WOSA/XFS
Programmer's Reference Manual

B006-0000-6001
Issue 1
09/1997

The product described in this book is a licensed product of NCR Corporation.

Trademark Information

It is the policy of NCR Corporation (NCR) to improve products as new technology, components, software, and firmware become available. NCR, therefore, reserves the right to change specifications without prior notice.

All features, functions, and operations described herein may not be marketed by NCR in all parts of the world. In some instances, photographs are of equipment prototypes. Therefore, before using this document, consult with your NCR representative or NCR office for information that is applicable and current.

To maintain the quality of our publications, we need your comments on the accuracy, clarity, organization, and value of this book.

Address correspondence to:

NCR (Scotland) Ltd.
Information Products
Kingsway West
Dundee
Scotland
DD2 3XX

Federal Communications Commission (FCC) Radio Frequency Interference Statement

Note: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Canadian Class A Device Declaration

This digital apparatus does not exceed the Class A limits for radio noise emissions from digital apparatus set out in the Radio Interference Regulations of the Canadian Department of Communications.

Le présent appareil numérique n'émet pas de bruits radioélectriques dépassant les limites applicables aux appareils numériques de la classe A prescrites dans le Réglement sur le brouillage radioélectrique édicté par le ministrère des Communications du Canada.

Information to User

This equipment must be installed and used in strict accordance with the manufacturer's instructions. However, there is no guarantee that interference to radio communications will not occur in a particular commercial installation. If this equipment does cause interference, which can be determined by turning the equipment off and on, the user is encouraged to consult an NCR service representative immediately.

Caution

NCR Corporation is not responsible for any radio or television interference caused by unauthorised modifications of this equipment or the substitution or attachment of connecting cables and equipment other than those specified by NCR. Such unauthorized modifications, substitutions, or attachments may void the user's authority to operate the equipment. The correction of interference caused by such unauthorized modifications, substitutions, or attachments will be the responsibility of the user.

Contents

References

Overview

Purpose and Audience

This publication provides programmers with NCR specific information needed to develop WOSA/XFS compliant applications to run on 56XX and 58XX NCR Self-Service Financial Terminals.

This book assumes that the reader has a working knowledge of writing applications for the operating environment being used, and that appropriate manuals and references for this environment are available.

What is in this Publication

Chapter 1 - Introduction

This chapter presents a high level description of the WOSA/XFS architecture and components, and lists the requirements for running a WOSA/XFS application on NCR Self-Service Terminals.

Chapter 2 - Developing an Application Using WOSA/XFS

This chapter describes the process of developing, debugging and installing a WOSA/XFS application on an SST.

Chapter 3 - Device Class Interface

This section describes NCR specific implementation details of the WOSA/XFS Service Provider set.

How To Use This Guide

This guide is not intended to be used in isolation but in conjunction with the WOSA/XFS Revision 2.00 Programmer's Reference documents which are freely available and contained in the WOSA/XFS Software Developers Kit (SDK) which can be found on the Microsoft World Wide Web page http://www.microsoft.com/, as well as in Microsoft Developer Network (MSDN) products.

Pre-requisite

The WOSA/XFS Manager Version 2.00 or above must be installed on the Self Service Terminal before the NCR Service Provider set can be used. The Manager provides the API interface for the application. The WOSA/XFS Manager is contained within the WOSA/XFS Software Developers Kit (SDK).

Revision Record

This is the initial release of the Publication.

Chapter 1

Table of Contents

WOSA/XFS - An Introduction

What is WOSA/XFS ?

'Windows Open Services Architecture' (WOSA), comprises a family of stable, open-ended interfaces for enterprise computing environments. These interfaces hide system complexities from users and application developers.

Using WOSA, you can seamlessly integrate Windows and Windows-based applications with all the services and enterprise capabilities that you need. WOSA includes the following interfaces:

- Open Database Connectivity (ODBC) for standard access to databases.
- Messaging Application Programming Interface (MAPI) for standard access to messaging services.
- Communications support, including Windows SNA, RPC and Sockets.

The Banking Solutions Vendor Council (BSVC), an organization of leading vendors of information technology to the financial services industry, has extended WOSA by defining a Windows-based client-server architecture for financial applications. These extensions, appropriately called Extensions for Financial Services (XFS), include a set of APIs and SPIs common to multiple financial applications.

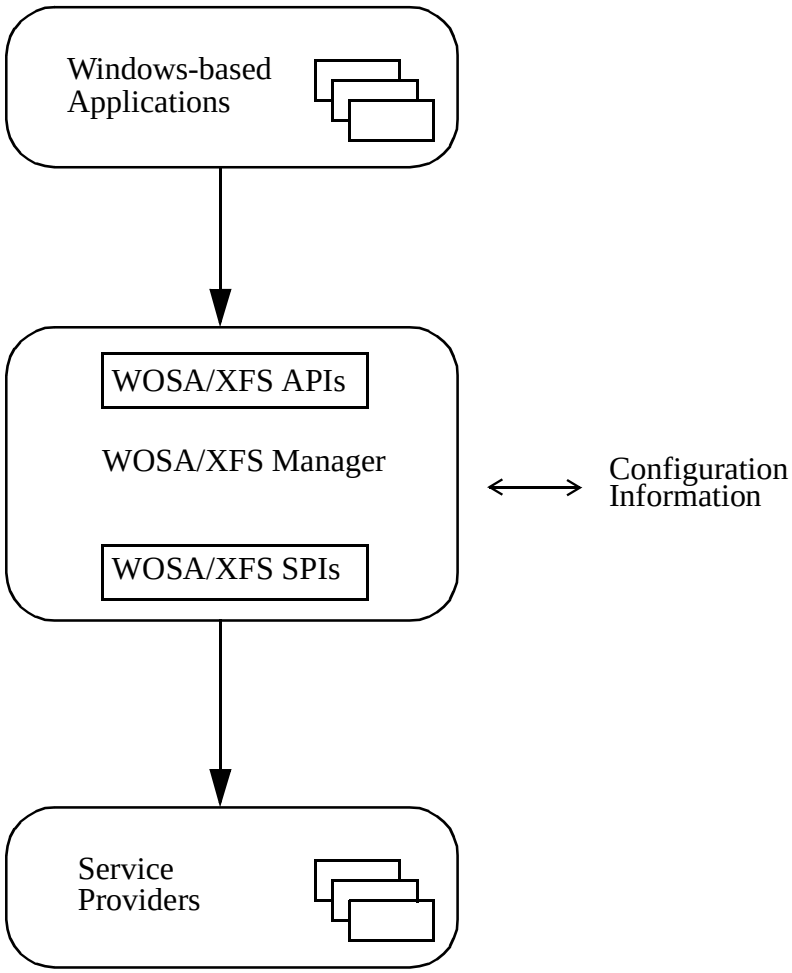
At the time of developing this manual, WOSA/XFS includes specifications to the following financial peripherals:

Device Class	**Class Name**
Printers	PTR
Identification Card Units	IDC
Cash Dispensers	CDM
PINpads	PIN
Check Readers and Scanners	CHK
Depository Units	DEP
Text Terminal Units	TTU
Sensors and Indicators Units	SIU
Vendor Dependent Mode	VDM
Cameras	CAM

The WOSA/XFS Architecture

As depicted below, WOSA/XFS, just like other WOSA elemets, defines the following:

- a set of APIs
- a corresponding set of SPIs
- supporting services providing access to financial services for Windows-based applications.



WOSA Extensions for Financial Services Architecture

The XFS Manager

A pre-requiste with all implementations of the WOSA/XFS is the vendor-independent XFS Manager. The application communicates with service providers, via the XFS Manager, using the API set. You can invoke most of these APIs either "synchronously" or "asynchronously".

In the former case, the Manager blocks the application until the API completes its function. In the latter, the application regains control immediately. However, the requested function is performed in parallel.

The XFS Manager maps the specified API to the corresponding SPI. It then routes this request to the appropriate service provider. The Manager uses configuration information to route the API that is directed at a "logical service" to the proper service provider entry point. The entry point is always local, although the final target may be remote.

The Service Provider

The Service Provider is the vendor-dependent component of the WOSA/XFS solution. The primary functions of the service providers are to:

- translate generic service requests to service-specific commands,
- route the requests to either a local service or device, or to one on a remote system, effectively defining a peer-to-peer interface among service providers,
- arbitrate access by multiple applications to a single service or device, providing exclusive access when requested,
- manage hardware interfaces to services or devices, and
- manage the asynchronous nature of the services and devices in an appropriate manner, always presenting this capability to the XFS Manager and the applications via Windows messages.

Look up Reference 1 for the definition of the functionality of services, of the architecture, and of the API and SPI sets.

Requirements For WOSA/XFS

Platforms

NCR supports a service provider set for the *person*S Platform for Windows NT D531-0300-000 running on 56XX & 58XX SSTs.

In addition, the WOSA/XFS solution requires the WOSA/XFS Manager Ver. 2.00 (or above), and one or more components of the NCR WOSA/XFS Windows-NT Service Provider set (Product ID:D531-0289-0000), which supports the following devices:

- Currency Dispenser Service Provider
- Envelope Depository/Dispenser Service Provider
- Night Safe Depository Service Provider
- Identification Card Service Provider
- Passbook Printer Service Provider
- Receipt/Journal Printer Service Provider
- Statement Printer Service Provider
- Pinpad Service Provider
- Sensors and Indicators Service Provider
- Text Terminal Service Provider
- Vendor Dependent Mode Service Provider

Application Requirements

NCR's SP set supports user-defined, WOSA compliant applications.

Hardware Requirements

The following is the minimum recommended configuration for the PC core of the SST on which the NCR Service Providers run:

56XX/58XX SSTs

- PC/AT 486 DX-2
- 32 MB RAM
- 1.44 MB FDD
- 540 MB Hard Disk
- VGA Color/Mono Monitor
- CD-ROM Drive

Chapter 2

Table of Contents

Developing an Application Using WOSA/XFS

This chapter describes how to develop a WOSA/XFS application on your Self-Service Terminals.

Setting up the Development Environment

This section describes how to set up your PC for developing WOSA/XFS application. It also provides information on the different libraries, header files and the compiler options to be used.

It is mandatory to have the following software installed in your PC before you start developing a WOSA/XFS application:

- WINDOWS NT 4.00 or higher
- Microsoft Visual C++ 4.0 or higher
- WOSA/XFS Manager, version 2.XX or higher, with the following components:

Libraries:

MSXFS.LIB	-	Basic XFS API and SPI functions
XFS_CONF.LIB	-	Configuration functions
XFS_SUPP.LIB	-	Support functions

Header files:

XFSSPI.H
XFSAPI.H
XFSADMIN.H
XFSCONF.H
XFSCDM.H
XFSDEP.H
XFSIDC.H
XFSPIN.H
XFSPTR.H
XFSSIU.H
XFSTTU.H
XFSVDM.H

Compiler Options

All structures passed to/returned by the WOSA/XFS sub-system are byte aligned. Byte alignment MUST be set to 1 either by using the #pragma pack(1) pragma, or by setting the byte alignment to 1 under the Build->Settings->Compiler->Code Generation Option.

Sample Application with Description

The 'C' code in the figure below illustrates how you would program your application to use the WOSA/XFS API to gain control of the NCR CDM SP and make the dispenser dispense £120. After this dispense, for the sake of the example, the application releases the SP for use by other applications and closes its session with the XFS Manager. With the CDM SP locked, your application could have gone on to control a series of CDM activities. Having closed the session with the CDM, it could also have gone on and worked with other financial peripherals before "cleaning up" and ending its session with the XFS manager. Note that this application should be linked with MSXFS.LIB.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
      HSERVICE       hService=0;
      BOOL           fSuccess=EXIT_SUCCESS;               // Default return

      if(Wfs_Startup())
      {
            // Session established with XFS Manager
            if(Wfs_Open(&hService))
            {
                  // Session established with CDM SP
                  if(Wfs_Lock(hService))                 // CDM SP locked for exclusive use
                  {
                        Wfs_ExecuteDispense(hService);// Dispense bills
                        Wfs_Unlock(hService);          // Release the CDM SP
                  }
                  else                                   // Lock failed
                  {
                        fSuccess=EXIT_FAILURE;
                  }
                  Wfs_Close(hService);
            }
            else                                         // Wfs_Open failed
            {
                  fSuccess=EXIT_FAILURE;
            }
            Wfs_Cleanup();
      }
      else                                               // Wfs_Startup failed
      {
            fSuccess=EXIT_FAILURE;
      }

      return fSuccess;
}
```

Bringing in the WOSA/XFS headers and definitions

Before your application could use the WOSA/XFS API to perform the task illustrated above, you would first have to bring in the appropriate headers and make the following definitions:

```
#include <xfscdm.h>                          // For the cash dispenser
#include <xfsspi.h>                          // Service Provider interface
#include <xfsadmin.h>                        // XFS Manager support functions

#define RECOGNISED_VERSIONS                  0x00000202    // See Note
#define TWO_MINUTES                          02*60*1000
#define TEN_MINUTES                          10*60*1000
#define TEN_SECONDS                          10*1000
#define EXIT_SUCCESS                         0
#define EXIT_FAILURE                         -1
#define WFS_TRACE NONE                       0
```

Note: RECOGNISED_VERSIONS specifies the range of service provider interface versions that can be accommodated.

Function Prototypes

Before you can compile the sample code the compiler requires that the following function prototypes be declared:

```
BOOL          Wfs_Startup    (void);
BOOL          Wfs_Open(LPHSERVICE lphService);
BOOL          Wfs_Lock(HSERVICE hService);
BOOL          Wfs_ExecuteDispense(HSERVICE hService);
BOOL          Wfs_Unlock(HSERVICE hService);
BOOL          Wfs_Close(HSERVICE hService);
BOOL          Wfs_Cleanup(void);
```

Functions Used

The functions to dispense £ 120 are as follows

```
BOOL          Wfs_Startup(void)
{
        WFSVERSIONWfsVersion;

        return (WFSStartUp(RECOGNISED_VERSIONS,
                           &WfsVersion) == WFS_SUCCESS);
}
```

```
BOOL            Wfs_Open(LPHSERVICE lphService)
{
        WFSVERSIONSvcVersion, SpiVersion;
        char szLogicalName[]="CurrencyDispenser1";

        return (WFSOpen  (szLogicalName,        // Logical name
                    WFS_DEFAULT_HAPP,           // App handle
                    "WOSA_TEST",                // App ID
                    WFS_TRACE_NONE,             // Trace Level
                    TWO_MINUTES,                // Timeout
                    RECOGNISED_VERSIONS,        // Srvc Versions
                    &SvcVersion,                // Returned version
                    &SpiVersion,                // Returned SPI version
                    lphService                  // Returned service handle
                     ) == WFS_SUCCESS);
}
```

```
BOOL            Wfs_Lock(HSERVICE hService)
{
        LPWFSRESULTlpResult=NULL;

        if(WFSLock(hService, TEN_SECONDS, &lpResult) == WFS_SUCCESS)
        {
              // Free the result structure returned by SP
              if(WFSFreeResult(lpResult) != WFS_SUCCESS)
              {
                    // Failed to free the result structure
                    return FALSE;
              }
        }
        else
        {
              // WFSLock Failed
              return FALSE;
        }
        return TRUE;
}
```

```
BOOL           Wfs_ExecuteDispense(HSERVICE hService)
{
      WFSCDMDISPENSE             tDispense;
      WFSCDMDENOMINATION         tDenomination;
      LPWFSRESULT                lpResult=NULL;
      HRESULT                    hResult=WFS_SUCCESS;
      ULONG                      ulaValues[5];

      tDispense.usTellerID       =    0;
      tDispense.usMixNumber      =    WFS_CDM_INDIVIDUAL;
      tDispense.bPresent         =    TRUE;
      tDispense.usPosition       =    WFS_CDM_POSCENTER;

      ulaValues[0]               =    0;
      ulaValues[1]               =    2;             // 2 £5  bills
      ulaValues[2]               =    2;             // 2 £10 bills
      ulaValues[3]               =    2;             // 2 £20 bills
      ulaValues[4]               =    1;             // 1 £50 bill

      tDenomination.lpulValues   =    ulaValues;

      strncpy(tDenomination.cCurrencyID, "GBP", 3);

      tDenomination.ulAmount     =    120;
      tDenomination.usCount      =    5;
      tDenomination.ulCashBox    =    0;
      tDispense.lpDenomination   =    &tDenomination;

      hResult  =   WFSExecute( hService,
                               WFS_CMD_CDM_DISPENSE,
                               &tDispense,
                               TWO_MINUTES,
                               &lpResult
                           );

      // Free the result structure allocated by the SP
      WFSFreeResult(lpResult);

      return(hResult == WFS_SUCCESS);
}
```

```
BOOL       Wfs_Unlock(HSERVICE hService)
{
      return(WFSUnlock(hService) == WFS_SUCCESS);
}
```

```
BOOL       Wfs_Close(HSERVICE hService)
{
      return(WFSClose (hService) == WFS_SUCCESS);
}
```

```
BOOL       Wfs_Cleanup(void)
{
      return(WFSCleanUp () == WFS_SUCCESS);
}
```

Note:

1. The range of the Service Provider versions that the application can support is specified in the sixth parameter of the WFSOpen( ) call. For example, if the range is from 1.07 to 2.00, then it should be specified as 0x02000107.

2. The range of the XFS Manager versions that the application can support is from 1.0 to 2.0. This is specified as the first parameter in the WFSStartUp( ) call.

Installing the SP set on an SST

NCR's WOSA/XFS Service Provider set is packaged using the Component Definition Tool and is installed in a manner similar to the Ulysses platform. To install the SP set, insert the product diskette into drive Aand run SETUP.EXE. Setup can also be invoked via the 'Add/Remove Programs' option of the Control Panel.

Once the SP set is installed, a new option Wosa_Xfs is added to NT's Start Menu with options WOSA_XFS Install and WOSA_XFS DeInstall to update/de-install the SP set respectively.

Installing the application on an SST

There is nothing special about the way one installs an application on an SST. Installation consists of simply coping the necessary files to the SST disk and setting up the environment, or using standard packaging tools to do the same.

Troubleshooting WOSA/XFS

The following points should be borne in mind while running a WOSA/XFS application:

1. The path containing the Service Provider DLLs and executables must be included in the PATH variable. If it is not, the SPs will fail to initialize.

2. If an improper shutdown of a Service Provider occurs (the Service Provider was not closed using WFSClose), the next attempt to startup the application will fail. Although terminating the SP using 'PVIEW' is a quick remedy, it may have undesirable side effects. Re-booting the SST is recommended in these circumstances.

3 Service Providers do not co-operate with the Ulysses System Application (UlSysApp.exe) in maintaining counts. The implication of this is that, if notes for example, are dispensed using UlSysApp, then the actual count (*ulCount*) of notes reported by the WFS_INF_CDM_CASH_UNIT_INFO command are not guaranteed to be accurate.

Debugging a WOSA/XFS Application

Extensive validation and tracing has been built into the WOSA/XFS Service Providers to aid the debugging of WOSA/XFS applications. All the input parameters are validated, and errors detected are logged in a per-service provider trace file, provided that tracing has been enabled. The trace files also contain a log of the EXECUTE/GETINFO command output structure (if applicable), which is sent along with the EXECUTE/GETINFO command completion message.

Tracing can be enabled in the following two ways:

- By specifying an appropriate dwTraceLevel in WFSOpen/AsyncOpen
- By using the WFMSetTraceLevel function

(For more details refer to WFSOpen/AsyncOpen & WFMSetTraceLevel API definitions in Reference 1).

The trace files, whose name and location are stored in the registry under SERVICE_PROVIDERS\XXX\GENERAL_CONFIGS\TraceFileName, where XXX is the service class (CDM, PIN, PTR etc.), are circular files with a maximum size of 500 K Bytes. The point in the file at which the last write has occurred is followed by the line,

"**** WRAP STARTS FROM HERE ****"

Since tracing is a performance overhead, it is recommended that it be turned on only while developing an application.

In addition to the trace files described above, .LOG files with names like XXX_SPX, XXX_WFP, XXX_SPP & XXX_DBG may be created in the log file directory specified in the registry. These files which are created by various components of the SP, contain low level debugging information for use by NCR support personnel. These files should be purged periodically.

Chapter 3

Table of Contents

Device Class Interface

This chapter describes the NCR Service Providers from various perspectives. The information presented here is not intended to be used in isolation, but together with the WOSA/XFS Device Class Interface Specifications, published by the BSVC.

Each device class description has the following sections:

Service Provider

Configurable Parameters

Capabilities

Conformance Matrices

These matrices tabulate the interpretation of the WOSA specifications for the Commands, Errors and Events. A Conformance Level assigned to each command/error/event, that indicates the extent to which the command/error/event conforms to the specifications. Conformance levels can take values 2 (fully compliant), through 1 (compliant with some deviations) to 0 (not supported).

Note: 'Conformance Level' is abbreviated as 'CL' throughout these tables.

Deviations

Application Guidelines

This section provides tips/guidelines to application developers.

Currency Dispenser Module

Service Provider Components

Device	DLL Name	SP Executable
Currency Dispenser	cdm_spx.dll cdm_wfp.dll cdm_ipc.dll cdm.dll	cdm.exe

Default Logical Service Names

Logical Name	Description
CurrencyDispenser1	The logical name of CDM service provider

Configurable Parameters

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS\CDM key:

Parameter	Description	Permissible Values
GENERAL_CONFIGS\MaxBills	Maximum number of notes that the dispenser can dispense. Configurable only on 56XX and 58XX series machines.	Permissible values range from 40 bills to a maximum of 50 bills. Defaults to 40 if not/incorrectly specified. Specifying a value greater than 40 must only be used where the currency has been pre-qualified for that number on the Currency Dispenser device. Otherwise this may lead to seriously degraded performance from the currency dispenser.
GENERAL_CONFIGS\SuspendTimeout	Duration in minutes, for which the device should be suspended when user tampering is suspected.	1-15 minutes with a default of 5 minutes.
GENERAL_CONFIGS\CurrencyIDType1-4	Used internally by the SP to store the Currency denomination of the notes in cassettes 1-4. May be pre-set for applications that do not issue a WFS_CMD_CDM_SET_CASH_UNIT_INFO or a WFS_CMD_CDM_END_EXCHANGE command prior to performing a WFS_CMD_CDM_DENOMINATE or a WFS_CMD_CDM_DISPENSE.	Currency IDs as specified by *ISO*. Note that no error checking is performed on this string.
GENERAL_CONFIGS\ValuesType1-4	Used internally by the SP to store the Currency denomination of the notes in cassettes 1-4. ID of the notes in cassettes 1-4. May be pre-set for applications that do not issue a WFS_CMD_CDM_SET_CASH_UNIT_INFO or a WFS_CMD_CDM_END_EXCHANGE command prior to performing a WFS_CMD_CDM_DENOMINATE or a WFS_CMD_CDM_DISPENSE.	Value of bills in cassette types 1-4.

Capabilities

Capability	Value
Logical Service Class	WFS_SERVICE_CLASS_CDM
Type Of Physical Device	WFS_CDM_TYPEATMACHINE
MaxBills	40-50
MaxCoins	0
Compound Device	FALSE
Shutter	FALSE
Retract (Mechanism)	TRUE
Safe Door	FALSE
Coins	FALSE
Cylinders	FALSE
CashBox	FALSE
CashIn	FALSE
Refill	FALSE
AutoDeposit	FALSE
VandalCheck	TRUE
IntermediateStacker	TRUE
BillsTakenSensor	TRUE
OutputPositions	WFS_CDM_POSCENTER

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_CDM_STATUS	2	• *fwDevice* of WFS_CDM_DEVPOWEROFF, WFS_CDM_DEVBUSY and WFS_CDM_DEVNODEVICE are never returned. • *fwSafeDoor* is always WFS_CDM_DOORNOTSUPPORTED. • *fwDispenser* reflects the state of the dispenser cash unit as of the last execute command that had device interaction. • *lppOutputPositions->fwPosition* is always WFS_CDM_POSCENTER. • *lppOutputPositions->fwShutter* will be WFS_CDM_SHTUNKNOWN if the device was off-line when the command was issued. • *lppOutputPositions->fwOutputPosition* is always WFS_CDM_CTNOTSUPPORTED. • *lppOutputPositions->fwTransport* will be WFS_CDM_TPUNKNOWN if the device was off-line when the command was issued.
WFS_INF_CDM_CAPABILITIES	2	None
WFS_INF_CDM_CASH_UNIT_INFO	2	• *usNumber*, the logical number of the cash unit, begins at 0 for the structure pointed to by the first element of *lppList*. This value is also, implicitly, the type of the cash unit structure. Therefore, if there are no cash units of type 1, there will still be a cash unit structure returned for type 1 (although it will specify that this cash unit type is missing). The Purge bin is the first element in the array as it does not have a logical Type like the rest of the cassettes. • *cUnitID* is not used. • *lppList->usStatus* of WFS_CDM_STATCUNOVAL and WFS_CDM_STATCUNOREF are never returned. • *lppList->lpPhysicalPositionName* indicates the name and position of the physical cash unit, when there is only one physical cassette associated with the logical cassette. • *lppList->lppPhysical->ulCount* is always zero (0).
WFS_INF_CDM_TELLER_INFO	0	None
WFS_INF_CDM_TELLER_POSITIONS	0	None
WFS_INF_CDM_CURRENCY_EXP	2	This command returns the currency exponent structures of the currencies specified the currency exponents file. (Refer Application Guidelines.)

WOSA Command	CL	Comments
WFS_INF_CDM_MIX_TYPES	2	In the case of MixTables (*usMixType* = WFS_CDM_MIXTABLE), *usSubType* is always 0. This is because command WFS_CMD_CDM_SET_MIX_TABLE, has no provision to specify whether the MixTable is of the WFS_CDM_MIX_MINIMUM_NUMBER_OF_BILLS type or the WFS_CDM_MIX_EQUAL_EMPTYING_OF_CASH_UNITS type.
WFS_INF_CDM_MIX_TABLE	2	*usCols* is always 4, as there are 4 types of cash units supported by the 56XX dispenser.
WFS_INF_CDM_PRESENT_STATUS	2	None
WFS_CMD_CDM_DENOMINATE	1	• *usTellerID* is not used. • *lpDenomination->usCount* must always be 5. • *lpDenomination->ulValues[]*, the value of notes to be denominated, should be specified in the order returned by the WFS_INF_CDM_CASH_UNIT_INFO command. Hence, the first element of the array should always be 0 for the Reject Bin, the second should indicate the number of notes from cassette type 1 and so on. • *lpDenomination->ulCashBox* must always be zero (0).
WFS_CMD_CDM_DISPENSE	1	• *usTellerID* is not used. • *usPosition* is ignored as there is only one dispense position. • *lpDenomination->usCount* must always be 5. • *lpDenomination->ulValues[]*, the value of notes to be dispensed, should be specified in the order returned by the WFS_INF_CDM_CASH_UNIT_INFO command. Hence, the first element of the array should always be 0 for the Reject Bin, the second should indicate the number of notes from cassette type 1 and so on. • *lpDenomination->ulCashBox* must always be zero (0).
WFS_CMD_CDM_PRESENT	2	None
WFS_CMD_CDM_REJECT	2	None
WFS_CMD_CDM_RETRACT	2	*lpusRetractArea* is ignored as there is only one reject bin.
WFS_CMD_CDM_CASH_IN	0	None
WFS_CMD_CDM_OPEN_SHUTTER	0	None

WOSA Command	CL	Comments
WFS_CMD_CDM_CLOSE_SHUTTER	0	None
WFS_CMD_CDM_SET_TELLER_INFO	0	None
WFS_CMD_CDM_SET_CASH_UNIT_INFO	2	• *usTellerID* is ignored • *usCount* should always be 5. • *lppList->usNumber*, the logical number of the cash unit, begins at 0 for the structure pointed to by the first element of *lppList*. This value is also, implicitly, the type of the cash unit structure. Therefore, if there are no cash units of type 1, there will still be a cash unit structure for type 1 (although it will specify that this cash unit type is missing). The Reject Bin is considered as Type 0, hence it corresponds to the first structure pointed to by *lppList*. • *lppList->cUnitID* is not used. • Structure *lppList->lppPhysical* is ignored.
WFS_CMD_CDM_START_EXCHANGE	2	• Input structure WFSCDMSTARTEX is ignored. • *usTellerID* should be ignored. • *usCount* is always 5. • *lppList->lpPhysicalPositionName* reflects the name of the Physical Cassette if the logical cassette refers to one physical cassette. • *lppList->lppPhysical-> ulCount* will always be zero (0).
WFS_CMD_CDM_END_EXCHANGE	2	• *usTellerID* is ignored. • *usCount* must always be 5. • *lppList->usNumber*, the logical number of the cash unit, begins at 0 for the structure pointed to by the first element of *lppList*. This value is also, implicitly, the type of the cash unit structure. Therefore, if there are no cash units of type 1, there will still be a cash unit structure for type 1 (although it will specify that this cash unit type is missing). The Reject Bin is considered as Type 0. • *lppList->cUnitID* is not used. • Structure *lppList->lppPhysical* is ignored.
WFS_CMD_CDM_OPEN_SAFE_DOOR	0	None
WFS_CMD_CDM_CHECK_VANDALISM	2	None

WOSA Command	CL	Comments
WFS_CMD_CDM_CALIBRATE_CASH_UNIT	2	• *usNumber* assumes values from 1 to 4 corresponding to the Type of the cassette that needs to be configured. • *usNumOfBills*, the maximum number of bills that can be used to configure the cassette is 40.
WFS_CMD_CDM_SET_TELLER_POSITIONS	0	None
WFS_CMD_CDM_CASH_IN_START	0	None
WFS_CMD_CDM_CASH_IN_END	0	None
WFS_CMD_CDM_CASH_IN_ROLLBACK	0	None
WFS_CMD_CDM_SET_MIX_TABLE	1	• If the SST file system is FAT, the mixtable filename *lpszName*, should not exceed eight characters. • *usRows* cannot exceed 500. • *usCols* must always be 4 as there a 4 Types of cash units.

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_INF_CDM_STATUS	None	2	None
WFS_INF_CDM_CAPABILITIES	None	2	None
WFS_INF_CDM_CASH_UNIT_INFO	None	2	None
WFS_INF_CDM_CURRENCY_EXP	None	2	None
WFS_INF_CDM_MIX_TYPES	None	2	None
WFS_INF_CDM_MIX_TABLE	WFS_ERR_CDM_INVALIDMIXNUMBER	2	None
WFS_INF_CDM_PRESENT_STATUS	None	2	None
WFS_CMD_CDM_DENOMINATE	WFS_ERR_CDM_CASHUNITERROR	2	None
	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
	WFS_ERR_CDM_INVALIDCURRENCY	2	None
	WFS_ERR_CDM_INVALIDDENOMINATION	2	None
	WFS_ERR_CDM_INVALIDMIXNUMBER	2	None
	WFS_ERR_CDM_TELLERID	0	Not supported
	WFS_ERR_CDM_NOCURRENCYMIX	2	None
	WFS_ERR_CDM_NOTDISPENSABLE	2	None
	WFS_ERR_CDM_TOOMANYBILLS	2	None
	WFS_ERR_CDM_TOOMANYCOINS	0	Not supported
WFS_CMD_CDM_DISPENSE	WFS_ERR_CDM_CASHUNITERROR	2	None
	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
	WFS_ERR_CDM_INVALIDCURRENCY	2	None
	WFS_ERR_CDM_INVALIDDENOMINATION	2	None
	WFS_ERR_CDM_INVALIDMIXNUMBER	2	None

continued...

WOSA Command	Error Codes	CL	Comments
WFS_CMD_CDM_DISPENSE	WFS_ERR_CDM_INVALIDPOSITION	0	Not supported
	WFS_ERR_CDM_INVALIDTELLERID	0	Not supported
	WFS_ERR_CDM_NOCURRENCYMIX	2	None
	WFS_ERR_CDM_NOTDISPENSABLE	2	None
	WFS_ERR_CDM_POSITIONLOCKED	2	None
	WFS_ERR_CDM_SAFEDOOROPEN	0	Not supported
	WFS_ERR_CDM_TOOMANYBILLS	2	None
	WFS_ERR_CDM_TOOMANYCOINS	0	Not supported
WFS_CMD_CDM_PRESENT	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
	WFS_ERR_CDM_NOBILLS	2	None
	WFS_ERR_CDM_SHUTTERNOTOPEN	2	None
	WFS_ERR_CDM_SHUTTEROPEN	2	None
	WFS_ERR_CDM_PRERRORNOBILLS	2	None
	WFS_ERR_CDM_PRERRORBILLS	0	Not supported
	WFS_ERR_CDM_PRERRORUNKNOWN	2	None
WFS_CMD_CDM_REJECT	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
	WFS_ERR_CDM_NOBILLS	0	Not supported
WFS_CMD_CDM_RETRACT	WFS_ERR_CDM_BILLSTAKEN	2	None
	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
	WFS_ERR_CDM_INVALIDRETRACT	0	None
	WFS_ERR_CDM_SHUTTERNOTCLOSED	2	None
	WFS_ERR_CDM_NOBILLS	2	None

WOSA Command	Error Codes	CL	Comments
WFS_CMD_CDM_SET_CASH_UNIT_INFO	WFS_ERR_CDM_CASHUNITERROR	0	None
	WFS_ERR_CDM_INVALIDCASHUNIT	2	None
	WFS_ERR_CDM_INVALIDTELLERID	0	None
	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
WFS_CMD_CDM_START_EXCHANGE	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
	WFS_ERR_CDM_INVALIDTELLERID	0	None
WFS_CMD_CDM_END_EXCHANGE	WFS_ERR_CDM_NOEXCHANGEACTIVE	2	None
	WFS_ERR_CDM_INVALIDCASHUNIT	2	None
	WFS_ERR_CDM_INVALIDTELLERID	0	None
WFSCMD_CDM_CHECK_VANDALISM	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
WFS_CMD_CDM_CALIBRATE_CASH_UNIT	WFS_ERR_CDM_EXCHANGEACTIVE	2	None
	WFS_ERR_CDM_CASHUNITERROR	2	None
WFS_CMD_CDM_SET_MIX_TABLE	WFS_ERR_CDM_INVALIDMIXNUMBER	2	None
	WFS_ERR_CDM_INVALIDMIXTABLE	2	None

Conformance Matrix - Events

WOSA Event	CL	Comments
WFS_SRVE_CDM_SAFEDOOROPEN	0	None
WFS_SRVE_CDM_SAFEDOOORCLOSED	0	None
WFS_USRE_CDM_CASHUNITTHRESHOLD	2	This event is generated when the number of notes in a currency cassette falls below *ulMinimum* for that cassette. It is also generated when the hardware detects a cassette low on bills. In the case of the Reject Cassette, the event is generated when the number of notes in the Reject Bin exceeds *ulMaximum* of the Reject Cassette.
WFS_SRVE_CDM_CASHUNITINFOCHANGED	2	None
WFS_SRVE_CDM_TELLERINFOCHANGED	0	None
WFS_EXEE_CDM_DELAYEDDISPENSE	0	None
WFS_EXEE_CDM_STARTDISPENSE	0	None
WFS_EXEE_CDM_CASHUNITERROR	2	None
WFS_SRVE_CDM_BILLSTAKEN	2	None
WFS_EXEE_CDM_PARTIALDISPENSE	0	None
WFS_EXEE_CDM_SUBDISPENSEOK	0	None
WFS_EXEE_CDM_INPUTREFUSE	0	None

Application Guidelines

1 Data concerning cassette types, currencies, values and note counts are persistent across sessions. They can be set at any time using the either of the following commands:
WFS_CMD_CDM_END_EXCHANGE
or
WFS_CMD_CDM_SET_CASH_UNIT_INFO
Since the default values for these quantities are inappropriate for a working system, an application should be capable of performing either of the following:
WFS_CMD_CDM_END_EXCHANGE
or
WFS_CMD_CDM_SET_CASH_UNIT_INFO.
This has to be performed when you run the application for the first time, after the dispenser NVRAM has been cleared or the relevant registry entries have been deleted.

2 Developers should be careful of the following three hardware related issues that have a bearing on the accuracy of cash totals:

- The CDM will regard a cassette as empty if it has a low level of cash and if it fails to pick a note successfully.
- If a cassette containing *n* notes has exactly *n* notes picked up i.e. it is emptied of notes, the CDM will not detect that the cassette as empty until the next attempt to pick the notes is made.
- The CDM cannot determine the number of notes present in a cassette. This implies that there is no way for the SP to be 100% accurate about how many notes are in the CDM. However, the counts will be accurate.

  The implications regarding the behaviour of the SPI are significant. If the SP-maintained totals differ from the apparent state of the cassette, then it has to be resolved. If the firmware detects a cassette as empty, the cassette's *usStatus* field (in the WFSCDMCASHUNIT structure) will be set to WFS_CDM_STATCUEMPTY. If there is a discrepancy between the two, the *ulCount* data could be non-zero. The hardware cannot guarantee that either of the totals is accurate.

3 The WFS_INF_CDM_CURRENCY_EXP command returns an array of pointers to the structures giving the currency exponent for the currency in each of these structures. This information is not maintained within the SP, but is stored in an external Currency Exponents file called "currency.exp" by default. Alternatively, a name could be created in the registry, such as WOSA/XFS\SERVICE_PROVIDERS\CDM\EXPONENTS\Filename,which contains the path\name of the currency exponents file. This file should contain only one currency and exponent on a line. A sample exponents file is shown below.

```
; Lines beginning with semi-colons, and blank lines are ignored. Each line
; should be of the form:
;               XXX:n
; where XXX is the three-letter ISO currency ID and n is the (signed) exponent.
; If less than three letters are used, spaces should not be inserted.
;
; Only one currency and exponent is allowed per line. However, the number
; of exponents is limited only by system resources.

; The SP performs no consistency checking on this file. So, if multiple
; definitions are included, both will be read and both will be presented by
; the WFS_INF_CDM_CURRENCY_EXP function. The SP will, however, perform a syntax
; check (if the file is found) and will fail to initialize if an error is found.
GBP:0
DEM:-2
LIT:2
```

4 The WFS_INF_CDM_MIX_TABLE command retrieves the house mix table specified by the input parameter *lpusMixNumber* as a pointer to a WFSCDMMIXTABLE structure. The SP maintains house mix tables as flat files, whose names are stored in the Windows NT registry as values under the "WOSA/XFS\SERVICE_PROVIDERS\CDM\MixTable" key. A sample is given below:

Level 1	WOSA/XFS\
Level 2	SERVICE_PROVIDERS\
Level 3	CDM\
Level 4	MixTable\ Mixtable1 = <File Name 1> . . Mixtable *i* = <File Name *i* >

The application refers to a particular mix table by setting *lpszName* (in structures WFSCDMMIXTABLE/ WFSCDMMIXTYPE) to the name of the mix table file.

During initialization, the SP reads and validates the mix tables, and if no errors are encountered, adds them to an internal database. Mix tables that contain either format or content errors, are discarded and will not be available to the application.

If the "MixTable" key does not exist, or if there are no values under it, then no mix tables are available to the application immediately after startup. The application should use the WFS_CMD_CDM_SET_MIX_TABLE command to define them. When this is done, the SP creates a new mix table file and adds this file name to the Windows Registry. The application may also use the

WFS_CMD_CDM_SET_MIX_TABLE command to modify an existing mix table.

The following is a sample mix table file:

```
COMMENT: Lines beginning with the keyword comment are ignored.
COMMENT: MIXNUMBER should be unique for each mix table and greater
COMMENT: than 2. (0-2 are predefined algorithms).
COMMENT: A mix table must define all 4 MIXHEADER columns. If less
COMMENT: than 4 cash units are used, the values in the unused
COMMENT: column should be equated to zero.
COMMENT: A mix table can have a maximum of 500 rows.
NAME: MixTble3
MIXNUMBER:  3
ROWS:      10
COLS:       4
MIXHEADER:                500;    100;     50;      10;
AMOUNT:        500;         1;      0;      0;       0;
AMOUNT:       1000;         2;      0;      0;       0;
AMOUNT:       1500;         2;      5;      0;       0;
AMOUNT:       2000;         4;      0;      0;       0;
AMOUNT:       2000;         0;     20;      0;       0;
AMOUNT:       2500;         4;      5;      0;       0;
AMOUNT:       2500;         4;      0;     10;       0;
AMOUNT:       3000;         6;      0;      0;       0;
AMOUNT:       3000;         0;     30;      0;       0;
AMOUNT:       3500;         7;      0;      0;       0;
END:
```

5 A WFS_CMD_CDM_REJECT command must be performed after any FATAL device faults are cleared by operator intervention.

6 If the Currency Dispenser is configured, there is no need to calibrate the device. If the device is not configured, the application should issue a WFS_CMD_CDM_CALIBRATE_CASH_UNIT command on initial power-up. If this is not done, the Bill Widths and Singularities will be set to default (Bill width will be set to USA test dollar width and singularity to 0FFH). This ensures that ALL bills are rejected prior to the correct configuration.

7 Bills of the same type but with different widths, (for example old style and new issue bills, should be put in different cassette types. The different sizes can then be configured using the WFS_CMD_CDM_CALIBRATE_CASH_UNIT command.

8 The transaction time for a dispense can be reduced if an application performs a STACK (WFS_CMD_CDM_DISPENSE with *bPresent* = FALSE) followed by a PRESENT instead of performing a WFS_CMD_CDM_DISPENSE with *bPresent* = TRUE, when there are operations which can be done between the

STACK and PRESENT commands. If this is done, the STACK command will move the bills to a position near the exit after it has reported the response to the STACK command. This reduces the time to PRESENT the bills.

9 The STACK can be optimized by carefully selecting the pick module where each cassette will be inserted. The cassette type with most bills picked per transaction should be inserted in the top pick module. The cassette type with the least bills picked per transaction should be placed in the bottom pick module. If no cassette type is to be used more than any other, the cassettes should be so arranged that the one to be picked first is in the top pick module and the cassette to be picked from next is below it.

10 The CDM SP and device drivers attempt to recover from shutter jam conditions. If after repeated attempts, the error condition persists, the device goes into a fatal state. A condition that can be cleared only by operator intervention via the VDM. The following table lists the error conditions from which the SP attempts to recover, and the action to be taken for other commonly occurring errors.

Error Condition	Recovery Action
Exit Shutter Jammed Closed prior to a Present	If in a Present sequence the exit shutter is jammed shut, or forced open while the bills are moving towards the exit, the bills are purged and three shutter recovery attempts are made. If the shutter still fails to operate on the retries, the CDM SP enters a suspend state for 'SuspendTimeout' minutes, during which all CDM commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the CDM SP will attempt to clear the transport, and if successful, resumes normal operation. If the shutter continues to remain jammed, the SP will attempt to clear the error once again. If this attempt fails, the device goes into a fatal state, following which, all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Exit Shutter Jammed Open prior to a Retract/Reject	If the exit shutter is jammed open before a RETRACT or a REJECT command when notes were presented by a previous command, three attempts are made to close the shutter. If this is unsuccessful, any stacked bills will be purged, and three more shutter recovery attempts are made. If the shutter continues to remain jammed, the CDM SP enters a suspend state for 'SuspendTimeout' minutes, during which all CDM commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the CDM SP will attempt to clear the transport and if successful, resumes normal operation. If the shutter continues to remain jammed, the SP will attempt to clear the error once again. If this attempt fails, the device goes into a fatal state, following which, all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Purge Bin Overfill sensor blocked	The Transport is run for an additional 3 seconds in an attempt to clear the bills into the bin. If this fails, no further automatic recovery is attempted by the SP. Recovery now requires operator intervention via VDM.

Error Condition	Recovery Action
Reject Bin missing	No automatic recovery by SP. Requires operator intervention via VDM.
Pick failure due to bad media	A cassette low and pick failure indicates that the cassette is empty. Otherwise, the fault is a failure to pick. If more than one cassette of the same Type is installed, an attempt will be made to dispense from the other cassettes when one goes empty. If a pick attempt fails, all picked bills will be purged and the operation will be repeated up to three times. If these attempts fail, the SP reports a Cash Unit error (WFS_ERR_CDM_CASHUNITERROR). No further recovery attempts will be made. If a cassette is not low on bills, and there are repeated pick fails due to bad notes, the status of the cassette will be marked as WFS_CDM_STATCUINOP on the third consecutive failed WFS_CMD_CDM_DISPENSE command which failed due to a pick fail from a particular cassette. In this case, recovery of the cassette type marked as WFS_CDM_STATCUINOP, requires operator intervention via VDM.
Cassette Inoperable due to an attempt to pick from an empty cassette.	The SP performs no automatic recovery. After the cassette is replenished and the Start Exchange and End Exchange commands are performed, the state of the cassette will be set to WFS_CDM_STATCUOK or WFS_CDM_STATCULOW, as the case may be.

Depository

Service Provider Components

Device	DLL Name	SP Executable
Envelope Depository/Dispenser	edep_spx.dll edep_wfp.dll edep_ipc.dll edep.dll	edep.exe
Nightsafe Depository	ndep_spx.dll ndep_wfp.dll ndep_ipc.dll ndep.dll	ndep.exe

Default Logical Service Names

Logical Name.	Description
Envelope1	The logical name of the Envelope Depository service provider
NightSafe1	The logical name of the Nightsafe Depository service provider

Configurable Parameters

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS\EDEP (NDEP) key.

Parameter	Description	Permissible Values
GENERAL_CONFIGS\Variant	Variant of the device.	STANDARD_DEP - Standard Depository BASIC_NSD - Basic nightsafe ENHANCED_NSD - Enhanced nightsafe
GENERAL_CONFIGS\DispenserType	Type of dispenser used.	NONE - dispenser is not present MOTORIZED - dispenser is motor-driven and envelopes are dispensed to the user
GENERAL_CONFIGS\SuspendTimeout	Duration in seconds for which the device should be suspended when user tampering is suspected.	Permissible values range from 1 - 15 minutes with a default of 5 minutes. Applicable only to the Envelope Depository SP.
GENERAL_CONFIGS\Timeout	Duration in milliseconds for which the Nightsafe Depository SP waits for a bag-drop switch transition before locking the depository door, following a deposit command. This timeout assumes significance if the bag-drop switch is faulty.	If not specified, default to 60 seconds.

Capabilities

Capability	Value (Envelope Depository)	Value (Nightsafe Depository)
Depository Type	ENVELOPE	NIGHTSAFE
Envelope Supply	ENVMOTORIZED	ENVNONE
Dep Transport	TRUE	FALSE
Printer	TRUE	FALSE
Toner	TRUE	FALSE
Shutter	TRUE	TRUE
PrintOnRetracts	FALSE	FALSE
RetractToDeposit	FALSE	FALSE
MaxNumChars	80	0

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_DEP_STATUS	1	*fwToner* reports WFS_DEP_TONLOW even if the ink cartridge is empty. (WFS_DEP_TONEMPTY is not reported).
WFS_INF_DEP_CAPABILITIES	2	None
WFS_CMD_DEP_ENTRY	1	If a deposit could not be completed due to an incorrect envelope size, an attempt is made to clear the transport and deposit the envelope in the depository bin. The envelope is not ejected.
WFS_CMD_DEP_DISPENSE	2	None
WFS_CMD_DEP_RETRACT	1	Printing on a retracted envelope is not supported.
WFS_CMD_DEP_CLEAR_TRANSPORT	2	None
WFS_CMD_DEP_RESET_COUNT	2	None

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_CMD_DEP_ENTRY	WFS_ERR_DEP_ENVJAMMED	2	None
	WFS_ERR_DEP_DEPFULL	2	None
	WFS_ERR_DEP_CONTMISSING	2	None
	WFS_ERR_DEP_ENVSIZE	2	None
	WFS_ERR_DEP_PTRFAIL	2	Returned when the printer fails because the printhead is removed. Failure of the printer for other reasons will not be reported.
	WFS_ERR_DEP_SHTNOTCLOSED	2	None
	WFS_ERR_DEP_SHTNOTOPENED	2	None
	WFS_ERR_DEP_DEPUNKNOWN	2	None
WFS_CMD_DEP_DISPENSE	WFS_ERR_DEP_ENVEMPTY	2	None
	WFS_ERR_DEP_ENVJAMMED	2	None
	WFS_ERR_DEP_SHTNOTOPENED	2	None

Conformance Matrix - Events

WOSA Event	CL	Comments
WFS_SRVE_DEP_TAKEN	2	None
WFS_EXEE_DEP_ENVDEPOSITED	2	None
WFS_EXEE_DEP_DEPOSITERROR	2	None
WFS_USRE_DEP_DEPTHRESHOLD	2	None
WFS_USRE_DEP-TONERTHRESHOLD	1	Generated when the printer toner is low (*fwToner*=WFS_DEP_TONLOW). Not generated when the printer runs out of toner
WFS_USRE_DEP-ENVTHRESHOLD	2	None
WFS_USRE_DEP-CONTINSERTED	2	None
WFS_USRE_DEP-CONTREMOVED	2	None

Application Guidelines

1 In the Nightsafe depository, if the bag-drop switch is stuck in the ON position, it is not possible to determine whether the deposit has been made or not. This may cause the deposit count to be inaccurate. In this case, the WFS_CMD_DEP_ENTRY command will return a WFS_ERR_DEP_DEPUNKNOWN error, irrespective of whether the deposit was completed successfully or not.

2 In the Envelope depository, the data to be printed on the envelope by the customer should be in uppercase. The lowercase alphabets are not supported in the NCR ASCII character set.

3 The Envelope Depository SP and device drivers attempt to recover from Envelope Jam conditions. If after repeated attempts, the error condition persists, the device goes into a fatal state, a condition that can be cleared only by operator intervention via the VDM. The following table lists the error conditions from which the SP attempts to recover, and the action to be taken for other commonly occurring errors.

4 In the event an envelope is accepted in response to a WFS_CMD_DEP_ENTRY command, and this envelope causes the deposit bin to overfill, the SP generates a WFS_EXEE_DEP_ENVDEPOSITED event and returns WFS_ERR_DEP_DEPFULL.

Error Condition	Recovery Action
Envelope jammed in transport with user access.	The SP suspends operation for 'SuspendTimeout' seconds, during which all EDEP commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the depository is considered to be healthy until the next command is issued. If the jam condition persists, the device goes into a fatal state, following which, all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Shutter Jammed Shut/Open	The SP makes 15 attempts to open/close the shutter. If these attempts fail, the SP suspends operation for 'SuspendTimeout' seconds, during which all EDEP commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the depository is considered to be healthy until the next command is issued. If the jam condition persists, the device goes into a fatal state, following which, all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Transport Jam without access, caused by an internal envelope jam.	No automatic recovery. Requires operator intervention via VDM.
Deposit Bin overfilled	No automatic recovery. Requires operator intervention via VDM.

Receipt and Journal Printers

Service Provider Components

Device	DLL Name	SP Executable
Receipt Printer (all variants)	rptr_spx.dll rptr_wfp.dll rptr_ipc.dll rptr.dll	rptr.exe
Journal Printer (all variants)	jptr_spx.dll jptr_wfp.dll jptr_ipc.dll jptr.dll	jptr.exe

Default Logical Service Names

Logical Name	Description
ReceiptPrinter1	The logical name of the receipt printer service provider.
JournalPrinter1	The logical name of the journal printer service provider.

Configurable Parameters

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS key

Parameter	Description	Permissible Values
FORMS\FORM_FILE_PATH	The path to the form file definitions directory.	Any valid absolute path
MEDIA\MEDIA_FILE_PATH	The path to the media file definitions directory	Any valid absolute path
ERROR_FILE_PATH\PTR_ERROR_FILE_PATH	Path of the directory in which initialization errors are logged.	Any valid absolute path.

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS\RPTR (JPTR) key:

Parameter	Description	Permissible Values
GENERAL_CONFIGS\Variant	Variant of the receipt/journal printer.	The following variants are supported by the receipt printer service provider: l NO_BLACK_MARK (no black mark receipt printer) l NO_BLACK_MARK_THERMAL (no black mark, thermal receipt printer) l BLACK_MARK (black mark receipt printer) l BLACK_MARK_THERMAL (black mark, thermal receipt printer) l NO_BLACK_MARK_SIDEWAYS_NORMAL (no black mark, combined sideways/normal mode receipt printer) l NO_BLACK_MARK_THERMAL_SIDEWAYS_NORMAL (no black mark, thermal, combined sideways/normal mode receipt printer) l BLACK_MARK_SIDEWAYS_NORMAL (black mark, combined sideways/normal mode receipt printer) l BLACK_MARK_THERMAL_SIDEWAYS_NORMAL (black mark, thermal, combined sideways/normal mode receipt printer). The following variants are supported by the journal printer service provider: l STD (standard journal printer) l STD_THERMAL (standard thermal journal printer).

Capabilities

Capability	Value (Receipt Printer)	Value (Journal Printer)
Compound	FALSE	FALSE
Resolution	LOW	LOW
Read Form Support	FALSE	FALSE
Write Form Support	TRUE (text only)	TRUE (text only)
Extents Support	FALSE	FALSE
Supported Controls	EJECT, FLUSH	FLUSH
Retract Bin capacity	0	0
Stacker capacity	0	0

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_PTR_STATUS	2	*fwDevice* of the WFS_PTR_DEVPOWEROFF, WFS_PTR_DEVBUSY and WFS_PTR_DEVNODEVICE are never returned.
WFS_INF_PTR_CAPABILITIES	2	None
WFS_INF_PTR_FORM_LIST	2	None
WFS_INF_PTR_MEDIA_LIST	2	None
WFS_INF_PTR_QUERY_FORM	2	None
WFS_INF_PTR_QUERY_MEDIA	2	None
WFS_INF_PTR_QUERY_FIELD	2	None
WFS_CMD_PTR_CONTROL_MEDIA	1	• Does not generate the event WFS_SRVE_PTR_MEDIA_TAKEN for the receipt printer Service Provider. This event is not applicable to the journal printer. • The only control codes supported are WFS_PTR_CTRLEJECT and WFS_PTR_CTRLFLUSH. The code WFS_PTR_CTRLCUT is implemented like WFS_PTR_CTRLEJECT.

WFS_CMD_PTR_PRINT_FORM	1	• Supports only low resolution.
		• Does not return the error code WFS_ERR_PTR_MEDIASKEWED.
		• Only supports control codes WFS_PTR_CTRLEJECT and WFS_PTR_CTRLFLUSH. WFS_PTR_CTRLCUT is implemented like WFS_PTR_CTRLEJECT.
		• The form/field definitions should satisfy the following conditions:
		  ◦ Orientation cannot be LANDSCAPE for the journal printer.
		  ◦ SKEW should not be greater than zero.
		  ◦ SIDE can only be FRONT.
		  ◦ TYPE can only be TEXT.
		  ◦ GRAPHICS can only be BESTFIT (default).
		  ◦ BARCODE can only be NONE (default).
		  ◦ STYLE can be NORMAL (default) or DOUBLE.
		  ◦ HORIZONTAL justification can only be LEFT (default), RIGHT or CENTER.

continued...

WOSA Command	CL	Comments
WFS_CMD_PTR_PRINT_FORM	1	• COLOR can only be BLACK. • FONT should be one of the following: INTERNATIONAL1 INTERNATIONAL2 INTERNATIONAL3 INTERNATIONAL4 INTERNATIONAL5 ARABIC1 ARABIC2 ARABIC3 ARABIC4 ARABIC5 • POINTSIZE is not supported (ignored). • CPI can be 7.0 or 14.0 for the receipt and journal printers. However, if CPI is specified as 14.0, then CASE should not be DOUBLE. Note that a CPI of 14.0 or 7.0 is invalid for LANDSCAPE orientation for the receipt printer. Here, the default is 8.5 and if specified, the CPI should be 8.5. • LPI can only take values between 1.0 and 9.0. • FORMAT is not supported (ignored). • Media definitions should satisfy the following conditions: • TYPE can only be GENERIC (default). • PRINTAREA, RESTRICTED, FOLD, STAGGERING, PAGE and LINES are not supported and should not be specified. • SIZE *width* should be <= 40.
WFS_CMD_PTR_READ_FORM	0	Not applicable
WFS_CMD_PTR_RAW_DATA	2	None
WFS_CMD_PTR_MEDIA_EXTENTS	0	Not applicable
WFS_CMD_PTR_RESET_COUNT	0	Not applicable
WFS_CMD_PTR_READ_IMAGE	0	Not applicable

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_INF_PTR_ FORM_LIST	None	2	None
WFS_INF_PTR_ MEDIA_LIST	None	2	None
WFS_INF_PTR_ QUERY_FORM	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_ FORMINVALID	2	Returned when the form definition specified does not satisfy the conditions given in the command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_ FORMNOTFOUND	2	Returned when the form name specified was not found in the form definition files at startup.
WFS_INF_PTR_ QUERY_MEDIA	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_ MEDIANOTFOUND	2	Returned when the media name specified was not found in the media definition files at startup.
	WFS_ERR_PTR_ MEDIAINVALID	2	Returned when the media definition specified does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
WFS_INF_PTR_ QUERY_FIELD	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_ FORMINVALID	2	Returned when the form definition specified in the input structure does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_ FORMNOTFOUND	2	Returned when the form name specified in the input structure was not found in the form files at startup.

continued....

WOSA Command	Error Codes	CL	Comments
WFS_INF_PTR_ QUERY_FIELD	WFS_ERR_PTR_ FIELDNOTFOUND	2	Returned when the field name specified in the input structure does not belong to the specified form.
	WFS_ERR_PTR_ FIELDINVALID	2	Returned when the field definition specified does not satisfy the conditions given in command conformance matrix for CMD_PTR_PRINT_FORM command.
WFS_CMD_PTR_ CONTROL_MEDIA	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_ NOMEDIAPRESENT	2	Returned when the printer is out of paper.
	WFS_ERR_PTR_ FLUSHFAIL	2	Never generated since the data is always flushed to the device.
	WFS_ERR_PTR_ RETRACTBINFULL	0	Not applicable
	WFS_ERR_PTR_ STACKERFULL	0	Not applicable
	WFS_ERR_PTR_ PAGETURNFAIL	0	Not applicable
	WFS_ERR_PTR_ MEDIATURNFAIL	0	Not applicable
WFS_CMD_PTR_ PRINT_FORM	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_ FLUSHFAIL	2	Additional error code. Returned when the print form command attempts to print data above or before the current position of the print head.
	WFS_ERR_PTR_ FORMINVALID	2	Returned when the form definition specified in the input structure does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_ FORMNOTFOUND	2	Returned when the form name specified in the input structure was not found in the form definition files at startup.

continued...

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PTR_ PRINT_FORM	WFS_ERR_PTR_ MEDIANOTFOUND	2	Returned when the media name specified in the input structure was not found in the media definition files at startup.
	WFS_ERR_PTR_ MEDIAINVALID	2	Returned when the media definition specified in the input structure does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_ MEDIASKEWED	0	Never returned. as Skew cannot be detected
	WFS_ERR_PTR_ MEDIAOVERFLOW	2	Returned when the print data overflows the media boundaries.
	WFS_ERR_PTR_ FIELDSPECFAILURE	2	Returned when a field is specified twice in the input.
	WFS_ERR_PTR_ FIELDERROR	2	Returned if an error occurs while processing a field.
WFS_CMD_PTR_ READ_FORM	WFS_ERR_PTR_READNOTSUPPORTED	2	Read is not supported by the receipt/journal printer.
	WFS_ERR_PTR_ FORMINVALID	0	Never returned
	WFS_ERR_PTR_ FORMNOTFOUND	0	Never returned
	WFS_ERR_PTR_ MEDIANOTFOUND	0	Never returned
	WFS_ERR_PTR_ MEDIAINVALID	0	Never returned
	WFS_ERR_PTR_ MEDIASKEWED	0	Never returned
	WFS_ERR_PTR_ FIELDSPECFAILURE	0	Never returned
	WFS_ERR_PTR_ FIELDERROR	0	Never returned
WFS_CMD_PTR_ RAW_DATA	None	2	None
WFS_CMD_PTR_ MEDIA_EXTENTS	WFS_ERR_PTR_EXTENTNOTSUPPORTED	2	Device does not report extents.
WFS_CMD_PTR_ RESET_COUNT	None	2	None
WFS_CMD_PTR_ READ_IMAGE	None	2	None

Conformance Matrix - Events

WOSA Event	CL	Comments
WFS_EXEE_PTR_NOMEDIA	2	None
WFS_EXEE_PTR_MEDIAINSERTED	0	Not applicable
WFS_EXEE_PTR_FIELDERROR	2	None
WFS_EXEE_PTR_FIELDWARNING	2	None
WFS_USRE_PTR_RETRACTBINTHRESHOLD	0	Not applicable
WFS_SRVE_PTR_MEDIATAKEN	0	Not generated for the receipt printer. Not applicable to the journal printer.
WFS_SRVE_PTR_MEDIAINSERTED	0	Not applicable
WFS_USRE_PTR_PAPERTHRESHOLD	2	None
WFS_USRE_PTR_TONERTHRESHOLD	1	The ribbon being ‘Close to Replace’/‘Replace Now’ is reported as TONER LOW/TONER OUT.

PRINTER FORMS

In the implementation of forms, the form and field definitions are stored in files called "Form Definition Files". The names of the directories containing these files are stored separately in the Windows NT Registry. Each file may contain one or more form definitions. The form definition files are read and parsed during the initialization of the service provider, and are validated in the following two stages:

Start-up validation:

When the SP is started, all the form definition files are opened and verified for syntax, 'Required Fields' and content. Here, the syntax refers to the valid keywords, non-duplication of keywords and matching BEGIN/END. The required fields are form **SIZE**, form **LANGUAGE**, field **SIZE** and field **POSITION**. Content is the value of the keywords that must be within the capabilities of the device. The form definitions are converted to an 'in-core' representation called as the 'Forms Database'. Errors found during this initial validation are recorded in the Forms Database and are logged in an error log file. The location of the error log file is specified in the registry.

Runtime Validation:

Runtime validation is performed when an INFO or an EXECUTE command concerned with forms such as QUERY_FORM, QUERY_FIELD, PRINT_FORM or READ_FORM is received. It verifies that the form **SIZE** and field **POSITION** values contained in the definition do not conflict with the device capabilities. For example, it verifies that the width of the form specified is not greater than 40 for the 40-column printers.

Runtime validation also checks the following:

- A value has been provided for **REQUIRED** fields.
- No value is supplied for the **STATIC** fields and **READ** only fields.
- A **WRITE** only field is not specified in the READ_FORM command.

Errors encountered while parsing the form definition files are logged in an error logfile.

Form Parser Error Messages and Limits

Form parser error messages are stored in error log files with names of the form "XXXX_FRM.LOG"

where

XXXX = RPTR for the receipt printer

JPTR for the journal printer

SPTR for the statement printer

PPTR for the passbook printer

The path where these files are created is a default to root ('\'). But, it can be configured in the Registry.

Form Parser Messages

The Form definition files are parsed during the startup. Any information regarding the files read and errors encountered (if any), is stored in the printer error logfile. The messages generated by the form definition parser are as following:

Parser Messages:

Message	**Description**
"Textfile Parser for XFSFORMS Version x.x"	The first line of the error log file, giving the version number of the read function.
"File <file name>"	Indicates the opened files. The messages following thereafter concern this file.
"Form <form name> read successfully"	Indicates the successful reading of the form.
"Parsed OK"	Indicates the successful reading and parsing of the last listed file.
"*n* error(s) found while parsing"	Indicates the unsuccessful reading of the last listed file, and contains *n* errors.

When the read function encounters an error, it is recorded in the error log file. An error message specifying the type of error and the line number is also recorded. The read function tries to continue as far as possible. In order to do so, it aborts the processing of a line or a keyword section, and continues with the next line. This may result in further errors.

Error Messages

Error Message	Description
"Error found in line *n*"	An error has occurred in the *n*th line of the form/media definition file.
"Error while trying to open file"	An error has occurred while opening an input file.
"Error: could not allocate memory"	An error has occurred while allocating memory.
"Error while trying to read from file"	An error has occurred while reading the input file.
"Line *n*: line continuation character invalidated"	The read function has found a character after the line continuation symbol (\). The rest of the line is ignored.
"Line *n*: string termination character not found."	The read function did not find a string termination character, i.e. a CR/LF was encountered first. A string termination character is inserted at the end of the data that has been read for this line.
"Line *n*: keyword expected."	The read function expected to find a keyword with its values. The entire keyword section is ignored.
"Line *n*: argument is missing."	The read function expected to find another argument for this keyword. The whole keyword section is ignored.
"Line *n*: wrong type of argument."	The read function expected to find a different sort of argument for this keyword. The whole keyword section is ignored.
"Line *n*: the name of the form is missing."	The form name is missing, i.e. the read function has found only the keyword XFSFORM and nothing following it. The form will not be inserted into the forms database. The parsing function continues looking for the next XFSFORM *formname*. A missing form name may result in further errors.
"Line *n*: the rest of line is not empty."	The read function has read a keyword section successfully, but the rest of the line is not empty. The rest is ignored.
"Line *n*: a string was expected."	The read function is looking for a string ("string") as an input. The whole keyword section is ignored.
"Line *n*: this is the second occurrence of a keyword"	The read function has encountered a second occurrence of a keyword. This keyword (and its values) is ignored.
"Line *n*: the keyword XFSFORM is expected."	The read function was expecting a line XFSFORM *formname*. It ignores this line, but continues expecting XFSFORM *formname*.

Error Message	Description
"Line *n*: the name of the field is missing."	The read function has found the line XFSFIELD *fieldname*, with the field name missing. This line is ignored. The parsing function will still expect to read form keyword sections and will produce errors for field keyword sections it does not recognize.
"Line *n*: a word is too long to be processed."	A word in the input file is too long, i.e. > 100 characters. This section is ignored.
"Line *n*: an integer was expected."	The read function expected to read an integer. This keyword section is ignored.
"Line *n*: a required keyword is missing."	The read function did not find all the REQUIRED keywords.
"Line *n*: the keyword UNIT is missing in the form definition."	The keyword section UNIT is missing in the form definition. The form will not be inserted into the forms database.
"Line *n*: the keyword SIZE is missing in the form definition."	The keyword section SIZE is missing in the form definition. The form will not be inserted into the forms database.
"Line *n*: the keyword LANGUAGE is missing in the form definition."	The keyword section LANGUAGE is missing from the form definition. The form will not be inserted into the forms database.
"Line *n*: the keyword POSITION is missing in the field definition."	One of the two keyword sections POSITION or FOLLOWS is mandatory for a field definition. Neither one has been defined for this field. The field will not be inserted into the forms database record of the current form.
"Line *n*: the keyword SIZE is missing in the field definition."	The keyword SIZE is missing in the field definition. The field will not be inserted into the definition of the current form.
"Line *n*: a field cannot FOLLOW itself."	The value of the keyword FOLLOWS is equal to the name of this field., i.e. the field would follow itself. This keyword section is ignored.
"Line *n*: two fields have the same field names."	This field has the same field name as another field that has already been read successfully into the current form definition. This field is ignored.
"Line *n*: two forms have the same form names."	This form has the same form name as another form that has already been read successfully, either from the current input file or from a previous input file. This form is ignored.
"Line *n*: an OR operator was expected in the definition of the field style."	An OR operator was expected when reading values for the STYLE keyword of a field definition. This keyword section is ignored.
"Line *n*: a value has occurred twice in the definition of the field style."	One value was given twice while reading the values for the keyword STYLE in a field definition.
"Line *n*: keyword BEGIN was expected."	The keyword BEGIN was expected and not found. The parsing function will ignore the whole field or the form that it has started reading. This will result in further errors.

Error Message	Description
"Line *n*: no forms have been read. check if file is empty."	No forms have been read successfully from the input file. This might be due to the input file being empty.
"Line *n*: line is too long to process."	A line in the input file is too long, > 500 characters. The parsing function ignores the rest of the line and continues reading.
"Line *n*: the field specified with FOLLOWS does not exist."	The position of a field was specified with FOLLOWS *fieldX*. *fieldX* however could not be found in the definition of the current form. The read function checks whether the field has a valid POSITION definition and uses this instead. If the field does not have a valid POSITION definition, it is discarded.
"Line *n*: no fields have been read for this form."	No fields are defined or no fields are successfully read for the current form.
"Line *n*: there are fields following each other in a cycle."	Several fields that use the FOLLOWS definition and create a cycle. (e.g. field1 FOLLOWS field2, field2 FOLLOWS field3, field3 FOLLOWS field1). This error is generated for each field in the cycle. The read function checks whether each field has a valid POSITION definition and uses this instead. If the field does not have a valid POSITION definition, it is discarded.
"Line *n*: a field has overflowed the form definition."	A field overflows the form, i.e. fieldposition + fieldsize > formsize. This field is discarded.
"Line *n*: this is not a valid value for this keyword."	An integer > 0 is expected as a value.

Form Parser Limits

The following are the limits of the Form Parser:

Maximum line length (including line continuation)	500 characters
Maximum token (terminated by white space characters) length	100 characters
Maximum quoted (in "") string length	100 characters
Maximum filename length	100 characters
Maximum number of components in an OR list	10
Maximum Length of a component in an OR list	20

PRINTER MEDIA

The WOSA/XFS Printer Device class functionality is based on the "forms" model. Media definitions are the descriptions of the print media present on the Self Service Terminal. These definitions contain details about the media such as the printable area, restricted area and type of fold (for passbooks). A PRINT_FORM command uses this information for printing on media with the help of a form definition as a format.

In the implementation of forms, the media definitions are stored in files called "Media Definition Files". The names of these files are stored separately in the Windows NT Registry. Each file may contain one or more media definitions. Media definition files are read and parsed during initialization of the service provider.

As in the case of form definitions, media validation is performed in the following two stages:

Start-up validation:

When the SP starts up, all media definition files are opened and verified for syntax, 'required fields' and content. Here, syntax refers to valid keywords, non-duplication of keywords and matching BEGIN/END. Required fields are media **SIZE**, media **UNIT** and media **TYPE**. Content is the value of the keywords which must be within the capabilities of the device. The media definitions are converted to an 'in-core' representation called the 'Media Database'. Errors found during this initial validation, are recorded in the Media Database and logged in an error log file. The location of the error log file is specified in the registry.

Runtime validation:

Runtime validation is done when the info or execute command, concerned with media (such as QUERY_MEDIA, PRINT_FORM or READ_FORM) are received. It verifies that the media SIZE, etc. do not conflict with the capabilities of the device. For example, it verifies that the width of the media specified is not greater than 40 for the 40 - column printers.

Errors encountered while parsing media definition files are logged in a media error logfile.

Media Parser Error Messages and Limits

Media parser error messages are stored in error log files with names of the form "XXXX_MED.LOG"

where

XXXX	=	RPTR for the receipt printer
		JPTR for the journal printer
		SPTR for the statement printer
		PPTR for the passbook printer

The path where these files are created is default to root ('\'). But it can be configured in the Registry.

Media Parser Messages

Media definition files are parsed during startup. Any information regarding the files that are read and the errors encountered (if any), is stored in the printer media error. The messages generated by the media definition parser are as following:

Parser Messages:

"Textfile Parser for XFSMEDIA Version x.x"	First line of the error log file, giving the version number of the read function.
"File <file name>"	Indicates the opened files. The messages following thereafter concerns this file.
"Form <media file name> read successfully"	Indicates the successful reading of the media file.
"Parsed OK"	Indicates the successful reading and parsing of the last listed file.
"*n* error(s) found while parsing"	Indicates the unsuccessful reading of the last listed file, and contains *n* errors.

When the read function encounters an error, it is recorded in the error log file. An error message specifying the type of error and the line number is also recorded. The read function tries to continue as far as possible. In order to do so, it aborts the processing of a line or a keyword section and continues with the next line. This may result in further errors.

Error Messages:

Error Message	Description
"Error found in line *n*"	An error has occurred on the *n*th line of the media definition file.
"Error while trying to open file"	An error has occurred while opening an input file.
"Error: could not allocate memory"	An error has occurred while allocating memory.
"Error while trying to read from file"	An error has occurred while reading from the input file.
"Line *n*: line continuation character invalidated"	The read function has found a character after the line continuation symbol (\). The rest of the line is ignored.
"Line *n*: string termination character not found."	The read function did not find a string termination character, i.e. a CR/LF was encountered first. A string termination character is inserted at the end of the data that has been read for this line.
"Line *n*: keyword expected."	The read function expected to find a keyword with its values. The entire keyword section is ignored.
"Line n: argument is missing."	The read function expected to find another argument for this keyword. The whole keyword section is ignored.
"Line n: wrong type of argument."	The read function expected to find a different sort of argument for this keyword. The whole keyword section is ignored.
"Line n: the name of the form is missing."	The name of a form is missing, i.e. the read function found only the keyword XFSMEDIA and nothing following it. The form will not be inserted into the media database. The parsing function continues looking for the next XFSMEDIA *medianame*. A missing form name results in further errors.
"Line n: the rest of line is not empty."	The read function has read a keyword section successfully, but the rest of the line is not empty. The rest of the line is ignored.
"Line n: a string was expected."	The read function was looking for a string ("string") as an input. The whole keyword section is ignored.
"Line n: this is the second occurrence of a keyword"	The read function has encountered with the second occurrence of a keyword. This keyword (and its values) is ignored.
"Line n: the keyword XFSMEDIA is expected."	The read function was expecting a line XFSMEDIA *medianame*. It ignores this line, but continues expecting XFSMEDIA *medianame*.
"Line n: a word is too long to be processed."	A word in the input file is too long, i.e. > 100 characters. This section is ignored.

Error Message	Description
"Line n: an integer was expected."	The read function expected to read an integer. This keyword section is ignored.
"Line n: a required keyword is missing."	The read function did not find all the required keywords.
"Line n: the keyword UNIT is missing in the form definition."	The keyword section UNIT is missing in the media definition. The media will not be inserted into the media database.
"Line n: the keyword SIZE is missing in the form definition."	The keyword section SIZE is missing in the media definition. The media will not be inserted into the media database.
"Line n: two medias have the same media names."	This media has the same media name as another media, that has already been read successfully either from the current input file or from a previous input file. This media is ignored.
"Line n: keyword BEGIN was expected."	The keyword BEGIN was expected and not found. The parsing function will ignore the whole media it has started reading. This may result in further errors.
"Line n: no forms have been read. check if file is empty."	No media has been read successfully from the input file. This might be due to the input file being empty.
"Line n: line is too long to process."	A line in the input file is too long, > 500 characters. The parsing function ignores the rest of the line and continues reading.
"Line n: this is not a valid value for this keyword."	An integer > 0 is expected as a value.

Media Parser Limits

The following are limits of the Media Parser

Maximum line length (including line continuation)	500 characters
Maximum token (terminated by white space characters) length	100 characters
Maximum quoted (in "") string length	100 characters
Maximum filename length	100 characters
Maximum number of components in an OR list	10
Maximum Length of a component in an OR list	20

Application Guidelines

1 If two fields overlap, the field defined first in the form definition has precedence. Consequently, only a part of the second field will be printed.

2 Double width characters always start in odd numbered columns. Consequently, a space may precede a change from single to double width mode.

3 The following table lists the limits for the Print command. If these limits are crossed, an error code WFS_ERR_INVALID_DATA will be returned and an error message is written to the trace file.

Description	**Limit**
Print Buffer size	2000
Maximum number of batches (batch - string on a particular line for a particular field) that can be printed	100
Maximum number of fields that can be specified as input to the WFS_CMD_PTR_PRINT_FORM command	50
Maximum length of an input string for a field	100

4 If while printing a form, the SP encounters a data that overflows a field, it truncates the field and logs a message in the trace file. But, it still continues printing returning WFS_ERR_INVALID_DATA after printing all the specified fields.

5 Case change will take place only if the active font supports it.

6 The Receipt and Journal Printer SPs and device drivers attempt to recover from some transport and media related errors. If after repeated attempts, the error condition persists, the device goes into a fatal state. A condition that can be cleared only by operator intervention via the VDM. The following table lists the error conditions from which the SP attempts to recover, and the action to be taken for other commonly occurring errors.

Error Condition	**Recovery Action**
Black Mark Error caused by 8 successive failures to feed to a black mark.*	No automatic recovery. Requires operator intervention via VDM
Transport Jam caused by the failure of a receipt to clear the knife edge after a print operation.*	Driver makes 3 attempts to clear transport. If these attempts fail, the condition must be removed by operator intervention via VDM.
Partial printer receipt in transport caused by a powerfailure during a print operation.*	Driver automatically feeds to black mark, and cuts and presents the receipt before the next print. Can also be cleared by pressing the paper load switch.
Paper Low during a print operation.	Replenish the paper roll. The driver automatically detects replacement.
Paper Out caused by paper running out during a print operation or, in the case of the receipt printer, 80 receipts being printed since a paper low condition is detected.	Replenish the paper roll. The driver automatically detects replacement.
Paper Out or Jam before Transport caused by a print command being issued when no paper is loaded.	The driver treats this as a mechanism error. No automatic recovery. Requires operator intervention via VDM.
Receipt/Journal Printer Open	Automatically cleared when the printer unit is closed.

* Receipt Printer only

Statement Printer

Service Provider Components

Device	DLL Name	SP Executable
Statement Printer (all variants)	sptr_spx.dll sptr_wfp.dll sptr_ipc.dll sptr.dll	sptr.exe

Default Logical Service Names

Logical Name	Description
StatementPrinter1	Logical name of the statement printer service provider.

Configurable Parameters

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS key.

Parameter	Description	Permissible Values
\FORMS\FORM_FILE_PATH	The path to the form file definitions.	Any valid absolute path.
\MEDIA\MEDIA_FILE_PATH	The path to the media file definitions.	Any valid absolute path.
\ERROR_FILE_PATH\PTR_ERROR_FILE_PATH	Path to store the error file for logging errors that occur during initialization.	Any valid absolute path.

The following configurable parameter values are stored under the WOSA/XFS_ROOT\SERVICE_PROVIDERS\SPTR key:

Parameter	Description	Permissible Values
GENERAL_CONFIGS\Variant	Variant of the hardware installed on ATM.	The following variants are supported by the statement printer SP. STD_STATEMENT - standard statement BUNCHER_STATEMENT - buncher statement COMBINATION_STATEMENT_PASSBOOK - combination statement/passbook
GENERAL_CONFIGS\SuspendTimeout	Time for which the device is suspended when customer tampering is suspected.	Time in seconds. Defaults to 300 secs.
GENERAL_CONFIGS\MediaLength	Statement printer media length.	Statement paper length in inches. Values of MediaLength can be 4” or 6”. Defaults to 4”.
GENERAL_CONFIGS\Combined	Specifies whether the statement printer is a combined statement passbook printer.	Values can be • 0 if not combined • 1 if combined Defaults to 0.
GENERAL_CONFIGS\MaxMediaRetracted	Maximum media items that the retract bin can hold.	10 by default for the statement printer (6 bunches, i.e. maximum of 10 x 6 = 60 statements for the buncher variant).
GENERAL_CONFIGS\MaxMediaOnStacker	Maximum media items that can be stacked by the buncher statement printer.	10 by default for the buncher variant of the statement printer.

Capabilities

Capability	Value
Compound	TRUE (for compound statement / passbook printer) FALSE (for standard and buncher statement-only printers)
Resolution	LOW
Read Form Support	FALSE
Write Form Support	TRUE (only text)
Extents Support	FALSE
Supported Controls	CUT, EJECT, FLUSH, RETRACT, STACK (STACK only for buncher variants)
Asynchronous media acceptance	FALSE
Retract Bin capacity	10 for the standard and the combination statement/passbook printer. 6 x 10 = 60 for the buncher statement printer.
Stacker capacity	10 x 10 = 100 for buncher statement printer. 10 for standard statement and combined statement/passbook printer.

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_PTR_STATUS	1	l *fwDevice* of WFS_PTR_DEVPOWEROFF, WFS_PTR_DEVBUSY and WFS_PTR_DEVNODEVICE are never returned. l *usRetractCount* field indicates the total number of items in the retract bin. For the combination statement/passbook, this is the sum of the statements and passbooks present in the retract bin.
WFS_INF_PTR_CAPABILITIES	2	None
WFS_INF_PTR_FORM_LIST	2	None
WFS_INF_PTR_MEDIA_LIST	2	None
WFS_INF_PTR_QUERY_FORM	2	None
WFS_INF_PTR_QUERY_MEDIA	2	None
WFS_INF_PTR_QUERY_FIELD	2	None
WFS_CMD_PTR_CONTROL_MEDIA	1	l Supports only the following control codes: l WFS_PTR_CTRLCUT, l WFS_PTR_CTRLEJECT, l WFS_PTR_CTRLRETRACT l WFS_PTR_CTRLFLUSH l WFS_PTR_CTRLSTACK (Buncher variant only.) l WFS_PTR_CTRLCUT will result in cutting as well as ejecting the statement. l Unsupported control codes are ignored. l If the control codes WFS_PTR_CTRLRETRACT and WFS_PTR_CTRLEJECT are specified together, the error code WFS_ERR_INVALID_DATA is returned. l If WFS_PTR_CTRLFLUSH is specified, without any printed statement being present from a previous WFS_CMD_PTR_PRINT_FORM command, no error is generated. Instead, a new statement is moved to the print position. l However, in the above case, WFS_PTR_CTRLRETRACT and WFS_PTR_CTRLEJECT will return the error code WFS_ERR_PTR_NOMEDIAPRESENT.

WOSA Command	CL	Comments

WFS_CMD_PTR_PRINT_FORM	1	• Right aligned fields are not supported.
		• Supports only low resolution. Other resolutions are ignored.
		• Does not return error code WFS_ERR_PTR_MEDIASKEWED.
		• Supports only the following control codes:
		  • WFS_PTR_CTRLCUT,
		  • WFS_PTR_CTRLEJECT
		  • WFS_PTR_CTRLRETRACT
		  • WFS_PTR_CTRLFLUSH
		  • WFS_PTR_CTRLSTACK (Buncher variant only)
		• WFS_PTR_CTRLCUT will result in cutting as well as ejecting the statement.
		• Unsupported control codes are ignored.
		• If control codes WFS_PTR_CTRLRETRACT and WFS_PTR_CTRLEJECT are specified together, the error code WFS_ERR_INVALID_DATA is returned.
		• The form and fields specified for printing should satisfy the following conditions:
		  • Orientation cannot be LANDSCAPE.
		  • SKEW can only be 0.
		  • SIDE can only be FRONT.
		  • TYPE can only be TEXT.
		  • GRAPHICS can only be BESTFIT (default).
		  • BARCODE can only be NONE (default).
		  • STYLE can only be NORMAL (default) or DOUBLE for the combination statement/passbook printer.
		  However, for the standard statement-only and buncher statement variants, STYLE can have values NORMAL (default), DOUBLE, BOLD and UNDER.
		  • HORIZONTAL justification can only be LEFT (default), RIGHT or CENTER.
		  • VERTICAL justification can be TOP (default), CENTER, BOTTOM or JUSTIFY.
		  • COLOR can only be BLACK.

continued...

WOSA Command	CL	Comments
WFS_CMD_PTR_PRINT_FORM	1	• FONT should be one of the following: INTERNATIONAL1 INTERNATIONAL2 INTERNATIONAL3 INTERNATIONAL4 INTERNATIONAL5 ARABIC1 ARABIC2 ARABIC3 ARABIC4 ARABIC5 • POINTSIZE is not supported and if specified is (ignored). • CPI can be 5.0, 10.0 (default), 12.0 or 17.14. However, if CPI is specified as other than 10.0 (default), then CASE should not be DOUBLE. • CASE change is effective only if the font supports it. • LPI can only take values between 1.0 and 9.0. The following restrictions also apply: • Two multi-line fields (fields with height > 1), say field1 and field2 having different values for LPI should satisfy the following conditions: - field1y!= field2y and - field1y >= field2y + field2height - 1 i.e. two multi-line fields having different values for LPI should not overlap. • FORMAT is not supported, and if specified is ignored. • The media specified for printing should satisfy the following conditions: • TYPE can only be GENERIC (default). • RESTRICTED, FOLD, STAGGERING, PAGE and LINES are not supported, and if specified are ignored • SIZE *width* should be <= 80 • SIZE *height* should be <= 23 for Statement printer with 4" media, • SIZE *height* <= 39 for Statement printer with 6" media.
WFS_CMD_PTR_PRINT_FORM	1	None
WFS_CMD_PTR_READ_FORM	0	None
WFS_CMD_PTR_RAW_DATA	1	The raw data input is passed to the printer without interpretation. Unsupported control codes are replaced by the character ']'. The raw data is always printed from the first line, first column onwards.

WOSA Command	CL	Comments
WFS_CMD_PTR_MEDIA_EXTENTS	0	None
WFS_CMD_PTR_RESET_COUNT	2	None
WFS_CMD_PTR_READ_IMAGE	0	None

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_INF_PTR_FORM_LIST	None	2	None
WFS_INF_PTR_MEDIA_LIST	None	2	None
WFS_INF_PTR_QUERY_FORM	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_FORMINVALID	2	Returned when the form definition specified does not satisfy the conditions given in the command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_FORMNOTFOUND	2	Returned when the form name specified is not found in the form files at startup.
WFS_INF_PTR_QUERY_MEDIA	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_MEDIANOTFOUND	2	Returned when the media name specified is not found in the media files.
	WFS_ERR_PTR_MEDIAINVALID	2	Returned when the media definition specified does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
WFS_INF_PTR_QUERY_FIELD	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_FORMINVALID	2	Returned when the form definition specified in the input structure does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_FORMNOTFOUND	2	Returned when the form name specified in the input structure is not found in the form files.

continued....

WOSA Command	Error Codes	CL	Comments
WFS_INF_PTR_QUERY_FIELD	WFS_ERR_PTR_FIELDNOTFOUND	2	Returned when the field name specified in the input structure is not found in the form definition.
	WFS_ERR_PTR_FIELDINVALID	2	Returned when the field definition specified does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
WFS_CMD_PTR_CONTROL_MEDIA	WFS_ERR_INVALID_POINTER	2	Returned when the control input is not passed to the command, or if the printer is NULL or invalid.
	WFS_ERR_INVALID_DATA	2	Returned when an unknown control code for *dwMediaControl* is specified in the input. Also returned when WFS_PTR_CTRLEJECT and WFS_PTR_CTRLRETRACT commands are specified together.
	WFS_ERR_PTR_NOMEDIAPRESENT	2	Generated when the printer is out of paper.
	WFS_ERR_PTR_FLUSHFAIL	0	Never generated since the data is always flushed to the device.
	WFS_ERR_PTR_RETRACTBINFULL	2	Generated when the retract bin has more than 10 media items in it. For the combination statement/passbook, this is the sum of the statements and passbooks in the retract bin.
	WFS_ERR_PTR_STACKERFULL	2	Generated when the stacker has stacked 10 statements (buncher statement printer only). Returned only when WFS_PTR_CTRLSTACK command is specified in the input.
	WFS_ERR_PTR_PAGETURNFAIL	0	None
	WFS_ERR_PTR_MEDIATURNFAIL	0	None

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PTR_PRINT_FORM	WFS_ERR_INVALID_POINTER	2	Returned when the input structure is not passed to the command, or if the printer is NULL or invalid.
	WFS_ERR_INVALID_DATA	2	Returned when an unknown control code for *dwMediaControl* is specified in the input. Also returned when WFS_PTR_CTRLEJECT and WFS_PTR_CTRLRETRACT commands are specified together.
	WFS_ERR_PTR_FORMINVALID	2	Returned when the form definition specified in the input structure does not satisfy the conditions given in the command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_FORMNOTFOUND	2	Returned when the form name specified in the input structure is not found in the form files.
	WFS_ERR_PTR_MEDIANOTFOUND	2	Returned when the media name specified in the input structure is not found in the media files.
	WFS_ERR_PTR_MEDIAINVALID	2	Returned when the media definition specified in the input structure does not satisfy the conditions given in the command conformance matrix for the WFS_CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_MEDIASKEWED	0	Never returned. Skew cannot be detected
	WFS_ERR_PTR_MEDIAOVERFLOW	2	Returned when the print data overflows the media boundaries.
	WFS_ERR_PTR_FIELDSPECFAILURE	2	Returned when a field is specified twice in the input.
	WFS_ERR_PTR_FIELDERROR	2	Returned if an error occurred while processing a field.

continued...

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PTR_PRINT_FORM	WFS_ERR_PTR_FLUSHFAIL	2	Additional error code. Returned when the print form command tries to print data above or before the current position of the print head.
	WFS_ERR_PTR_RETRACTBINFULL	2	Generated after 10 captures have been performed.
	WFS_ERR_PTR_STACKERFULL	2	Generated when the stacker has stacked 10 statements (buncher statement printer only). No printing is done if the stacker is full.
	WFS_ERR_PTR_PAGETURNFAIL	0	None
	WFS_ERR_PTR_MEDIATURNFAIL	0	None
WFS_CMD_PTR_READ_FORM	WFS_ERR_PTR_READNOTSUPPORTED	2	Read is not supported by the statement printer.
	WFS_ERR_PTR_FORMINVALID	0	Never returned
	WFS_ERR_PTR_FORMNOTFOUND	0	Never returned
	WFS_ERR_PTR_MEDIANOTFOUND	0	Never returned
	WFS_ERR_PTR_MEDIAINVALID	0	Never returned
	WFS_ERR_PTR_MEDIASKEWED	0	Never returned
	WFS_ERR_PTR_FIELDSPECFAILURE	0	Never returned
	WFS_ERR_PTR_FIELDERROR	0	Never returned
	WFS_ERR_PTR_RETRACTBINFULL	0	Never returned
	WFS_ERR_PTR_STACKERFULL	0	Never returned
	WFS_ERR_PTR_PAGETURNFAIL	0	Never returned
	WFS_ERR_PTR_MEDIATURNFAIL	0	Never returned
WFS_CMD_PTR_RAW_DATA	None	0	None
WFS_CMD_PTR_MEDIA_EXTENTS	WFS_ERR_PTR_EXTENTNOTSUPPORTED	2	The device does not report extents.
WFS_CMD_PTR_RESET_COUNT	None	0	None
WFS_CMD_PTR_READ_IMAGE	None	0	None

Conformance Matrix - Events

WOSA Event	CL	Comments
WFS_EXEE_PTR_NOMEDIA	2	None
WFS_EXEE_PTR_MEDIAINSERTED	0	None
WFS_EXEE_PTR_FIELDERROR	2	None
WFS_EXEE_PTR_FIELDWARNING	2	None
WFS_SRVE_PTR_MEDIATAKEN	2	None
WFS_SRVE_PTR_MEDIAINSERTED	0	None
WFS_USRE_PTR_PAPERTHRESHOLD	2	Generated only when a print (WFS_CMD_PTR_PRINT_FORM or WFS_CMD_PTR_RAW_DATA) is attempted.
WFS_USRE_PTR_TONERTHRESHOLD	2	Generated only when a print (WFS_CMD_PTR_PRINT_FORM or WFS_CMD_PTR_RAW_DATA) is attempted.
WFS_USRE_PTR_RETRACTBINTHRESHOLD	2	Generated after 10 captures have been performed.

PRINTER FORMS

Refer to the Printer Forms section of the Receipt and Journal Printers Device Class.

Application Guidelines

The following guidelines are in addition to those for the Receipt and Journal printers.

1 An error file (sptr_frm.log) is created during the parsing of the form definition files. All the errors during the parse run are logged into this error file. The cause of error/s can be found by examining this file. Similarly, an error file (sptr_med.log) is created while parsing of the media definition files. All the errors during the parse run are logged into this error file. The cause of error/s can be found by examining this file.

2 Any statement present at the time of SP initialization is captured. Similarly, for the combination statement/passbook printer, a statement or passbook present is captured.

3 For the combination statement/passbook printers, ensure that all the transactions with the statement printer end with an eject or a capture before issuing commands to the passbook printer. If a printed statement is present under the head, and a print command is given to the passbook printer, a system escape might take place.

4 A statement is printed with the line "INITIALIZING STATEMENT PRINTER" during initialization. This is then captured and retracted into the retract bin to update the status.

5 The timeouts specified should be long enough (> 30s approx.) for printing, stacking, capturing and ejecting a statement. Especially, the buncher variant takes a long time. The correct timeouts can be found out by trial-and-error.

6 In case a customer tampering is suspected, the commands issued after the command that have sensed the conditions are not accepted for the period (*SuspendTimeout*) specified in the registry. These commands will return with the error code WFS_ERR_DEV_NOT_READY. Also, if any statement is present in the transport, it is captured. The status is checked at intervals of *SuspendTimeout* seconds, and a system status change event is generated if the device status changes.

7 The Statement SP and device driver attempt to recover from some transport and media related errors. If after repeated attempts, the error condition persists, the device goes into a fatal state. A condition that can be cleared only by operator intervention via the VDM. The following table lists the error conditions from which the SP attempts to recover, and the action to be taken for other commonly occurring errors.

Error Condition	Recovery Action
Paper Low during a print operation.	Replenish the media. The driver automatically detects replacement.
Capture Bin full	Empty the capture bin. The driver automatically resets the capture count.
Form Jam during capture	No automatic recovery. Requires operator intervention via VDM.
Paper Load Error	No automatic recovery. Requires operator intervention via VDM.
Media Jam	No automatic recovery. Requires operator intervention via VDM.
Customer tampering during capture.	When the driver detects a statement stuck in the throat of the printer which cannot be pulled in, the SP assumes that customer tampering has occurred and enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Clamp Jammed Closed (Buncher variant only)	If the printer driver detects the clamp in a down position before paper is fed to the print-head at the start of a statement print operation, the SP assumes that customer tampering has occurred and enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.

Passbook Printer

Service Provider Components

Device	DLL Name	SP Executable
Passbook Printer	pptr_spx.dll pptr_wfp.dll pptr_ipc.dll pptr.dll	pptr.exe

Default Logical Service Names

Logical Name	Description
PassbookPrinter1	Logical name of the passbook printer service provider.

Configurable Parameters

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS key.

Parameter	Description	Permissible Values
\FORMS\FORM_FILE_PATH	The path to the form file definitions.	Any valid absolute path.
\MEDIA\MEDIA_FILE_PATH	The path to the media file definitions.	Any valid absolute path.
\ERROR_FILE_PATH\PTR_ERROR_FILE_PATH	Path to store the error file for logging errors that occur during initialization.	Any valid absolute path.

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS\PPTR key:

Parameter	Description	Permissible Values
GENERAL_CONFIGS\Variant	Variant of the hardware installed on the ATM.	The following variants are supported for the passbook printer: MAGNETIC - Magnetic passbook printer NON_MAGNETIC- Non-magnetic passbook printer
GENERAL_CONFIGS\SuspendTimeout	Time for which the device is suspended when customer tampering is suspected.	Time in seconds. Defaults to 300 seconds.

Parameter	Description	Permissible Values
GENERAL_CONFIGS\MagFormat	Magnetic format to be used while accepting a passbook, reading from magnetic track from a passbook or writing to magnetic track of a passbook.	Can be one of the following: • DIN • IBM • ISO • NONE (if no magnetic stripe) If not specified, defaults to NONE
GENERAL_CONFIGS\BarcodeFormat	Barcode format to be used while defining the barcode scan.	Can be one of the following: • NCR • FUJITSU_STD • IBM_STD • FUJITSU_FREE • IBM_FREE If not specified, defaults to FUJITSU_STD
GENERAL_CONFIGS\MaxMediaRetracted	Maximum media items that the retract bin can hold.	Device capability is currently 10.

Capabilities

Capability	Value
Compound	TRUE
Resolution	LOW
Read Form Support	TRUE. (MSF and Pagemark. MSF is supported by MAGNETIC variants only)
Write Form Support	TRUE. (TEXT and MSF. MSF is supported by MAGNETIC variants only)
Extents Support	FALSE
Supported controls	EJECT, FLUSH, RETRACT
Asynchronous media acceptance	FALSE
Retract Bin capacity	10
Stacker capacity	0

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_PTR_STATUS	1	• *.fwDevice* of WFS_PTR_DEVPOWEROFF, WFS_PTR_DEVBUSY and WFS_PTR_DEVNODEVICE are never returned • *usRetractCount* field indicates the total number of items in the retract bin. For the combination statement/passbook, this is the sum of the statements and passbooks present in the retract bin. • The status command, if not preceded by a PRINT_FORM or a READ_FORM command will not return the latest replenishment information.
WFS_INF_PTR_CAPABILITIES	2	None
WFS_INF_PTR_FORM_LIST	2	None
WFS_INF_PTR_MEDIA_LIST	2	None
WFS_INF_PTR_QUERY_FORM	2	None
WFS_INF_PTR_QUERY_MEDIA	2	None
WFS_INF_PTR_QUERY_FIELD	2	None

WFS_CMD_PTR_CONTROL_MEDIA	1	• Supports only the following control codes:
		  ◦ WFS_PTR_CTRLEJECT
		  ◦ WFS_PTR_CTRLRETRACT
		  ◦ WFS_PTR_CTRLFLUSH.
		• Non-supported control codes are ignored.
		• If the control codes WFS_PTR_CTRLRETRACT and WFS_PTR_CTRLEJECT are specified together, the error code WFS_ERR_INVALID_DATA is returned.
		• A CONTROL_MEDIA command, without a passbook present in the printer will return the error code WFS_ERR_PTR_NOMEDIAPRESENT.

WOSA Command	CL	Comments

WFS_CMD_PTR_PRINT_FORM	1	• Right aligned fields are not supported.
		• Supports only low resolution. Other resolutions are ignored.
		• Does not return error code WFS_ERR_PTR_MEDIASKEWED.
		• If there are multiple MSF fields in a form definition, only POSITION, SIZE, TYPE, CLASS and ACCESS need to be specified in the field definition of type MSF. Other values are ignored.
		• An MSF field definition should satisfy the following conditions:
		  • POSITION *x* should specify the beginning of the stored data. POSITION *y* is ignored.
		  • TYPE should be MSF.
		  • ACCESS should be WRITE or READWRITE.
		  • If CLASS is STATIC, INITIALVALUE must be specified.
		• It is assumed that if there are multiple fields of type MSF in a form, they will be defined consecutively in the form definition file.
		• Supports only the following control codes:
		  • WFS_PTR_CTRLEJECT,
		  • WFS_PTR_CTRLRETRACT
		  • WFS_PTR_CTRLFLUSH.
		• Unsupported control codes are ignored.
		• If control codes WFS_PTR_CTRLRETRACT and WFS_PTR_CTRLEJECT are specified together, the error code WFS_ERR_INVALID_DATA will be returned.
		• The form and fields specified for printing should adhere to following:
		  • Orientation cannot be LANDSCAPE
		  • SKEW can only be 0.
		  • SIDE can only be FRONT.
		  • TYPE can only be TEXT, MSF or PAGEMARK.
		  • GRAPHICS can only be BESTFIT (default).
		  • BARCODE can only be NONE (default).
		  • STYLE can only be NORMAL (default) and DOUBLE for the combination statement/passbook printer.

continued....

WOSA Command	CL	Comments
WFS_CMD_PTR_PRINT_FORM	1	l HORIZONTAL justification can only be LEFT (default), RIGHT or CENTER. l VERTICAL justification can be TOP (default), CENTER, BOTTOM or JUSTIFY. l COLOR can only be BLACK. l If a print or a read is attempted to or from a field of TYPE MSF and if the horizontal magnetic stripe reader is absent, the field error WFS_ERR_PTR_FIELDINVALID is returned. l FONT in the field definition should be one of the following: INTERNATIONAL1 INTERNATIONAL2 INTERNATIONAL3 INTERNATIONAL4 INTERNATIONAL5 ARABIC1 ARABIC2 ARABIC3 ARABIC4 ARABIC5 l POINTSIZE is not supported and if specified, is ignored. l CPI can be 5.0, 10.0 (default), 12.0 or 17.14. However, if CPI is specified as 10.0 (default), the CASE should not be DOUBLE. l CASE change is effective only if the active font supports it. l LPI is not supported, and if specified, is ignored. l FORMAT is not supported, and if specified, is ignored. l The media specified for printing should satisfy the following conditions: l TYPE can only be PASSBOOK (default). l SIZE *width* should be <= 80.

continued....

WOSA Command	CL	Comments
WFS_CMD_PTR_PRINT_FORM	1	l PRINTAREA *width* should be <= SIZE *width*. l PRINTAREA *height* should be <= SIZE *height*. l PRINTAREA *x* should be <= SIZE *width*. l SIZE *width* should be >= PRINTAREA *x* + PRINTAREA *width*. l PRINTAREA *y* should be <= SIZE *height*. l SIZE *height* >= PRINTAREA *y* + PRINTAREA *height* + 2 * STAGGERING. l RESTRICTED *x* should be <= SIZE *width*. l SIZE *width* >= RESTRICTED *x* + RESTRICTED *width*. l RESTRICTED *y* <= RESTRICTED *height*. l SIZE *height* >= RESTRICTED *y* + RESTRICTED *height* + 2 * STAGGERING.
WFS_CMD_PTR_READ_FORM	1	l Does not return error code WFS_ERR_PTR_MEDIASKEWED. l A CONTROL_MEDIA command, without a passbook present inside the printer will return the error code WFS_ERR_PTR_NOMEDIAPRESENT. l Supports only the following control codes: l WFS_PTR_CTRLEJECT, l WFS_PTR_CTRLRETRACT l WFS_PTR_CTRLFLUSH. l Non-supported control codes are ignored. l If control codes WFS_PTR_CTRLRETRACT and WFS_PTR_CTRLEJECT are specified together, the error code WFS_ERR_INVALID_DATA will be returned. l Only MSF fields (those stored on the magnetic stripe) and the pagemark can be read. l Only POSITION, SIZE, CLASS and ACCESS need to be specified in the field definition of MSF type fields. Other values are ignored. l The values in the field definition of an MSF field should satisfy the following conditions: l POSITION *x* should specify the beginning of the stored data. POSITION *y* is ignored. l TYPE should be MSF. l ACCESS can be READ or READWRITE. l If CLASS is STATIC, INITIALVALUE must be specified.

continued...

WOSA Command	CL	Comments
WFS_CMD_PTR_READ_FORM	1	l It is assumed that if there are multiple fields of type MSF in a form, they will be defined consecutively in the form definition file. l All values in the field definition with type PAGEMARK are ignored. For 'FREE' barcode formats, the value returned by the PAGE_NUMBER command will be returned as described in Ref 24**.** It is the applications responsibility to interpret the codes. l The form and fields specified for printing should satisfy the following conditions: l Orientation cannot be LANDSCAPE l SKEW can only be 0. l TYPE can only be MSF or PAGEMARK. l If a read is attempted from a field of TYPE MSF and if the horizontal magnetic stripe reader is absent, the field error WFS_ERR_PTR_FIELDINVALID is returned. l The media specified for printing should satisfy the following conditions: l TYPE can only be PASSBOOK (default). l SIZE *width* should be <= 80. l PRINTAREA *width* should be <= SIZE *width*. l PRINTAREA *height* should be <= SIZE *height*. l PRINTAREA *x* should be <= SIZE *width*. l SIZE *width* should be >= PRINTAREA *x* + PRINTAREA *width*. l PRINTAREA *y* should be <= SIZE *height*. l SIZE *height* >= PRINTAREA *y* + PRINTAREA *height* + 2 * STAGGERING *staggering*. l RESTRICTED *x* should be <= SIZE *width*. l SIZE *width* >= RESTRICTED *x* + RESTRICTED *width*. l RESTRICTED *y* <= RESTRICTED *height*. l SIZE *height* >= RESTRICTED *y* + RESTRICTED *height* + 2 * STAGGERING *staggering*.
WFS_CMD_PTR_RAW_DATA	1	The raw data input is passed to the printer without interpretation. Unsupported control codes are replaced by the character ']'.The raw data is always printed from the first line, first column onwards.
WFS_CMD_PTR_MEDIA_EXTENTS	0	None
WFS_CMD_PTR_RESET_COUNT	2	None
WFS_CMD_PTR_READ_IMAGE	0	None

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_INF_PTR_FORM_LIST	None	2	None
WFS_INF_PTR_MEDIA_LIST	None	2	None
WFS_INF_PTR_QUERY_FORM	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_FORMINVALID	2	Returned when the form definition specified does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_FORMNOTFOUND	2	Returned when the form name specified is not found in the form files.
WFS_INF_PTR_QUERY_MEDIA	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_PTR_MEDIANOTFOUND	2	Returned when the media name specified is not found in the media files.
	WFS_ERR_PTR_MEDIAINVALID	2	Returned when the media definition specified does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
WFS_INF_PTR_QUERY_FIELD	WFS_ERR_INVALID_POINTER	2	Returned when the input pointer is NULL or invalid.
	WFS_ERR_PTR_FORMINVALID	2	Returned when the form definition specified in the input structure does not satisfy the conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_FORMNOTFOUND	2	Returned when the form name specified in the input structure is not found in form file.

continued....

WOSA Command	Error Codes	CL	Comments
WFS_INF_PTR_QUERY_FIELD	WFS_ERR_PTR_FIELDNOTFOUND	2	Returned when the field name specified in the input structure is not found in the form definition.
	WFS_ERR_PTR_FIELDINVALID	2	Returned when the field definition specified does not satisfy the conditions given in the command conformance matrix for CMD_PTR_PRINT_FORM command.
WFS_CMD_PTR_CONTROL_MEDIA	WFS_ERR_INVALID_POINTER	2	Returned if the input pointer is NULL or invalid.
	WFS_ERR_INVALID_DATA	2	Returned when an unknown control code for *dwMediaControl* is specified in the input. Also returned when WFS_PTR_CTRLEJECT and WFS_PTR_CTRLRETRACT commands are specified together.
	WFS_ERR_PTR_NOMEDIAPRESENT	2	Generated when no passbook is present in the printer.
	WFS_ERR_PTR_FLUSHFAIL	0	Never generated since the data is always flushed to the device.
	WFS_ERR_PTR_RETRACTBINFULL	2	Generated when the retract bin has more than 10 media items in it. For the combination statement/passbook, this is the sum of the statements and passbooks present in the retract bin.
	WFS_ERR_PTR_STACKERFULL	0	None
	WFS_ERR_PTR_PAGETURNFAIL	0	None
	WFS_ERR_PTR_MEDIATURNFAIL	0	None

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PTR_PRINT_FORM	WFS_ERR_INVALID_POINTER	2	Returned when the input structure pointer is NULL or invalid.
	WFS_ERR_INVALID_DATA	2	Returned when an unknown control code is specified for *dwMediaControl* in the input. Also returned when WFS_PTR_CTRLEJECT and WFS_PTR_CTRLRETRACT are specified together.
	WFS_ERR_PTR_FORMINVALID	2	Returned when the form definition specified in the input structure does not adhere to conditions given in command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_FORMNOTFOUND	2	Returned when the form name specified in the input structure is not found in the forms file.
	WFS_ERR_PTR_MEDIANOTFOUND	2	Returned when the media name specified in the input structure is not found in the media files.
	WFS_ERR_PTR_MEDIAINVALID	2	Returned when the media definition specified in the input structure does not satisfy the conditions given in the command conformance matrix for the WFS_CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_MEDIASKEWED	0	Never returned. Skew cannot be detected
	WFS_ERR_PTR_MEDIAOVERFLOW	2	Returned whenever the print data overflows the media boundaries.

continued...

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PTR_PRINT_FORM	WFS_ERR_PTR_FIELDSPECFAILURE	1	Returned when a field is specified twice in the input.
	WFS_ERR_PTR_FIELDERROR	2	Returned if an error occurred while processing a field.
	WFS_ERR_PTR_FLUSHFAIL	2	Additional error code. Returned when the print form command tries to print data above or before the current position of the print head.
	WFS_ERR_PTR_RETRACTBINFULL	2	Generated after 10 captures have been performed.
	WFS_ERR_PTR_STACKERFULL	0	None
	WFS_ERR_PTR_PAGETURNFAIL	0	None
	WFS_ERR_PTR_MEDIATURNFAIL	0	None
WFS_CMD_PTR_READ_FORM	WFS_ERR_PTR_READNOTSUPPORTED	0	None
	WFS_ERR_PTR_FORMINVALID	2	Returned when the form definition specified in the input structure does not satisfy the conditions given in the command conformance matrix for the CMD_PTR_PRINT_FORM command.
	WFS_ERR_PTR_FORMNOTFOUND	2	None
	WFS_ERR_PTR_MEDIANOTFOUND	2	None
	WFS_ERR_PTR_MEDIAINVALID	2	Returned when the media definition specified in the input structure does not satisfy the conditions given in command conformance matrix for the WFS_CMD_PTR_PRINT_FORM command.

continued...

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PTR_READ_FORM	WFS_ERR_PTR_MEDIASKEWED	0	Never returned since skew cannot be detected.
	WFS_ERR_PTR_FIELDSPECFAILURE	2	Returned when a field is specified twice in the input.
	WFS_ERR_PTR_FIELDERROR	2	Returned if an error occurred while processing a field.
	WFS_ERR_PTR_RETRACTBINFULL	2	Generated after 10 captures have been performed.
	WFS_ERR_PTR_STACKERFULL	0	None
	WFS_ERR_PTR_PAGETURNFAIL	0	Not supported
	WFS_ERR_PTR_MEDIATURNFAIL	0	Not supported
WFS_CMD_PTR_RAW_DATA	None		None
WFS_CMD_PTR_MEDIA_EXTENTS	WFS_ERR_PTR_EXTENTNOTSUPPORTED	2	The device does not report extents.
WFS_CMD_PTR_RESET_COUNT	None	0	None
WFS_CMD_PTR_READ_IMAGE	None	0	None

Conformance Matrix - Events

WOSA Event	CL	Comments
WFS_EXEE_PTR_NOMEDIA	2	None
WFS_EXEE_PTR_MEDIAINSERTED	2	None
WFS_EXEE_PTR_FIELDERROR	2	None
WFS_EXEE_PTR_FIELDWARNING	2	None
WFS_SRVE_PTR_MEDIATAKEN	2	None
WFS_SRVE_PTR_MEDIAINSERTED	0	None
WFS_USRE_PTR_PAPERTHRESHOLD	0	None
WFS_USRE_PTR_TONERTHRESHOLD	1	Generated only if a print (WFS_CMD_PTR_PRINT_FORM or WFS_CMD_PTR_RAW_DATA) is attempted.
WFS_USRE_PTR_RETRACTBINTHRESHOLD	2	Generated after 10 captures have been performed.

PRINTER FORMS

Refer to the Printer Forms section of the Receipt and Journal Printers device class.

Application Guidelines

1 An error file (pptr_frm.log) is created during the parsing of the form definitions files. All the errors during the parse run are logged into this error file. The cause of error/s can be found by examining this file.

2 Similarly, an error file (pptr_med.log) is created during parsing of the media definitions files. All the errors during the parse run are logged into this error file. The cause of error/s can be found by examining this file.

3 Any passbook present at the time of SP initialization is captured. Similarly, for the combination statement/passbook printer, a statement or passbook present is captured.

4 For the combination statement/passbook printer, ensure that all the transactions with the passbook printer end with an eject or capture, before issuing commands to the statement printer. If a passbook is present under the head and a print command is given to the statement printer, a system escape may take place.

5 The timeouts specified should be long enough (> 20s approx) for printing, capturing and ejecting a passbook. The correct timeouts can be found out by trial-and-error.

6 When customer tampering is suspected, commands issued after the command that have sensed the conditions are not accepted for the suspend period (*SuspendTimeout)* specified in the registry. These commands will return with the error code WFS_ERR_DEV_NOT_READY. Also, if any statement is present in the transport, it is captured. The status is checked every *SuspendTimeout* seconds and a system status change event is generated if the status changes.

7 If two fields overlap, the field defined first in the form definition has precedence. Consequently, only part of the second field will be printed.

8 Double width characters always start in odd numbered columns. Consequently, a space may precede a change from single to double width mode.

9 Case change will take place only if the active font supports it.

10 Buffer limits are as described in the 'Print Command Limits' of the Statement Printer section of this document.

11 The Passbook SP and device driver attempt to recover from some transport and media related errors. If after repeated attempts, the error condition persists, the device goes into a fatal state, a condition that can be cleared only by operator intervention via the VDM. The following table lists the error conditions from which the SP attempts to recover, and the action to be taken for other commonly occurring errors.

Error Condition	Recovery Action
Passbook jam during Eject	When the device driver detects a passbook jam during an eject operation, it makes 3 attempts to clear the jam. If these attempts fail, the SP assumes that customer tampering has occurred and enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM. The driver also maintains a count of consecutive jams that gets cleared on a successful accept-capture/eject sequence. If this count exceeds 20, the SP returns WFS_ERR_HARDWARE_ERROR operator intervention via VDM is required to clear the error.
Passbook Jam during Accept	When the driver detects a passbook stuck in the throat of the printer which cannot be pulled in, it makes 7 attempts to move the passbook into the printer. If these attempts fail, the SP assumes that customer tampering has occurred and enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Read Errors	If the device driver encounters a read failure, it make 2 more read attempts before reporting a read error. The SP maintains a count of consecutive read errors for each track which is cleared by a successful read from the appropriate track. If this count exceeds 20, the SP returns WFS_ERR_HARDWARE_ERROR and continues to do so until the condition is cleared by operator intervention via VDM.

Error Condition	Recovery Action
Write Errors	If the device driver encounters a write failure, it make 2 more write attempts before reporting a write error. The SP maintains a count of consecutive write errors for each track which is cleared by a successful write to the appropriate track. If this count exceeds 20, the SP returns WFS_ERR_HARDWARE_ERROR and continues to do so until the condition is cleared by operator intervention via VDM.
Blank Track Error	The SP maintains a count of consecutive blank tracks for all tracks which is cleared by a successful read to the appropriate track. If this count exceeds 24, the SP returns WFS_ERR_HARDWARE_ERROR and continues to do so until the condition is cleared by operator intervention via VDM.
Passbook Jam during accept	When the driver detects a passbook stuck in the throat of the printer which cannot be pulled in, it makes 7 attempts to move the passbook into the printer. If these attempts fail, the SP assumes that customer tampering has occurred and enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.

Text Terminal Unit

Service Provider Components

Device	DLL Name	SP Executable
Enhanced Operator Panel	ttu_spx.dll ttu_wfp.dll ttu_ipc.dll ttu.dll	ttu.exe

Default Logical Service Names

Logical Name	Description
TextTerminalUnit1	The logical name of the TTU service provider

Configurable Parameters

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS\TTU key.

Parameter	Description	Permissible Values
GENERAL_CONFIGS\Variant	Device variant.	EOP - Enhanced Operator Panel
GENERAL_CONFIGS\Keyclick	Enables/Disables the keyclick.	0- Disable keyclick 1- Enables keyclick
GENERAL_CONFIGS\Charset	Selects the character set.	SET1 - Selects set 1 SET2 - Selects set 2 SET3 - Selects set 3 SET4 - Selects set 4 SET5 - Selects set 5
GENERAL_CONFIGS\Keymap	Keymap file.	Any valid file, structured as described in note 1 below.
GENERAL_CONFIGS\FormTrace	Parser trace file.	Any valid path/filename.
GENERAL_CONFIGS\Forms Dir	The path to the form file definitions.	Any valid absolute path.

NOTE:

The keymap file defines the mapping of keys to keycodes and has the following structure:

Key Position	Key code	Enable/Disable
<0-24>	<0-FF>	<0 or 1>

A line with a semicolon (';') in the first column is treated as a comment.

Capabilities

Capability	Value
Type	FIXED
Resolutions	32x16
No. of LEDs	3
Supported Keys	NUMERIC + HEXADECIMAL

Text Terminal Unit

Capability	Value
Keyboard Lock	FALSE
Display Lights	FALSE
Cursor	FALSE
Forms	TRUE
Beep	TRUE
Keyboard Buffer size	33

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_TTU_STATUS	2	l *fwDevice* of WFS_TTU_DEVPOWEROFF, WFS_TTUDEVNODEVICE and WFS_TTU_DEVUSERERROR are never returned. l *wKeyboard* is always WFS_TTU_KBDON.
WFS_INF_TTU_CAPABILITIES	2	l The LEDs are mapped as follows: *wLEDs*[0] = Supervisor LED *wLEDs*[1] = In-service LED *wLEDs*[2] = Error LED
WFS_INF_TTU_FORM_LIST	2	If no forms are defined, *lpszFormList* will point to an empty double null character terminated string (i.e. "\0")
WFS_INF_TTU_QUERY_FORM	2	None
WFS_INF_TTU_QUERY_FIELD	2	None
WFS_CMD_TTU_BEEP	1	l Different types of beeps are not-supported. All beep types default to a 540 Hz, 100ms beep l WFS_TTU_BEEPCONTINUOUS is actioned as a periodic beep with a periodicity of 200 ms and a duty cycle of 50%.
WFS_CMD_TTU_CLEARSCREEN	2	None
WFS_CMD_TTU_DISPLIGHT	2	The command always returns WFS_SUCCESS but is not actioned.
WFS_CMD_TTU_SET_LED	1	Blinking LEDs are not supported. Hence, WFS_SET_LEDSLOWFLASH, WFS_SET_LEDMEDIUMFLASH and WFS_SET_LEDQUICKFLASH are actioned as WFS_SET_LEDCONTINUOUS.
WFS_CMD_TTU_SET_RESOLUTION	2	None
WFS_CMD_TTU_DISPLAY_FORM	2	None
WFS_CMD_TTU_READ_FORM	2	l If no fields are read, *lpszFields* will point to an empty double null character terminated string. (i.e. "\0"). l If the CANCEL key is pressed, the returned string will consist of the fields already read. l If a timeout occurs or the command is cancelled via WFSCancelAsyncRequest, the returned string is NULL.

WOSA Command	CL	Comments
WFS_CMD_TTU_WRITE	1	l WFS_TTU_POSRELATIVE is not supported. l Attributes WFS_TTU_TEXTUNDERLINED, WFS_TTU_TEXTINVERTED and WFS_TTU_TEXTFLASH are accepted but actioned as normal text.
WFS_CMD_TTU_READ	1	l WFS_TTU_POSRELATIVE is not supported. l Attributes WFS_TTU_TEXTUNDERLINED, WFS_TTU_TEXTINVERTED and WFS_TTU_TEXTFLASH are accepted and actioned as normal text. l WFS_TTU_KEYALPHANUMERIC is not supported. l If the CANCEL key is pressed, the returned string will consist of the characters already read. l If a timeout occurs or the command is cancelled via WFSCancelAsyncRequest, the returned string is NULL.

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_INF_TTU_QUERY_FORM	WFS_ERR_TTU_FORMNOTFOUND	2	None
	WFS_ERR_TTU_FORMINVALID	2	Returned under the following conditions: l A required keyword is found to be missing in the form definition during startup. l No fields are found for the form during startup. l The dimensions of the form exceed the current resolution of the screen.
WFS_INF_TTU_QUERY_FIELD	WFS_ERR_TTU_FORMNOTFOUND	2	None
	WFS_ERR_TTU_FORMINVALID	2	Returned under the following conditions: l A “required” keyword is found to be missing in the form definition during startup. l No fields are found for the form during startup. l The dimensions of the form exceed the current resolution of the screen. l A request is made to return details of all the fields, and at least one of them is invalid as described above.
	WFS_ERR_TTU_FIELDNOTFOUND	2	None

continued...

WOSA Command	Error Codes	CL	Comments
WFS_INF_TTU_QUERY_FIELD	WFS_ERR_TTU_FIELDINVALID	2	Returned under the following conditions: • A required keyword is found to be missing in the field definition during startup. • A field height greater than 1 is specified in the form definition. • An invalid value is specified for one or more of the keywords. • TYPE, CLASS, KEYS, ACCESS, OVERFLOW, STYLE and HORIZONTAL. • Displaying the field at the position specified in the field definition causes the field to overflow the screen.
WFS_CMD_TTU_BEEP	WFS_ERR_INVALIDDATA	0	Returned under the following conditions: • BEEP_ON and BEEP_OFF are requested together. • An invalid beep type is specified.
WFS_CMD_TTU_CLEARSCREEN	WFS_ERR_INVALIDDATA	0	Returned when the rectangle to be cleared is not within the current screen resolution.
WFS_CMD_TTU_DISPLIGHT	WFS_SUCCESS	2	Always returned but not actioned.
WFS_CMD_TTU_SET_LED	WFS_ERR_TTU_INVALIDLED	2	Returned when *wLed* is not between 0 and 2 (both inclusive).
	WFS_ERR_INVALID_DATA	0	Returned when *fwCommand* is invalid.
WFS_CMD_TTU_SET_RESOLUTION	WFS_ERR_TTU_RESNOTSUPP	2	None

WOSA Command	Error Codes	CL	Comments
WFS_CMD_TTU_DISPLAY_FORM	WFS_ERR_TTU_FORMNOTFOUND	2	None
	WFS_ERR_TTU_FORMINVALID	2	Returned under the following conditions: l A required keyword is found to be missing in the form definition during startup. l No fields are found for the form during startup. l The dimensions of the form exceed the current resolution of the screen.
	WFS_ERR_TTU_MEDIAOVERFLOW	2	Returned when displaying the provided input would causes the field to overflow the form.
	WFS_ERR_TTU_FIELDSPECFAILURE	2	Returned under the following conditions: l A field is specified more than once in the input. l Input has not been provided for a REQUIRED field. l A field specified in the input does not belong to the form. l A field specified in the input is found to be invalid. (See comment for WFS_ERR_TTU_FIELDINVALID in WFS_INF_QUERY_FIELD)
	WFS_ERR_TTU_FIELDERROR	2	Returned when the provided input exceeds the field width and an action of TERMINATE has been specified for OVERFLOW in the form definition.
	WFS_ERR_INVALID_DATA	0	Returned under the following conditions: l An invalid pointer is specified for *lpszFormName* or *lpszFields*. l An invalid value is provided for *bClearScreen*.

WOSA Command	Error Codes	CL	Comments
WFS_CMD_TTU_READ_FORM	WFS_ERR_TTU_FORMNOTFOUND	2	None
	WFS_ERR_TTU_FORMINVALID	2	Returned under the following conditions: l A required keyword is found to be missing in the form definition during startup. l No fields are found for the form during startup. l The dimensions of the form exceed the current resolution of the screen
	WFS_ERR_TTU_FIELDSPECFAILURE	2	Returned under the following conditions: l A field is specified more than once in the input. l A required field is not specified in the input. l A field supplied in the input is not part of the form. l A field specified in the input was found to be invalid. (See comment for WFS_ERR_TTU_FIELDINVALID in WFS_INF_QUERY_FIELD) l A field supplied in the input overflows the screen.
	WFS_ERR_TTU_KEYCANCELED	2	Returned when the READ_FORM command is terminated by pressing the CANCEL key. The returned string contains the characters entered before the CANCEL key was pressed.

WOSA Command	Error Codes	CL	Comments
WFS_CMD_TTU_WRITE	WFS_ERR_INVALIDDATA	3	Returned under the following conditions: l WFS_TTU_POSRELATIVE is specified for *fwMode*. l An invalid location is specified for the write. l The supplied string overflows the screen. l An invalid attribute is specified for the text. l A buffer of length greater than 2000 bytes is specified for *lpsText*.
WFS_CMD_TTU_READ	WFS_ERR_TTU_KEYCANCELED	2	The keys entered upto the time the CANCEL key is pressed will be returned in *lpszInput*.
	WFS_ERR_INVALIDDATA	0	Returned under the following conditions l WFS_TTU_POSRELATIVE is specified for *fwMode*. l A field overflows the screen. l An invalid value is specified for *fwEchoMode*. l An invalid value is specified for *fwKeys*.

Forms Interpretation

The implementation of forms is based on the following basic assumptions about fields and their attributes:

- **ACCESS** of a field is its primary attribute. Other attributes are validated with respect to **ACCESS**.
- **CLASS** is an attribute of **WRITE** only fields and has no relevance while reading a form.
- **TYPE** and **KEYS** are attributes of **READ** only fields and have no relevance while displaying a form.
- All other field attributes apply to both **READ** and **WRITE** fields.
- A read form command is usually preceded by a display form command. If an application provides a value for a read-write field in a display form command, the SP 'remembers' this value. If no keys are entered for that field in the following read form, it returns that value. If a key is pressed, the entered key/s are superimposed on the 'remembered' value, and returned to the application. For example, if a value of '100,000.00' was specified for a read-write field in a display form, and then you enter '452' in the read form following the display form, the string '452,000.00' will be returned to the application at the end of the read form.
- If there is no 'remembered' initial value when a read form command is issued, then the INITIALVALUE, (if one is specified in the form definition), is used instead. If neither is available, the field is blank at the start of the read-form.
- If an initial value is supplied but the format (an echo image as described in Reference 25) is not, a default echo image consisting of *n* asterisks will be used, where *n* equals the number of characters in the INITIALVALUE string.
- If a format is provided without an initial value, then the field will be blank at the start of the read operation.
- Initial values for fields of TYPE PASSWORD are ignored.

Forms Validation

Forms validation is performed in two stages; during start-up and at runtime.

Start-up validation:

When the SP starts up, all form definition files are opened and verified for syntax, 'required fields' and content. Here, syntax refers to valid keywords, non-duplication of keywords, and matching BEGIN/END. Required fields are form **SIZE**, form **LANGUAGE**, field **SIZE** and field **POSITION**, and content is the value of the keywords, which must be within the capabilities of the device. The form definitions are converted to an 'in-core' representation called the 'Forms Database'. Errors found during this initial validation are recorded in the Forms Database and logged in an error log file 'ttu_form.log'. The location of this error log file is specified in the registry.

Runtime validation:

Runtime validation is performed when QUERY_FORM, QUERY_FIELD, DISPLAY_FORM or READ_FORM are requested. It verifies that the form **SIZE** and field **POSITION** values contained in the definition do not conflict with the capabilities of the device. For example, either of the co-ordinates of a field position exceeding the resolution of the device. Runtime validation is required because the resolution of the TTU may be modified at any time using the WFS_TTU_SET_RESOLUTION command, which could render a form that is valid for one resolution, invalid for another.

Runtime validation also verifies the following:

- All REQUIRED fields have been specified in the input.
- No value is supplied for **STATIC** fields. (See note 3 below).
- **WRITE** only fields are not specified in the READ_FORM command.
- **READ** only fields are not specified in the DISPLAY_FORM command.

The ACCESS attribute of a field is interpreted as follows:

WRITE	Writing relates to displaying data on the screen. Thus, **WRITE** fields are those which the application can update on the display.
READ	Reading relates to reading from the keyboard. Thus, **READ** fields are those which the application can read from the keyboard.
READ/WRITE	These fields have a combination of the properties of **READ** and **WRITE** fields. The fields would typically be used to display an initial value and then read data from that field.

The CLASS of a field is interpreted as follows:

OPTIONAL	These are fields for which the application need not supply values. If a value is supplied, the field is processed, else it is ignored.
STATIC	These are fields which the application cannot manipulate. The values of these fields are contained in the INITIALVALUE of the field. The application MUST NOT provide values for such fields. If it does, the value is ignored. No error is generated as the specification does not provide an appropriate error code
REQUIRED	These are fields for which the application MUST provide values.

Form Parser Error Messages and Limits

This section describes the limits of the TTU forms parser as well as the error messages that may be generated by the forms parser. These error messages are logged in the forms trace file, whose location is specified in the registry.

Form Parser Limits

The following are limits of the Form Parser:

Maximum line length (including line continuation)	500 characters
Maximum token (terminated by white space characters) length	100 characters
Maximum quoted (in "") string length	100 characters
Maximum filename length	100 characters
Maximum number of components in an OR list	10
Maximum Length of a component in an OR list	20

Form Parser Messages

The following table lists the errors/warning messages generated by the form parser. These errors get logged in the forms tracefile whose name and location are specified in the registry.

Error Message	**Description**
Error:Form Definition file not found - <form filename>	The specified form definition could not be found.
Error:Line *n*:Internal Error. Unknown field type	An internal error has occurred which prevents further processing.
Error:Line *n*:Internal Error <error no.>	An internal memory allocation error has occurred which prevents further processing.
Warning:Line *n*:Unexpected keyword <keyword>	An unexpected keyword <keyword> has been encountered. A keyword is unexpected if it appears out of place in the form definition.
Warning:Form <form name>:Required Keyword <keyword> Missing	Required keyword <keyword> is not specified in the form definition.
Warning:Form <form name>:Multiple occurrences of keyword <keyword>	The keyword <keyword> has appeared multiple times in the form definition. The last occurrence overwrites all previous occurrences.
Warning:Form <form name>:No Fields	No fields have been defined for <form name>.
Warning:Form <form name>:Field <field name>: Required Keyword <keyword> Missing	Required keyword <keyword> is not specified in the definition of field <field name>.
Warning:Form <form name>:Field <field name>: Multiple Occurrences of Keyword <keyword>	The keyword <keyword> has appeared multiple times in the definition of field <field name>.
Warning:Line *n*:Invalid TYPE <string>	An invalid value <string> is specified for the attribute TYPE.
Warning:Line *n*:Invalid CLASS <string>	An invalid value <string> is specified for the attribute CLASS.
Warning:Line *n*:Invalid KEYS <string>	An invalid value <string> is specified for the attribute KEYS.
Warning:Line *n*:Invalid ACCESS <string>	An invalid value <string> is specified for the attribute ACCESS.
Warning:Line *n*:Invalid OVERFLOW <string>	An invalid value <string> is specified for the attribute OVERFLOW.
Warning:Line *n*:Invalid STYLE <string>	An invalid value <string> is specified for the attribute STYLE.
Warning:Line *n*:Invalid HORIZONTAL <string>	An invalid value <string> is specified for the attribute HORIZONTAL.
Warning:Line *n*:Line too long	The line is too long to be processed.

Error Message	Description
Warning:Line *n*:Invalid keyword <string>	An invalid keyword <string> is encountered. A keyword is invalid if it is not one of the keywords defined in the form 'Definition Syntax' described in Reference 7.
Warning:Line *n*:Invalid integer value	An invalid string is specified for an integer value. The string is invalid if it is of 0 length or greater than the maximum length of a token specified in the previous section, or if it contains non-numeric characters.
Warning:Line *n*:Invalid quoted string	An invalid string is specified for a quoted string. The string is invalid if it is of 0 length or greater than the maximum length of a token specified in the previous section, or if it is not enclosed in double quotes ("").
Warning:Line *n*:Invalid token	A token in the input is invalid. A token is invalid if its length is 0 or greater than the maximum length of a token specified in the previous section.
Warning:Line *n*:Invalid OR list	An invalid OR combination of strings is specified. An OR list is invalid if the number of component strings is greater than the maximum, or the length of any one component string is greater than the maximum specified in the previous section.

Default Keyboard Mapping

At startup, the Operator keyboard keys are mapped according to the key map file specified in the registry. Keys not specified in the key map file will be mapped as follows:

Key Position	EOP	
	Code	Description
01	31H	'1'
02	32H	'2'
03	33H	'3'
04	F2H	CANCEL
05	34H	'4'
06	35H	'5'
07	36H	'6'
08	FFH	CLEAR
09	37H	'7'
10	38H	'8'
11	39H	'9'
12	F1H	ENTER
13	E0H	'00'
14	30H	'0'
15	2EH	'.'
16	FEH	RUBOUT
17	41H	'A'
18	42H	'B'
19	43H	'C'
20	44H	'D'
21	45H	'E'
22	46H	'F'
23	NA	UNUSED
24	NA	UNUSED

Application Guidelines

1 The OVERWRITE overflow action is not supported for any type of field.

2 Only the NORMAL attribute is supported. UNDER, INVERTED and FLASHING are accepted, but treated as NORMAL.

3 CENTER alignment is supported only for WRITE fields.

4 The Format string is relevant only to READ and READWRITE fields. 'Echo Images' are as defined in the Echo commands section of Reference 25.

5 The Initial value is relevant only to WRITE and READWRITE fields.

Vendor Dependent Mode

Service Provider Components

Device	DLL Name	SP Executable
None	vdm_spx.dll vdm_wfp.dll vdm_ipc.dll vdm.dll	vdm.exe

Default Logical Service Names

Logical Name	Description
VendorDependentMode1	The logical name of the Vendor Dependent Mode Service Provider

Configurable Parameters

The VDM Service Provider has no configurable parameters.

Capabilities

The VDM Device Class has no defined capabilities.

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_VDM_STATUS	1	*wDevice* that specifies the status of Vendor Dependent Mode service class will always be WFS_VDM_DEVONLINE.
WFS_INF_VDM_CAPABILITIES	2	None
WFS_CMD_VDM_ENTER_MODE_REQ	2	None
WFS_CMD_VDM_ENTER_MODE_ACK	2	None
WFS_CMD_VDM_EXIT_MODE_REQ	2	None
WFS_CMD_VDM_EXIT_MODE_ACK	2	None

Conformance Matrix - Events

WOSA Event	CL	Comments
WFS_SRVE_VDM_ENTER_MODE_REQ	2	None
WFS_SRVE_VDM_EXIT_MODE_REQ	2	None
WFS_SYSE_VDM_MODEENTERED	2	None
WFS_SYSE_VDM_MODEEXITED	2	None

Application Guidelines

1. When a VDM Entry/Exit request is issued, the SP posts a service event to all the registered applications, in response to which, all registered applications are expected to send in an acknowledgment. The SP waits indefinitely for the acknowledgments. When in this state, the VDM will ignore all Entry/Exit requests. It is therefore mandatory for all registered applications to acknowledge.

2. The VDM SP also expects an acknowledgment from the application that issued the VDM Entry/Exit request.

3. The VDM SP does not interact with the hardware of the SST. It does not sense supervisor switch changes, nor does it provide any additional functionality apart from that specified in reference 8.

4. The VDM SP should be used together with a Vendor Dependent Application (VDA) which provides the functionality for the vendor dependent mode. This functionality may include sensing the supervisor switch and invoking Ulysses's System Application. The structure of a VDA is depicted as follows:

```
#include <xfsapi.h>
#include <xfsvdm.h>

HSERVICE hVDMSp;// hService of the VDM SP
WinMain()
{
      . . . .
      . . . .

      WFSStartUp(.., ..);

      WFSOpen("VendorDependentMode1" , . . . ., &hVDMSp );

      WFSRegister(hVDMSp, SERVICE_EVENTS|SYSTEM_EVENTS);

      while(GetMessage(. ., . ., 0, 0))
      {
           TranslateMessage(. . .);
           DispatchMessage(. . .);
      }
}

long PASCAL VDA_WndProc(HWND hWnd,UINT uiMsg,WPARAM wParam,LPARAM lParam)
{
      LPWFSRESULT lpWFSResult=(LPWFSRESULT)lParam;

      switch (uiMsg){
           case WFS_SERVICE_EVENT:
                switch(lpWFSResult->u.dwEventID)
                {
                     case WFS_SRVE_VDM_EXIT_MODE_REQ:
                          WFSExecute(hVDMSp, WFS_CMD_VDM_EXIT_MODE_ACK, . . .);
                          break;
                     case WFS_SRVE_VDM_ENTER_MODE_REQ:
                          WFSExecute(hVDMSp, WFS_CMD_VDM_ENTER_MODE_ACK, . . .);
                          break;
                }
                break;
           case WFS_SYSTEM_EVENT:
                switch( lpWFSResult->u.dwEventID )
                {
                     case WFS_SYSE_VDM_MODEENTERED:
                          // Invoke Diagnostic Program
                          break;
                          case WFS_SYSE_VDM_MODEEXITED:
                          // Cleanup
                          break;
                }
                break;
           case WM_DESTROY:
                CloseVDMService(hVDMSp, . . .);
                lRetVal = WFSCleanUp();
                PostQuitMessage(0);
                break;
            default:
                return DefWindowProc(hWnd,uiMsg,wParam,lParam);
      }
      return (0L);
}
```

Pinpad and Key Library

Pinpad

Service Provider Components

Devices	DLL Name(s)	SP Executable
Basic Alpha Pinpad Encryptor (BAPE) Encryptor Keyboard Controller (EKC)	pin_spx.dll pin_wfp.dll pin_ipc.dll pin.dll	pin.exe

Default Logical Service Name

Logical Name	Description
Pinpad1	The logical name of PIN Keypad Service Provider

Configurable Parameters

The following configurable parameters, stored in the registry under the WFS_CFG_HKEY_XFS_ROOT\SERVICE_PROVIDERS\PIN key, are used by the PIN Keypad Service Provider and the Key Library.

Parameter	Description	Permissible Values
GENERAL_CONFIGS\Variant	Variant of the encryptor device.	01 for BAPE variant, and 05 for EKC variant.
GENERAL_CONFIGS\NumOfInitialKeys	Number of initial keys that should be imported before a WOSA application comes up.	In the range 1 to 100, both inclusive. If NumOfInitialKeys is 0, then no initial/master key is required to be loaded before starting up the application.
GENERAL_CONFIGS\KeyID1	Temporary encryption key-id 1.	In the range 1 to 100, both inclusive. It should be different from keyid2, ivid1 and ivid2.
GENERAL_CONFIGS\KeyID2	Temporary encryption key-id 2.	In the range 1 to 100, both inclusive. It should be different from keyid1, ivid1 and ivid2.
GENERAL_CONFIGS\IVID1	Temporary initialization vectorid1.	In the range 1 to 100, both inclusive. It should be different from keyid1, keyid2 and ivid2.
GENERAL_CONFIGS\IVID2	Temporary initialization vector id 2.	In the range 1 to 100, both inclusive. It should be different from keyid1, keyid2 and ivid1.

The following parameters are used only by the PIN Keypad Service Provider software and are stored in the registry under the WFS_CFG_HKEY_XFS_ROOT/SERVICE_PROVIDERS/PIN key

Parameter	Description	Permissible Values
KEYBOARD\Variant	Variant of the cardholder keyboard (CKM) device.	Should be either 21, 22, 23, 41, 42, 43, 61, 62, 63, A1, A2, A3, C1, C2, C3, E1, E2 or E3.
KEYBOARD\FKEnter	Key position code for the ENTER function key.	In the range 0 to 64. Value 0 indicates that the ENTER function key is not supported for the CKM variant.
KEYBOARD\FKEnterCC	Control code for ENTER function key.	In the range 0 to 255.
KEYBOARD\FKCancel	Key position code for the CANCEL function key.	In the range 0 to 64. Value 0 indicates that the CANCEL function key is not supported for the CKM variant.
KEYBOARD\FKCancelCC	Control code for CANCEL function key.	In the range 0 to 255.
KEYBOARD\FKClear	The key position code for the CLEAR function key.	In the range 0 to 64. Value 0 indicates that the CLEAR function key is not supported for the CKM variant.
KEYBOARD\FKClearCC	Control code for CLEAR function key.	In the range 0 to 255.
KEYBOARD\FKBackspace	Key position code for the BACKSPACE function key.	In the range 0 to 64. Value 0 indicates that the BACKSPACE function key is not supported for the CKM variant.
KEYBOARD\FKBackspaceCC	Control code for BACKSPACE function key.	In the range 0 to 255.
KEYBOARD\FKHelp	Key position code for the HELP function key.	In the range 0 to 64. Value 0 indicates that the HELP function key is not supported for the CKM variant.
KEYBOARD\FKHelpCC	Control code for HELP function key.	In the range 0 to 255.
KEYBOARD\FKDecPoint	Key position code for the DECPOINT function key.	In the range 0 to 64. Value 0 indicates that the DECPOINT function key is not supported for the CKM variant.
KEYBOARD\FKDecPointCC	Control code for DECPOINT function key.	In the range 0 to 255.
KEYBOARD\FK00	Key position code for the 00 function key.	In the range 0 to 64. Value 0 indicates that the 00 function key is not supported for the CKM variant.
KEYBOARD\FK00CC	Control code for 00 function key.	In the range 0 to 255.

Parameter	Description	Permissible Values
KEYBOARD\FK000	Key position code for the 000 function key.	In the range 0 to 64. Value 0 indicates that the 000 function key is not supported for the CKM variant.
KEYBOARD\FK000CC	Control code for 000 function key.	In the range 0 to 255.
KEYBOARD\FDK01CC-08CC	Control code for FDK01-08 key.	In the range 0 to 255.
KEYBOARD\Activator01-16	Control code for ACTIVATOR_01-16	In the range 0 to 255. Value 0 indicates it as unsupported.

The following configurable parameters are used only by the Key Library, and are stored in the registry under the WFS_CFG_HKEY_XFS_ROOT/SERVICE_PROVIDERS/PIN/KEYLIB key.

Parameter	Description	Permissible Value
Repository	Name and location of the Key Repository.	Any absolute path+filename.

Capabilities

Capability	Value (EKC/BAPE)
fwType	TYPEEPP | TYPEEDM
bCompound	FALSE
usKeyNum	100 for BAPE encryptor, and 300 for EKC encryptor
fwAlgorithms	CRYPTDESECB | CRYPTDESCBC | CRYPTDESMAC
fwPinFormats	FORM3624 | FORMANSI | FORMISO0 | FORMISO1 | FORMECI2 | FORMECI3 | FORMVISA
fwDerivationAlgorithms	0
fwPresentationAlgorithms	0
fwDisplay	DISPNONE
bIDConnect	FALSE

Capability	Value (EKC/BAPE)
fwType	TYPEEPP | TYPEEDM
fwIDKey	IDKEYIMPORT
fwValidationAlgorithms	DES | VISA

Conformance Matrix - Commands

WOSA Command	CL	Comments

WFS_INF_PIN_STATUS	1	• *fwDevice* of WFS_PIN_DEVPOWEROFF and WFS_PIN_DEVNODEVICE are never returned. WFS_PIN_DEVUSERERROR will be returned if the key repository configuration is incorrect or an error occurs while retrieving the encryptor status from the key repository.
		• *fwEncStat* can assume the following values:
		  ◦ WFS_PIN_ENCUNDEFINED - if the key repository configuration is incorrect or has an error while getting the encryptor status from the key repository.
		  ◦ WFS_PIN_ENCREADY - if the encryptor is initialized, and at least one working key is imported in the encryptor.
		  ◦ WFS_PIN_ENCINITIALIZED - if the encryptor is initialized, and no working key is imported in the encryptor.
		  ◦ WFS_PIN_ENCNOTINITIALIZED - if the encryptor is not initialized.
		  ◦ WFS_PIN_ENCNOTREADY - if the encryptor is not initialized, and at least one key is imported in the encryptor.

WFS_INF_PIN_CAPABILITIES	2	None

WFS_INF_PIN_KEY_DETAIL	1	The input parameter *lpsKeyName* will be interpreted as follows:
		  ◦ First the key repository will be searched for the encryption key as specified by the *lpsKeyName*.
		  ◦ If the encryption key *lpsKeyName* is not present in the key repository, then the key repository will be searched for the key-couple as specified by the *lpsKeyName*.
		  ◦ If the encryption key and key-couple, both could not be found in the key repository, then WFS_ERR_PIN_KEYNOTFOUND error is returned.
		  ◦ In case of a key-couple, two encryption key details (three WFSPINKEYDETAIL pointers) are returned in output parameters, each one giving encryption key details for keys forming the key-couple.

WFS_INF_PIN_FUNCKEY_DETAIL	2	None

WOSA Command	CL	Comments
WFS_CMD_PIN_CRYPT	1	• The input parameters *lpsKey* and *lpsStartValueKey* should specify the encryption key name as registered with the key repository. • The input parameter *lpxKeyEncKey* (*lpxStartValue*), if used, should have *lpxKeyEncKey->usLength* as ENCRYPTION_KEY_LENGTH (INITIALIZATION_VECTOR_LENGTH) and *lpxKeyEncKey->lpbData* specifying the actual value. • The input parameter *lpxCryptData* should not be NULL. • For MACing, *lpxCryptData->usLength* should not be 0 or compressed data length should not be 0. • For decryption, *lpxCryptData->usLength* should be multiple of 8 bytes.
WFS_CMD_PIN_IMPORT_KEY	2	• The input parameter *lpsKey* should specify the encryption key name or initialization vector name as registered with the key repository. • The input parameter *lpsEncKey* should specify the encryption key name as registered with the key repository. • The input parameter *lpxIdent* can be used to pass a secret keyspace password when importing a key for the EKC encryptor. In this case, *lpxIdent->usLength* should be set to PASSWORD_LENGTH and *lpxIdent->lpbValue* should specify the value of password in hex. • The input parameter *lpxValue->usLength* should be set to ENCRYPTION_KEY_LENGTH (or INITIALIZATION_VECTOR_LENGTH) and *lpxValue->lpbData* should specify the actual key value. • The value of input parameter *fwUse* should match that stored in the key repository for the encryption key being imported.
WFS_CMD_PIN_DERIVE_KEY	0	None

WFS_CMD_PIN_GET_PIN	1	Input parameters *usMinLen*, *bAutoEnd* and *cEcho* are ignored.
		• The following values will be returned for the output parameter *wCompletion* even though they are defined only for the execute event:
		  ◦ WFS_PIN_COMPCLEAR
		  ◦ WFS_PIN_COMPBACKSPACE
		• If any unused key is pressed, or any unknown key code is received, then the execute event will show *ulDigit* parameter as 0x00 and the command will be completed as if it is cancelled.

WOSA Command	CL	Comments
WFS_CMD_PIN_LOCAL_DES	1	• Input parameter *lpsValidationData*, if non-NULL, should contain characters having values in the range 0x30-0x39 and 0x41-0x46. • Input parameter *lpsOffset*, if non-NULL, should contain characters having values in the range 0x30-0x39. • Input parameter *bPadding* should have a value in the range 0x30-0x39. Otherwise 0x30 will be used as the default value, with no error code returned. • Input parameter *usMaxPIN* should be in the range MIN_CHECK_LENGTH and MAX_CHECK_LENGTH. • Input parameter *usValDigits* should be in the range MIN_LEFT_SHIFT_VALUE and MAX_LEFT_SHIFT_VALUE. • Input parameter *lpsKey* should specify the encryption key name as registered with the key repository. • Input parameter *lpxKeyEncKey*, if used, should have *lpxKeyEncKey->usLength* as ENCRYPTION_KEY_LENGTH and *lpxKeyEncKey->lpbData* specifying the actual value.
WFS_CMD_PIN_CREATE_OFFSET	0	None
WFS_CMD_PIN_LOCAL_EUROCHEQUE	0	None
WFS_CMD_PIN_LOCAL_VISA	1	• Input parameter *lpsPAN*, should contain characters having values in the range 0x30-0x39 and 0x41-0x46. • Input parameter *lpsPVV*, if non-NULL, should contain characters having values in the range 0x30-0x39. • Input parameter *wPVVDigits* should be in the range MIN_LEFT_SHIFT_VALUE and MAX_LEFT_SHIFT_VALUE. • Input parameter *lpsKey* should specify a key-couple name if *lpxKeyEncKey* input is not being used. • Input parameter *lpxKeyEncKey*, if used, should have *lpxKeyEncKey->usLength* as 2 times ENCRYPTION_KEY_LENGTH and *lpxKeyEncKey->lpbData* specifying the actual value.
WFS_CMD_PIN_PRESENT_IDC	0	None

WOSA Command	CL	Comments
WFS_CMD_PIN_GET_PINBLOCK	1	• Input parameter *lpsCustomerData*, if used, should contain characters having values in the range 0x30-0x39 and 0x41-0x46. • Input parameter *lpsXORData* is ignored. • Input parameter *bPadding* will be used only for formats WFS_PIN_FORM3624, WFS_PIN_FORMISO1 and WFS_FORM_ECI2. It should be in the range 0x30-0x39 and 0x41-0x46. • Input parameter *lpsKey* cannot be NULL and should specify key name as registered with the key repository. • Input parameter *lpsKeyEncKey*, if used, should specify key name as registered with the key repository.

WFS_CMD_PIN_GET_DATA	1	• Output parameter *lpsData* will only contain numeric digits; and decimal point(s), if any.
		• The following values will also be returned for the output parameter *wCompletion* even though they are defined only for execute event:
		  ◦ WFS_PIN_COMPCLEAR
		  ◦ WFS_PIN_COMPBACKSPACE
		• If any unused key is pressed, or any unknown key code is received, then the execute event will show *ulDigit* parameter as 0x00 and the command will be completed as if it is cancelled.

WFS_CMD_PIN_INITIALIZATION	1	The input pointer should be NULL.

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_INF_PIN_STATUS	None	0	None
WFS_INF_PIN_CAPABILITIES	None		None
WFS_INF_PIN_KEY_DETAIL	WFS_ERR_PIN_KEYNOTFOUND	1	Returned under the following conditions: l Key name is invalid. l Encryption key is not registered with the key repository and key-couple with name specified is also not registered with the key repository.
	WFS_ERR_PIN_ACCESSDENIED	3	Returned under the following conditions: l Key repository is empty. l Severe error while accessing the key repository.
WFS_INF_PIN_FUNCKEY_DETAIL	None	0	None
WFS_CMD_PIN_CRYPT	WFS_ERR_PIN_KEYNOTFOUND	2	Returned under the following conditions: l *lpsKey* is not registered with the key repository. l *lpsStartValueKey* is not registered with the key repository.
	WFS_ERR_PIN_KEYNOVALUE	2	Returned under the following conditions: l *lpsKey* is not imported in encryptor; as indicated by key repository information for the key. l *lpsStartValueKey* is not imported in encryptor; as indicated by key repository information for the key.
	WFS_ERR_PIN_USEVIOLATION	2	Returned under the following conditions: l *lpsKey* does not have proper type of access for encryption in the key repository. l *lpsStartValueKey* does not have proper type of access in the key repository.

continued....

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PIN_CRYPT	WFS_ERR_PIN_MODENOTSUPPORTED	2	Returned under the following conditions: l Invalid encryption mode is specified in *wMode* input parameter. l Invalid encryption algorithm is specified in *wAlgorithm* input parameter.
	WFS_ERR_PIN_ACCESSDENIED	2	Returned under the following conditions: l Key repository is empty. l Encryptor is non-initialized. l Severe error while accessing the key repository.
	WFS_ERR_PIN_INVALIDKEYLENGTH	2	Returned under the following conditions: l *lpxKeyEncKey* input pointer is non-NULL and *lpxKeyEncKey->usLength* is not equal to ENCRYPTION_KEY_LENGTH. l *lpxKeyEncKey* input pointer is non-NULL and *lpxKeyEncKey->lpbData* is NULL. l *lpxStartValue* input pointer is non-NULL and *lpxStartValue->usLength* is not equal to INITIALIZATION_VECTOR_LENGTH. l *lpxStartValue* input pointer is non-NULL and *lpxStartValue->lpbData* is NULL.
WFS_CMD_PIN_IMPORT_KEY	WFS_ERR_PIN_KEYNOTFOUND	2	None
	WFS_ERR_PIN_KEYNOVALUE	2	None
	WFS_ERR_PIN_USEVIOLATION	2	None
	WFS_ERR_PIN_INVALIDID	2	Returned under the following conditions: l *lpxIdent* input pointer is non-NULL and *lpxIdent->usLength* is not equal to PASSWORD_LENGTH. l *lpxIdent* input pointer is non-NULL and *lpxIdent->lpbData* is NULL pointer.

continued...

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PIN_IMPORT_KEY	WFS_ERR_PIN_DUPLICATEKEY	2	The key has already been imported; cannot be re-imported.
	WFS_ERR_PIN_INVALIDKEYLENGTH	2	Returned under the following conditions: l *lpxValue->usLength* is not equal to ENCRYPTION_KEY_LENGTH. l *lpxValue->lpbData* is NULL pointer.
	WFS_ERR_PIN_ACCESSDENIED	3	Returned under the following conditions: l Key repository is empty. l Encryptor is non-initialized. l Severe error while accessing the key repository.
WFS_CMD_PIN_GET_PIN	WFS_ERR_PIN_KEYINVALID	0	This error code will not be generated.
	WFS_ERR_PIN_KEYNOTSUPPORTED	2	Returned under the following conditions: l *ulActiveKeys* input specifies a key that is not supported. l *ulTerminateFDKs* input specifies a key that is not supported.
	WFS_ERR_PIN_NOACTIVEKEYS	2	*ulActiveKeys* input is equal to 0.
	WFS_ERR_PIN_NOTERMINATEKEYS	0	This error code will not be generated.
	WFS_ERR_PIN_MINIMUMLENGTH	0	This error code will not be generated.
WFS_CMD_PIN_LOCAL_DES	WFS_ERR_PIN_KEYNOTFOUND	2	*lpsKey* is not registered with the key repository.
	WFS_ERR_PIN_KEYNOVALUE	2	*lpsKey* is not imported in encryptor; as indicated by key repository information for the key.
	WFS_ERR_PIN_USEVIOLATION	2	*lpsKey* does not have proper type of access in the key repository.
	WFS_ERR_PIN_ACCESSDENIED	2	Returned under the following conditions: l Key repository is empty. l Encryptor is non-initialized. l Severe error while accessing the key repository.

continued

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PIN_LOCAL_DES	WFS_ERR_PIN_NOPIN	2	Returned under the following conditions: l The PIN buffer is empty; no previous issue of WFS_CMD_PIN_GET_PIN command. l The PIN buffer is cleared before this command.
	WFS_ERR_PIN_INVALIDKEYLENGTH	3	Returned under the following conditions: l *lpxKeyEncKey->usLength* is not equal to ENCRYPTION_KEY_LENGTH. l *lpxKeyEncKey->lpbData* is NULL pointer.
WFS_CMD_PIN_LOCAL_VISA	WFS_ERR_PIN_KEYNOTFOUND	2	*lpsKey* is not registered with the key repository as encryption key or key-couple.
	WFS_ERR_PIN_KEYNOVALUE	2	Returned under the following conditions: l *lpsKey* is not imported in encryptor; as indicated by key repository information for the key. l At least one or both of the keys in key-couple *lpsKey* is not imported in encryptor; as indicated by key repository information for the key.
	WFS_ERR_PIN_USEVIOLATION	2	*lpsKey* does not have proper type of access in the key repository.
	WFS_ERR_PIN_ACCESSDENIED	2	Returned under the following conditions: l Key repository is empty. l Encryptor is non-initialized. l Severe error while accessing the key repository.
	WFS_ERR_PIN_NOPIN	2	Returned under the following conditions: l The PIN buffer is empty; no previous issue of WFS_CMD_PIN_GET_PIN command. l The PIN buffer is cleared before this command.

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PIN_GET_PINBLOCK	WFS_ERR_PIN_KEYNOTFOUND	2	Returned under the following conditions: l *lpsKey* is not registered with the key repository as encryption key. l *lpsKeyEncKey* is not registered with the key repository as encryption key.
	WFS_ERR_PIN_KEYNOVALUE	2	Returned under the following conditions: l *lpsKey* is not imported in encryptor; as indicated by key repository information for the key. l *lpsKeyEncKey* is not imported in encryptor; as indicated by key repository information for the key.
	WFS_ERR_PIN_USEVIOLATION	2	Returned under the following conditions: l *lpsKey* does not have proper type of access in the key repository. l *lpsKeyEncKey* does not have proper type of access in the key repository.
	WFS_ERR_PIN_MODENOTSUPPORTED	2	Returned under the following conditions: l *wFormat* input specifies unsupported format for PIN. l The entered PIN length is not valid for the format specified in *wFormat* input.
	WFS_ERR_PIN_ACCESSDENIED	2	Returned under the following conditions: l Key repository is empty. l Encryptor is non-initialized. l Severe error while accessing the key repository.
	WFS_ERR_PIN_NOPIN	2	Returned under the following conditions: l The PIN buffer is empty; no previous issue of WFS_CMD_PIN_GET_PIN command. l The PIN buffer is cleared before this command.

WOSA Command	Error Codes	CL	Comments
WFS_CMD_PIN_GET_DATA	WFS_ERR_PIN_KEYINVALID	0	This error code will not be generated.
	WFS_ERR_PIN_KEYNOTSUPPORTED	2	Returned under the following conditions: l *ulActiveFDKs* input specifies a key that is not supported. l *ulActiveKeys* input specifies a key that is not supported. l *ulTerminateFDKs* input specifies a key that is not supported. l *ulTerminateKeys* input specifies a key that is not supported.
	WFS_ERR_PIN_NOACTIVEKEYS	2	*ulActiveKeys* input is equal to 0.
WFS_CMD_PIN_INITIALIZATION	WFS_ERR_PIN_ACCESSDENIED	2	Returned under the following conditions: l Encryptor is non-initialized. l Severe error while accessing the key repository.
	WFS_ERR_PIN_INVALIDID	0	This error code will not be generated.

Conformance Matrix - Events

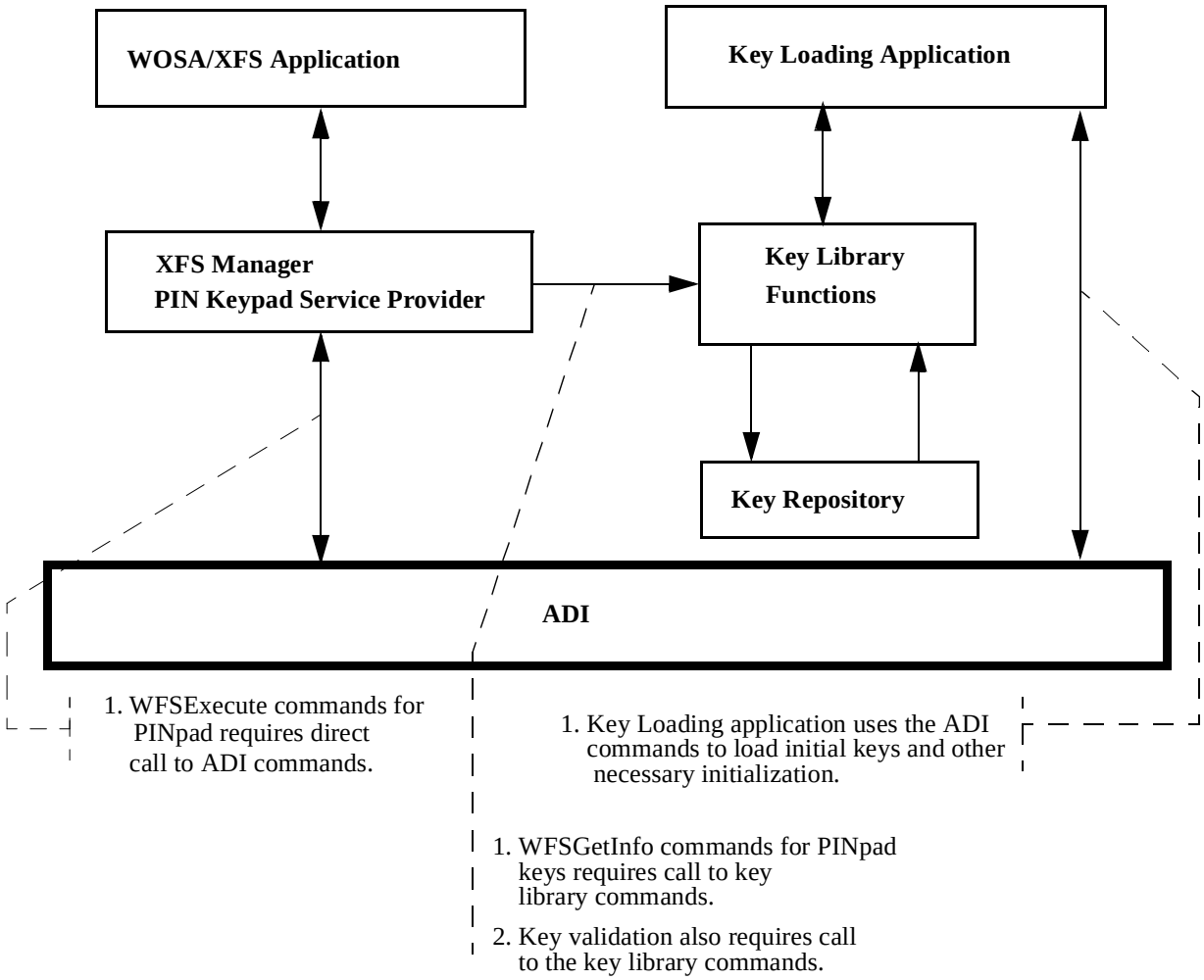
WOSA/XFS Events	CL	Comments
WFS_EXEE_PIN_KEY	2	None
WFS_SRVE_PIN_INITIALIZED	2	None
WFS_SRVE_PIN_ILLEGAL_KEY_ACCESS	2	None

Application Guidelines

1 Position and Key codes are stored in the registry and should be set according to the physical layout of the keyboard. Thus for example, if the CLEAR key is located on the top-right hand corner of the keyboard, FKClear should be set to 04 and FKClearCC to 255 (if default codes are being used). Position codes are documented in the "Control Codes And Keycode Parameters" section of the Programer's Manual for the Cardholder Keyboard Manager (Ref. 26). This document also discusses constraints on the assignment of keycodes.

Key Library and WOSA/XFS MVSS Application

The Key Library API maintains an encryption key repository, that stores all the required information concerning the hardware encryption keys for the WOSA/XFS MVSS system. The following diagram illustrates the location of the Key Library API within the WOSA/XFS MVSS system, a Key Loading application and ADI for NCR SSTs:



A WOSA/XFS MVSS application works on NCR SSTs, and conforms to the standards developed by the Banking Solutions Vendor Council. A Key Loading application is developed by the WOSA/XFS MVSS application developers for use with a WOSA/XFS MVSS application, to perform encryptor initialization at the customer site.

The Key Library API provides an interface for recording names, usage and location of encryption keys which are used/managed by the WOSA/XFS application. It also stores the encryptor status (initialized/not initialized). The WOSA/XFS application starts executing only after the encryptor is initialized.

Using the Key Library

Encryption keys are classified into two categories - keys that are loaded before any WOSA/XFS MVSS application starts on SST, called 'working' keys,

'initial or master' keys, and keys that are loaded/imported by the WOSA/XFS MVSS application during its execution, called 'working' keys.

It is the Key Loading application's task to load the initial keys before starting a WOSA/XFS MVSS application on an NCR SST. This task is performed when the SST is installed at the customer site and before the WOSA/XFS MVSS application is started on it. It can be repeated when the encryptor device is again required to be initialized for whatever reason.

The Key Library is used along with the WOSA/XFS MVSS application as follows:

- First, Key Library commands are used to create key repository records that identify the name, id, type of access of the encryption key and whether it is a master (initial) key. The names of all the initial keys and working keys are recorded in the key repository. The WOSA/XFS MVSS application knows well in advance, the names of keys that it will be using, and these are found in the key repository.
- Next, the Initial keys are loaded. You need to perform some additional vendor-specific tasks like creating a keyspace structure for the WOSA/XFS MVSS application, or entering a password for the secret secure keyspace, in order to be capable of loading the initial keys.
- After performing these initialization tasks, the key loading application uses a key library command to set the encryptor status to 'initialized'. This indicates that the encryptor is ready to run a WOSA/XFS MVSS application.
- The WOSA/XFS MVSS application can now be started.

A Key Loading application can be developed for each customer site that will automate the process of initial key loading. But, you may still have to intervene - for example, to enter a password.

Apart from encryption keys, the Key Library also maintains Starting Vectors or Initialization Vectors (IV). The IV has the same properties as that of the encryption keys, except that it is used only for MAC generation and CBC encryption.

Key Library Information

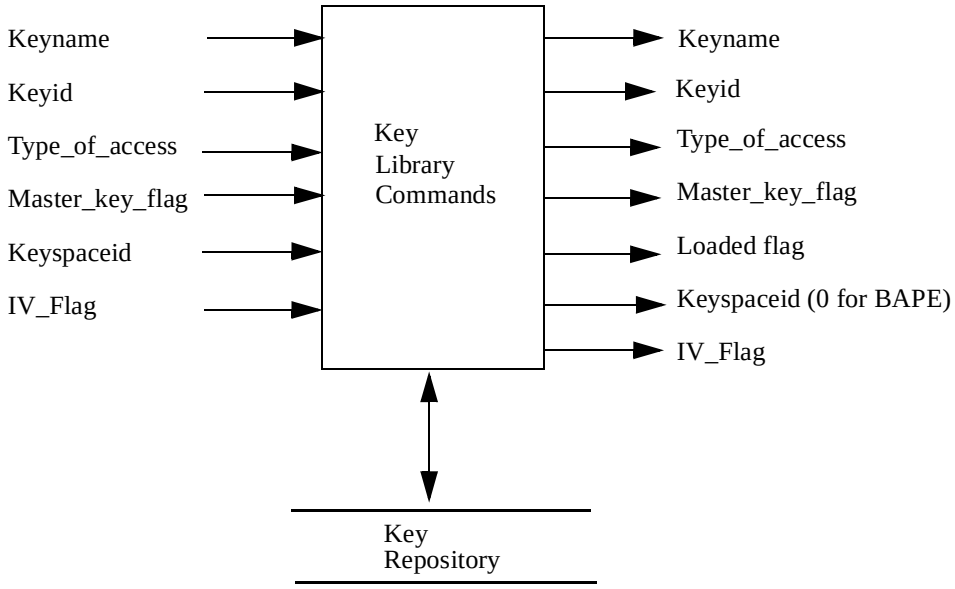
The Key Library stores all information concerning the encryption keys used by the WOSA/XFS MVSS system. It also keeps track of the encryptor status (initialized or not), and stores vendor-specific keyspace management information needed for each key.

From the WOSA/XFS MVSS point of view, the following information is associated with the encryption key:

- A keyname
- A keyid
- A keyspaceid
- Type of access information
- An Initialization Vector (IV) flag
- A master_key_flag

In addition to this, the 'loaded_flag' is also stored in the key repository:

The following diagram depicts the possible inputs/outputs to or from Key Library commands:



The application/s (Key Loading or WOSA/XFS MVSS) identifies a key by its ***Keyname***. The keyname is unique within the Key Library.

The ***Keyid*** information specifies the key identifier as required by the encryptor ADI commands. It is in the range of 1-100 for BAPE and 0-299 for EKC variant of the encryptor. The keyid is also unique within the Key Library.

The ***Type_of_access*** information identifies how the encryption key has been used in the WOSA/XFS MVSS application. It identifies the functions that are allowed to use this encryption key. It is a combination of the following values:

WFS_PIN_USECRYPT

WFS_PIN_USEFUNCTION

WFS_PIN_USEMACING

WFS_PIN_USEKEYENCKEY

WFS_PIN_USESVENCKEY

WFS_PIN_USENODUPLICATE.

Note that for EKC variant, the combination of WFS_PIN_USECRYPT and WFS_PIN_USEFUNCTION is not allowed.

The ***Master_flag*** identifies an initial key for the WOSA/XFS MVSS application. Initially, a master key is loaded in the encryptor. The encryptor status is then set to initialize and the WOSA/XFS MVSS application is started.

The ***Keyspaceid*** identifies the identifier of the keyspace to which the key belongs, in case of EKC encryptor. The keyspaceid is in the range 0-15 for EKC encryptor. It is 0 for BAPE encryptor.

The ***Loaded_flag*** information is maintained by the Key Library to indicate whether the key is currently loaded in the encryptor or not.

The ***IV_flag*** information identifies whether the key will act as IV for MAC generation or CBC encryption.

All the key repository information is available outside the Key Library as a result of an information command.

The value of an encryption key is not stored in the key repository. The actual value of a key is never available externally, even through Key Library commands. You have to maintain it through the Key Loading application, or any other application that uses the Key Library and allows loading/exchanging of the key in the encryptor. The application actually loads/exchanges/deletes encryption keys in the encryptor and updates the key repository information using the Key Library commands.

The key repository also stores maintenance information to check the integrity of the repository, and its *Sync* S*tatus*, that indicates whether or not the repository contents are in sync with the encryptor contents. The sync status is set by the application before using any ADI command that loads/deletes encryption keys in the encryptor. It is reset after the successful updating of the key repository. When sync status indicates that the repository is out of sync with the encryptor, the encryptor needs to be re-initialized.

The key repository also maintains information about *Key Couple*'. A key couple is a pair of encryption keys. Note that IVs cannot be specified as one of the keys of the key couple.

Key Library Configuration Information

The configuration information for the Key Library which may vary for different customer sites or different installations of NCR SSTs, is stored in a separate configuration repository.The name and location of this repository is stored under WOSA/XFS_ROOT\SERVICE_PROVIDERS\PIN\KEYLI\Repository.

In addition to the above, the Key Library shares parameters Variant, KeyID1, KeyID2, IVID1 & IVID2 described earlier in this chapter with the Pinpad Service Provider. Note that the configuration information for the Key Library is a subset of that of the PIN Keypad Device Service Provider. The configuration repository must be present in order to use Key Library commands.

CMD_KEYLIB_INITIALIZE

Purpose

This command initializes the Key Library for the calling application.

Synopsis

```
#include <key_lib.h>
int CMD_KEYLIB_INITIALIZE ( ) ;
```

Description

This command initializes the Key Library for the calling application. It reads configuration information from the registry, validates it and checks the integrity of the key repository. This function should be called before issuing any other Key Library function calls. It returns the encryptor status (i.e. whether the encryptor is initialized or not) or the initialization error code.

Parameters

None.

Return Value

INT iResult

The return value is either the error code for the Key Library initialization failure or the encryptor status, if the Key Library initialization is successful. The value TRUE (1) indicates that the encryptor is initialized and the value FALSE (0) indicates it to be non-initialized.

If the function returned is not TRUE (1) or FALSE (0), it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_KEYLIB_CONFIGURATION_ERROR

KL_ERR_ENCRYPTOR_VARIANT_CONFIGURATION_ERROR

KL_ERR_INVALID_ENCRYPTOR_VARIANT

KL_ERR_TEMPKEY_1_CONFIGURATION_ERROR

KL_ERR_TEMPKEY_2_CONFIGURATION_ERROR

KL_ERR_TEMPIV_1_CONFIGURATION_ERROR

KL_ERR_TEMPIV_2_CONFIGURATION_ERROR

KL_ERR_REPOSITORY_NAME_CONFIGURATION_ERROR

KL_ERR_REPOSITORY_OUT_OF_SYNC

KL_ERR_REPOSITORY_INTEGRITY_ERROR

KL_ERR_REPOSITORY_ACCESS_ERROR

KL_ERR_INTERNAL_ERROR

CMD_CREATE_KEY_NAME

Purpose

This command registers an encryption key with the specified attributes with the Key Library. (Adds an encryption key record in the key repository).

Synopsis

```
#include <key_lib.h>

int CMD_CREATE_KEY_NAME (LPSTR     lpszKeyName,
                         USHORT    usKeyId,
                         USHORT    usKeyspaceId,
                         WORD      fwTypeOfAccess,
                         BOOL      bIsIV,
                         BOOL      bMasterKeyFlag);
```

Description

This command registers an encryption key with the specified attributes with the Key Library. (Adds an encryption key record in the key repository). The key is not loaded into the encryptor. It is only made known to the Key Library.

Note:

For the encryption keys in BAPE, all combinations of type of access values are allowed, except WFS_PIN_USENODUPLICATE.

For encryption keys in EKC, all combinations of type of access values are allowed, except OR'ing of WFS_PIN_USECRYPT and WFS_PIN_USEFUNCTION.

For IVs, only combinations of WFS_PIN_USECRYPT, WFS_PIN_USEMACING and WFS_PIN_USENODUPLICATE are allowed.

The master key flag cannot be TRUE for IVs. That is, the master key flag and IV flag cannot be TRUE simultaneously.

If the master key flag is TRUE for an encryption key, the type of access should be WFS_PIN_USEKEYENCKEY and/or WFS_PIN_USESVENCKEY.

Parameters

LPSTR lpszKeyName

The name of the encryption key. A string of up to MAX_KEYNAME_LENGTH characters.

USHORT usKeyId

The keyid of the encryption key. It should be in the range 1-100 for BAPE and 0-299 for an EKC encryptor.

USHORT usKeyspaceId

The keyspaceid of the encryption key. It should be in the range 0-15 for EKC and 0 for a BAPE encryptor.

WORD fwTypeOfAccess

WOSA/XFS type of access for the encryption key specified as a combination of WFS_PIN_USECRYPT, WFS_PIN_USEFUNCTION, WFS_PIN_USEMACING, WFS_PIN_USEKEYENCKEY, WFS_PIN_USESVENCKEY and WFS_PIN_USENODUPLICATE. Note that for IVs, only combinations of WFS_PIN_USECRYPT, WFS_PIN_USEMACING and WFS_PIN_USENODUPLICATE are allowed.

BOOL bIsIV

The flag indicating whether this key is to be loaded as an IV in the encryptor or not. A TRUE value indicates that it should be IV.

BOOL bMasterKeyFlag

The flag indicating whether this key is a master key or not. A TRUE value indicates a master key and a FALSE value indicates that this key is not a master key.

Return Value

INT iSuccess

The return value indicating success of an encryption key registration. A zero value indicates success.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_ACCESS_ERROR

KL_ERR_INTERNAL_ERROR

KL_ERR_INVALID_KEY_NAME

KL_ERR_DUPLICATE_KEY_NAME

KL_ERR_DUPLICATE_KEY_ID

KL_ERR_INVALID_KEY_ID

KL_ERR_INVALID_TYPEOFACCESS

KL_ERR_INVALID_KEYSPACE_ID

KL_ERR_INVALID_MASTER_KEY_FLAG_FOR_IV

CMD_CREATE_KEY_COUPLE

Purpose

This command creates a key couple with the specified key names pair.

Synopsis

```
#include <key_lib.h>
int CMD_CREATE_KEY_COUPLE (LPSTR  lpszKeyCouple,
                           LPSTR  lpszKeyName1,
                           LPSTR  lpszKeyName2);
```

Description

This command creates a key couple with the specified key names pair. (Adds a key couple record in the key repository). The key names should already exist in the key repository and none of the two keys should be IVs. The same key-name can be specified for both the keys in the key couple.

Parameters

LPSTR lpszKeyCouple

The name of the key couple. A character string of up to MAX_KEYNAME_LENGTH characters.

LPSTR lpszKeyName1

The name of the already registered encryption key.

LPSTR lpszKeyName2

The name of the already registered encryption key.

Return Value

INT iSuccess

The return value indicating success of key couple registration. A zero value indicates success.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_EMPTY

KL_ERR_REPOSITORY_ACCESS_ERROR

KL_ERR_INTERNAL_ERROR

KL_ERR_INVALID_KEY_COUPLE_NAME

KL_ERR_INVALID_KEY_NAME_1

KL_ERR_INVALID_KEY_NAME_2

KL_ERR_DUPLICATE_KEY_COUPLE_NAME

KL_ERR_KEY_NAME_1_NOT_FOUND

KL_ERR_KEY_NAME_2_NOT_FOUND

KL_ERR_KEY_NAME_1_IS_IV

KL_ERR_KEY_NAME_2_IS_IV

CMD_DELETE_KEY_NAME

Purpose

This command de-registers the specified encryption key with the Key Library. (Deletes the key repository record for the key).

Synopsis

```
#include <key_lib.h>

int CMD_DELETE_KEY_NAME (LPSTR       lpszKeyName);
```

Description

This command de-registers the specified encryption key from the Key Library. The key should be deleted from the encryptor before de-registering it from the Key Library. The key couple(s), which use the key to be deleted, should also be deleted before deleting the encryption key.

Parameters

LPSTR lpszKeyName

The name of the encryption key. A character string of up to MAX_KEYNAME_LENGTH characters.

Return value

INT iSuccess

The return value indicating success of encryption key deletion. A zero value indicates success, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_EMPTY

KL_ERR_REPOSITORY_ACCESS_ERROR

KL_ERR_INVALID_KEY_NAME

KL_ERR_KEY_NOT_FOUND

KL_ERR_INVALID_KEY_STATUS

KL_ERR_KEY_NAME_IN_USE

CMD_DELETE_KEY_COUPLE

Purpose

This command de-registers the specified key couple from the Key Library. (Deletes the key repository record for the key couple).

Synopsis

```
#include <key_lib.h>

int CMD_DELETE_KEY_COUPLE (LPSTR lpszKeyCouple);
```

Description

This command de-registers the specified key couple from the Key Library.

Parameters

LPSTR lpszKeyCouple

The name of the key couple. A character string of up to MAX_KEYNAME_LENGTH characters.

Return Value

INT iSuccess

The return value indicating success of key couple deletion. A zero value indicates success, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_NO_KEY_COUPLE_RECORDS

KL_ERR_REPOSITORY_ACCESS_ERROR

KL_ERR_INVALID_KEY_COUPLE_NAME

KL_ERR_KEY_COUPLE_NOT_FOUND

CMD_SET_KEY_STATUS

Purpose

This command updates the Loaded_flag information for the specified encryption key in the key repository.

Synopsis

```
#include <key_lib.h>

int CMD_SET_KEY_STATUS (LPSTR       lpszKeyName,
                        BOOL        bLoadedFlag);
```

Description

This command updates the loaded_flag information for the specified encryption key in the key repository. This command unconditionally updates the key repository record (without checking the current status of the loaded_flag information for the key).

Parameters

LPSTR lpszKeyName

The name of the encryption key. A character string of up to MAX_KEYNAME_LENGTH characters.

BOOL bLoadedFlag

The flag indicating whether the loaded_flag information for the specified key is to be set or reset. TRUE indicates that the key is currently loaded in the encryptor and FALSE indicates that the key is deleted from the encryptor or that it can be overwritten.

Return Value

INT iSuccess

The return value indicating success of loaded_flag information update. A zero value indicates success, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_EMPTY

KL_ERR_REPOSITORY_ACCESS_ERROR

KL_ERR_INVALID_KEY_NAME

KL_ERR_KEY_NOT_FOUND

CMD_SET_ENCRYPTOR_STATUS

Purpose

This command sets the encryptor state to either initialized or non-initialized.

Synopsis

#include <key_lib.h>

int CMD_SET_SYNC_STATUS (BOOL bSyncStatus);

Description

This command sets the encryptor state to either initialized or non-initialized. It unconditionally updates the key repository (without checking the current encryptor status).

Parameters

BOOL bEncryptorStatus

The flag indicating whether the encryptor status is to be set to initialized or non-initialized. TRUE indicates that the encryptor status is to be set to initialized and FALSE indicates the encryptor status is to be set to non-initialized.

Return Value

INT iSuccess

The return value indicating success of encryptor status update. A zero value indicates success, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_EMPTY

KL_ERR_REPOSITORY_ACCESS_ERROR

CMD_SET_SYNC_STATUS

Purpose

This command sets the sync status to either 'in-sync' or 'out-of-sync'.

Synopsis

```
#include <key_lib.h>

int CMD_SET_ENCRYPTOR_STATUS (BOOL        bEncryptorStatus);
```

Description

This command sets the sync status to either 'in-sync' or 'out-of-sync'. It is used by the application to set the sync status to 'out-of-sync' before issuing an ADI command to load/delete keys into/from the encryptor. After the ADI command is complete and the corresponding key repository is successfully updated, this command should be used to set the sync status to 'in-sync'.

Parameters

BOOL bSyncStatus

The flag indicating whether the sync status is to be set to 'in-sync' or 'out-of-sync'. TRUE indicates that the sync status is to be set to 'in-sync' and FALSE indicates that the sync status is to be set to 'out-of-sync'.

Return Value

INT iSuccess

The return value indicating success of sync status update. A zero value indicates success, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_ACCESS_ERROR

INFO_GET_KEY_DETAIL

Purpose

This command returns key repository information for an encryption key.

Synopsis

```
#include <key_lib.h>

int INFO_GET_KEY_DETAIL   (LPSTR              lpszKeyName,
                           LPKLKEYDETAIL      lptKeyDetail);
```

Description

This command returns key repository information for an encryption key. It does not produce any interaction with the encryptor. It can be used by a Key Loading or WOSA/XFS MVSS applications to obtain key information.

Parameters

LPSTR lpszKeyName

The name of the encryption key. A character string of up to MAX_KEYNAME_LENGTH characters.

LPKLKEYDETAIL lptKeyDetail

The key detail structure KLKEYDETAIL that stores the key repository information defined as follows:

```
typedef struct   kl_key_detail
{
   LPSTR          lpszKeyName;
   USHORT         usKeyId;
   WORD           fwTypeOfAccess;
   BOOL           bMasterKey;
   BOOL           bLoadedFlag;
   USHORT         usKeyspaceId;
   BOOL           bIsIV;
} KLKEYDETAIL, *LPKLKEYDETAIL;
```

Note, the application should release the returned data using WFMFreeBuffer.

LPSTR lpszKeyName

The name of the encryption key. A character string of up to MAX_KEYNAME_LENGTH characters.

USHORT usKeyId

The keyid of the encryption key. It should be in the range 1-100 for BAPE and 0-299 for EKC.

WORD fwTypeOfAccess

The WOSA/XFS type of access for an encryption key specified as a combination of WFS_PIN_USECRYPT, WFS_PIN_USEFUNCTION, WFS_PIN_USEMACING, WFS_PIN_USEKEYENCKEY, WFS_PIN_USESVENCKEY and WFS_PIN_USENODUPLICATE.

BOOL bMasterKeyFlag

The flag indicating whether this key is a master key or not. TRUE indicates a master key and FALSE indicates that it is not a master key.

BOOL bLoaded

The flag indicating whether the key is currently loaded in the encryptor or not. TRUE indicates that the key is currently loaded in the encryptor and FALSE indicates that the key is not loaded in the encryptor.

USHORT usKeyspaceId

The keyspaceid to which the key belongs.

BOOL bIsIV

The flag indicating whether this key is IV or not. TRUE indicates that it is an IV.

Return Value

INT iSuccess

The return value indicating success of a key query. A zero value indicates that the key exists in the key library and that repository information is returned, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_EMPTY

KL_ERR_INVALID_KEY_NAME

KL_ERR_KEY_NOT_FOUND

INFO_GET_KEY_COUPLE_DETAIL

Purpose

This command returns the key repository information on a key couple.

Synopsis

```
#include <key_lib.h>

int INFO_GET_KEY_COUPLE_DETAIL  (LPSTR                  lpszKeyCouple,
                                 LPKLKEYCOUPLE          ptKeyCouple);
```

Description

This command returns the key repository information for a key couple.

Parameters

LPSTR lpszKeyCouple

The name of the key couple. A null terminated character string of up to MAX_KEYNAME_LENGTH characters.

LPKLKEYCOUPLE lptKeyCouple

A key couple structure KLKEYCOUPLE that stores the key repository information. It is defined as follows:

```
typedef struct    kl_key_couple

{
    LPSTR         lpszKeyCouple;
    LPSTR         lpszKeyName1;
    LPSTR         lpszKeyName2;
} KLKEYCOUPLE, *LPKLKEYCOUPLE;
```

Note, the application should release the returned data using WFMFreeBuffer.

LPSTR lpszKeyCouple

The name of the key couple.

LPSTR lpszKeyName1

The name of the encryption key 1 in the key pair.

LPSTR lpszKeyCouple

The name of the encryption key 2 in the key pair.

Return Value

INTi Success

The return value indicating success of a key couple query. A zero value indicates that the key couple exists in the key library and that repository information is returned, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_NO_KEY_COUPLE_RECORDS

KL_ERR_INVALID_KEY_COUPLE_NAME

KL_ERR_KEY_COUPLE_NOT_FOUND

INFO_GET_KEY_NAMES

Purpose

This command returns all the registered keynames in the key library.

Synopsis

#include <key_lib.h>

int INFO_GET_KEY_NAMES (LPPKEYNAMES *lppsKeyNames);

Description

This command returns all the registered keynames in the key library. It does not produce any interaction with the encryptor. It can be used by Key Loading and/or WOSA/XFS MVSS applications to retrieve all available keynames, and then use INFO_GET_KEY_DETAIL on each of them.

Parameters

LPPKEYNAMES lppsKeyNames

A pointer to a NULL terminated array of pointers to keynames that are registered with the Key Library. If iSuccess is zero and this pointer is NULL, then the key repository is empty. Note, the application should release the returned data using WFMFreeBuffer.

Return Value

INT iSuccess

The return value indicating success of keynames query. A zero value indicates that at least one key exists in the key library and keyname/s is/are returned, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_EMPTY

KL_ERR_INTERNAL_ERROR

INFO_GET_KEY_COUPLES

Purpose

This command returns all the registered key couples in the key library.

Synopsis

#include <key_lib.h>

int INFO_GET_KEY_COUPLES (LPPKEYCOUPLES *lppsKeyCouples);

Description

This command returns all the registered key couples in the key library.

Parameters

LPPKEYCOUPLES lppsKeyCouples

A pointer to a NULL terminated array of pointers to key couple names that are registered with the Key Library. If iSuccess is zero and this pointer is NULL, then the key repository contains no key couple records. Note, the application should release the returned data using WFMFreeBuffer.

Return Value

INT iSuccess

The return value indicating success of key couples query. A zero value indicates that at least one key exists in the key library and keyname/s is/are returned, otherwise an error code is returned.

If the function returned is not zero, it is one of the following error codes described in the Key Library Error Codes section:

KL_ERR_REPOSITORY_EMPTY

KL_ERR_INTERNAL_ERROR

INFO_GET_ENCRYPTOR_STATUS

Purpose

Determines the encryptor status.

Synopsis

```
#include <key_lib.h>

int INFO_GET_ENCRYPTOR_STATUS (BOOL    *bpEncryptorStatus);
```

Description

This command returns the encryptor status. It does not produce any interaction with the encryptor. It can be used by Key Loading and/or WOSA/XFS MVSS applications to determine the encryptor status.

Parameters

BYTE bEncryptorStatus

The flag indicating whether the encryptor is initialized or non-initialized. TRUE indicates that it is initialized and FALSE indicates that it is non-initialized.

Return Value

INT iSuccess

Always 0.

INFO_GET_SYNC_STATUS

Purpose

This command returns the sync status of the key repository.

Synopsis

#include <key_lib.h>

int INFO_GET_SYNC_STATUS (BOOL *bpSyncStatus);

Description

This command returns the sync status of the key repository as 'in-sync' or 'out-of-sync'. It is used by the Key Library DLL at initialization, to check that the repository and encryptor are synchronized. If the repository is found to be ‘out-of-sync’, the key library will fail to initialize.

Parameters

BYTE bSyncStatus

The flag indicating whether the sync status is set to 'in-sync' or 'out-of-sync'. TRUE indicates that the sync status is set to 'in-sync' and FALSE indicates that the sync status is set to 'out-of-sync'.

Return Value

INT iSuccess

Always 0.

Key Library and WOSA/XFS MVSS Encryption Keys

WOSA/XFS MVSS applications refer to the encryption keys using keynames. All the keynames that the WOSA/XFS application will use should be registered with the Key Library.

Working keys are loaded into the encryptor by the WOSA/XFS MVSS application using the WFS_CMD_PIN_IMPORT_KEY command. The key value is either specified in a clear text form or an encrypted form. In case of an encrypted key value, a key exchange key is specified to decrypt the encrypted key value, and the encryption key is loaded with the decrypted value. The key repository needs to be updated for this newly loaded key for which the CMD_SET_KEY_STATUS command is used.

The other WOSA/XFS MVSS commands for PIN Keypad refer to keys either by keyname alone, or by a combination of keyname and encrypted key value.

- When the reference is only by a keyname, the key should be already loaded/ imported in the encryptor; either initially before starting the WOSA/XFS MVSS application or within the application, using the WFS_CMD_PIN_IMPORT_KEY command. The Key Library command INFO_GET_KEY_DETAIL is used to obtain the key status and other required information.
- When the reference is by a combination of keyname and encrypted key value, the actual encryption key for executing the command is to be derived. The specified keyname refers to a key exchange key that should be already loaded/imported in the encryptor. There are two keys to be noted - the specified key exchange key and the new key that is to be loaded into the encryptor with the decrypted value. However, the keyname for this new key is not specified by the WOSA/XFS MVSS application. The PIN Keypad Service Provider and Key Library have to take care of such 'temporary keys' for the WOSA/XFS MVSS application.

Temporary Keys

A "temporary key" refers to the key (id) that is used by the PINpad Service Provider when the WOSA/XFS MVSS PIN Keypad command supplies an encrypted key value, and provides a name of the stored key to be used as a key exchange key. The PINpad Service Provider uses this key exchange key to decrypt the encrypted key value and load the decrypted value at the temporary keyid as specified in the configuration repository of the Key Library.

At least two temporary key (id)s are needed to be registered in the configuration repository of the Key Library for the WOSA/XFS MVSS application. The PIN Keypad Service Provider needs to know these temporary key (id)s in order to be able to execute the PINpad commands issued by the WOSA/XFS MVSS application.

A temporary key has the same type of access as that of the key exchange key, and is loaded in the same keyspace as that of the key exchange key.

The above discussion is also applicable to IVs.

Key Library Error Codes

All the Key Library functions return an integer error value on an error. The following are the error codes that are generated by the Key Library functions:

KL_ERR_KEYLIB_CONFIGURATION_ERROR

The configuration repository information is not available or is incorrect.

KL_ERR_ENCRYPTOR_VARIANT_CONFIGURATION_ERROR

The encryptor variant is not present in the configuration repository.

KL_ERR_INVALID_ENCRYPTOR_VARIANT

The encryptor variant in the configuration repository is invalid.

KL_ERR_TEMPKEY_1_CONFIGURATION_ERROR

The temporary key 1 id specified in the configuration repository is not available or is invalid.

KL_ERR_TEMPKEY_2_CONFIGURATION_ERROR

The temporary key 2 id specified in the configuration repository is not available or is invalid.

KL_ERR_TEMPIV_1_CONFIGURATION_ERROR

The temporary IV 1 id specified in the configuration repository is not available or is invalid.

KL_ERR_TEMPIV_2_CONFIGURATION_ERROR

The temporary IV 2 id specified in the configuration repository is not available or is invalid.

KL_ERR_REPOSITORY_NAME_CONFIGURATION_ERROR

The key repository file name is not available or is invalid.

KL_ERR_REPOSITORY_OUT_OF_SYNC

The repository is out of sync with the encryptor.

KL_ERR_REPOSITORY_INTEGRITY_ERROR

The number of encryptor key records in the key repository does not match with the file size.

KL_ERR_REPOSITORY_EMPTY

The key repository is empty.

KL_ERR_NO_KEY_COUPLE_RECORDS

The key repository does not contain any key couple information.

KL_ERR_REPOSITORY_COPY_ERROR

There is a file copy error when making a backup of the key repository.

KL_ERR_REPOSITORY_ACCESS_ERROR

There is an error in updating the key repository.

KL_ERR_INTERNAL_ERROR

There is an error when allocating memory or any other internal error.

KL_ERR_REPOSITORY_LOCKED

The key repository is locked or there is an error when locking the key repository.

KL_ERR_REPOSITORY_UNLOCK_ERROR

There is an error when unlocking the key repository.

KL_ERR_INVALID_KEY_NAME

The key-name is of invalid length or is NULL.

KL_ERR_INVALID_KEY_COUPLE_NAME

The key couple name is of invalid length or is NULL.

KL_ERR_INVALID_KEY_NAME_1

The key-name 1 in key couple pair is of invalid length or is NULL.

KL_ERR_INVALID_KEY_NAME_2

The key-name 2 in key couple pair is of invalid length or is NULL.

KL_ERR_DUPLICATE_KEY_NAME

There is an encryptor key record with the same key-name.

KL_ERR_DUPLICATE_KEY_COUPLE_NAME

The key couple name already exists in the key repository.

KL_ERR_INVALID_KEY_ID

The key-id is out of range or is invalid.

KL_ERR_DUPLICATE_KEY_ID

There is an encryptor key record with the same key-id.

KL_ERR_INVALID_KEYSPACE_ID

The keyspace-id is out of range or is invalid.

KL_ERR_INVALID_TYPEOFACCESS

The type of access is invalid for the key.

KL_ERR_KEY_NOT_FOUND

The encryptor key record is not found with the specified key-name.

KL_ERR_KEY_COUPLE_NOT_FOUND

The key couple record is not found with the specified key couple name.

KL_ERR_KEY_NAME_1_NOT_FOUND

The encryptor key record is not found with the specified key-name for the key couple.

KL_ERR_KEY_NAME_2_NOT_FOUND

The encryptor key record is not found with the specified key-name for the key couple.

KL_ERR_INVALID_KEY_STATUS

The key status (whether it is loaded in encryptor or not) is invalid for update.

KL_ERR_KEY_NAME_IN_USE

The encryptor key record is in use (in key couple). You cannot delete it.

KL_ERR_KEY_NAME_1_IS_IV

The key name 1 in key couple is IV and not an encryption key.

KL_ERR_KEY_NAME_2_IS_IV

The key name 2 in key couple is IV and not an encryption key.

KL_ERR_INVALID_MASTER_KEY_FLAG_FOR_IV

The master key flag is TRUE for IV which is not allowed.

Sensors and Indicators Unit

Service Provider Components

Device	DLL Name	SP Executable
Non TCM based TCM based	siu_spx.dll siu_wfp.dll siu_ipc.dll siu.dll	siu.exe

Default Logical Service Names

Logical Name	Description
SIU	The logical name of the SIU service provider

Configurable Parameters

The following configurable parameters are stored in the registry under the WOSA/XFS_ROOT\SERVICE_PROVIDERS\SIU\GENERAL_CONFIGS\CONFIG.

Parameter	Description	Permissible Values
ALARMS\COMPOSITE\Conf	Specifies whether or not the composite sensor is installed and configured.	TRUE FALSE
ALARMS\SAFE\Conf	Specifies whether or not the safe sensor is installed and configured.	TRUE FALSE
ALARMS\SILENT\Conf	Specifies whether or not the silent sensor is installed and configured.	TRUE FALSE
ALARMS\TAMPER\Conf	Specifies whether or not the tamper sensor is installed and configured.	TRUE FALSE
INDICATORS\IN_SERVICE\Available	Specifies whether or not the In-Service indicator is installed.	TRUE FALSE
INDICATORS\FASCIA_LIGHT\Available	Specifies whether or not the Fascia Light indicator is installed.	TRUE FALSE
INDICATORS\FASCIA_LIGHT\Linked	Specifies whether or not the Fascia Light indicator is linked with the In-Service indicator.	TRUE FALSE

Capabilities

Capability	Value
Operator Switch	TRUE
Tamper Sensor	TRUE*
Silent Alarm Sensor	TRUE*
Composite Sensor	TRUE*
Proximity Sensor	FALSE
Ambient Light Sensor	FALSE
Cabinet Door Sensor	FALSE
Safe Door Sensor	TRUE*
Vandal Shield Sensor	FALSE
Open/Close Indicator	TRUE*
Fascia Light Indicator	TRUE*
Audio Indicator	TRUE
Heating Indicator	FALSE
Volume Control	FALSE
UPS Control	FALSE
Card Unit Guidance Light	TRUE
PIN Pad Unit Guidance Light	FALSE
Notes Dispenser Unit Guidance Light	FALSE
Coin Dispenser Unit Guidance Light	FALSE
Receipt Printer Unit Guidance Light	FALSE
Passbook Printer Unit Guidance Light	TRUE
Envelope Depository Guidance Light	TRUE
Envelope Dispenser Guidance Light	FALSE
Cheque Unit Guidance Light	FALSE
Bill Acceptor Unit Guidance Light	FALSE

NOTE: Those capabilities marked with asterisks are dependent on the registry settings.

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_SIU_STATUS	2	• *fwDevice* of WFS_SIU_DEVPOWEROFF, WFS_SIU_DEVNODEVICE, WFS_SIU_DEVHWERROR and WFS_SIU_DEVUSERERROR are never returned. • *fwDoors* of WFS_SIU_CLOSED and WFS_SIU_LOCKED are never reported. • The condition WFS_SIU_NOT_AVAILABLE indicates that the port whose status is requested is not supported or does not exist.
WFS_INF_SIU_CAPABILITIES	2	None
WFS_CMD_SIU_ENABLE_EVENTS	2	None
WFS_CMD_SIU_SET_PORTS	1	The following sensors/indicators are not supported: Cabinet Doors, Safe Doors, Vandal Shield, Heating device, Volume Control and Guidance Lights on the PIN pad unit, Notes Dispenser unit, Coin Dispenser unit, Receipt Printer unit, Cheque unit, Bill Acceptor unit and Envelope Dispenser unit.
WFS_CMD_SIU_SET_DOOR	0	None
WFS_CMD_SIU_SET_INDICATOR	1	The Heating Indicator is not supported.
WFS_CMD-SIU_SET_AUXILLARY	0	None
WFS_CMD_SIU_SET_GUIDLIGHT	1	The following indicators are not supported: PIN pad unit, Notes Dispenser unit, Coin Dispenser unit, Receipt Printer unit, Cheque unit, Bill Acceptor unit and Envelope Dispenser unit.

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_CMD_SIU_ENABLE_EVENTS	WFS_ERR_SIU_INVALID_PORT	1	Returned when the requested port: l Does not exist and is not supported by the service provider. l Does not exist but is supported by the service provider. l Exists but is not supported by the service provider.
	WFS_ERR_SIU_SYNTAX	2	None
WFS_CMD_SIU_SET_PORTS	WFS_ERR_SIU_INVALID_PORT	1	Returned when the requested port: l Does not exist and is not supported by the service provider. l Does not exist but is supported by the service provider. l Exists but is not supported by the service provider.
	WFS_ERR_SIU_SYNTAX	2	None
	WFS_ERR_SIU_PORT_ERROR	3	Returned when an attempt to set the beep sequence fails.

WOSA Command	Error Codes	CL	Comments
WFS_CMD_SIU_SET_INDICATOR	WFS_ERR_SIU_INVALID_PORT	1	Returned when the requested port: • Does not exist and is not supported by the service provider. • Does not exist but is supported by the service provider. • Exists but is not supported by the service provider.
	WFS_ERR_SIU_SYNTAX	2	None
	WFS_ERR_SIU_PORT_ERROR	1	Returned when an attempt to set the beep sequence fails.
WFS_CMD_SIU_SET_GUIDLIGHT	WFS_ERR_SIU_INVALID_PORT	1	Returned when the requested port: • Does not exist and is not supported by the service provider. • Exists but is not supported by the service provider.
	WFS_ERR_SIU_SYNTAX	2	None

Conformance Matrix - Events

WOSA Event	CL	Comments
WFS_SRVE_SIU_PORT_STATUS	2	Generated only for the Card Unit, Passbook Printer and Depository guidance lights, the Safe Sensor, Composite Sensor, Tamper Sensor, Silent Alarm Sensor, Audio Indicator, Fascia Light Indicator, Open/Close Indicator and the Operator (Mode) Switch.
WFS_EXEE_SIU_PORT_ERROR	2	l Generated by errors on the following ports: Guidance Lights (Card Unit, Depository, Passbook Printer) Audio Indicator Operator Switch Fascia Light Open/Close Indicator l Never generated by the WFS_CMD_SIU_ENABLE_EVENTS command.

Application Guidelines

1 The WFS_CMD_SIU_ENABLE_EVENTS command enables the application to register for change-of-state events from various sensors and indicators by setting the corresponding flags in the input. If the flag for even a single sensor/indicator that can be supported is set in the input, the command is considered as valid. The command is considered as unsuitable for execution, only if none of the set sensors/indicators is supported. The same is true for the WFS_CMD_SIU_SET_PORTS command.

2 When the WFS_CMD_SIU_SET_GUIDLIGHT and WFS_CMD_SIU_SET_PORTS commands are issued to set the blinking speed of the guidance lights, they affect ALL the guidance lights that are on. This command cannot set the speed of an individual guidance light.

3 The WFS_CMD_SIU_SET_DOOR and WFS_CMD_SIU_SET_AUXILIARY commands are unsupported.

Identity Card Unit

Service Provider Components

Device	DLL Name	SP Executable
IDC	idc_spx.dll idc_wfp.dll idc_ipc.dll idc.dll	idc.exe

Default Logical Service Names

Logical Name	Description
IDCardUnit1	The logical name of the IDC service provider.

Configurable Parameters

The following configurable parameters are stored in the registry under the WOSA/XFS ROOT\SERVICE_PROVIDERS\IDC key.

Parameter	Description	Permissible Values
GENERAL_CONFIGS\Variant	Variant of the device.	Can be one of the following: l MCRW - Motorised Magnetic Card Reader/Writer. l SCRW - Motorised Magnetic Card Reader/Writer with a Smart Card Reader/Writer Unit attached. l DIP - DIP type Magnetic Card Reader. l SWIPE - SWIPE type Magnetic Card Reader.
GENERAL_CONFIGS\MaxRetainedCards	Maximum number of cards that can be accommodated by the Capture bin. Relevant only to MCRW and SCRW.	Device dependent parameter. Defaults to 10 if not specified in the registry.
GENERAL_CONFIGS\HighRetainedCards	High capture count threshold. Relevant only to MCRW and SCRW.	l A number less than or equal to MaxRetainedCards. Defaults to 7 if not specified in the registry.

Parameter	Description	Permissible Values
GENERAL_CONFIGS\PowerOnFlag	Determines the action to be performed when a card is detected in the device at powerup.	Can be set to one of the following values: l NOACTION: If Power On options not supported. This will be the case for non-Motorised units. l EJECT: If Card present, it will be ejected on Power On. l RETAIN: If Card present, it will be captured on Power On. l EJECTTHENRETAIN: If Card present, it will be ejected on Power On. If the Card is not taken within a specified time, it is captured. l READPOSITION: If the Card is present, it will be moved on Power On so that it is in a position to be read. If not specified in the registry, defaults to NOACTION.
GENERAL_CONFIGS\PowerOffFlag	Determines the action to be performed when a card is detected in the device at powerdown.	Can be set to one of the following: l NOACTION: If Power Off options are not supported, this will be the case for the non-motorised units. l EJECT: If Card present, it will be ejected on Power Off. l RETAIN: If Card present, it will be captured on Power Off. l EJECTTHENRETAIN: If Card present, it will be ejected on Power Off. If the card is not taken within a specified time, it is captured. l READPOSITION: If Card present, it will be moved on Power Off so that it is in a position to be read. If not specified in the registry, defaults to NOACTION.

Parameter	Description	Permissible Values
GENERAL_CONFIGS\ReadTracks	Tracks the device can read	l A combination of strings TRACK1, TRACK2 & TRACK3. The separator can be '+', '|' or the space character. l If no track can be read, this should be set to NO_TRACK. l If this value is not specified in the registry, a default of TRACK2 is assumed.
GENERAL_CONFIGS\WriteTracks	Tracks the device can write	l A combination of strings TRACK1, TRACK2 & TRACK3. The separator can be '+', '|' or the space character. l If no track can be written, as in the case of the DIP & SWIPE variants, this should be set to NO_TRACK. l If this value is not specified in the registry, a default of NO_TRACK is assumed.
GENERAL_CONFIGS\SecurityType	Security module type	Should be set to SEC_NOT_SUPP as the current implementation does not support the security module.
GENERAL_CONFIGS\SuspendTimeout	Time in seconds for which the device should be suspended when customer tampering is suspected.	If not specified, a default of 200 seconds is assumed.
GENERAL_CONFIGS\FormsDir	Directory under which IDC form definition files are stored.	Any valid directory name.
GENERAL_CONFIGS\ChipProtocol	Supported Chip protocols	l PROT_NOT_SUPP if the device cannot handle Chip Cards. (This will be the case if variant is not SCRW). l A space or '+' or '|' separated combination of strings of the form T<XX> where XXranges from 00 to 15. (e.g. "T00" or "T00+T01+T02"). If not specified, a default value of PROT_NOT_SUPP is assumed

Capabilities

Capability	Value (MCRW)	Value (SCRW)	Value (DIP/SWIPE)
Track Read Capability	TRUE	TRUE	TRUE
Track Write Capability	H/W Dependant	H/W Dependant	FALSE
Chip I/O	FALSE	TRUE	FALSE
Power On options	H/W Dependant	H/W Dependant	FALSE
Power Off options	H/W Dependant	H/W Dependant	FALSE
Security Module	FALSE	FALSE	FALSE

Conformance Matrix - Commands

WOSA Command	CL	Comments
WFS_INF_IDC_STATUS	1	• *fwDevice* of WFS_IDC_DEVPOWEROFF and WFS_IDC_DEVNODEVICE are never returned. • *fwSecurity* is always WFS_IDC_SECNOTSUPP even if a security module is present. • If the device has been suspended and this command is issued, the returned status will be that which existed at the start of the suspend operation and will also be the case if the card is staged in the smart card unit.
WFS_INF_IDC_CAPABILITIES	2	• *fwType* of WFS_IDC_TYPE_CONTACTLESS is never returned. • *fwSecType* is always WFS_IDC_SECNOTSUPP even if a security module is present.
WFS_INF_IDC_FORM_LIST	2	None
WFS_INF_IDC_QUERY_FORM	2	None
WFS_CMD_IDC_READ_TRACK	2	• The command can result in the card being captured if a card jam occurs or the card being ejected if the firmware suspects customer tampering. • WFS_ERR_HARDWARE_ERROR may be returned if the card is staged in the Smart Card section of an SCRW when the command is given. • WFS_ERR_DEV_NOT_READY may be returned if the device enters a state which could cause a system escape or when the SP is in a suspended state. If such an error is returned, and if the SP is not in a suspended state, the next command sent should be an EJECT_CARD or a RETAIN_CARD. • Should an error occur because the data on the track does not match the form definition, the output will contain as much field data as could be read.
WFS_CMD_IDC_WRITE_TRACK	2	• The command can result in the card being captured if a card jam occurs or the card being ejected if the firmware suspects customer tampering. • WFS_ERR_HARDWARE_ERROR may be returned if the card is staged in the Smart Card section of an SCRW when the command is given. • WFS_ERR_DEV_NOT_READY may be returned if the device enters a state which could cause a system escape or when the SP is in a suspended state. If such an error is returned, and if the SP is not in a suspended state, the next command sent should be an EJECT_CARD or a RETAIN_CARD.

WOSA Command	CL	Comments
WFS_CMD_IDC_EJECT_CARD	2	• WFS_ERR_HARDWARE_ERROR may be returned if the card is staged in the Smart Card section of an SCRW when the command is given. • In the event a card is jammed in the device when this command is issued, the card is captured by the SP, the Cards Retained count is incremented, and an error code of WFS_ERR_IDC_MEDIARETAINED is returned. Note however, that no events are generated if this causes the cards retained count to cross a threshold as the WOSA/XFS specification does not define any events for this command.
WFS_CMD_IDC_RETAIN_CARD	1	• WFS_ERR_HARDWARE_ERROR may be returned when a card is jammed inside the card reader which sends the device into a FATAL state. This condition must be cleared using the Ulysses System Application. • WFS_ERR_IDC_MEDIAJAMMED is returned when the SP enters a suspended state after it detects possible customer tampering.
WFS_CMD_IDC_RESET_COUNT	2	None
WFS_CMD_IDC_SET_KEY	0	Security module is not supported.

WOSA Command	CL	Comments
WFS_CMD_IDC_READ_RAW_DATA	2	l The command can result in the card being captured if a card jam occurs or the card being ejected if the firmware suspects customer tampering. l WFS_ERR_HARDWARE_ERROR may be returned if the card is staged in the Smart Card section of an SCRW when the command is given. l WFS_ERR_DEV_NOT_READY may be returned if the device enters a state which could cause a system escape or when the SP is in a suspended state. If such an error is returned, and if the SP is not in a suspended state, the next command sent should be an EJECT_CARD or a RETAIN_CARD.
WFS_CMD_IDC_WRITE_RAW_DATA	2	l The command can result in the card being captured if a card jam occurs or the card being ejected if the firmware suspects customer tampering. l WFS_ERR_HARDWARE_ERROR may be returned if the card is staged in the Smart Card section of an SCRW when the command is given. l WFS_ERR_DEV_NOT_READY may be returned if the device enters a state which could cause a system escape or when the SP is in a suspended state. If such an error is returned, and if the SP is not in a suspended state, the next command sent should be an EJECT_CARD or a RETAIN_CARD.
WFS_CMD_IDC_CHIP_IO	2	l A card must be present in the unit if this command is to succeed. l A READ_RAW_DATA command should have been issued in some previous command to get the ATR of the chip. If the ATR is not obtained correctly, this command will fail. l Communication with the chip takes place transparently. All the data passed as input for the CHIP_IO command is sent to the chip. All the data returned by the chip is returned as output data. No attempt is made to interpret this data.

Conformance Matrix - Errors

WOSA Command	Error Codes	CL	Comments
WFS_INF_IDC_QUERY_FORM	WFS_ERR_IDC_FORMNOTFOUND	2	None
	WFS_ERR_IDC_FORMINVALID	2	None
WFS_CMD_IDC_READ_TRACK	WFS_ERR_IDC_MEDIAJAM	2	None
	WFS_ERR_IDC_SHUTTERFAIL	2	None
	WFS_ERR_IDC_INVALIDDATA	2	None
	WFS_ERR_IDC_NOMEDIA	1	Returned under the following conditions: l On timeout. l If the command has caused a card to be accepted into the unit. But, if when an attempt is made to read it, the card is not present. l A card is expected to be present in the unit but is not detected at the start of this command. The next call to the command will however wait for a card to be inserted.
	WFS_ERR_IDC_INVALIDMEDIA	0	Never returned
	WFS_ERR_IDC_FORMNOTFOUND	2	None
	WFS_ERR_IDC_FORMINVALID	2	Returned when the data on the track does not conform to the form definition. Here, the benefit of doubt should be given to the form. (For example, a non-existent start/end delimiter).
	WFS_ERR_IDC_SECURITYFAIL	0	Never returned.
WFS_CMD_IDC_WRITE_TRACK	WFS_ERR_IDC_MEDIAJAM	2	None
	WFS_ERR_IDC_SHUTTERFAIL	2	None

WOSA Command	Error Codes	CL	Comments
	WFS_ERR_IDC_INVALIDDATA	2	Returned under the following conditions: l A track could not be read as written to, or if the existing data on the track is invalid. l Data on a card does not conform to the form definition.
	WFS_ERR_IDC_NOMEDIA	2	Returned under the following conditions: l The command has caused a card to be accepted into the unit. But, while attempting to read/write to it, the card is not present. l A card is expected to be present in the unit but is not detected at the start of this command. The next call to the command will wait for a card to be inserted.
	WFS_ERR_IDC_INVALIDMEDIA	2	Returned when the track exists but does not contain valid data when a read is performed to verify the written data.
	WFS_ERR_IDC_DATASYNTAX	2	None
	WFS_ERR_IDC_FORMNOTFOUND	2	None
	WFS_ERR_IDC_FORMINVALID	2	Returned under the following conditions: l The specified form was found to be invalid at startup. l The form specified is meant only for reading l Data specified in the input conflicts with the form definition.

WOSA Command	Error Codes	CL	Comments
WFS_CMD_IDC_EJECT_CARD	WFS_ERR_IDC_MEDIAJAM	2	None
	WFS_ERR_IDC_SHUTTERFAIL	0	Never returned
	WFS_ERR_IDC_MEDIARETAINED	2	None
	WFS_ERR_IDC_NOMEDIA	2	None
WFS_CMD_IDC_RETAIN_CARD	WFS_ERR_IDC_MEDIAJAM	1	This code is returned when a card is jammed in the throat of the card reader which indicates possible customer tampering. This causes the SP to go into a suspended state for SuspendTimeout seconds specified in the registry.
	WFS_ERR_IDC_NOMEDIA	2	None
	WFS_ERR_IDC_RETAINBINFULL	0	This code is never returned. Instead, when the hardware senses that the retain bin is full, WFSExecute returns and error code of WFS_ERR_HARDWAREERROR
	WFS_ERR_HARDWARE_ERROR	1	This code is returned when a card is jammed inside the card reader which causes the device to go into a fatal state. This condition must be cleared via the Ulysses System Application.
	WFS_ERR_DEV_NOT_READY	1	This code is returned when the SP is in a suspended state.
WFS_CMD_IDC_RESET_COUNT	WFS_ERR_INTERNAL_ERROR	2	An internal error occurred while processing this command.
WFS_CMD_IDC_SET_KEY	None	2	None
WFS_CMD_IDC_READ_RAW_DATA	WFS_ERR_IDC_MEDIAJAM	2	None
	WFS_ERR_IDC_SHUTTERFAIL	2	None

WOSA Command	Error Codes	CL	Comments
	WFS_ERR_IDC_NOMEDIA	2	Returned under the following conditions: l The command has caused a card to be accepted into the unit, but when an attempt is made to read it, the card is not present. l A card is expected to be present in the unit but is not detected at the start of this command. The next call to the command however, will wait for a card to be inserted.
WFS_CMD_IDC_READ_RAW_DATA	WFS_ERR_IDC_INVALIDMEDIA	2	In the case of a SMART card, this error is returned under the following conditions: l The Smart Card command format is invalid l The SP was unable to communicate with the Smart Card. l The Smart Card is powered off. l An invalid card is inserted.
WFS_CMD_IDC_WRITE_RAW_DATA	WFS_ERR_IDC_MEDIAJAM	2	None
	WFS_ERR_IDC_SHUTTERFAIL	2	None

WOSA Command	Error Codes	CL	Comments
	WFS_ERR_IDC_NOMEDIA	2	Returned under the following conditions: l When the command has caused a card to be accepted into the unit, but when an attempt is made to read it, the card is not present. l When a card is expected to be present in the unit but is not detected at the start of this command. The next call to the command however, will wait for a card to be inserted.
	WFS_ERR_IDC_INVALIDMEDIA	1	Returned under the following conditions: l The specified track is not supported. l The supplied data was found to be invalid during validation. l An error occurred while writing to the card l An error occurred while verifying the data written to the card.
WFS_CMD_IDC_CHIP_IO	WFS_ERR_IDC_MEDIAJAM	0	Never returned. If a jam occurs in a smart card related operation, it is treated as a hardware error.
	WFS_ERR_IDC_NOMEDIA	2	None
	WFS_ERR_IDC_INVALIDMEDIA	2	Returned when the chip has been powered-off by the firmware for some reason.

WOSA Command	Error Codes	CL	Comments
	WFS_ERR_IDC_INVALIDDATA	2	None
	WFS_ERR_IDC_PROTOCOLNOTSUPP	2	None
	WFS_ERR_IDC_ATRNOTOBTAINED	2	None

Conformance Matrix - Events

WOSA Event	CL	Comments
WFS_EXEE_IDC_INVALIDTRACKDATA	2	None
WFS_EXEE_IDC_MEDIAINSERTED	2	Generated after a card has been properly staged.
WFS_SRVE_IDC_MEDIAREMOVED	2	Generated when a card is taken by a user following a successful card eject. Also generated by Read/Write commands under the following conditions: l A card is expected to be in unit at the start of the command but is not. (This can happen if the card is reeled out after the previous command). l An invalid card has been ejected by the firmware and the user has taken it.
WFS_EXEE_IDC_INVALIDMEDIA	2	None
WFS_SRVE_IDC_CARDACTION	0	Not supported
WFS_USRE_IDC_RETAINBINTHRESHOLD	2	l WFS_IDC_RETAINBINOK is never posted. l An event with event data of WFS_IDC_RETAINBINHIGH is posted when the number of captured cards exceeds the value of HighRetainedCards specified in the registry. This event is posted each time a card is captured, until the number of captured cards reaches MaxRetainedCards, at which time the SP posts an event with event data of WFS_IDC_RETAINBINFULL. Beyond this point, the SP will continue to capture cards and post events until the hardware detects that the retain bin is full.

IDC Forms

The WOSA/XFS IDC class functionality supports the "forms" model. However, non-forms based functionality is also provided for by the WOSA/XFS specification.

In the implementation of forms, both the form and the field definition are stored in files called "Form Definition Files", and the name of the directory containing these files is stored in the Windows NT Registry. Each file may contain one or more form definitions, and the registry may contain any number of Form Definition File paths, limited only by the system resources.

Forms Validation

Form validation is performed in two stages; at start-up and at runtime.

Start-up validation:

When the SP is started up, all form definition files are opened and validated. The syntax errors are checked if they are categorized under the following:

- Invalid keywords
- Duplication of keywords
- Unexpected placement of keywords
- Invalid data associated with a keyword
- Invalid comments

The form definitions are converted to an 'in-core' representation called the 'Forms Database'. Errors found during this initial validation are logged in the form error log file whose location is specified in the registry. If an error is detected in a form during this validation, an entry will be made in the 'Forms Database', but the form will be marked as invalid.

Runtime validation:
Runtime validation is performed when the QUERY_FORM, READ_TRACK and WRITE_TRACK commands are given. The type of errors detected at this stage are:

- User defined fields used but not defined.
- User defined fields not found on track.
- User defined fields out of bounds for track.
- Field requested to be read/written has not been defined in form.

Errors found during this validation process may be logged either in the form error log file or the trace file whose locations are specified in the registry.

Interpretation of Reserved Keywords:

The following table describes how Keywords are interpreted by the SP. Keywords not listed below are interpreted as described in ref. 5.

Keyword	Interpretation
SECURE	This keyword is ignored as Security Modules are not supported.
FIELDSEPPOS*n*	This is the *n*th occurrence of field separator in the track. Here, *n* must be greater than 0.
DEFAULT	The value for this keyword must be a single character other than the character '?', which signifies that fields not specified in the form are to be left unchanged. This character is used to pad up fields in the track not explicitly mentioned in the form definition.
ALL	For Write forms, if the input specifies the field name as ALL, the entire track is written to, irrespective of the fields defined in the form.Similarly, if the input names certain fields, but the form specifies ALL for the track, the entire track is written with the original contents or with the Default character.

Forms Guidelines

- The form name defined between the delimiters '['and']' should not contain blank spaces. There can be blank spaces elsewhere. Therefore,
  [READ FORM] is an invalid form name, but [READFORM] is valid.
- The form name cannot exceed 20 characters.
- User defined keywords cannot have blank spaces in between. Therefore a keyword 'ACC NO' will result in a syntax error for the form.
- User defined keywords cannot exceed 20 characters.
- The maximum size of data that can be associated with a keyword is 320 characters.
- All user defined fields must have a start and an end position. These positions must always be of the form, <Field Separator> +/- <Field Offset>.
  For example, a form which contains MII= FIELDSEPPOS1, ENDTRACK will be invalid. The correct format would be as MII= FIELDSEPPOS1 + 0, ENDTRACK + 0.
- If the read algorithm does not make use of brackets to take care of precedence, a check will occur from left to right, irrespective of the type of operand. Thus, read algorithm 'TRACK1|TRACK2&TRACK3', it is equivalent to '(TRACK1|TRACK2)&TRACK3'.
- The absence of a DEFAULT character being defined in the form is equivalent to the statement "DEFAULT = ?" being present in the form.

Application Guidelines

1. If the READ_RAW_DATA command requests the ATR of the chip to be read, this will be the last action performed. When the command is complete, the card will be staged in the Smart card reader. If a chip is present, it will be in a powered-up condition.

2. After the READ_RAW on Chip or the CHIP_IO command is completed successfully, the card remains staged in the Smart Card unit and will be in a powered-on condition. To switch off, issue any other command which requires tracks to be read/written. The Eject/Retain operations will also switch the chip off.

3. Errors detected in form files will be logged in the form error file named in the registry. During initialization of the SP, the forms are read from the files, formatted into structures and placed into memory. Only certain classes of errors can be detected at this stage. If an error is found in any form, it will be logged in the file. In order to check for basic syntactical correctness of forms, examine the form error file just after the SP has been initialized. After form related commands have been executed, semantical errors are likely to be detected. This, too, will be logged into the form error file.

4. In the READ_RAW command, the data will not contain track delimiters.

5. In the WRITE_RAW command, the data passed to write on card should not contain track delimiters.

6. The maximum data that can be stored on tracks 1, 2 and 3 are 78, 39 and 106 respectively minus two delimeters.

7. If the read algorithm is "TRACK1 | (TRACK2 & TRACK3)", the output data will contain data from all the tracks successfully read. Reading will not stop immediately after the read algorithm is satisfied.

8. When a QUERY_FORM command returns the error of WFS_ERR_FORMINVALID, there is definitely a problem with the form. However, the same cannot be said of the READ_TRACK and WRITE_TRACK commands. Here, WFS_ERR_FORMINVALID error may be returned even if command or data does not suit the form being used.
   For example - A Read form being used in a WRITE_TRACK command, a filed defined in a form is not present on card data to be written on a field overflows the field, etc. Messages in the trace file will indicate the true nature of the error.

9. Following a WFSCancelAsyncRequest of any command that results in the SP waiting for a card to be inserted, the application should issue a GET_STATUS command to determine if a card is staged or not. This is recommended because there is a finite possibility that the cancel request is processed as the card is being staged in which case the SP does not generate a WFS_EXEE_IDC_MEDIAINSERTED event.

10 The IDC SP and device drivers attempt to recover from card jam and other card conditions. If after repeated attempts, the error condition persists, the device goes into a fatal state, a condition that can be cleared only by operator intervention via the VDM. The following table lists the error conditions from which the SP attempts to recover, and the action to be taken for other commonly occurring errors.

Error Condition	**Recovery Action**
Card Jam on Entry	Following a WFS_CMD_IDC_READ_RAW_DATA command, if the SP detects a card entering the throat but is unable to complete the ACCEPT operation, the SP assumes that customer tampering has occurred and enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Read errors	The device driver maintains a count of consecutive read errors for each track. The count is cleared by a good read from the appropriate track, but a blank track is not considered a good read in this case. The driver also maintains an overall error count of consecutive read errors on any track which can be cleared by a good read on any track. When this count reaches 15, 30 or 45, the SP assumes that customer tampering has occurred and attempts to eject the card. Following this, the SP enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR Recovery now requires operator intervention via VDM.
Blank tracks	The device driver maintains a count of consecutive blank tracks returned from any track, plus an overall count.When this count reaches 15, the SP assumes that customer tampering has occurred and attempts to eject the card. Following this, the SP enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.

Error Condition	Recovery Action
Invalid Cards	The device driver detects long cards on entry, and ejects them as invalid. However, if 5 consecutive invalid cards are detected, the SP assumes that customer tampering has occurred and attempts to eject the card. Following this, the SP enters a suspend state for 'SuspendTimeout' minutes, during which all commands that have device interaction will return WFS_ERR_DEV_NOT_READY. On expiry of this period, the SP checks the status of the device, and if it is found to be healthy, resumes normal operation. If user tampering persists, the device goes into a fatal state, following which all commands issued to the device will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Card jammed inside device	When the device driver detects a card jam it makes 3 attempts to complete the operation at 3-second intervals. If these attempts are unsuccessful, the SP returns WFS_ERR_IDC_MEDIAJAM and attempts to capture the card. If the attempt to capture the card fails the device goes into a fatal state in which the device is inoperable. All Execute command requiring device interaction issued to the SP will return WFS_ERR_HARDWARE_ERROR. If following a successful capture of a card, three consecutive card jams occur in either the forward or backward direction, the device enters a fatal state in which the device is rendered inoperable. All Execute command requiring device interaction issued to the SP will return WFS_ERR_HARDWARE_ERROR. Recovery now requires operator intervention via VDM.
Write errors due to bad media/hardware errors.	The device driver maintains a count of consecutive write errors for each track, and one for the card overall. The overall count is cleared by a good write to any track. If this count reaches six, the device enters a fatal state in which the device is rendered inoperable. The SP make no attempt at recovery. Recovery now requires operator intervention via VDM.
Shutter Jam	When the device detects a card entering the throat, the shutter solenoid is energized for up to 25 seconds. If the shutter fails to open, the SP returns WFS_ERR_IDC_SHUTTERFAIL. If the driver detects this condition five times in a row, the device enters a fatal state in which the device is rendered inoperable. The SP makes no attempt at recovery. Recovery now requires operator intervention via VDM.

References

Table of Contents

References

1 WOSA/XFS Application Programming Interface (API)/Service Provider Interface (SPI) Specification, Programmer’s Reference, Revision 2.00, September 12, 1996.

2 WOSA/XFS Sensors and Indicators Unit Device Class Interface, Programmer's Reference, Revision 2.00, September 26, 1996.

3 WOSA/XFS Programming Interface Specification for Depository Service Class, Revision 2.00, September 19, 1996.

4 WOSA/XFS Cash Dispenser Device Class Interface, Programmer's Reference, Revision 2.00, September 25th, 1996.

5 WOSA/XFS Magnetic Stripe Reader/Writer Device Class Interface, Programmer's Reference, Revision, 2.00 September 11th, 1996.

6 WOSA/XFS Printer Device Class Interface, Programmer's Reference, -- Revision 2.00, September 19, 1996.

7 WOSA/XFS Text Terminal Unit Service Class, Revision 2.00, Sept. 26, 1996.

8 WOSA/XFS Vendor Dependent Mode Unit Service Class-Revision 2.00, Sept. 26, 1996.

9 WOSA/XFS PIN Keypad Device Class Interface, Programmer's Reference, Revision 2.04, Sep 11, 1996.

10 NCR Self-Service Platform Software, Programmer's Manual for the Media Entry Indicators, D1-4597A.

11 NCR Self-Service Platform Software, Programmer's Manual for the Indicators Services , D1-4596-A.

12 NCR Self-Service Platform Software, Programmer's Manual for the Alarms, D1-4601-A.

13 56XX ATMs Diagnostic Status Code Note Book, SP-5065.

14 NCR 56XX ATMs Security Enclosures' Alarms and Locks, D1-2472-A.

15 Programmer's Manual for the Envelope Dispenser - NCR Self-Service Platform Software, D1-4610-A.

16 Programmer's Manual for the Envelope Depository - NCR Self-Service Platform Software, D1-4607-A.

17 Programmer's Manual for the Night safe Depository - NCR Self-Service Platform Software, D1-4609-A.

18 Self-Service Platform Software Programmer's Manual for the Currency Dispenser, D1-4606-A.

19 Programmer’s Manual for the Motorised Magnetic Card Reader/Writer NCR S4 D1-4604-A

20 Smart Card Programmer’s Reference Manual. NCR S4 D1-3084-B.

21 Programmer’s Manual for the Swipe Magnetic Card Reader NCR S4 D1-4741-A.

22 Programmer’s Manual for the DIP Magnetic Stripe Reader NCR S4 D1-4605-A.

23 Programmer's Manual for the 40 Column Printer, NCR: --D1-4612-A.

24 Programmer's Manual for the 80 Column Printer, NCR: --D1-4611-A.

25 NCR Self-Service Platform Software, Programmer's Manual for the Enhanced Operator Panel, D1-4598-A.

26 NCR Self-Service Platform Software, Programmer's Manual for the Cardholder Keyboard Manager, D1-4594-C.

Cut

User Feedback Form

Title: WOSA/XFS - Programmer's Reference Manual

Number: B006-0000-6001 **Release** 01.00.00 **Date:** September 1997

NCR welcomes your feedback on this publication. Your comments can be of great value in helping us improve our information products.

You may send your comments, electronically, to the Information Products Department at Dundee. See over for details.

Circle the numbers below that best represent your opinion of this publication.

Ease of use	5	4	3	2	1	0
Accuracy	5	4	3	2	1	0
Clarity	5	4	3	2	1	0
Completeness	5	4	3	2	1	0
Organization	5	4	3	2	1	0
Appearance	5	4	3	2	1	0
Examples	5	4	3	2	1	0
Illustrations	5	4	3	2	1	0
Job performance	5	4	3	2	1	0
Question resolution	5	4	3	2	1	0
Overall satisfaction	5	4	3	2	1	0

5 = Excellent
4 = Good
3 = Adequate
2 = Fair
1 = Poor

0 = Not applicable

Indicate the ways you feel we could improve this publication.

- ❑ Improve the table of contents
- ❑ Improve the overview/introduction
- ❑ Improve the organization
- ❑ Improve the index
- ❑ Make it less technical
- ❑ Make it more concise/brief
- ❑ Add more/better quick reference aids
- ❑ Add more examples
- ❑ Add more illustrations
- ❑ Add more step-by-step procedures
- ❑ Add more troubleshooting information
- ❑ Add more detail

Write any additional comments you may have below and on additional sheets, if necessary. Include page numbers where applicable.

Use the following addresses to send your comments, electronically, to the Information Products Department at Dundee:

WWW - http://www.ncr.com/product/infoprod/dundeeip/
e-mail - Information.Products@Dundee.ncr.com

Cut

Fold

If we may contact you concerning your comments, please fill in the information below:

Name: _____

Organization: _____

Company: _____

Address: _____

_____

Phone: _____ Fax: _____

Thank you for your evaluation of this publication. Fold the form where indicated, tape (please do not staple), and drop in the mail.

F 8763-0695

Fold

NCR

Affix Postage Stamp Here

NCR Financial Systems Ltd
Information Products
Kingsway West
Dundee
Scotland
DD2 3XX