTHROUGH             (2)
#define WFS_PIN_DISPDISPLAY                (3)

/* values of WFSPINCAPS.fwIDKey */

#define WFS_PIN_IDKEYINITIALIZATION        (0x0001)
#define WFS_PIN_IDKEYIMPORT                (0x0002)

/* values of WFSPINCAPS.fwValidationAlgorithms */

#define WFS_PIN_DES                        (0x0001)
#define WFS_PIN_EUROCHEQUE                 (0x0002)
```

```
#define WFS_PIN_VISA                      (0x0004)
#define WFS_PIN_DES_OFFSET                (0x0008)
#define WFS_PIN_BANKSYS                   (0x0010)

/* values of WFSPINCAPS.fwKeyCheckModes and
          WFSPINIMPORTKEYEX.wKeyCheckMode */

#define WFS_PIN_KCVNONE                   (0x0000)
#define WFS_PIN_KCVSELF                   (0x0001)
#define WFS_PIN_KCVZERO                   (0x0002)

/* values of WFSPINKEYDETAIL.fwUse */

#define WFS_PIN_USECRYPT                  (0x0001)
#define WFS_PIN_USEFUNCTION               (0x0002)
#define WFS_PIN_USEMACING                 (0x0004)
#define WFS_PIN_USEKEYENCKEY              (0x0020)
#define WFS_PIN_USENODUPLICATE            (0x0040)
#define WFS_PIN_USESVENCKEY               (0x0080)
#define WFS_PIN_USEPINLOCAL              (0x10000)
#define WFS_PIN_USERSAPUBLIC             (0x20000)
#define WFS_PIN_USERSAPRIVATE            (0x40000)
#define WFS_PIN_USECHIPINFO             (0x100000)
#define WFS_PIN_USECHIPPIN              (0x200000)
#define WFS_PIN_USECHIPPS               (0x400000)
#define WFS_PIN_USECHIPMAC              (0x800000)
#define WFS_PIN_USECHIPLT              (0x1000000)
#define WFS_PIN_USECHIPMACLZ           (0x2000000)
#define WFS_PIN_USECHIPMACAZ           (0x4000000)

/* values of WFSPINFUNCKEYDETAIL.ulFuncMask */

#define WFS_PIN_FK_0                      (0x00000001)
#define WFS_PIN_FK_1                      (0x00000002)
#define WFS_PIN_FK_2                      (0x00000004)
#define WFS_PIN_FK_3                      (0x00000008)
#define WFS_PIN_FK_4                      (0x00000010)
#define WFS_PIN_FK_5                      (0x00000020)
#define WFS_PIN_FK_6                      (0x00000040)
#define WFS_PIN_FK_7                      (0x00000080)
#define WFS_PIN_FK_8                      (0x00000100)
#define WFS_PIN_FK_9                      (0x00000200)
#define WFS_PIN_FK_ENTER                  (0x00000400)
#define WFS_PIN_FK_CANCEL                 (0x00000800)
#define WFS_PIN_FK_CLEAR                  (0x00001000)
#define WFS_PIN_FK_BACKSPACE              (0x00002000)
#define WFS_PIN_FK_HELP                   (0x00004000)
#define WFS_PIN_FK_DECPOINT               (0x00008000)
#define WFS_PIN_FK_00                     (0x00010000)
#define WFS_PIN_FK_000                    (0x00020000)
#define WFS_PIN_FK_RES1                   (0x00040000)
#define WFS_PIN_FK_RES2                   (0x00080000)
#define WFS_PIN_FK_RES3                   (0x00100000)
#define WFS_PIN_FK_RES4                   (0x00200000)
#define WFS_PIN_FK_RES5                   (0x00400000)
#define WFS_PIN_FK_RES6                   (0x00800000)
#define WFS_PIN_FK_RES7                   (0x01000000)
#define WFS_PIN_FK_RES8                   (0x02000000)
#define WFS_PIN_FK_OEM1                   (0x04000000)
#define WFS_PIN_FK_OEM2                   (0x08000000)
#define WFS_PIN_FK_OEM3                   (0x10000000)
#define WFS_PIN_FK_OEM4                   (0x20000000)
#define WFS_PIN_FK_OEM5                   (0x40000000)
#define WFS_PIN_FK_OEM6                   (0x80000000)

/* values of WFSPINFUNCKEY.ulFDK */

#define WFS_PIN_FK_FDK01                  (0x00000001)
#define WFS_PIN_FK_FDK02                  (0x00000002)
#define WFS_PIN_FK_FDK03                  (0x00000004)
#define WFS_PIN_FK_FDK04                  (0x00000008)
#define WFS_PIN_FK_FDK05                  (0x00000010)
#define WFS_PIN_FK_FDK06                  (0x00000020)
#define WFS_PIN_FK_FDK07                  (0x00000040)
```

```
#define WFS_PIN_FK_FDK08                  (0x00000080)
#define WFS_PIN_FK_FDK09                  (0x00000100)
#define WFS_PIN_FK_FDK10                  (0x00000200)
#define WFS_PIN_FK_FDK11                  (0x00000400)
#define WFS_PIN_FK_FDK12                  (0x00000800)
#define WFS_PIN_FK_FDK13                  (0x00001000)
#define WFS_PIN_FK_FDK14                  (0x00002000)
#define WFS_PIN_FK_FDK15                  (0x00004000)
#define WFS_PIN_FK_FDK16                  (0x00008000)
#define WFS_PIN_FK_FDK17                  (0x00010000)
#define WFS_PIN_FK_FDK18                  (0x00020000)
#define WFS_PIN_FK_FDK19                  (0x00040000)
#define WFS_PIN_FK_FDK20                  (0x00080000)
#define WFS_PIN_FK_FDK21                  (0x00100000)
#define WFS_PIN_FK_FDK22                  (0x00200000)
#define WFS_PIN_FK_FDK23                  (0x00400000)
#define WFS_PIN_FK_FDK24                  (0x00800000)
#define WFS_PIN_FK_FDK25                  (0x01000000)
#define WFS_PIN_FK_FDK26                  (0x02000000)
#define WFS_PIN_FK_FDK27                  (0x04000000)
#define WFS_PIN_FK_FDK28                  (0x08000000)
#define WFS_PIN_FK_FDK29                  (0x10000000)
#define WFS_PIN_FK_FDK30                  (0x20000000)
#define WFS_PIN_FK_FDK31                  (0x40000000)
#define WFS_PIN_FK_FDK32                  (0x80000000)

/* values of WFSPINCRYPT.wMode */

#define WFS_PIN_MODEENCRYPT               (1)
#define WFS_PIN_MODEDECRYPT               (2)
#define WFS_PIN_MODERANDOM                (3)

/* values of WFSPINENTRY.wCompletion */

#define WFS_PIN_COMPAUTO                  (0)
#define WFS_PIN_COMPENTER                 (1)
#define WFS_PIN_COMPCANCEL                (2)
#define WFS_PIN_COMPCONTINUE              (6)
#define WFS_PIN_COMPCLEAR                 (7)
#define WFS_PIN_COMPBACKSPACE             (8)
#define WFS_PIN_COMPFDK                   (9)
#define WFS_PIN_COMPHELP                  (10)
#define WFS_PIN_COMPFK                    (11)
#define WFS_PIN_COMPCONTFDK               (12)


/* values of WFSPINSECMSG.wProtocol */
#define WFS_PIN_PROTISOAS                 (1)
#define WFS_PIN_PROTISOLZ                 (2)
#define WFS_PIN_PROTISOPS                 (3)
#define WFS_PIN_PROTCHIPZKA               (4)
#define WFS_PIN_PROTRAWDATA               (5)

/* values of WFSPINENCIO.wProtocol */
#define WFS_PIN_ENC_PROT_CH               (1)

/* XFS PIN Errors */

#define WFS_ERR_PIN_KEYNOTFOUND           (-(PIN_SERVICE_OFFSET + 0))
#define WFS_ERR_PIN_MODENOTSUPPORTED      (-(PIN_SERVICE_OFFSET + 1))
#define WFS_ERR_PIN_ACCESSDENIED          (-(PIN_SERVICE_OFFSET + 2))
#define WFS_ERR_PIN_INVALIDID             (-(PIN_SERVICE_OFFSET + 3))
#define WFS_ERR_PIN_DUPLICATEKEY          (-(PIN_SERVICE_OFFSET + 4))
#define WFS_ERR_PIN_KEYNOVALUE            (-(PIN_SERVICE_OFFSET + 6))
#define WFS_ERR_PIN_USEVIOLATION          (-(PIN_SERVICE_OFFSET + 7))
#define WFS_ERR_PIN_NOPIN                 (-(PIN_SERVICE_OFFSET + 8))
#define WFS_ERR_PIN_INVALIDKEYLENGTH      (-(PIN_SERVICE_OFFSET + 9))
#define WFS_ERR_PIN_KEYINVALID            (-(PIN_SERVICE_OFFSET + 10))
#define WFS_ERR_PIN_KEYNOTSUPPORTED       (-(PIN_SERVICE_OFFSET + 11))
#define WFS_ERR_PIN_NOACTIVEKEYS          (-(PIN_SERVICE_OFFSET + 12))
#define WFS_ERR_PIN_NOTERMINATEKEYS       (-(PIN_SERVICE_OFFSET + 14))
#define WFS_ERR_PIN_MINIMUMLENGTH         (-(PIN_SERVICE_OFFSET + 15))
#define WFS_ERR_PIN_PROTOCOLNOTSUPP       (-(PIN_SERVICE_OFFSET + 16))
#define WFS_ERR_PIN_INVALIDDATA           (-(PIN_SERVICE_OFFSET + 17))
```

```
#define WFS_ERR_PIN_NOTALLOWED            (-(PIN_SERVICE_OFFSET + 18))
#define WFS_ERR_PIN_NOKEYRAM              (-(PIN_SERVICE_OFFSET + 19))
#define WFS_ERR_PIN_NOCHIPTRANSACTIVE     (-(PIN_SERVICE_OFFSET + 20))
#define WFS_ERR_PIN_ALGORITHMNOTSUPP      (-(PIN_SERVICE_OFFSET + 21))
#define WFS_ERR_PIN_FORMATNOTSUPP         (-(PIN_SERVICE_OFFSET + 22))
#define WFS_ERR_PIN_HSMSTATEINVALID       (-(PIN_SERVICE_OFFSET + 23))
#define WFS_ERR_PIN_MACINVALID            (-(PIN_SERVICE_OFFSET + 24))
#define WFS_ERR_PIN_PROTINVALID           (-(PIN_SERVICE_OFFSET + 25))
#define WFS_ERR_PIN_FORMATINVALID         (-(PIN_SERVICE_OFFSET + 26))
#define WFS_ERR_PIN_CONTENTINVALID        (-(PIN_SERVICE_OFFSET + 27))


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

typedef struct _wfs_pin_status
{
    WORD                fwDevice;
    WORD                fwEncStat;
    LPSTR               lpszExtra;
} WFSPINSTATUS, * LPWFSPINSTATUS;

typedef struct _wfs_pin_caps
{
    WORD                wClass;
    WORD                fwType;
    BOOL                bCompound;
    USHORT              usKeyNum;
    WORD                fwAlgorithms;
    WORD                fwPinFormats;
    WORD                fwDerivationAlgorithms;
    WORD                fwPresentationAlgorithms;
    WORD                fwDisplay;
    BOOL                bIDConnect;
    WORD                fwIDKey;
    WORD                fwValidationAlgorithms;
    WORD                fwKeyCheckModes;
    LPSTR               lpszExtra;
} WFSPINCAPS, * LPWFSPINCAPS;

typedef struct _wfs_pin_key_detail
{
    LPSTR               lpsKeyName;
    WORD                fwUse;
    BOOL                bLoaded;
} WFSPINKEYDETAIL, * LPWFSPINKEYDETAIL;

typedef struct _wfs_pin_fdk
{
    ULONG               ulFDK;
    USHORT              usXPosition;
    USHORT              usYPosition;
} WFSPINFDK, * LPWFSPINFDK;

typedef struct _wfs_pin_func_key_detail
{
    ULONG               ulFuncMask;
    USHORT              usNumberFDKs;
    LPWFSPINFDK       * lppFDKs;
} WFSPINFUNCKEYDETAIL, * LPWFSPINFUNCKEYDETAIL;

typedef struct _wfs_pin_key_detail_ex
{
    LPSTR         lpsKeyName;
    DWORD         dwUse;
    BYTE          bGeneration;
    BYTE          bVersion;
    BYTE          bActivatingDate[4];
    BYTE          bExpiryDate[4];
    BOOL          bLoaded;
} WFSPINKEYDETAILEX, * LPWFSPINKEYDETAILEX;

/*=================================================================*/
```

```
/* PIN Execute Command Structures */
/*=================================================================*/

typedef struct _wfs_hex_data
{
    USHORT              usLength;
    LPBYTE              lpbData;
} WFSXDATA, * LPWFSXDATA;

typedef struct _wfs_pin_crypt
{
    WORD                wMode;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
    WORD                wAlgorithm;
    LPSTR               lpsStartValueKey;
    LPWFSXDATA          lpxStartValue;
    BYTE                bPadding;
    BYTE                bCompression;
    LPWFSXDATA          lpxCryptData;
} WFSPINCRYPT, * LPWFSPINCRYPT;

typedef struct _wfs_pin_import
{
    LPSTR               lpsKey;
    LPSTR               lpsEncKey;
    LPWFSXDATA          lpxIdent;
    LPWFSXDATA          lpxValue;
    WORD                fwUse;
} WFSPINIMPORT, * LPWFSPINIMPORT;

typedef struct _wfs_pin_derive
{
    WORD                wDerivationAlgorithm;
    LPSTR               lpsKey;
    LPSTR               lpsKeyGenKey;
    LPSTR               lpsStartValueKey;
    LPWFSXDATA          lpxStartValue;
    BYTE                bPadding;
    LPWFSXDATA          lpxInputData;
    LPWFSXDATA          lpxIdent;
 } WFSPINDERIVE, * LPWFSPINDERIVE;

typedef struct _wfs_pin_getpin
{
    USHORT              usMinLen;
    USHORT              usMaxLen;
    BOOL                bAutoEnd;
    CHAR                cEcho;
    ULONG               ulActiveFDKs;
    ULONG               ulActiveKeys;
    ULONG               ulTerminateFDKs;
    ULONG               ulTerminateKeys;
} WFSPINGETPIN, * LPWFSPINGETPIN;

typedef struct _wfs_pin_entry
{
    USHORT              usDigits;
    WORD                wCompletion;
} WFSPINENTRY, * LPWFSPINENTRY;

typedef struct _wfs_pin_local_des
{
    LPSTR               lpsValidationData;
    LPSTR               lpsOffset;
    BYTE                bPadding;
    USHORT              usMaxPIN;
    USHORT              usValDigits;
    BOOL                bNoLeadingZero;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
    LPSTR               lpsDecTable;
} WFSPINLOCALDES, * LPWFSPINLOCALDES;
```

```
typedef struct _wfs_pin_create_offset
{
    LPSTR               lpsValidationData;
    BYTE                bPadding;
    USHORT              usMaxPIN;
    USHORT              usValDigits;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
    LPSTR               lpsDecTable;
} WFSPINCREATEOFFSET, * LPWFSPINCREATEOFFSET;

typedef struct _wfs_pin_local_eurocheque
{
    LPSTR               lpsEurochequeData;
    LPSTR               lpsPVV;
    WORD                wFirstEncDigits;
    WORD                wFirstEncOffset;
    WORD                wPVVDigits;
    WORD                wPVVOffset;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
    LPSTR               lpsDecTable;
} WFSPINLOCALEUROCHEQUE, * LPWFSPINLOCALEUROCHEQUE;

typedef struct _wfs_pin_local_visa
{
    LPSTR               lpsPAN;
    LPSTR               lpsPVV;
    WORD                wPVVDigits;
    LPSTR               lpsKey;
    LPWFSXDATA          lpxKeyEncKey;
} WFSPINLOCALVISA, * LPWFSPINLOCALVISA;

typedef struct _wfs_pin_presentidc
{
    WORD                wPresentAlgorithm;
    WORD                wChipProtocol;
    ULONG               ulChipDataLength;
    LPBYTE              lpbChipData;
    LPVOID              lpAlgorithmData;
} WFSPINPRESENTIDC, * LPWFSPINPRESENTIDC;

typedef struct _wfs_pin_present_result
{
    WORD                wChipProtocol;
    ULONG               ulChipDataLength;
    LPBYTE              lpbChipData;
} WFSPINPRESENTRESULT, * LPWFSPINPRESENTRESULT;

typedef struct _wfs_pin_presentclear
{
    ULONG               ulPINPointer;
    USHORT              usPINOffset;
} WFSPINPRESENTCLEAR, * LPWFSPINPRESENTCLEAR;

typedef struct _wfs_pin_block
{
    LPSTR               lpsCustomerData;
    LPSTR               lpsXORData;
    BYTE                bPadding;
    WORD                wFormat;
    LPSTR               lpsKey;
    LPSTR               lpsKeyEncKey;
} WFSPINBLOCK, * LPWFSPINBLOCK;

typedef struct _wfs_pin_getdata
{
    USHORT              usMaxLen;
    BOOL                bAutoEnd;
    ULONG               ulActiveFDKs;
    ULONG               ulActiveKeys;
    ULONG               ulTerminateFDKs;
    ULONG               ulTerminateKeys;
} WFSPINGETDATA, * LPWFSPINGETDATA;
```

```
typedef struct _wfs_pin_key
{
    WORD          wCompletion;
    ULONG         ulDigit;
} WFSPINKEY, * LPWFSPINKEY;

typedef struct _wfs_pin_data
{
    USHORT              usKeys;
    LPWFSPINKEY        *lpPinKeys;
    WORD                wCompletion;
} WFSPINDATA, * LPWFSPINDATA;

typedef struct _wfs_pin_init
{
    LPWFSXDATA          lpxIdent;
    LPWFSXDATA          lpxKey;
} WFSPININIT, * LPWFSPININIT;

typedef struct _wfs_pin_local_banksys
{
    LPWFSXDATA          lpxATMVAC;
} WFSPINLOCALBANKSYS, * LPWFSPINLOCALBANKSYS;

typedef struct _wfs_pin_banksys_io
{
    ULONG               ulLength;
    LPBYTE              lpbData;
} WFSPINBANKSYSIO, * LPWFSPINBANKSYSIO;

typedef struct _wfs_pin_secure_message
    {
    WORD          wProtocol;
    ULONG         ulLength;
    LPBYTE        lpbMsg;
} WFSPINSECMSG, * LPWFSPINSECMSG;

typedef struct _wfs_pin_import_key_ex
{
    LPSTR       lpsKey;
    LPSTR       lpsEncKey;
    LPWFSXDATA  lpxValue;
    LPWFSXDATA  lpxControlVector;
    DWORD       dwUse;
    WORD        wKeyCheckMode;
    LPWFSXDATA  lpxKeyCheckValue;
} WFSPINIMPORTKEYEX, * LPWFSPINIMPORTKEYEX;

typedef struct _wfs_pin_enc_io
{
    WORD          wProtocol;
    ULONG         ulDataLength;
    LPVOID        lpvData;
} WFSPINENCIO, *LPWFSPINENCIO;

/*=================================================================*/
/* PIN Message Structures */
/*=================================================================*/

typedef struct _wfs_pin_access
{
    LPSTR         lpsKeyName;
    LONG          lErrorCode;
} WFSPINACCESS, * LPWFSPINACCESS;


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

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

```
#endif    /* __INC_XFSPIN__H */
```

8. German ZKA GeldKarte

The PIN service is able to handle the German "Geldkarte", which is an electronic purse specified by the ZKA (Zentraler Kreditausschuß).

For anyone attempting to write an application that handles these chip cards, it is essential to read and understand the specifications published by

Bank-Verlag, Köln
Postfach 30 01 91
D-50771 Köln

Phone: +49 221 5490-0

Fax: +49 221 5490-120

8.1 How to use the SECURE_MSG commands

This is to describe how an application should use the WFS_CMD_PIN_SECURE_MSG_SEND and WFS_CMD_PIN_SECURE_MSG_RECEIVE commands for transactions involving chipcards with a German ZKA GeldKarte chip.

- Applications must call SECURE_MSG_SEND for every command they send to the chip or to a host system, including those commands that do not actually require secure messaging. This enables the service provider to remember security-relevant data that may be needed or checked later in the transaction.
- Applications must pass a complete message as input to SECURE_MSG_SEND, with all fields - including those that will be filled by the service provider - being present in the correct length. All fields that are not filled by the service provider must be filled with the ultimate values in order to enable MACing by the service provider.
- Every command SECURE_MSG_SEND that an application issues must be followed by exactly one command SECURE_MSG_RECEIVE that informs the service provider about the response from the chip or host. If no response is received (timeout or communication failure) the application must issue a SECURE_MSG_RECEIVE command with **`lpSecMsgIn->lpbMsg = NULL`** to inform the service provider about this fact.
- If a system is restarted after a SECURE_MSG_SEND was issued to the service provider but before the SECURE_MSG_RECEIVE was issued, the restart has the same effect as a SECURE_MSG_RECEIVE command with **`lpSecMsgIn->lpbMsg = NULL`**.
- Between a SECURE_MSG_SEND and the corresponding SECURE_MSG_RECEIVE no SECURE_MSG_SEND with the same **`lpSecMsgIn->wProtocol`** must be issued. Other WFS_CMD_PIN... commands – including SECURE_MSG_SEND / RECEIVE with different **`wProtocol`** – may be used.

8.2 Protocol WFS_PIN_PROTISOAS

This protocol handles ISO8583 messages between an ATM and an authorization system (AS).

Only messages in the new ISO format, with new PAC/MAC-format using session keys and Triple-DES are supported.

Authorization messages may be used to dispense the amount authorized in cash or to load the amount into an electronic purse (GeldKarte).

For loading a GeldKarte the only type of authorization supported is a transaction originating from track 3 of a German ec-card (message types 0200/0210 for authorization and 0400/0410 for reversal)

For dispensing cash, transactions originating from international cards (message types 0100/0110 and 0400/0410) are supported as well.

The following bitmap positions are filled by the service provider:

- BMP11 Trace-Nummer
- BMP52 PAC
- BMP57 Verschlüsselungsparameter (only the challenge values RND_{MES} and RND_{PAC})
- BMP64 MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

The following bitmap positions are checked by the service provider and have to be filled by the application:

- Nachrichtentyp
- BMP3 Abwicklungskennzeichen (only for GeldKarte, not for cash)
- BMP4 Transaktionsbetrag (only for GeldKarte, not for cash)
- BMP41 Terminal-ID
- BMP42 Betreiber-BLZ

For a documentation of authorization messages see:

Regelwerk für das deutsche ec-Geldautomaten-System
Stand: 22. Nov. 1999

Bank-Verlag, Köln
Autorisierungszentrale GA/POS der privaten Banken
Spezifikation für GA-Betreiber
Version 3.12
31. Mai 2000

dvg Hannover
Schnittstellenbeschreibung für Autorisierungsanfragen bei nationalen GA-Verfügungen unter Verwendung der Spur 3
Version 2.5
Stand: 15.03.2000

dvg Hannover
Schnittstellenbeschreibung für Autorisierungsanfragen bei internationalen Verfügungen unter Verwendung der Spur 2
Version 2.6
Stand: 30.03.2000

8.3 Protocol WFS_PIN_PROTISOLZ

This protocol handles ISO8583 messages between a „Ladeterminal" and a „Ladezentrale" (LZ).

Only messages in the new ISO format, with new MAC-format using session keys and Triple-DES are supported.

Both types of GeldKarte chip (type 0 = DEM, type 1 = EUR) are supported.

The following bitmap positions are filled by the service provider:

- BMP11: Trace-Nummer
- BMP57: Verschlüsselungsparameter (only the challenge value RND_{MES})
- BMP64: MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

The following bitmap positions are checked by the service provider and have to be filled by the application:

- Nachrichtentyp
- BMP3: Abwicklungskennzeichen
- BMP4: Transaktionsbetrag
- BMP12: Uhrzeit
- BMP13: Datum
- BMP25: Konditionscode
- BMP41: Terminal-ID
- BMP42: Betreiber-BLZ (caution: "Ladeentgelt" also in BMP42 is not set by the EPP)
- BMP61: Online-Zeitpunkt
- BMP62: Chipdaten

The following bitmap positions are only checked if they are available:

- BMP43: Standort
- BMP60: Kontodaten Ladeterminal

For a documentation of the Ladezentrale interface see:

ZKA / Bank-Verlag, Köln
Schnittstellenspezifikation für die ec-Karte mit Chip
Geldkarte Ladeterminals
Version 3.0
2. 4. 1998

8.4 Protocol WFS_PIN_PROTISOPS

This protocol handles ISO8583 messages between a terminal and a "Personalisierungsstelle" (PS). These messages are about OPT.

The service provider creates the whole message with WFS_CMD_PIN_SECURE_MSG_SEND, including message type and bitmap.

For a documentation of the Personalisierungsstelle interface see:

ZKA / Bank-Verlag, Köln
Schnittstellenspezifikation für die ec-Karte mit Chip
Online-Personalisierung von Terminal-HSMs
Version 3.0
2. 4. 1998

8.5 Protocol WFS_PIN_PROTCHIPZKA

This protocol is intended to handle messages between the application and a GeldKarte.

Both types of GeldKarte are supported.

Both types of load transactions ("Laden vom Kartenkonto" and "Laden gegen andere Zahlungsmittel") are supported.

See the chapter "Command Sequence" below for the actions that service providers take for the various chip card commands.

Only the command APDUs to and the response APDUs from the chip must be passed to the service provider, the ATR (answer to reset) data from the chip is not passed to the service provider.

For a documentation of the chip commands used to load a GeldKarte see:

ZKA / Bank-Verlag, Köln
Schnittstellenspezifikation für die ec-Karte mit Chip
Ladeterminals
Version 3.0
2. 4. 1998

8.6 Protocol WFS_PIN_PROTRAWDATA

This protocol is intended for vendor-specific purposes. Generally the use of this protocol is not recommended and should be restricted to issues that are impossible to handle otherwise.

For example a HSM that requires vendor-specific, cryptographically secured data formats for importing keys or terminal data may use this protocol.

Applicaton programmers should be aware that the use of this command may prevent their applications from running on different hardware.

8.7 Command Sequence

The following list shows the sequence of actions an application has to take for the various GeldKarte Transactions. Please note that this is a summary and is just intended to clarify the purpose of the chipcard-related WFS_CMD_PIN_... commands. In no way it can replace the ZKA specifications mentioned above.

Command **WFS_CMD_PIN_...**	**wProtocol** **WFS_PIN_PROT...**	**lpbMsg**	**Service Provider´s actions**
Preparation for Load/Unload			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	recognize type of chip
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_ID	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_ID	store EF_ID
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_LLOG	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_LLOG	
SECURE_MSG_SEND	CHIPZKA	Command APDU READ_RECORD EF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_BÖRSE	
SECURE_MSG_SEND	CHIPZKA	Command APDU READ_RECORD EF_BETRAG	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_BETRAG	
Load against other ec-Card			
SECURE_MSG_SEND	CHIPZKA	**for type 0 chips only** Command APDU READ RECORD EF_KEYD	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_KEYD	
SECURE_MSG_SEND	CHIPZKA	**for type 1 chips only** Command APDU GET KEYINFO	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND1 from Chip	store RND1
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN with Secure Msg.	fill -Terminal ID -Traceno. -RND2 -MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	store response APDU for later check of ISOLZ message, BMP 62
SECURE_MSG_SEND	ISOAZ	ISO8583 message 0200 Authorization Request	fill - Traceno. (BMP 11) - PAC (BMP 52) - RND$_{MES}$ + RND$_{PAC}$ (BMP 57) - MAC (BMP 64) check other security relevant fields
SECURE_MSG_RECEIVE	ISOAZ	ISO8583 message 0210 Authorization Response	check MAC and other security relevant fields
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0200 Ladeanfrage	fill - Traceno. (BMP 11) - RND$_{MES}$ (BMP 57) - MAC (BMP 64) check other security relevant fields.

Command **WFS_CMD_PIN_...**	**wProtocol** **WFS_PIN_PROT...**	**lpbMsg**	**Service Provider´s actions**
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0210 Ladeantwort	check MAC and other security relevant fields, store BMP62 for later use in LADEN command.
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND3 from chip	store RND3
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN with Secure Msg.	provide complete command from BMP62 of ISOLZ response , compute command MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check response MAC
GET_JOURNAL	ISOLZ	Vendor specific	
GET_JOURNAL	ISOAZ	Vendor specific	
Reversal of a Load against other ec-Card			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND5 from chip	store RND5
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN with Secure Msg.	fill -Terminal ID -Traceno. -RND6 -Keyno. KGK_{LT} -MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	store response APDU for later check of ISOLZ message, BMP 62
SECURE_MSG_SEND	ISOAZ	ISO8583 message 0400 Storno	fill - Traceno. (BMP 11) - PAC (BMP 52) - RND_{MES} + RND_{PAC} (BMP 57) - MAC (BMP 64) check other security relevant fields
SECURE_MSG_RECEIVE	ISOAZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields.
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0400 Storno	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields, store BMP62 for later use in LADEN command.
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND7 from chip	store RND7
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN with Secure Msg.	provide complete command from BMP62 of ISOLZ response , compute command MAC
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check response MAC
GET_JOURNAL	ISOLZ	Vendor specific	
GET_JOURNAL	ISOAZ	Vendor specific	

PIN Verification Type 0			
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND0 from chip	store RND0
SECURE_MSG_SEND	CHIPZKA	Command APDU EXTERNAL AUTHENTICATE	fill -Keyno. K_{INFO} -ENCRND
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU PUT DATA	fill RND1
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU READ RECORD EF_INFO with Secure Messaging	
SECURE_MSG_RECEIVE	CHIPZKA	record EF_INFO	check MAC
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND2 from chip	store RND2
SECURE_MSG_SEND	CHIPZKA	Command APDU VERIFY	provide complete command APDU
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
PIN Verification Type 1			
SECURE_MSG_SEND	CHIPZKA	Command APDU GET KEYINFO	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU GET CHALLENGE	
SECURE_MSG_RECEIVE	CHIPZKA	Random number RND0 from chip	store RND0
SECURE_MSG_SEND	CHIPZKA	Command APDU MUTUAL AUTHENTICATE	fill ENC0
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check ENC1
SECURE_MSG_SEND	CHIPZKA	Command APDU VERIFY	provide complete command APDU
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	check MAC
„Laden vom Kartenkonto“ (both types)			
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0200 Ladeanfrage	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0210 Ladeantwort	check MAC and other security relevant fields.
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	

Reversal of a „Laden vom Kartenkonto“			
SECURE_MSG_SEND	CHIPZKA	Command APDU SELECT FILE DF_BÖRSE	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN EINLEITEN	fill -Terminal ID -Traceno.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message 0400 Storno	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	Command APDU LADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	
Unload			
SECURE_MSG_SEND	CHIPZKA	ENTLADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message Entladeanfrage 0200	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Entladeantwort 0210	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	ENTLADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	CHIPZKA	ENTLADEN EINLEITEN	fill -Terminal ID -Trace No.
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
SECURE_MSG_SEND	ISOLZ	ISO8583 message Entladequittung 0202	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Entladebestätigung 0212	check MAC and other security relevant fields
SECURE_MSG_SEND	CHIPZKA	Command APDU ENTLADEN	
SECURE_MSG_RECEIVE	CHIPZKA	Response APDU	
GET_JOURNAL	ISOLZ	Vendor specific	

Repeated Messages (Stornowiederholung / Entladequittungswiederholung)			
SECURE_MSG_SEND	ISOLZ	ISO8583 message Stornowiederholung 0401 or Entladequittungswiederholung 0203	fill - Traceno. (BMP 11) - RND_{MES} (BMP 57) - MAC (BMP 64) check other security relevant fields.
SECURE_MSG_RECEIVE	ISOLZ	ISO8583 message Stornoantwort 410 or Entladebestätigung 0212	check MAC and other security relevant fields
GET_JOURNAL	ISOLZ	Vendor specific	