

# Advanced HART/Fieldbus USB Interface HFI400

## DLL Software Development Application Notes

HART®  
Fieldbus  
Profibus  
Intrinsic Safety  
Configuration Tools  
Semiconductors  
Training  
Custom Design



### Features

- General Overview of HFI400
- Overview of HFI400 hardware
- Detailed discussion of the HFI400 DLL
- Comprehensive list of DLL functions used for interfacing the HFI400 to your HART/Fieldbus/Profibus configuration software
- Example startup, send and receive routines

## General Overview

The Smar Research HFI400 is the first HART/Fieldbus/Profibus to USB interface in the world to support all three protocols simultaneously from the same set of terminals. This breakthrough in technology is a result of Smar Research's rich history in the process control industry and diligent research and development of cutting edge ASIC's. The HFI400 is the first product to use the latest ASIC in the Smar Research family of IC's, the HTFB5050.

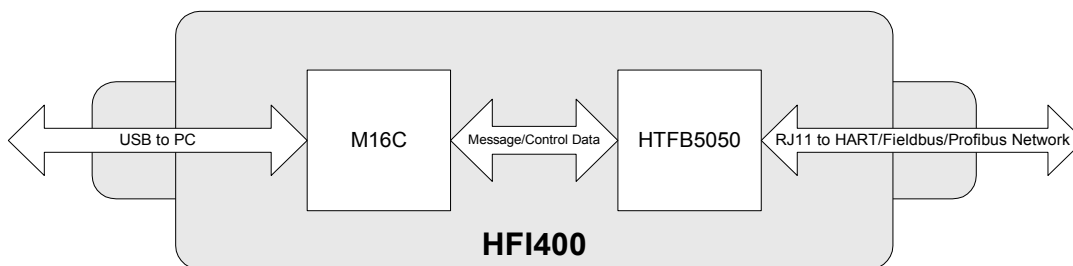
Using the HTFB5050, the HFI400 allows the end user the unprecedented flexibility of interfacing to multiple network protocols. This results in a reduction of in-the-field tools necessary while simultaneously reducing cost. In addition, the HFI400 will grow with the plant as it migrates from analog 4-20mA HART to digital Fieldbus or Profibus communications. From a developer's viewpoint, the HFI400 facilitates an easier integration with your current configuration software packages, as one device, one set of software commands and one DLL can now be used for all HART, Fieldbus and Profibus software packages. This has the added benefit of allowing multiprotocol software packages to use a single hardware device for all protocols.

It is the purpose of this document to provide software developers with the knowledge needed of the HFI400 at the application layer to effectively integrate it into their HART/Fieldbus/Profibus configuration software. With this purpose in mind, this document will only briefly discuss an overview of the physical layer of the HFI400. This will be followed by an in depth discussion of the software methods required to effectively send and receive data and configure the modes of the HFI400 using the DLL functions. The software of the HFI400 was designed with easy integration in mind. Due to this, integration of the HFI400 into configuration software that previously used other devices should be simple and straightforward.

It is assumed that the reader has previous knowledge of the C programming language, DLL interfacing and is familiar with the fundamentals of the HART, Fieldbus and Profibus network protocols. Knowledge of the USB protocol is advantageous but not required. Due to the structuring of the software methods used to interface applications to the HFI400, the USB protocols are invisible to the application developer.

## Hardware Overview

The HFI400 has two main components and two main connectors. The Mitsubishi M16C and the Smar Research HTFB5050 comprise the two main internal components. These components correlate directly to the two main connections of the HFI400, the USB connector and the RJ11 connector to the network. The Mitsubishi M16C handles all USB communications through the USB connector. Incoming and outgoing USB messages are decoded/encoded and processed accordingly inside the M16C. The HTFB5050 serves as the interface to the HART/Fieldbus/Profibus network by encoding outgoing messages in byte format into their appropriate protocol format and decoding incoming messages into byte format to be sent to the M16C. In this way, these two components combined with supporting analog circuitry effectively create the USB to HART/Fieldbus/Profibus network interface. Transmission and reception of messages and control and configuration information to and from the HFI400 are all accommodated through the DLL functions. These commands will be discussed in greater detail in the proceeding sections.



**Fig. 1 – HFI400 Hardware Block Diagram**

## Integrating the DLL into your Project

In order to interface to the HFI400, the following files are required:

**HFI400.dll** - the Dynamic Link Library for the HFI400  
**HFI400.h** - the header file for the HFI400

For a successful integration of the HFI400 into your project, the following guidelines are suggested.

- The header file HFI400.h must be included in your source code (i.e. `#include "HFI400.h"`) and located in the compilers header path
- The HFI400.dll must be in the path (i.e. `<windows system dir>\system32`) or the same directory as the resulting executable
- The DLL must be explicitly linked before use of the DLL functions.

Following these guidelines will facilitate a smooth integration of the HFI400 into your project. A sample piece of code detailing the explicit loading of the HFI400 DLL functions into your project is included in the implementation section of this document. When successfully integrated into your project, these files will provide all necessary methods for sending and receiving message data to and from the HART/Fieldbus/Profibus network as well as all necessary local management functions.

## DLL functions

The HFI400 DLL contains five main functions that are used directly by the developer:

- *HFI Transmit Request* - Transmits a HART command to the network via the HFI400 and receives the device response message. Specific addresses and device commands can be specified here.
- *HFI Read Full Message* - Listens to the HART network and reads any message on the line. This function allows the developer to monitor the line without sending any commands.
- *HFI Local Management* - Used to configure Data Link Layer properties in the HFI400 as per the HART data link layer specification HCF\_SPEC-81 master device management commands. These commands allow the configuration and management of such parameters as re-try limits, master address primary/secondary, burst-frame capture on/off and master frame capture on/off.
- *HFI Open* - Opens a connection between the HFI400 and the application. For use at Start-up. This function must be called before any communication between the HFI400 and the application begins.
- *HFI Close* - Closes the connection between the HFI400 and the application. For use at the close of the application or when communication between the application and the HFI400 is desired to terminate.

These five operations comprise all necessary functions of communication between the application and the HFI400. The specific implementation and syntax of each function will be discussed in greater detail in the following sections.

### NOTE:

As of the publication of this document, the HFI400.dll supports only HART communication. As the Foundation Fieldbus and Profibus functionality is developed, the HFI400.dll will evolve to include additional functions and possibly additional features within existing functions.

## DLL Function `hfiTransmitRequest()`

This function is used to send HART commands to a specified device and receive the device response. After being called, the function returns a structure `RECEIVESTRUCT`, which contains the data returned from the device. The fields of this function correspond as much as possible to the structure of a HART message. Below is the declaration of the `hfiTransmitRequest()` function from the `HFI400.h` header file and a detailed explanation of how to use each of its parameters.

```
RECEIVESTRUCT hfiTransmitRequest(handle hfi_h, int numOfPreambles, unsigned
char* address, unsigned char cmd, unsigned char dataSize, unsigned char* data)
```

### Input Variables

`hfi_h`

This variable is of type `handle` and is generated from the `hfiOpen()` function. This variable is necessary due to cases where more than one HFI400 is being used at a time. In these cases each HFI400 would have a unique handle, each generated by a separate `hfiOpen()` function call. `hfi_h` should contain the handle of the HFI400 you wish to send the request to.

`numOfPreambles`

This variable is an integer value that represents the number of preambles you wish to send with the message onto the HART line. `numOfPreambles` can be any integer value. However, the HART specification strongly recommends a minimum of 5 preambles.

`address`

This variable is a pointer to a 5 byte array that contains the address of the device the request is to be sent to. HART supports two addressing modes, long address and short address. Both addressing modes are supported. Long addresses are composed of 5 bytes (which contain the manufacturer id, device id and extended address) and are to be contained as such in the 5 byte array. Short addresses can be any value between 0 and 15 and should be contained in a 5 byte array as well. The first byte is to be the short address followed by 4 bytes of `0x00`. For example, if you desire to send a request to short address 7, the address array will be declared as such: `unsigned char shortAdd = {0x07,0x00,0x00,0x00,0x00}`

`cmd`

This variable is an unsigned char that contains the HART command number you wish to send. Any valid HART command number can be used here.

`dataSize`

This variable is an unsigned char that contains the data byte count of the data array to be sent. This value should correspond to the length in bytes of the following variable, `data`.

`data`

This variable is a pointer to a byte array of varying length that contains the HART command data bytes to be sent. Dependant on the command number and other parameters, this array may be `NULL`. Refer to the HART specification for specifics of the data contained in this field.

### Output Variable

`RECEIVESTRUCT`

This structure is returned from the `hfiTransmitRequest()` function call. The structure contains the following information: local status, address (5 bytes), command and data. It is defined in the `HFI400.h` header file as:

```
typedef struct {
    unsigned char localStatus;
    unsigned char address[5];
    unsigned char command;
    unsigned char byteCount;
    unsigned char data[128];
} RECEIVESTRUCT;
```

#### NOTE:

The structure of `RECEIVESTRUCT` is designed to mimic the `TRANSMIT.request` message as outlined in the HART HCF\_SPEC-81 : Data Link Layer Specification.

The definitions of each of the `RECEIVESTRUCT` variables are discussed on the following page.

*transmitRequest()* cont'd.

For clarity and example purposes, the following definitions are provided using a sample instance of the RECEIVESTRUCT structure:

```
struct RECEIVESTRUCT received;
```

```
received.localstatus
```

This unsigned char variable contains the local status information from the HF1400. This information is to be used to ensure the success of the response and report any problems. Table 1 contains the list of possible local status values.

localStatus value	Description	Suggested Course of Corrective Action
0	Response Successful	None
1	Timeout occurred (after set # of retries) Message was sent, but no reply was received	Resend Message / Report Error
2	Communication Error Message was sent, but comm. errors detected	Examine first byte of data to determine cause of error (see received.data below)
3	Another Master detected at same address as HF1400 Transmission not possible	Change HF1400 Master address using hfiLocalManagement() function
0xFF	HF1400 in debug mode. data will contain raw HART message. See Table 4 for more information	If debug mode not desired, use hfiLocalManagement() function to stop debug read
7	Unable to send data / lost connection / invalid handle	Close current HF1400 handle and open another

**Table 1 – Summary of localStatus values for hfiTransmitRequest()**

```
received.address
```

This variable is an 5 byte unsigned char array that contains the long address of the HART device the received message is from. The long address is composed of 5 bytes (which contain the manufacturer code, device type and device id) as per the HART specification.

```
received.command
```

This variable is an unsigned char containing the HART command number of the message from the device.

```
received.byteCount
```

This variable is an unsigned char that contains the data byte count of the received data array. This value will correspond to the length in bytes of the following variable, data.

```
received.data
```

This variable is an unsigned integer array of 128 bytes that contains the HART command data bytes received from the HART device. Dependant on the command number and other parameters, this array may be NULL. Refer to the HART specification for specifics of the data contained in this field. If the HF1400 is in debug mode, data will contain the entire raw HART message including preambles and checksum bytes.

If a communication error was detected (localStatus = 2), the first byte of data will contain status information on the cause of the communication error. These values are summarized in Table 2 below.

Bit Number	Description
Bit 7	Equals 1 when there is any communication error present (localStatus = 2)
Bit 6	Parity error
Bit 5	Overrun error
Bit 4	Framing error
Bit 3	Checksum error
Bit 2	(reserved)
Bit 1	RX buffer overflow
Bit 0	(undefined)

**NOTE:**  
The communication error status byte follows the status coding of the HART specification

**Table 2 – Summary of the first byte in data when localStatus = 2**

## DLL Function *hfiReadFullMessage()*

This function is used to read any message stored in the HFI400 buffer from the HART network. This allows “quiet” monitoring of the HART network by the application. After being called, the function returns the structure `RECEIVESTRUCT`, which contains HART data captured from the network. Below is the declaration of the `hfiReadFullMessage()` function from the `HFI400.h` header file and a detailed explanation of how to use each of its parameters.

```
RECEIVESTRUCT hfiReadFullMessage(handle hfi_h)
```

### Input Variables

`hfi_h`

This variable is of type `handle` and is generated from the `hfiOpen()` function. This variable is necessary due to cases where more than one HFI400 is being used at a time. In these cases each HFI400 would have a unique handle, each generated by a separate `hfiOpen()` function call. `hfi_h` should contain the handle of the HFI400 you wish to send the request to.

### Output Variable

`RECEIVESTRUCT`

This structure is returned from the `hfiReadFullMessage()` function call. The structure contains the following information: local status, address (5 bytes), command and data. The `RECEIVESTRUCT` structure is discussed in detail in on the previous pages under the `hfiTransmitRequest()` output variable section. The structure and definitions of each sub-variable are the same with the exception of the local status definitions. To avoid excessive redundancy, only the `localStatus` definitions for the `hfiReadFullMessage()` function will be contained here. For full details of `RECEIVESTRUCT`, refer to the previous pages 4-5.

	Description	Values
Bit 7	Indicates if the message is a burst mode message (HART <code>cycle.indicate</code> ) or a receive indicate (HART <code>receive.indicate</code> ) message.	1 = burst mode message 0 = receive indicate message
Bit 6	Indicates if there is another master on the HART network with the same address as the HFI400	1 = message is from a master with same address 0 = message is not from a master with same address
Bits 5-0	These 6 bits report various states of the HFI400 receive buffer	4 = (000100) = message returned with more messages waiting in the HFI400 buffer 5 = (000101) = message returned with no more messages waiting in the HFI400 buffer 6 = (000110) = no message returned / no messages available 7 = (000111) = unable to send data / lost connection / invalid handle

**Table 3 – Summary of `localStatus` values for `hfiReadFullMessage()`**

#### NOTE:

If the HFI400 is in debug mode, `localStatus` will always return `0xFF`. In this mode, the data field of `RECEIVESTRUCT` will contain the raw HART message bytes. All preambles and checksum bytes will be included in the message with no deciphering done by the HFI400. To enter or exit the debug mode, use the `hfiLocalManagement()` function. For more details, see the `hfiLocalManagement()` section of this document.

## DLL Function *hfiLocalManagement()*

This function is used to configure the Data Link Layer properties of the HFI400 as per the HART Data Link Layer specification HCF\_SPEC-81. The HART specification defines certain master device management commands that must be contained in all HART compliant master devices. These commands allow the configuration and management of such parameters as re-try limits, master address primary/secondary, burst-frame capture on/off and master frame capture on/off. After being called, the function returns a boolean that describes the success of the transfer. Below is the declaration of the `hfiLocalManagement()` function and a detailed explanation of how to use each of its parameters.

```
BOOL hfiLocalManagement(handle hfi_h, unsigned char serviceReq, unsigned char param)
```

### Input Variables

`hfi_h`

This variable is of type `handle` and is generated from the `hfiOpen()` function. This variable is necessary due to cases where more than one HFI400 is being used at a time. In these cases each HFI400 would have a unique handle, each generated by a separate `hfiOpen()` function call. `hfi_h` should contain the handle of the HFI400 you wish to send the request to.

`serviceReq`

This variable contains the value of the device management command you wish to execute. Table 4 contains a detailed list of each device management command and its corresponding service request number.

`param`

This variable contains any command specific data byte to be sent. Table 4 contains details of the specific values of `param` for each different service request. For device management commands that have no parameters, `param` should be NULL and will be ignored.

Service	Description	serviceReq	param
Set Retry Limit	Sets the number of transmit retries the HFI400 will perform. By default the HFI400 will perform 3 retries, per the HART spec., before returning an error to the Application Layer (*Default = 3)	0x10	# of retries
Set Master Address	Configures the HFI400 to operate as either a Primary HART Master or Secondary HART Master (*Default = 2)	0x11	1 = primary 2 = secondary master
Capture Burst Frames	Sets the HFI400 to allow all incoming HART burst messages to be read (*Default)	0x12	NULL
Ignore Burst Frames	Sets the HFI400 to ignore all incoming HART burst messages	0x13	NULL
Capture Other Master Frames	Sets the HFI400 to allow all HART master frames from Masters other than the HFI400 to be read (*Default)	0x14	NULL
Ignore Other Master Frames	Sets the HFI to ignore incoming Master frames not intended for the HFI400	0x15	NULL
Start Debug Read	Puts the HFI400 in debug mode. All subsequent messages returned from the device will be in the format exactly seen on the HART line. All preambles and checksum bytes will be included in the message with no deciphering done by the HFI400	0x16	NULL
Stop Debug Read	Exits debug mode (*Default)	0x17	NULL

**Table 4 – Summary of Service Requests**

### Output Variable

The `hfiLocalManagement()` function returns the boolean variable to indicate to success or failure of the transfer. A value of TRUE indicates the transfer was a success. A value of FALSE indicates the transfer failed. In the case of a failed transfer, the management request should either be re-submitted, and/or an error message should be reported to the user.

## DLL Function *hfiOpen()*

This function is used to open a line of communication between the application and an HFI400 at a specific address. The default HFI400 device address is 0. Before the application can send or receive any network communications from the HFI400 this function must be called, therefore it is suggested that this command be included in the startup section of code. This variable is necessary due to cases where more than one HFI400 is being used at a time. The `hfiOpen()` function allows each HFI400 to have a unique handle, each generated by a separate `hfiOpen()` function call. After being called, `hfiOpen()` returns a handle that is device specific, to be used as a parameter for the other HFI400 DLL functions. `hfiOpen()` should be used in conjunction with `hfiClose()` upon termination of the application. Below is the declaration of the `hfiOpen()` function and the definition of each of its parameters.

```
HANDLE hfiOpen(int hfiAddress)
```

### **Input Variable**

`hfiAddress`

This variable is an integer that contains the HFI400 address you wish to open a connection with. Up to 10 devices are supported on the same USB hub (addresses 0-9). The default HFI400 address is 0.

### **Output Variable**

The `hfiOpen()` function returns a handle to be used as a parameter for other HFI400 DLL function calls. The resulting handle is to be used as a parameter for the other HFI400 DLL functions, `hfiLocalManagement()`, `hfiTransmitRequest()` and `hfiReadFullMessage()` to ensure the commands are being transmitted to the intended HFI400.

#### NOTE:

The HFI400 address is generated automatically by the HFI400. The default HFI400 address is 0. However, this address may be different when used on computers with multiple USB ports and/or with multiple USB devices connected. It is recommended to poll through addresses 0-9 to verify each HFI400 address.

## DLL Function *hfiClose()*

This function closes the connection between the application and a HFI400 and is to be used when communication between the application and the HFI400 is desired to terminate. Each open HFI400 is given a unique handle by the `hfiOpen()` function. This handle is used to specify which HFI400 you wish to close when calling `hfiClose()`. Below is the declaration of the `hfiClose()` function and the definition of each of its parameters.

```
hfiClose(handle hfi_h)
```

### **Input Variable**

`hfi_h`

This variable is of type handle and is generated from the `hfiOpen()` function. The handle of the specific HFI400 you wish to terminate communications with should be contained here. This variable is necessary due to cases where more than one HFI400 is being used at a time. In these cases each HFI400 would have a unique handle, each generated by a separate `hfiOpen()` function call.

### **Output Variable**

The `hfiClose()` function has no output variables. Nothing is returned from this function. Upon completion of this function call, communications with the specified HFI400 can be assumed closed.

#### NOTE:

Each instance of `hfiOpen()` should have a corresponding `hfiClose()` before termination of the application.

## Implementation

In the previous section, the HFI400 DLL functions and variables were explained. In this section, sample code will further clarify how these commands are used together to fully implement the HFI400 into your application. In particular, this section will detail a Startup routine for initializing the DLL and HFI400, a transmit routine for sending a HART command to a device and receiving the response, and local management call, a read full message monitor of the HART line, and a termination of the HFI400 connection routine. The files HFI400.h and HFI400exApp.cpp, included with this document, also contain sample code to be used in aiding the development of your application. HFI400.h should be included in any project using the HFI400 DLL functions.

### Startup

At the start of the application, or before beginning communication with the HFI400, the HFI400 DLL must be explicitly loaded and communication with the HFI400 must be established. In addition, it is recommended that the data link layer properties of the HFI400 be set at this time. Setting these properties at startup is not required and can be done at any time. The default values of the HFI400 can be seen in Table 4. A sample startup routine in pseudo code is shown below.

```
#include "HFI400.h"

HINSTANCE hDLL=NULL;           // Handle to the HFI400.dll library
HANDLE hfi_h=NULL;           //Handle to the actual HFI400 interface

typedef RECEIVESTRUCT (* HFITREQUEST) (HANDLE,int,unsigned char*,unsigned char,unsigned
char,unsigned char* );
HFITREQUEST hfitransmitrequest; //Function pointer

typedef HANDLE (* HFIOOPEN) (int);
HFIOOPEN hfiopen; //Function pointer

typedef void (* HFICLOSE) (HANDLE);
HFICLOSE hficlose; //Function pointer

typedef RECEIVESTRUCT (* HFIFMESSAGE) (HANDLE);
HFIFMESSAGE hfireadfullmessage; //Function pointer

typedef BOOL (* HFIFILM) (HANDLE, unsigned char,unsigned char);
HFIFILM hfilocalmanagement; //Function pointer

//LOAD HFI400.DLL FUNCTIONS
hDLL = LoadLibrary("HFI400");
if (hDLL != NULL) //HFI400 loaded sucessfully
{
    //get references for the functions defined on the DLL
    hfitransmitrequest = (HFITREQUEST)GetProcAddress(hDLL, "hfiTransmitRequest");
    hfiopen = (HFIOOPEN)GetProcAddress(hDLL, "hfiOpen");
    hfireadfullmessage = (HFIFMESSAGE)GetProcAddress(hDLL, "hfiReadFullMessage");
    hfilocalmanagement = (HFIFILM)GetProcAddress(hDLL, "hfiLocalManagement");
    hficlose = (HFICLOSE)GetProcAddress(hDLL, "hfiClose");

    if (hfitransmit==NULL || hfiopen==NULL || hfilocalmanagement==NULL
        || hfifullmessage==NULL || hficlose==NULL)
    {
        printf("one or more specified procedure could not be found,\n");
        printf("please check if you have the latest HFI400 DLL version\n");
        //add any error handling here
        FreeLibrary(hDLL);
        return 0; //return an error code
    }
}
else //HFI400 DLL library not found. Check if HFI400.DLL is on path

//OPEN CONNECTION WITH DFI400
hfi_h = hfiopen(0); //open a connection to the HFI400 at address 0
if(hfi_h == NULL){ //no connection found
    printf("could not establish connection to the HFI400 at address 0 \n");
    printf("please check if HFI400 is connected to the USB port or enabled.\n");
    return 0;} //return an error code
```

## Send Message / Receive Response Routine

The most frequent interaction between the application and the HFI400 is a send and receive routine, especially for master slave protocols such as the HART specification. When sending a message and then receiving the device reply via the HFI400 is required, the application should follow a routine similar to the sample code below. After sending the message and receiving the response, the `localStatus` should be checked for any errors in the transmission of the message.

```
//Send Message / Receive Response Routine

//setup message to be sent
RECEIVESTRUCT recvStr;
unsigned char address[5]={0x00,0x00,0x00,0x00,0x00};
int i;
address[0] = 1;//short address 1

//send command 0, 5 preambles, to short address 1, no data bytes
recvStr = hfitransmitrequest(hfi_h,5,address,0,0,NULL);

//check localStatus for any transmission errors
if(recvStr.localStatus == 0x01){
    //timeout occurred-resend and/or report error here
}
elseif(recvStr.localStatus == 0x07){
    //invalid handle or unable to send data or lost connection
}

//if we have gotten this far, message was sent with success
//The message in recvStr can now be deciphered and processed
```

## Local Management Call

In order to change the internal settings and parameters of the Data Link Layer of the HFI400, a local management call must be performed. This call is straightforward and simple to perform. A list of the local management parameters can be found in table 4 in the `hfiLocalManagement()` section of this document.

```
//Local Management Call

//this call will set the HFI400 to be a master primary
hfilocalmanagement(hfi_h,SET_MASTER_ADDRESS,1); //constants defined in HFI400.h header file
```

## Monitor the Line

This function is used to read any message stored in the HFI400 buffer from the HART network. This allows “quiet” monitoring of the HART network by the application. If continuous monitoring is desired, `hfiReadFullMessage()` must be continuously called. It is left up to the user to determine what will be done with the returned message from the line.

```
//Monitor the HART line

RECEIVESTRUCT recvStr;
recvStr = hfi_read_full_message(hfi_h);

//check if message was received
if(recvStr.localStatus != HFINOMESSAGE) //check localStatus to see if message retrieved
{
    printf(" Data received: %d",recvStr.byteCount);
    if(recvStr.byteCount != 0) //if message contains data bytes, print...
        printData(recvStr.byteCount,recvStr.data);
    printf("\n");
    //process the data here
}
```

## Close HFI Connection

Closing communication with the HFI400 is suggested at the termination of the application. This is accomplished by simply calling the `hfiClose` function for with the handle of the desired HFI400 as shown below.

```
if(hfi_h != NULL)
    hfiClose(hfi_h);
```

---

Smar Research reserves the right to make changes to design and functionality of any product without notice. Smar Research does not assume any liability arising out of the application or use of any product. Smar Research, Technology Source, and the SRC logo are registered trademarks of Smar Research Corporation. The HART, Fieldbus, and Profibus Foundation logos are trademarks of their respective owners.

## **Smar Research Corporation**

4250 Veterans Memorial Highway  
Holbrook, NY USA 11741  
Tel: 631.737.3111 Fax: 631.737.3892  
techinfo@SmarResearch.com  
www.SmarResearch.com



© Smar Research Corporation



11



HFI400DLLAPP - 0504