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

WORKSHOP AGREEMENT

CWA 14050-16

November 2000

ICS 35.200; 35.240.15; 35.240.40

Extensions for Financial Services (XFS) interface specification - Release 3.0 - Part 16: Application Programming Interface (API) - Service Provider Interface (SPI) - Migration from Version 2.0 (see CWA 13449) to Version 3.0 (this CWA) - Programmer's Reference

Ref. No CWA 14050-16:2000 E

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

Table of Contents

Foreword

This CWA is revision 3.0 of the XFS interface specification.

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

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

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

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

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

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

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

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

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

Part 2: Service Classes Definition; Programmer's Reference

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1. General

In the Configuration Information chapter the reference to Windows 3.1 and Windows for Workgroups is removed. The configuration information is now partly moved to HKEY_LOCAL_MACHINE and partly to HKEY_USERS.

The hardware and software error events now return information about the physical service name and an action that is required to manage the error.

2. New Chapters

2.1 Vendor Dependent Mode

XFS compliant applications must comply with the following:

- Every XFS application should open a session with the VDM ServiceProvider passing a valid ApplId and then register for all VDM entry and exit notices.
- Before opening any session with any other XFS Service Provider, check the status of the VDM Service Provider. If Vendor Dependent Mode is not “Inactive”, do not open a session.
- When getting a VDM entry notice, close all open sessions with all other XFS Service Providers as soon as possible and issue an acknowledgement for the entry to VDM.
- When getting a VDM exit notice, acknowledge at once.
- When getting a VDM exited notice, re-open any required sessions with other XFS Service Providers.

This is mandatory for self-service but optional for branch.

2.2 References

1. XFS Service Classes Definition, Programmer’s Reference Revision 3.00, October 18, 2000

3. Changes to existing Chapters

3.1 Configuration Information

The XFS Manager uses its configuration information to define the relationships among the applications and the service providers. In particular, this information defines the mapping between the logical service interface presented at the API (via logical service name) and the appropriate service provider entry points.

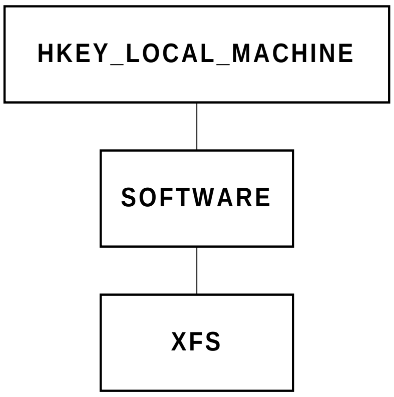
The configuration information also includes specific information about logical services and service providers, some of which is common to all solution providers; it may also include information about physical services, if any are present on the system, and vendor-specific information. The location of the information is transparent to both applications and service providers; they always store and retrieve it using the configuration functions provided by the XFS Manager, as described in Section 7, for portability across Windows platforms.

It is the responsibility of solution providers, and the developers of each service provider, to implement the appropriate setup and management utilities, to create and manage the configuration information about the XFS subsystem configuration and its service providers, using the configuration functions.

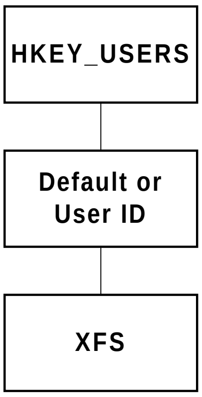
These functions are used by service providers and applications to write and retrieve the configuration information for a XFS subsystem, which is stored in a hierarchical structure called the Windows Registry. The structure and the functions are based on the Win32 Registry architecture and API functions, and are implemented in Windows NT/98 and future versions of Windows using the Registry and the associated functions.

Each node in the configuration registry is called a key, each having a name and (optionally) values. All values consist of a name and data pair, both null-terminated character strings. There are two logical groupings of XFS Registry information; local PC dependent configuration information and user dependent configuration information.

The local PC dependent configuration information is stored beneath the following Registry key.



User dependent configuration information is stored in the HKEY_USERS section of the Registry.



Within the local PC dependent configuration information are stored three XFS related keys;

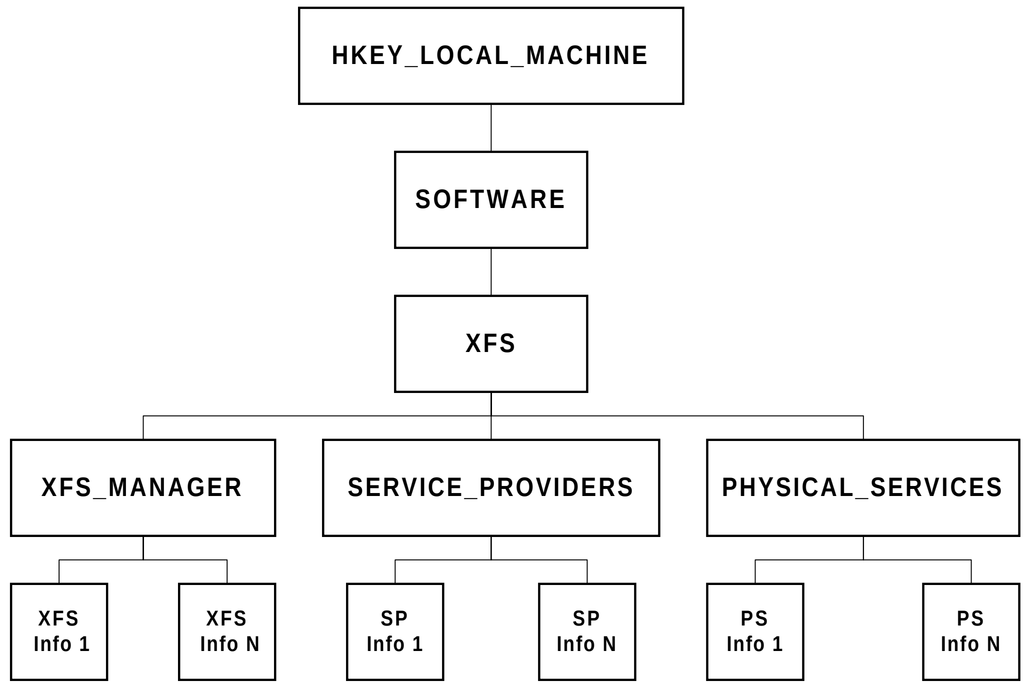
- XFS_MANAGER – Beneath this key are values and/or keys for information that the XFS Manager creates and uses.
- SERVICE_PROVIDERS – Beneath this key is a key for each XFS compliant service provider.
- PHYSICAL_SERVICES – Beneath this key are physical attachment configuration information, defined by the solution provider.

Within the User dependent configuration information is stored the following LOGICAL_SERVICES key:

- LOGICAL_SERVICES – Beneath this key is defined a key for each XFS logical service (ie: the *lpszLogicalName* parameter of the **WFSOpen**, **WFSAsyncOpen** and **WFPOpen** functions)

The configuration functions provide the capabilities to create, enumerate, open and delete keys, and to set, query and delete values within each key. Vendor-provided configuration utility programs set up the registry structure and its contents, using these functions. Configured Registry values and keys define how the XFS subsystem, services and providers are configured. These are used by the XFS Manager, applications and service providers. Note that vendor-specific information may be added to any key in this structure, using optional values.

The figure below illustrates the full structure of the local PC dependent configuration information.



The XFS_MANAGER key has the following optional values:

• TraceFile	the name of the file containing trace data. If this value is not set in the configuration, trace data is written to the default file path\name C:\XFSTRACE.LOG.
• ShareFilename	the name of the memory mapped file used by the memory management functions of the XFS Manager.
• ShareFilesize	the size of the memory mapped file used by the memory management functions of the XFS Manager.

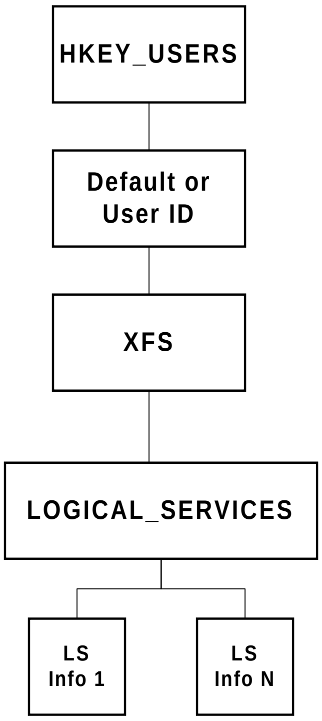
Some additional values could be also defined in the XFS SDK release notes. Please refer to the related document for more information.

Beneath the SERVICE_PROVIDERS key there are keys for each individual service providers, the keys are the service provider names. Each of these keys has three mandatory values:

• dllname	the name of the file containing the service provider DLL
• vendor_name	the name of the supplier of this service provider
• version	the version number of this service provider

The PHYSICAL_SERVICES keys are fully vendor dependent.

The figure below illustrates the full structure of the User dependent configuration information.



Beneath the LOGICAL_SERVICES key there are keys for each individual service providers, the keys are the logical service names. Each of these keys has two mandatory values:

- class the service class of the logical service; (see the Service Class Definition Document for the standard values)
- provider the name of the service provider that provides the logical service (the key name of the corresponding service provider key)

The 'User Id' key is only applicable to the Windows Terminal Server platform. The 'User Id' is the user name associated with the session in which the application is executing.

An example of the content of the configuration information for an actual system in exported REGEDIT form is shown below. See Section 7 for the definitions of the configuration functions.

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\XFS]

[HKEY_LOCAL_MACHINE\SOFTWARE\XFS\XFS_MANAGER]
"TraceFile"="C:\\XFSTRACE.LOG"

[HKEY_LOCAL_MACHINE\SOFTWARE\XFS\PHYSICAL_SERVICES]

[HKEY_LOCAL_MACHINE\SOFTWARE\XFS\SERVICE_PROVIDERS]

[HKEY_LOCAL_MACHINE\SOFTWARE\XFS\SERVICE_PROVIDERS\IBM4722]
"dllname"="IBM4722.DLL"
"vendor_name"="XFS Solutions Provider"
"version"="1.0.0"

[HKEY_LOCAL_MACHINE\SOFTWARE\XFS\SERVICE_PROVIDERS\IBM4777]
"dllname"="IBM4777.DLL"
"vendor_name"="XFS Solutions Provider"
"version"="1.0.0"
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\XFS\SERVICE_PROVIDERS\IBM4778]
"dllname"="IBM4778.DLL"
"vendor_name"="XFS Solutions Provoder"
"version"="1.0.0"

[HKEY_LOCAL_MACHINE\SOFTWARE\XFS\SERVICE_PROVIDERS\IBM4733]
"dllname"="IBM4733.DLL"
"vendor_name"="XFS Solutions Provider"
"version"="1.0.0"

[HKEY_USERS\.Default\XFS\LOGICAL_SERVICES]

[HKEY_USERS\.Default\XFS\LOGICAL_SERVICES\CashDispenser]
"class"="ATS"
"provider"="IBM4733"
"type"="ATS"

[HKEY_USERS\.Default\XFS\LOGICAL_SERVICES\Document]
"class"="PTR"
"provider"="IBM4722"
"type"="DOCUMENT"

[HKEY_USERS\.Default\XFS\LOGICAL_SERVICES\Magstripe]
"class"="IDC"
"provider"="IBM4777"
"type"="SWIPE"

[HKEY_USERS\.Default\XFS\LOGICAL_SERVICES\Passbook]
"class"="PTR"
"provider"="IBM4722"
"type"="PASSBOOK"

[HKEY_USERS\.Default\XFS\LOGICAL_SERVICES\Pinpad]
"class"="PIN"
"provider"="IBM4778"
"type"="EDM"

[HKEY_USERS\.Default\XFS\LOGICAL_SERVICES\Receipt]
"class"="PTR"
"provider"="IBM4722"
"type"="RECEIPT"
```

3.2 Exclusive Service and Device Access

. . .

An application should act in a cooperative manner when locking a service, by keeping it locked for the minimum time period that it requires exclusive access to the service. Typically, this means locking a set of services, performing a series of requests to the services to complete a transaction, and immediately unlocking the services. However, an application which has obtained a lock on a device will be informed via the WFS_SYSE_LOCK_REQUESTED system event whenever another application requests a lock on the device (i.e. Potentially multiple lock request events will occur – one for each request by another application). Therefore an alternative strategy is for the application to register for system events and unlock the device only when it receives the event notification that another application has requested a lock on the device.

. . .

3.3 Lock Policy for Independant Devices

. . .

Service state: LOCKED

- Arriving requests (except lock requests) are handled as follows:
  - Non-deferred requests are processed on arrival.
  - Deferred requests that are ***not*** **WFPExecute** requests are placed in the deferred queue.
  - **WFPExecute** requests from the owner of the lock are placed in the deferred queue.
  - **WFPExecute** requests that are not from the owner of the lock are rejected (with error code WFS_ERR_LOCKED).
  - **WFPUnlock** and **WFPClose** requests from the owner of the lock are placed in the deferred queue. (Note that a close request to a locked service is treated as an unlock followed by a close.)
  - **WFPUnlock** and **WFPClose** requests that are ***not*** from the owner of the lock are treated as non-deferred requests, i.e., processed on arrival.
- The deferred queue is processed FIFO.
- When a **WFPLock** request arrives:
  - If it is from the owner of the lock, it is granted.
  - If it is not from the owner of the lock, it is placed in the lock queue, an WFS_SYSE_LOCK_REQUESTED event is posted to the owner of the lock.
- When a **WFPUnlock** or **WFPClose** request is processed from the deferred queue, or the connection between the service and the owner of the lock is lost:
  - If the lock queue is not empty, the service state changes to **LOCK_PENDING**.
  - If the lock queue is empty, the service state changes to **UNLOCKED**.

. . .

3.4 Notification Mechanisms — Registering for Events

. . .
There are four classes of events:
- SERVICE_EVENTS
- USER_EVENTS
- SYSTEM_EVENTS
- EXECUTE_EVENTS

For the first three of these event classes, if a class is being monitored and an event occurs in that class, a message is broadcast to every *hWnd* registered for that class, specifying the service identified by the *hService* handle. The exception to this is the WFS_SYSE_LOCK_REQUESTED system event, this event is posted only to the application which owns the lock on the device. The events are generated when:
- the service status changes (SERVICE_EVENTS), e.g., a printer is suspended or is no longer available.
- the service needs an operation from the user to take place (USER_EVENTS), e.g., a device needs “abnormal” attention, such as adding paper or toner to a printer.
- a system event occurs (SYSTEM_EVENTS), e.g., a hardware error occurs, a version negotiation fails, the network is no longer available or there is no more disk space.

The EXECUTE_EVENTS class is different from the other three. These are events which occur as a normal part of processing an **WFSExecute** command and they are always sent before the completion of the command.

. . .

4. Changes to Application Programming Interface (API) Functions

4.1 *WFSExecute*

. . .

Error Codes If the function return is not WFS_SUCCESS, it is one of the following error conditions. Any service-specific errors that can be returned are defined in the specifications for each service class.

WFS_ERR_CANCELED
The request was canceled by **WFSCancelBlockingCall**.

WFS_ERR_CONNECTION_LOST
The connection to the service is lost.

WFS_ERR_DEV_NOT_READY
The function required device access, and the device was not ready or timed out.

WFS_ERR_HARDWARE_ERROR
The function required device access, and an error occured on the device.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_COMMAND
The *dwCommand* issued is not supported by this service class.

WFS_ERR_INVALID_DATA
The data structure passed as input parameter contains invalid data.

WFS_ERR_INVALID_POINTER
A pointer parameter does not point to accessible memory.

WFS_ERR_INVALID_HSERVICE
The *hService* parameter is not a valid service handle.

WFS_ERR_LOCKED
The service is locked under a different *hService*.

WFS_ERR_NOT_STARTED
The application has not previously performed a successful **WFSStartUp**.

WFS_ERR_OP_IN_PROGRESS
A blocking operation is in progress on the thread; only **WFSCancelBlockingCall** and **WFSIsBlocking** are permitted at this time.

WFS_ERR_SOFTWARE_ERROR
The function required access to configuration information, and an error occurred on the software.

WFS_ERR_TIMEOUT
The timeout interval expired.

WFS_ERR_UNSUPP_COMMAND
The *dwCommand* issued, although valid for this service class, is not supported by this service provider or device.

WFS_ERR_USER_ERROR
A user is preventing proper operation of the device.

WFS_ERR_UNSUPPORTED_DATA
The data structure passed as an input parameter although valid for this service class, is not supported by this service provider or device.

. . .

4.2 WFSAsyncExecute

. . .

Error Codes If the function return is not WFS_SUCCESS, it is one of the following error conditions, indicating that the asynchronous operation was not initiated:

WFS_ERR_CONNECTION_LOST
The connection to the service is lost.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_COMMAND
The *dwCommand* issued is not supported by this service class.

WFS_ERR_INVALID_HSERVICE
The *hService* parameter is not a valid service handle.

WFS_ERR_INVALID_HWND
The *hWnd* parameter is not a valid window handle.

WFS_ERR_INVALID_POINTER
A pointer parameter does not point to accessible memory.

WFS_ERR_NOT_STARTED
The application has not previously performed a successful **WFSStartUp**.

WFS_ERR_OP_IN_PROGRESS
A blocking operation is in progress on the thread; only **WFSCancelBlockingCall** and **WFSIsBlocking** are permitted at this time.

WFS_ERR_UNSUPP_COMMAND
The *dwCommand* issued, although valid for this service class, is not supported by this service provider or device.

The following error conditions are returned via the asynchronous command completion message, as the *hResult* from the WFSRESULT structure. Any service-specific errors that can be returned are defined in the specifications for each service class.

WFS_ERR_CANCELED
The request was canceled by **WFSCancelAsyncRequest**.

WFS_ERR_DEV_NOT_READY
The function required device access, and the device was not ready or timed out.

WFS_ERR_HARDWARE_ERROR
The function required device access, and an error occured on the device.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_DATA
The data structure passed as input parameter contains invalid data.

WFS_ERR_LOCKED
The service is locked under a different *hService*.

WFS_ERR_SOFTWARE_ERROR
The function required access to configuration information, and an error occured on the software.

WFS_ERR_TIMEOUT
The timeout interval expired.

WFS_ERR_UNSUPP_COMMAND
The *dwCommand* issued, although valid for this service class, is not supported by this service provider or device.

WFS_ERR_USER_ERROR
A user is preventing proper operation of the device.

WFS_ERR_UNSUPPORTED_DATA
The data structure passed as an input parameter although valid for this service class, is not supported by this service provider or device.

. . .

4.3 WFSGetInfo

. . .

Error Codes If the function return is not WFS_SUCCESS, it is one of the following error conditions. Any service-specific errors that can be returned are defined in the specifications for each service class.

WFS_ERR_CANCELED
The request was canceled by **WFSCancelBlockingCall**.

WFS_ERR_CONNECTION_LOST
The connection to the service is lost.

WFS_ERR_DEV_NOT_READY
The function required device access, and the device was not ready or timed out.

WFS_ERR_HARDWARE_ERROR
The function required device access, and an error occured on the device.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_CATEGORY
The *dwCategory* issued is not supported by this service class.

WFS_ERR_INVALID_DATA
The data structure passed as input parameter contains invalid data.

WFS_ERR_INVALID_HSERVICE
The *hService* parameter is not a valid service handle.

WFS_ERR_INVALID_POINTER
A pointer parameter does not point to accessible memory.

WFS_ERR_NOT_STARTED
The application has not previously performed a successful **WFSStartUp**.

WFS_ERR_OP_IN_PROGRESS
A blocking operation is in progress on the thread; only **WFSCancelBlockingCall** and **WFSIsBlocking** are permitted at this time.

WFS_ERR_SOFTWARE_ERROR
The function required access to configuration information, and an error occured on the software.

WFS_ERR_TIMEOUT
The timeout interval expired.

WFS_ERR_UNSUPP_CATEGORY
The *dwCategory* issued, although valid for this service class, is not supported by this service provider.

WFS_ERR_USER_ERROR
A user is preventing proper operation of the device.

WFS_ERR_UNSUPPORTED_DATA
The data structure passed as an input parameter although valid for this service class, is not supported by this service provider or device.

. . .

4.4 WFSAsyncGetInfo

. . .

Error Codes If the function return is not WFS_SUCCESS, it is one of the following error conditions, indicating that the asynchronous operation was not initiated:

WFS_ERR_CONNECTION_LOST
The connection to the service is lost.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_CATEGORY
The *dwCategory* issued is not supported by this service class.

WFS_ERR_INVALID_HSERVICE
The *hService* parameter is not a valid service handle.

WFS_ERR_INVALID_HWND
The *hWnd* parameter is not a valid window handle.

WFS_ERR_INVALID_POINTER
A pointer parameter does not point to accessible memory.

WFS_ERR_NOT_STARTED
The application has not previously performed a successful **WFSStartUp**.

WFS_ERR_OP_IN_PROGRESS
A blocking operation is in progress on the thread; only **WFSCancelBlockingCall** and **WFSIsBlocking** are permitted at this time.

WFS_ERR_UNSUPP_CATEGORY
The *dwCategory* issued, although valid for this service class, is not supported by this service provider.

The following error conditions are returned via the asynchronous command completion message, as the *hResult* from the WFSRESULT structure. Any service-specific errors that can be returned are defined in the specifications for each service class.

WFS_ERR_CANCELED
The request was canceled by **WFSCancelAsyncRequest**.

WFS_ERR_DEV_NOT_READY
The function required device access, and the device was not ready or timed out.

WFS_ERR_HARDWARE_ERROR
The function required device access, and an error occured on the device.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_DATA
The data structure passed as input parameter contains invalid data..

WFS_ERR_SOFTWARE_ERROR
The function required access to configuration information, and an error occured on the software.

WFS_ERR_TIMEOUT
The timeout interval expired.

WFS_ERR_USER_ERROR
A user is preventing proper operation of the device.

WFS_ERR_UNSUPPORTED_DATA
The data structure passed as an input parameter although valid for this service class, is not supported by this service provider or device.

. . .

4.5 WFSOpen

HRESULT **WFSOpen**(*lpszLogicalName, hApp, lpszAppID, dwTraceLevel, dwTimeOut, dwSrvcVersionsRequired, lpSrvcVersion, lpSPIVersion, lphService*)

Initiates a session (a series of service requests terminated with the **WFSClose** function) between the application and the specified service. This does not necessarily mean that the hardware is opened. This command will return with WFS_SUCCESS even if the hardware is inoperable, offline or powered off. The status of the device can be requested through a GetInfo command.

The synchronous version of **WFSAsyncOpen.**

. . .

The error codes WFS_ERR_DEV_NOT_READY and WFS_ERR_HARDWARE_ERROR are removed.

4.6 WFSAsyncOpen

HRESULT **WFSAsyncOpen**(*lpszLogicalName, hApp, lpszAppID, dwTraceLevel, dwTimeOut, lphService, hWnd, dwSrvcVersionsRequired, lpSrvcVersion, lpSPIVersion, lpRequestID*)

Initiates a session (a series of service requests terminated with the **WFSClose** or **WFSAsyncClose** function) between the application and the specified service. This does not necessarily mean that the hardware is opened. This command will return with WFS_SUCCESS even if the hardware is inoperable, offline or powered off. The status of the device can be requested through a GetInfo command.

The asynchronous version of **WFSOpen.**

. . .

5. Changes to Service Provider Interface (SPI) Functions

5.1 WFPExecute

. . .

Error Codes If the function return is not WFS_SUCCESS, it is one of the following error conditions, indicating that the asynchronous operation was not initiated. Any service-specific errors that can be returned are defined in the specifications for each service class.

WFS_ERR_CONNECTION_LOST
The connection to the service is lost.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_COMMAND
The *dwCommand* issued is not supported by this service class.

WFS_ERR_INVALID_HSERVICE
The *hService* parameter is not a valid service handle.

WFS_ERR_INVALID_HWND
The *hWnd* parameter is not a valid window handle.

WFS_ERR_INVALID_POINTER
A pointer parameter does not point to accessible memory.

WFS_ERR_UNSUPP_COMMAND
The *dwCommand* issued, although valid for this service class, is not supported by this service provider.

The following error conditions are returned via the asynchronous command completion message, as the *hResult* from the WFSRESULT structure. Any service-specific errors that can be returned are defined in the specifications for each service class.

WFS_ERR_CANCELED
The request was canceled by **WFSCancelAsyncRequest**.

WFS_ERR_DEV_NOT_READY
The function required device access, and the device was not ready or timed out.

WFS_ERR_HARDWARE_ERROR
The function required device access, and an error occured on the device.

WFS_ERR_INVALID_DATA
The data structure passed as input parameter contains invalid data..

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_LOCKED
The service is locked under a different *hService*.

WFS_ERR_SOFTWARE_ERROR
The function required access to configuration information, and an error occured on the software.

WFS_ERR_TIMEOUT
The timeout interval expired.

WFS_ERR_USER_ERROR
A user is preventing proper operation of the device.

WFS_ERR_UNSUPPORTED_DATA
The data structure passed as an input parameter although valid for this service class, is not supported by this service provider or device.

. . .

5.2 WFPGetInfo

. . .

Error Codes If the function return is not WFS_SUCCESS, it is one of the following error conditions, indicating that the asynchronous operation was not initiated. Any service-specific errors that can be returned are defined in the specifications for each service class.

WFS_ERR_CONNECTION_LOST
The connection to the service is lost.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_CATEGORY
The *dwCategory* issued is not supported by this service class.

WFS_ERR_INVALID_HSERVICE
The *hService* parameter is not a valid service handle.

WFS_ERR_INVALID_HWND
The *hWnd* parameter is not a valid window handle.

WFS_ERR_INVALID_POINTER
A pointer parameter does not point to accessible memory.

WFS_ERR_UNSUPP_CATEGORY
The *dwCategory* issued, although valid for this service class, is not supported by this service provider.

The following error conditions are returned via the asynchronous command completion message, as the *hResult* from the WFSRESULT structure. Any service-specific errors that can be returned are defined in the specifications for each service class.

WFS_ERR_CANCELED
The request was canceled by **WFSCancelAsyncRequest**.

WFS_ERR_DEV_NOT_READY
The function required device access, and the device was not ready or timed out.

WFS_ERR_HARDWARE_ERROR
The function required device access, and an error occured on the device.

WFS_ERR_INTERNAL_ERROR
An internal inconsistency or other unexpected error occurred in the XFS subsystem.

WFS_ERR_INVALID_DATA
The data structure passed as input parameter contains invalid data..

WFS_ERR_SOFTWARE_ERROR
The function required access to configuration information, and an error occured on the software.

WFS_ERR_TIMEOUT
The timeout interval expired.

WFS_ERR_USER_ERROR
A user is preventing proper operation of the device.

WFS_ERR_UNSUPPORTED_DATA
The data structure passed as an input parameter although valid for this service class, is not supported by this service provider or device.

. . .

6. Changes to Configuration Functions

The introduction of this section was removed. It now only refers to the chapter Configuration Information.

See Section 3.7 for the overall discussion of configuration information and how it is stored within the Windows Registry.

6.1 *WFMCreateKey*

HRESULT **WFMCreateKey** (*hKey, lpszSubKey, phkResult, lpdwDisposition*)

Creates a new key, or if the specified key exists, opens it.

The first use of hKey by a process sets the migration mode for that process.

If the current value WFS_CFG_XFS_ROOT is the first hKey used, the XFS_CONF.DLL will attempt to migrate values from CLASSES_ROOT to LOCAL_MACHINE. If either of the new values WFS_CFG_MACHINE_XFS_ROOT or WFS_CFG_USER_DEFAULT_XFS_ROOT are used then no migration will take place for this process. The assumption is that any process using the new key values will be doing it's own migration. The reason migration does not always take place is that some applications will require access to both the old and new key roots so that they can migrate their non-CEN keys and values.

Parameters

HKEY *hKey*
Handle to a currently open key, or the predefined handle value:
WFS_CFG_HKEY_XFS_ROOT
WFS_CFG_MACHINE_XFS_ROOT
WFS_CFG_USER_DEFAULT_XFS_ROOT
The key opened or created by this function is a subkey of the key identified by this parameter.

LPSTR *lpszSubKey*
Pointer to a null-terminated string containing the name of the key to be created or opened.

PHKEY *phkResult*
Pointer to a variable that receives the handle of the created or opened key.

LPDWORD *lpdwDisposition*
Pointer to a variable that receives one of the disposition values:
WFS_CFG_CREATED_NEW_KEY
WFS_CFG_OPENED_EXISTING_KEY

Comments

If this function creates a new key, it has no values. The **WFMSetValue** function is used to create values.

Error Codes

If the function return is not WFS_SUCCESS, it is one of the following error conditions:

WFS_ERR_CFG_INVALID_HKEY
The specified *hKey* parameter does not correspond to a currently open key.

WFS_ERR_INVALID_POINTER
A pointer parameter does not point to accessible memory.

7. New Events

7.1 Lock Requested

The Lock requested system event is sent to any application which currently has a device locked whenever a request for a lock on the same device is received from another application or service handle. Note that this event is generated each time another application requests a lock on the same device. This system event differs from other system events in that it is only posted to the owner of the lock, it is NOT posted to any other applications.

Field	Description
RequestID	(not used)
hService	Service handle identifying the device and session which has obtained the lock..
tsTimestamp	Time the status change occurred (local time, in a Win32 SYSTEMTIME structure)
hResult	(not used)
u.dwEventID	= WFS_SYSE_LOCK_REQUESTED
lpBuffer	(not used)

8. Changes to existing Events

8.1 Device Status Changes

Status changes of logical services (which typically reflect changes in physical devices) are reported as system events. This is in addition to being reported by the WFS_INF_xxx_STATUS query of the **WFSGetInfo** or **WFSAsyncGetInfo** functions. The WFSRESULT data structure (defined in Section 8.1) is utilized as follows:

Field	Description
RequestID	(not used)
hService	Service handle identifying the session that created the result
tsTimestamp	Time the status change occurred (local time, in a Win32 SYSTEMTIME structure)
hResult	(not used)
u.dwEventID	= WFS_SYSE_DEVICE_STATUS
lpBuffer	Pointer to a WFSDEVSTATUS structure:

```
typedef struct  wfs_devstatus {
    LPSTR       lpszPhysicalName;
    LPSTR       lpszWorkstationName;
    DWORD       dwState;
} WFSDEVSTATUS, * LPWFSDEVSTATUS;
```

The members of this structure are:

Field	Description
lpszPhysicalName	Pointer to the physical service name of the service that changed its state.
lpszWorkstationName	Pointer to the name of the workstation in which the logical service name is defined.
dwState	Specifies the new state of the physical device managed by the service as one of the following:

Value	Meaning
WFS_STAT_DEVONLINE	The device is online (i.e., powered on and operable).
WFS_STAT_DEVOFFLINE	The device is offline (e.g., the operator has taken the device offline by turning a switch or pulling out the device).
WFS_STAT_DEVPOWEROFF	The device is powered off or physically not connected.
WFS_STAT_DEVNODEVICE	There is no device intended to be there; e.g. this type of self service machine does not contain such a device or it is internally not configured.
WFS_STAT_DEVHWERROR	The device is inoperable due to a hardware error.

WFS_STAT_DEVUSERERROR	The device is inoperable because a person is preventing proper device operation.

8.2 Hardware and Software Errors

Hardware and software errors are reported as system events. In most cases, this is in addition to being reported via the WFS_ERR_HARDWARE_ERROR or the WFS_ERR_SOFTWARE_ERROR or WFS_ERR_USER_ERROR error code that is returned when a hardware or software or user error occurs in the course of executing a function. The WFSRESULT data structure (defined in Section 8.1), is utilized as follows:

Field	**Description**
RequestID	Request ID of the request being processed when the error occurred (if any)
hService	Service handle identifying the session associated with the error (if any)
tsTimestamp	Time the error occurred (local time, in a Win32 SYSTEMTIME structure)
hResult	Result handle of the request being processed when the error occurred (if any)
u.dwEventID	*The ID of the error*

Value	**Meaning**
WFS_SYSE_HARDWARE_ERROR	The error is a hardware error
WFS_SYSE_SOFTWARE_ERROR	Th error is a software error
WFS_SYSE_USER_ERROR	The error is a user error

lpBuffer Pointer to a WFSHWERROR structure:

```
    typedef struct _wfs_hwerror {
        LPSTR           lpszLogicalName;
        LPSTR           lpszPhysicalName;
        LPSTR           lpszWorkstationName;
        LPSTR           lpszAppID;
        DWORD           dwAction;
        DWORD           dwSize;
        LPBYTE          lpbDescription;
    } WFSHWERROR, * LPWFSHWERROR;
```

The members of this structure are:

Field	**Description**
lpszLogicalName	Pointer to the logical service name of the service that generated the error (if any)
lpszPhysicalName	Pointer to the physical service name of the service that generated the error (if any)
lpszWorkstationName	Pointer to the the name of the workstation in which the logical service name is defined (if any)
lpszAppID	Pointer to the application ID associated with the session that generated the error (if any)
dwAction	The action required to manage the error. Possible values are:

Value	**Meaning**
WFS_ERR_ACT_NOACTION	No action required. Error was autorecovered.
WFS_ERR_ACT_RESET	Reset device to attempt recovery.
WFS_ERR_ACT_SWERROR	A software error occurred. Contact software vendor.
WFS_ERR_ACT_CONFIG	A configuration error occurred. Check configuration.
WFS_ERR_ACT_HWCLEAR	Recovery is not possible. A manual intervention for clearing the device is required. This value is only used for hardware errors.
WFS_ERR_ACT_HWMAINT	Recovery is not possible. A technical maintenance intervention is required. This value is only used for hardware errors.
WFS_ERR_ACT_SUSPEND	Device will attempt auto recovery and will advise any further action required via a Device Status Event.

dwSize	The size in bytes of the following description
lpbDescription	Pointer to a vendor-specific description of the error

9. New Error Codes

WFS_ERR_USER_ERROR
A user is preventing proper operation of the device.

WFS_ERR_UNSUPPORTED_DATA
The data structure passed as an input parameter although valid for this service class, is not supported by this service provider or device.

10. Changes to C – Header files

10.1 XFSAPI.H

```
/*************************************************************************
*                                                                        *
* xfsapi.h      WOSA/XFS - API functions, types, and definitions         *
*                                                                        *
*               Version 2.01  --  17/04/97                               *
*                                                                        *
*************************************************************************/

#ifndef __inc_xfsapi__h
#define __inc_xfsapi__h

#ifdef __cplusplus
extern "C" {
#endif

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

/****** Common ********************************************************/

#include <windows.h>

typedef unsigned short USHORT;
typedef char CHAR;
typedef short SHORT;
typedef unsigned long ULONG;
typedef unsigned char UCHAR;
typedef SHORT * LPSHORT;
typedef LPVOID * LPLPVOID;
typedef ULONG * LPULONG;
typedef USHORT * LPUSHORT;

typedef HANDLE HPROVIDER;

typedef ULONG REQUESTID;
typedef REQUESTID * LPREQUESTID;

typedef HANDLE HAPP;
typedef HAPP * LPHAPP;

typedef USHORT HSERVICE;
typedef HSERVICE * LPHSERVICE;

typedef LONG HRESULT;
typedef HRESULT * LPHRESULT;

typedef BOOL (WINAPI * XFSBLOCKINGHOOK)(VOID);
typedef XFSBLOCKINGHOOK * LPXFSBLOCKINGHOOK;

/****** String lengths *************************************************/

#define WFSDDESCRIPTION_LEN                     256
#define WFSDSYSSTATUS_LEN                       256

/****** Values of WFSDEVSTATUS.fwState **********************************/

#define WFS_STAT_DEVONLINE                      (0)
#define WFS_STAT_DEVOFFLINE                     (1)
#define WFS_STAT_DEVPOWEROFF                    (2)
#define WFS_STAT_DEVNODEVICE                    (3)
#define WFS_STAT_DEVHWERROR                     (4)
#define WFS_STAT_DEVUSERERROR                   (5)
#define WFS_STAT_DEVBUSY                        (6)

/****** Value of WFS_DEFAULT_HAPP ***************************************/

#define WFS_DEFAULT_HAPP                        (0)
```

```
/****** Data Structures *************************************************/

typedef struct _wfs_result
{
    REQUESTID       RequestID;
    HSERVICE        hService;
    SYSTEMTIME      tsTimestamp;
    HRESULT         hResult;
    union {
        DWORD       dwCommandCode;
        DWORD       dwEventID;
    } u;
    LPVOID          lpBuffer;
} WFSRESULT, * LPWFSRESULT;

typedef struct _wfsversion
{
    WORD            wVersion;
    WORD            wLowVersion;
    WORD            wHighVersion;
    CHAR            szDescription[WFSDDESCRIPTION_LEN+1];
    CHAR            szSystemStatus[WFSDSYSSTATUS_LEN+1];
} WFSVERSION, * LPWFSVERSION;

/****** Message Structures **********************************************/

typedef struct _wfs_devstatus
{
    LPSTR           lpszPhysicalName;
    LPSTR           lpszWorkstationName;
    DWORD           dwState;
} WFSDEVSTATUS, * LPWFSDEVSTATUS;

typedef struct _wfs_undevmsg
{
    LPSTR           lpszLogicalName;
    LPSTR           lpszWorkstationName;
    LPSTR           lpszAppID;
    DWORD           dwSize;
    LPBYTE          lpbDescription;
    DWORD           dwMsg;
    LPWFSRESULT     lpWFSResult;
} WFSUNDEVMSG, * LPWFSUNDEVMSG;

typedef struct _wfs_appdisc
{
    LPSTR           lpszLogicalName;
    LPSTR           lpszWorkstationName;
    LPSTR           lpszAppID;
} WFSAPPDISC, * LPWFSAPPDISC;

typedef struct _wfs_hwerror
{
    LPSTR           lpszLogicalName;
    LPSTR           lpszPhysicalName;
    LPSTR           lpszWorkstationName;
    LPSTR           lpszAppID;
    DWORD           dwAction;
    DWORD           dwSize;
    LPBYTE          lpbDescription;
} WFSHWERROR, * LPWFSHWERROR;

typedef struct _wfs_vrsnerror
{
    LPSTR           lpszLogicalName;
    LPSTR           lpszWorkstationName;
    LPSTR           lpszAppID;
    DWORD           dwSize;
    LPBYTE          lpbDescription;
    LPWFSVERSION    lpWFSVersion;
} WFSVRSNERROR, * LPWFSVRSNERROR;

/****** Error codes ******************************************************/
```

```
#define WFS_SUCCESS                               (0)
#define WFS_ERR_ALREADY_STARTED                   (-1)
#define WFS_ERR_API_VER_TOO_HIGH                  (-2)
#define WFS_ERR_API_VER_TOO_LOW                   (-3)
#define WFS_ERR_CANCELED                          (-4)
#define WFS_ERR_CFG_INVALID_HKEY                  (-5)
#define WFS_ERR_CFG_INVALID_NAME                  (-6)
#define WFS_ERR_CFG_INVALID_SUBKEY                (-7)
#define WFS_ERR_CFG_INVALID_VALUE                 (-8)
#define WFS_ERR_CFG_KEY_NOT_EMPTY                 (-9)
#define WFS_ERR_CFG_NAME_TOO_LONG                 (-10)
#define WFS_ERR_CFG_NO_MORE_ITEMS                 (-11)
#define WFS_ERR_CFG_VALUE_TOO_LONG                (-12)
#define WFS_ERR_DEV_NOT_READY                     (-13)
#define WFS_ERR_HARDWARE_ERROR                    (-14)
#define WFS_ERR_INTERNAL_ERROR                    (-15)
#define WFS_ERR_INVALID_ADDRESS                   (-16)
#define WFS_ERR_INVALID_APP_HANDLE                (-17)
#define WFS_ERR_INVALID_BUFFER                    (-18)
#define WFS_ERR_INVALID_CATEGORY                  (-19)
#define WFS_ERR_INVALID_COMMAND                   (-20)
#define WFS_ERR_INVALID_EVENT_CLASS               (-21)
#define WFS_ERR_INVALID_HSERVICE                  (-22)
#define WFS_ERR_INVALID_HPROVIDER                 (-23)
#define WFS_ERR_INVALID_HWND                      (-24)
#define WFS_ERR_INVALID_HWNDREG                   (-25)
#define WFS_ERR_INVALID_POINTER                   (-26)
#define WFS_ERR_INVALID_REQ_ID                    (-27)
#define WFS_ERR_INVALID_RESULT                    (-28)
#define WFS_ERR_INVALID_SERVPROV                  (-29)
#define WFS_ERR_INVALID_TIMER                     (-30)
#define WFS_ERR_INVALID_TRACELEVEL                (-31)
#define WFS_ERR_LOCKED                            (-32)
#define WFS_ERR_NO_BLOCKING_CALL                  (-33)
#define WFS_ERR_NO_SERVPROV                       (-34)
#define WFS_ERR_NO_SUCH_THREAD                    (-35)
#define WFS_ERR_NO_TIMER                          (-36)
#define WFS_ERR_NOT_LOCKED                        (-37)
#define WFS_ERR_NOT_OK_TO_UNLOAD                  (-38)
#define WFS_ERR_NOT_STARTED                       (-39)
#define WFS_ERR_NOT_REGISTERED                    (-40)
#define WFS_ERR_OP_IN_PROGRESS                    (-41)
#define WFS_ERR_OUT_OF_MEMORY                     (-42)
#define WFS_ERR_SERVICE_NOT_FOUND                 (-43)
#define WFS_ERR_SPI_VER_TOO_HIGH                  (-44)
#define WFS_ERR_SPI_VER_TOO_LOW                   (-45)
#define WFS_ERR_SRVC_VER_TOO_HIGH                 (-46)
#define WFS_ERR_SRVC_VER_TOO_LOW                  (-47)
#define WFS_ERR_TIMEOUT                           (-48)
#define WFS_ERR_UNSUPP_CATEGORY                   (-49)
#define WFS_ERR_UNSUPP_COMMAND                    (-50)
#define WFS_ERR_VERSION_ERROR_IN_SRVC             (-51)
#define WFS_ERR_INVALID_DATA                      (-52)
#define WFS_ERR_SOFTWARE_ERROR                    (-53)
#define WFS_ERR_CONNECTION_LOST                   (-54)
#define WFS_ERR_USER_ERROR                        (-55)
#define WFS_ERR_UNSUPPORTED_DATA                  (-56)

#define WFS_INDEFINITE_WAIT                       0

/****** Messages ********************************************************/

/* Message-No = (WM_USER + No) */

#define WFS_OPEN_COMPLETE                         (WM_USER + 1)
#define WFS_CLOSE_COMPLETE                        (WM_USER + 2)
#define WFS_LOCK_COMPLETE                         (WM_USER + 3)
#define WFS_UNLOCK_COMPLETE                       (WM_USER + 4)
#define WFS_REGISTER_COMPLETE                     (WM_USER + 5)
#define WFS_DEREGISTER_COMPLETE                   (WM_USER + 6)
#define WFS_GETINFO_COMPLETE                      (WM_USER + 7)
#define WFS_EXECUTE_COMPLETE                      (WM_USER + 8)

#define WFS_EXECUTE_EVENT                         (WM_USER + 20)
```

```
#define WFS_SERVICE_EVENT                               (WM_USER + 21)
#define WFS_USER_EVENT                                  (WM_USER + 22)
#define WFS_SYSTEM_EVENT                                (WM_USER + 23)

#define WFS_TIMER_EVENT                                 (WM_USER + 100)

/****** Event Classes ***************************************************/

#define SERVICE_EVENTS                                  (1)
#define USER_EVENTS                                     (2)
#define SYSTEM_EVENTS                                   (4)
#define EXECUTE_EVENTS                                  (8)

/****** System Event IDs ************************************************/

#define WFS_SYSE_UNDELIVERABLE_MSG                      (1)
#define WFS_SYSE_HARDWARE_ERROR                         (2)
#define WFS_SYSE_VERSION_ERROR                          (3)
#define WFS_SYSE_DEVICE_STATUS                          (4)
#define WFS_SYSE_APP_DISCONNECT                         (5)
#define WFS_SYSE_SOFTWARE_ERROR                         (6)
#define WFS_SYSE_USER_ERROR                             (7)
#define WFS_SYSE_LOCK_REQUESTED                         (8)

/****** WOSA/XFS Trace Level ********************************************/

#define WFS_TRACE_API                                   0x00000001
#define WFS_TRACE_ALL_API                               0x00000002
#define WFS_TRACE_SPI                                   0x00000004
#define WFS_TRACE_ALL_SPI                               0x00000008
#define WFS_TRACE_MGR                                   0x00000010

/****** WOSA/XFS Error Actions ********************************************/

#define WFS_ERR_ACT_NOACTION                            (0x0000)
#define WFS_ERR_ACT_RESET                               (0x0001)
#define WFS_ERR_ACT_SWERROR                             (0x0002)
#define WFS_ERR_ACT_CONFIG                              (0x0004)
#define WFS_ERR_ACT_HWCLEAR                             (0x0008)
#define WFS_ERR_ACT_HWMAINT                             (0x0010)
#define WFS_ERR_ACT_SUSPEND                             (0x0020)

/****** API functions ***************************************************/

HRESULT extern WINAPI WFSCancelAsyncRequest ( HSERVICE hService, REQUESTID RequestID);

HRESULT extern WINAPI WFSCancelBlockingCall ( DWORD dwThreadID);

HRESULT extern WINAPI WFSCleanUp ();

HRESULT extern WINAPI WFSClose ( HSERVICE hService);

HRESULT extern WINAPI WFSAsyncClose ( HSERVICE hService, HWND hWnd, LPREQUESTID
lpRequestID);

HRESULT extern WINAPI WFSCreateAppHandle ( LPHAPP lphApp);

HRESULT extern WINAPI WFSDeregister ( HSERVICE hService, DWORD dwEventClass, HWND
hWndReg);

HRESULT extern WINAPI WFSAsyncDeregister ( HSERVICE hService, DWORD dwEventClass, HWND
hWndReg, HWND hWnd, LPREQUESTID lpRequestID);

HRESULT extern WINAPI WFSDestroyAppHandle ( HAPP hApp);

HRESULT extern WINAPI WFSExecute ( HSERVICE hService, DWORD dwCommand, LPVOID
lpCmdData, DWORD dwTimeOut, LPWFSRESULT * lppResult);

HRESULT extern WINAPI WFSAsyncExecute ( HSERVICE hService, DWORD dwCommand, LPVOID
lpCmdData, DWORD dwTimeOut, HWND hWnd, LPREQUESTID lpRequestID);

HRESULT extern WINAPI WFSFreeResult ( LPWFSRESULT lpResult);
HRESULT extern WINAPI WFSGetInfo ( HSERVICE hService, DWORD dwCategory, LPVOID
lpQueryDetails, DWORD dwTimeOut, LPWFSRESULT * lppResult);

HRESULT extern WINAPI WFSAsyncGetInfo ( HSERVICE hService, DWORD dwCategory, LPVOID
lpQueryDetails, DWORD dwTimeOut, HWND hWnd, LPREQUESTID lpRequestID);

BOOL extern WINAPI WFSIsBlocking ();

HRESULT extern WINAPI WFSLock ( HSERVICE hService, DWORD dwTimeOut , LPWFSRESULT *
lppResult);

HRESULT extern WINAPI WFSAsyncLock ( HSERVICE hService, DWORD dwTimeOut, HWND hWnd,
LPREQUESTID lpRequestID);

HRESULT extern WINAPI WFSOpen ( LPSTR lpszLogicalName, HAPP hApp, LPSTR lpszAppID,
DWORD dwTraceLevel, DWORD dwTimeOut, DWORD dwSrvcVersionsRequired, LPWFSVERSION
lpSrvcVersion, LPWFSVERSION lpSPIVersion, LPHSERVICE lphService);

HRESULT extern WINAPI WFSAsyncOpen ( LPSTR lpszLogicalName, HAPP hApp, LPSTR
lpszAppID, DWORD dwTraceLevel, DWORD dwTimeOut, LPHSERVICE lphService, HWND hWnd,
DWORD dwSrvcVersionsRequired, LPWFSVERSION lpSrvcVersion, LPWFSVERSION lpSPIVersion,
LPREQUESTID lpRequestID);

HRESULT extern WINAPI WFSRegister ( HSERVICE hService, DWORD dwEventClass, HWND
hWndReg);

HRESULT extern WINAPI WFSAsyncRegister ( HSERVICE hService, DWORD dwEventClass, HWND
hWndReg, HWND hWnd, LPREQUESTID lpRequestID);

HRESULT extern WINAPI WFSSetBlockingHook ( XFSBLOCKINGHOOK lpBlockFunc,
LPXFSBLOCKINGHOOK lppPrevFunc);

HRESULT extern WINAPI WFSStartUp ( DWORD dwVersionsRequired, LPWFSVERSION
lpWFSVersion);

HRESULT extern WINAPI WFSUnhookBlockingHook ();

HRESULT extern WINAPI WFSUnlock ( HSERVICE hService);

HRESULT extern WINAPI WFSAsyncUnlock ( HSERVICE hService, HWND hWnd, LPREQUESTID
lpRequestID);

HRESULT extern WINAPI WFMSetTraceLevel ( HSERVICE hService, DWORD dwTraceLevel);


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

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

#endif  /* __inc_xfsapi__h */
```

10.2 XFSCONF.H

```
/******************************************************************************
*                                                                             *
* xfsconf.h     WOSA/XFS - definitions for the Configuration functions        *
*                                                                             *
*               Version 2.00  --  11/11/96                                    *
*                                                                             *
******************************************************************************/

#ifndef __INC_XFSCONF__H
#define __INC_XFSCONF__H

#ifdef __cplusplus
extern "C" {
#endif

/******* Common *************************************************************/

#include    <xfsapi.h>

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

// following HKEY and PHKEY are already defined in WINREG.H
// so definition has been removed
// typedef HANDLE  HKEY;
// typedef HANDLE * PHKEY;

/******* Value of hKey ******************************************************/
#define     WFS_CFG_HKEY_XFS_ROOT               ((HKEY)1)
#define     WFS_CFG_HKEY_MACHINE_XFS_ROOT       ((HKEY)2)
#define     WFS_CFG_HKEY_USER_DEFAULT_XFS_ROOT  ((HKEY)3)

/******* Values of lpdwDisposition ******************************************/
#define     WFS_CFG_CREATED_NEW_KEY             (0)
#define     WFS_CFG_OPENED_EXISTING_KEY         (1)

/******* Configuration Functions ********************************************/
HRESULT extern  WINAPI  WFMCloseKey ( HKEY hKey);

HRESULT extern  WINAPI  WFMCreateKey ( HKEY hKey, LPSTR lpszSubKey, PHKEY phkResult,
LPDWORD lpdwDisposition);

HRESULT extern  WINAPI  WFMDeleteKey ( HKEY hKey, LPSTR lpszSubKey);

HRESULT extern  WINAPI  WFMDeleteValue ( HKEY hKey, LPSTR lpszValue );

HRESULT extern  WINAPI  WFMEnumKey ( HKEY hKey, DWORD iSubKey, LPSTR lpszName, LPDWORD
lpcchName, PFILETIME lpftLastWrite);

HRESULT extern  WINAPI  WFMEnumValue ( HKEY hKey, DWORD iValue, LPSTR lpszValue,
LPDWORD lpcchValue, LPSTR lpszData, LPDWORD lpcchData);

HRESULT extern  WINAPI  WFMOpenKey ( HKEY hKey, LPSTR lpszSubKey, PHKEY phkResult);

HRESULT extern  WINAPI  WFMQueryValue ( HKEY hKey, LPSTR lpszValueName, LPSTR
lpszData, LPDWORD lpcchData);

HRESULT extern  WINAPI  WFMSetValue ( HKEY hKey, LPSTR lpszValueName, LPSTR lpszData,
DWORD cchData);


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

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

#endif  /* __INC_XFSCONF__H */
```