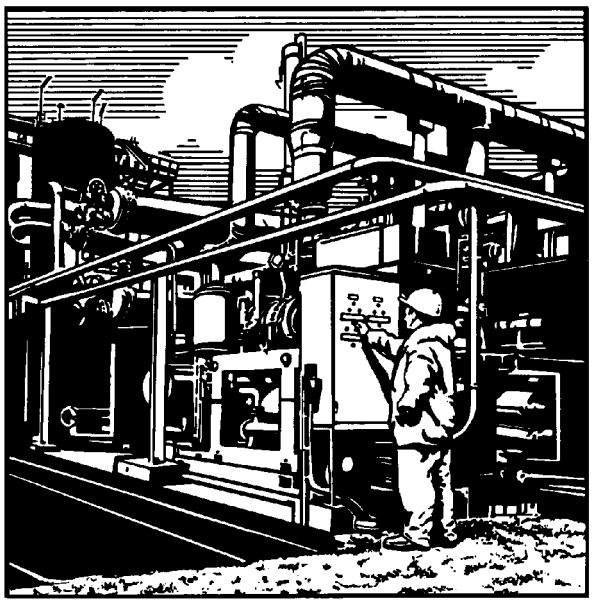
March 25, 1999
Version 1.1

Specification for

Field Calibrator Interface



Prepared by:

Kenneth Holladay
Danny Lents

SOUTHWEST RESEARCH INSTITUTE
Instrumentation and Space Research Division
6220 Culebra Road • San Antonio, Texas 78238-5166
(210) 684-5111 • FAX: (210) 647-4325

Table of Contents

1. Introduction

1.1 Background

The worldwide market for process control instruments has experienced excellent growth over the past several years. These instruments are used throughout manufacturing processes where the need for maintaining measurement accuracy while minimizing out of service time creates a demanding challenge.

A trend that has produced significant savings and reduced down time is in situ (field) calibration. Manufacturers of calibration equipment responded to this development by designing portable, intelligent, field calibrators. These calibrators often include a communication interface so that a Calibration Management System (CMS) program can download one or more test procedures into the calibrator. A technician then travels to each instrument location and uses the downloaded procedures to test and adjust each instrument without having to remove it from service. The test results are automatically stored in the calibrator for later retrieval by the CMS software.

1.2 Problem Description

Unfortunately, the calibrator manufacturers and calibration management software developers never developed any standards to achieve this common goal of downloading procedures and uploading results. Today's marketplace is fractured with a variety of proprietary communication protocols and program interfaces. Every CMS developer that wants to support these calibrators must write a different driver for each one. Every calibrator manufacturer that wants its hardware supported by a calibration management system must provide individual support to the developer.

1.3 Proposed Solution

This situation could be significantly improved by defining a standard interface between the calibration equipment and the calibration management software. To be successful, it must accomplish several goals.

1. It must be a binary standard so that both the driver developer and the CMS developer are free to use whatever language is most suitable, including Visual C++, Visual Basic (VB), Delphi, Power Builder, etc.

2. It must provide a semantically clear interface where data items such as unit codes and thermocouple probe types can be communicated in a completely unambiguous format.
3. It must provide a flexible data transfer mechanism that can be adapted by the driver to fit the format used by the calibrator hardware.
4. It must encompass sufficient data exchange to satisfy calibration management requirements, while being flexible enough to include the most common optional data fields.

This document describes the Field Calibrator Interface (FCINTF) specification designed to fulfill the above requirements. It is based on the Microsoft Component Object Model technology, and is thus a language independent binary specification. It uses tables of data originally developed for the HART communication protocol, thus providing unambiguous semantics.

The proposed FCINTF standard was originally presented to the Field Calibration Technology Committee (FCTC) of the Instrument Society of America (ISA) in May 1998. There was significant interest expressed in the specification, and several committee members volunteered to review the specification. This document presents the final results of that effort with formal release 1.0.

By using this specification, each calibrator manufacturer can develop a single driver that provides a uniform interface to any calibration management software application that also implements the specification.

The field calibrator interface is defined by three documents:

FCINTF.IDL	The Interface Definition Language file
INTEROP.H	Definitions for semantically unambiguous data based on HART tables.
FCINTF.DOC	This document

1.4 Audience

This specification is intended as reference material for developers of FCINTF compliant Clients (CMS) and Servers (Calibrator drivers). It is assumed that the reader is familiar with Microsoft OLE/COM technology and the needs of Field Calibration.

1.5 References

Rogerson, Dale E., Inside COM, Microsoft Press, Redmond, WA, 1997.

Brockschmidt, Kraig, Inside OLE, Microsoft Press, Redmond, WA, 1995.

1.6 Terminology

CMS Calibration Management System.

COM Component Object Model.

Client In this document, the client is usually the calibration management software.

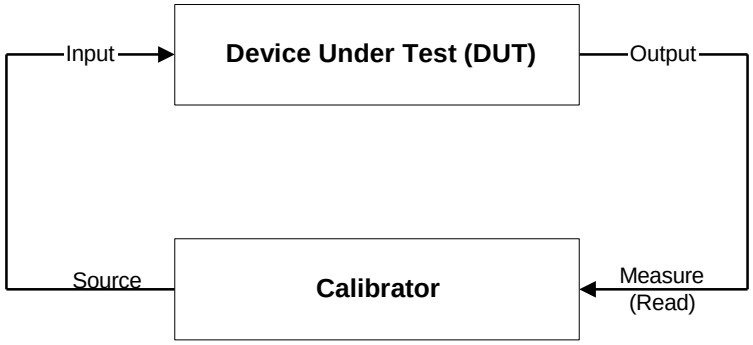
Driver In this document, the driver is the COM server component that provides the interface to the calibration hardware. The terms driver, component, and server may be used interchangeably.

Method The functions that can be called on an interface are often referred to as methods. In this document, method and function are used interchangeably.

Output In this document, output refers to the signal from the instrument (or device under test) that is measured by the calibrator.

Input In this document, input refers to the signal that goes into the instrument, often generated (or sourced) by the calibrator.

Figure 1 - Calibrator and Device Connection



2. FCINTF Implementation Fundamentals

2.1 FCINTF Component Object Model Overview

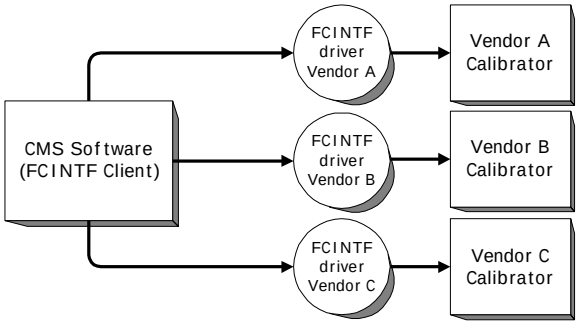
The Field Calibrator Interface is based on Microsoft's Component Object Model (COM). Since COM is a specification detailing a binary standard for building components, a COM interface contains no code. It is a specification, written on paper, which defines how a client and a server can communicate with each other.

The FCINTF Specification defines COM interfaces (what the interfaces are), not the implementation (not the how of the implementation) of those interfaces. It specifies

the behavior that the interfaces are expected to provide to the client applications that use them.

An FCINTF client can connect to a FCINTF driver provided by one or more vendors. Vendor supplied code in each driver determines how the server physically accesses the calibrator.

Figure 2 - Relationship between FCINTF Client and Drivers



Also, COM components can be loaded by an application at run time. This is a tremendous advantage for CMS software. Adding a new field calibrator driver is as simple as copying the driver files and registering the driver.

2.2 Calibrator Object and Interfaces

2.2.1 Anatomy of an Object

FCINTF defines only a single calibrator object with three interfaces. An FCINTF client connects to and communicates with the FCINTF driver through the interfaces.

Figure 3 - FCINTF Server Object

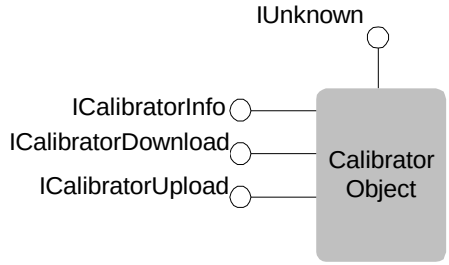


Table 1 FCINTF Functions

ICalibratorInfo	DriverProperties() Open() Close() Properties() GetId() GetCalDates() SetDateAndTime() SetTempStandard() GetTestResultsCount() ClearMemory() ValidateTag()
ICalibratorDownload	StartDownloading DownloadTag() FinishDownloading() AbortDownload()
ICalibratorUpload	StartUploading() UploadNextTag() FinishUploading() AbortUpload()

2.2.2 Finding an Object

The COM specification relies very heavily on Globally Unique Identifiers, or GUID's. These are 16 byte (128bit) identifiers that are assigned to interfaces, components, and categories.

To find FCINTF compatible drivers on a system, the client application can query the registry for all components that support the Field Calibrators category (defined by the GUID "FF5EFD40-7B15-11D1-B326-00001CBE02A").

Each component (FCINTF driver) that is listed by this request will have its own GUID and will be in the registry in a format similar to the following:

```
[HKEY_CLASSES_ROOT\CLSID\{FF5EFD47-7B15-11D1-B326-00001CBE02AA}]
@="Fluke 702 Calibrator"

[HKEY_CLASSES_ROOT\CLSID\{FF5EFD47-7B15-11D1-B326-00001CBE02AA}\InprocServer32]
@="E:\\FIELDCALIBRATORS\\FLK702\\BIN\\FLK702.DLL"

[HKEY_CLASSES_ROOT\CLSID\{FF5EFD47-7B15-11D1-B326-00001CBE02AA}\ProgID]
@="Fluke702.1"

[HKEY_CLASSES_ROOT\CLSID\{FF5EFD47-7B15-11D1-B326-00001CBE02AA}\VersionIndependentProgID]
@="Fluke702"
```

```
[HKEY_CLASSES_ROOT\CLSID\{FF5EFD47-7B15-11D1-B326-00001CBE02AA}\Implemented
Categories\{FF5EFD40-7B15-11D1-B326-00001CBE02AA}]

[HKEY_CLASSES_ROOT\CLSID\{FF5EFD47-7B15-11D1-B326-00001CBE02AA}\Data]
"CommPort"=dword:00000001
"CommRate"=dword:00002580
"Retries"=dword:00000003
"RetryTimeoutMs"=dword:000003e8
"PctMemUsable"=dword:00000032
```

2.2.3 Interface Identifiers (IID)

As with all COM objects, each interface defined by FCINTF is defined by a GUID (see Appendix A). Having defined interface definitions is critical to the COM philosophy. Once an interface has been published, that interface will never change. A new IID will be generated if an interface is updated.

After creating an instance of the COM server, the client can use standard COM functions to obtain pointers to the FCINTF interfaces: IUnknown, ICalibratorInfo, ICalibratorDownload, and ICalibratorUpload.

2.3 Local vs. Remote Servers

When building an FCINTF compliant server, there are a number of simplifying assumptions which fit this type of operation.

- It is very unlikely that multiple CMS applications (clients) will need to connect simultaneously to a single calibrator driver.
- It is most likely that the calibrator will be physically connected to the same computer that is running the CMS software.

Under these conditions, the most likely configuration for the driver will be an in-process COM object built as a dynamic link library. This does not preclude a vendor from using an executable process to provide a local or remote server, it simply implies that this level of complication is probably not necessary to accomplish the goals defined for the FCINTF.

2.4 UNICODE, NT and WIN95

All string parameters to the FCINTF Interfaces are UNICODE, because the native OLE APIs are all UNICODE.

At the time of this writing, MIDL 3.0 or later is required in order to correctly compile the IDL code and generate proxy/stub software. Microsoft Windows NT 4.0 (or later), or

Windows 95 with DCOM support is required to properly handle the marshaling of FCINTF parameters.

2.5 Threads and Multitasking

This specification does NOT require any particular threading model for the server. For in-process (DLL) servers, the threading model must be specified in the registry. For FCINTF drivers, it is unlikely that multiple thread support will result in any significant performance improvement.

An advantage to a single threaded approach is that it simplifies implementation of servers with respect to reentrancy issues. Since all method calls are serialized automatically by the message loop, methods are never reentered or interrupted by other methods. Another advantage is that it insures (as required by COM) that the thread which created the object is the only thread accessing the object.

If anything other than single threading is used, it is up to the driver vendor to ensure that access to all internal data is appropriately protected.

2.6 Persistent Storage

This revision of the specification assumes that the driver does not require any persistent storage relating to its communication with a calibrator. All information storage is the responsibility of the calibration management system software.

Simple driver related data items such as the communication rate or number of communication retries should be stored in the Windows registry.

2.7 Memory Management

COM interfaces define the parameters and return values passed between client and server. The field calibrator interface is defined in FCINTF.IDL such that the responsibilities of the client and the server are very clear.

Because the client application and FCINTF servers are developed independently, there must be strict rules governing the allocation and de-allocation of memory used to pass parameters across the COM interface. There are no special rules for *internal* memory allocation, so new/delete or malloc/free should be used for everything that does not pass through a COM interface.

The COM functions CoTaskMemAlloc() and CoTaskMemFree() are used for allocating and freeing memory that passes through the COM interface. It's important to note a C++ object's constructor is not automatically called if the object is dynamically allocated with

CoTaskMemAlloc(). Likewise, freeing a C++ object with CoTaskMemFree() does not call the destructor.

Each parameter of a function is classified as an **in parameter** or an **out parameter**. A third classification, an **in-out parameter**, is not used by this interface. In each class of parameter, the responsibility for allocating and freeing *dynamically allocated* parameters is as follows:

- **[in] parameter** must be allocated and freed by the client.
- **[out] parameter** must be allocated by the driver and freed by the client.
  A dynamically allocated [**out**] parameter requires special consideration. These parameters are usually passed as a pointer-to-pointer. Also note that a structure passed across a COM interface may contain a member that is a pointer to receive memory allocated by the server and must be freed by the client.

  Dynamically allocated parameters that are passed through a COM interface must be explicitly set to NULL or assigned a pointer to memory allocated by the server using CoTaskMemAlloc().

  As a precaution, the client should set a dynamically allocated [out] parameter to NULL before calling the interface function even though the server is expected to set the parameter to NULL if an error is encountered.

  The client must use CoTaskMemFree() to free memory that was dynamically allocated by the server. Attempting to call CoTaskMemFree() on a NULL pointer causes a memory access violation. In addition, the client should set the pointer to NULL after freeing it, as illustrated below.

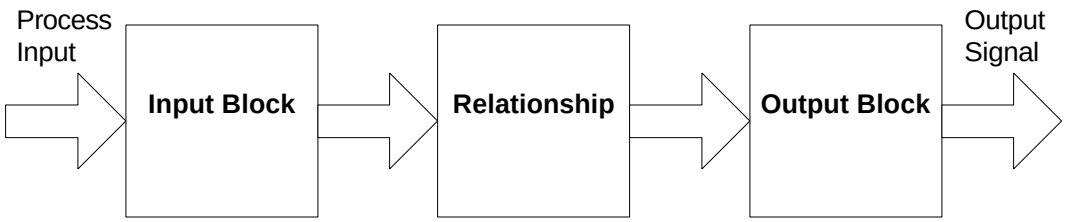
```
if( szwLocationInCalibrator != NULL )
{
        CoTaskMemFree( szwLocationInCalibrator );
        szwLocationInCalibrator = NULL;
}
```

3. FCINTF Model

3.1 Block Diagram

To achieve a high degree of flexibility, the field calibrator interface assumes that any field device that requires calibration can be defined by describing three blocks, as illustrated below.

Figure 4 - Instrument Block Diagram



3.1.1 Input Block

The input block defines the input to the instrument. For a transmitter, this might be pressure, temperature, flow, etc. Separate input block definitions are included in the FCINTF for each type of input that requires special configuration information. The goal was to capture only the information that a calibrator would require to perform tests and adjustments. For example, a temperature input that uses a thermocouple has different setup information than a pressure input. If no special setup information is required, a generic block type is provided.

For more detailed information, look under the "I/O Block Types" heading in appendix A.

3.1.2 Output Block

The output block defines the type of output signal produced by the instrument. A transmitter might produce a 4 to 20 milliamp signal. An I to P converter has a pressure output, while a switch has only a pair of contacts.

3.1.3 Relationship Block

The relationship block defines how the process input is converted to an output signal. There are two basic categories of input to output relations: proportional and switch.

Proportional instruments produce a continuous output signal that is proportional to the input signal. For the majority of these instrument instruments, this proportionality is linear. However, in this age of digital electronics, this is not always the case. For example, there are many differential pressure transmitters that use a square root relation, which when used with an orifice plate, produces a signal that is linearly proportional to flow.

The switch relation is only relevant when the output block type is a switch. This is for discreet or on/off devices.

3.2 Test Definitions

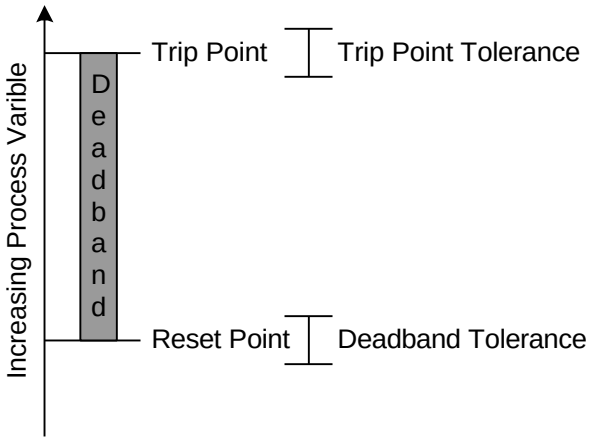
3.2.1 Proportional Tests

The "tagFCI_TEST_PROPORTIONAL" data structure defined in appendix A defines the information necessary to conduct a test on a proportional instrument. The items in this structure were defined to follow generally accepted industrial calibration practice as far as possible.

3.2.2 Switch Tests

There is less consensus in the industry when defining the parameters for a switch. Figure 5 illustrates the relation between the switch parameters used by the FCINTF for a switch that is defined to trip on an increasing signal. For a decreasing signal, simple invert everything.

Figure 5 - Switch Test Parameters



3.3 Parameter Semantics

Another problem that must be addressed by this type of interface is how to achieve semantically unambiguous definitions that can be translated by both the client and the driver. For example, how do we specify a thermocouple type for a temperature input block? How do we specify units? There is no common ground in the calibration industry for these items. If we use strings, how will the computer recognize that "TC J", "Type J", "Thermocouple J", and "Copper-Constantan" all refer to the same thing?

The solution adopted for this interface was to use a set of tables where the numeric entries in the tables are unambiguously defined, and where the tables are managed by a

controlling organization. The required tables were adopted from the HART Communication Foundation, and are included in Appendix B as the INTEROP.H file.

4. FCINTF Reference

4.1 Return Codes

4.1.1 HRESULT

All methods on all three FCINTF interfaces use an HRESULT return code. Since COM often uses this result to indicate operating system failures, no function related status is returned here. All functions should return S_OK. However, the client must check the return code since any other value than S_OK indicates a failure.

4.1.2 pStatus [out] Parameter

All methods on all three FCINTF interfaces have an [out] parameter that returns a status value to indicate the success or failure of the operation. The specific values for this status are taken from the FCISTATUS enumeration near the top of the FCINTF.IDL file.

Table 2 - Summary of pStatus Return Codes

Return Code	Description
FCI_OK	No error.
FCI_FAILED	The function failed for an unspecified reason.
FCI_COMM_ERROR	Unable to communicate with the calibrator.
FCI_REVISION_ERROR	The calibrator driver revision is incompatible with this calibrator.
FCI_CANCELED	The operation was canceled.
FCI_NOT_SUPPORTED	This calibrator does not support this function.
FCI_ALREADY_OPEN	There is already a calibrator open.
FCI_WRONG_SESSION	The session identifier is incorrect.
FCI_WRONG_CALIBRATOR	The calibrator currently attached is not the one associated with this session.
FCI_ERR_IN_RANGE	The input range is not supported by this calibrator.
FCI_ERR_IN_UNITS	The input engineering unit is not supported by this calibrator.
FCI_ERR_IN_TYPE	The input type is not supported by this calibrator.
FCI_ERR_IN_MANUAL	This calibrator does not support manual entry for input.
FCI_ERR_OUT_RANGE	The output range is not supported by this calibrator.
FCI_ERR_OUT_UNITS	The output engineering unit is not supported by this calibrator.

Return Code	Description
FCI_ERR_OUT_TYPE	The input type is not supported by this calibrator.
FCI_ERR_OUT_MANUAL	This calibrator does not support manual entry for output.
FCI_ERR_INOUT	Invalid combination of input and output types for this calibrator.
FCI_ERR_RELATION	The input to output relation is not supported by this calibrator.
FCI_ERR_PROBE	The temperature probe is not supported by this calibrator.
FCI_ERR_CJC	The cold junction compensation type is not supported.
FCI_ERR_ENUM	A given enumeration is not a valid value.
FCI_ERR_RESOURCE	Driver was unable to load a required resource.
FCI_ERR_TESTPOINT	Error in one or more test points. Out of range, not stable, or not valid.
FCI_ERR_TESTTYPE	The test type does not match the instrument or is not supported.
FCI_ERR_POWER	The loop power configuration is inconsistent or not supported.
FCI_ERR_TAG	A tag is required for this operation.
FCI_MEM_FULL	The calibrator memory is full.
FCI_MEM_EMPTY	The calibrator memory is empty.
FCI_MEM_HAS_DATA	The calibrator memory contains test data.
FCI_END_OF_UPLOAD	End of upload list has been reached.
FCI_ERR_DOWNLOAD	Error encountered when trying to finalize download.
FCI_ERR_TAGLENGTH	The tag length sent exceeds the length supported by this driver.
FCI_ERR_SNLENGTH	The serial number length exceeds the length supported by this driver.
FCI_ERR_MAXPOINTS	There were more AF or AL test points than could be uploaded in one packet.

4.2 ICalibratorInfo

The ICalibratorInfo interface has functions to collect information from the calibrator and any other calibrator-related functions that don't fit within the categories of Download or Upload.

4.2.1 Driver Properties()

```
HRESULT DriverProperties( [out] FCISTATUS *pStatus );
```

Description

Calling this method usually results in the display of a Driver Properties dialog box. The purpose of this dialog is to provide a user interface for defining driver related data that does not require communication with the calibrator. Any data in this box should be saved using the Windows registry.

It is recommended that the dialog display useful information such as:

driver software version number
communications baud rate (possibly a user-edited field)
number of retries (user-edited field)
retry timeout (user-edited field)

It is recommended that this dialog have *Ok* and *Cancel* buttons, with an optional *Help* button.

Parameters

Parameters	Description
pStatus	Receives status to return to calling function.

4.2.2 Open()

```
HRESULT Open( [in]  int  nPortNumber,
              [out] long  *plSessionId,
              [out] long  *plCapabilities,
              [out] FCISTATUS   *pStatus );
```

Description

The Open function must be called before using any other functions that communicate with the calibrator. This function returns a unique session ID that is required in all subsequent function calls to the calibrator. This ensures only one instance of the calibrator is opened at a time. This function should return FCI_ALREADY_OPEN if a session was already opened.

The plCapabilities parameter is one of the most important pieces of information returned by this function. It identifies the capabilities supported by the calibrator. The client can make decisions on subsequent interface function calls based on the calibrator capabilities. The plCapabilities parameter is assigned Ored combinations of CAPABILITY CODES from Table 3 that describe the driver's capabilities. E.g.:

*plCapabilities = (FCI_CAP_ID | FCI_CAP_UPLOAD | FCI_CAP_DOWNLOAD);

Table 3 - Calibrator Capabilities

CAPABILITY CODE	DESCRIPTION
FCI_CAP_ID	Reads calibrator manufacturer, model, & serial number
FCI_CAP_CALINFO	Reads calibrator last calibration and next calibration due date
FCI_CAP_DATESET	Sets the calibrator date & time
FCI_CAP_TEMPSTD	Sets the calibrator temperature standard. (IPTS-68 or ITS-90)
FCI_CAP_MEMCLEAR	Clears calibrator memory
FCI_CAP_DOWNLOAD	Downloads test setups to the calibrator
FCI_CAP_DLSETUP	Accepts text prompt to show before test
FCI_CAP_DLCLEANUP	Accepts text prompt to show after test
FCI_CAP_TECHNAME	Supplies technician name with results
FCI_CAP_UPLOAD	Uploads test data from the calibrator

Parameters

Parameters	Description
nPortNumber	Communications port number where the calibrator is attached.
plSessionId	Driver assigned number that identifies the current session.
plCapabilities	Bit-mapped value which indicates calibrator capabilities.
pStatus	Receives status to return to calling function.

4.2.3 Close()

```
HRESULT Close( [in]  long lSessionId,
               [out] FCISTATUS *pStatus );
```

Description

This function closes the calibrator session.

Any actions involved in closing the calibrator session are performed in this function. This typically includes releasing the serial port and setting the session ID to zero.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

4.2.4 Properties()

```
HRESULT Properties( [in]  long lSessionId,
                    [out] FCISTATUS *pStatus );
```

Description

This function is expected to raise a dialog box showing properties for the connected calibrator. It is recommended that the dialog box have *Ok*, *Help*, and *Driver Properties* buttons. The *Driver Properties* button raises the same dialog as the DriverProperties() function.

The FCINTF driver creates the dialog resource and determines what information is presented on the dialog box. It typically includes information such as:

calibrator manufacturer, model and serial number
number of download procedures in the calibrator
number of test results present in the calibrator

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

4.2.5 GetId()

```
HRESULT GetId( [in]  long  lSessionId,
               [out, string] LPWSTR *pszwCalManufacturer,
               [out, string] LPWSTR *pszwCalModel,
               [out, string] LPWSTR *pszwCalSerialNum,
               [out] FCISTATUS *pStatus );
```

Description

The function queries the calibrator for its manufacturer, model and serial number.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pszwCalManufacturer	Receives pointer to calibrator manufacturers name
pszwCalModel	Receives pointer to calibrator model name
pszwCalSerialNum	Receives pointer to calibrator serial number
pStatus	Receives status to return to calling function.

Comment

pszwCalManufacturer, pszwCalModel, pszwCalSerialNum are [out] strings (see paragraph 2.7). The driver should initialize these pointers to NULL, then use CoTaskMemAlloc() to allocate memory before assigning values. The client must call CoTaskMemFree() on any non-null returned values.

4.2.6 GetCalDates()

```
HRESULT GetCalDates( [in]  long lSessionId,
                     [out] SYSTEMTIME *pLastCalDate,
                     [out] SYSTEMTIME *pNextCalDueDate,
                     [out] FCISTATUS *pStatus );
```

Description

This function queries the calibrator for its last and next calibration dates.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pLastCalDate	Receives the calibrator's last calibration date.
pNextCalDueDate	Receives the calibrator's next calibration due date.
pStatus	Receives status to return to calling function.

Comment

The SYSTEMTIME format has sufficient precision to avoid all Y2K problems. It is the driver's responsibility to reformat any dates stored in the calibrator into this format. For example, some calibrators use a two-digit year with a specific algorithm that is applied to produce the correct four-digit year.

4.2.7 SetDateAndTime()

```
HRESULT SetDateAndTime( [in]  long lSessionId,
                        [out] FCISTATUS *pStatus );
```

Description

This function sets the date and time in the calibrator to match the system clock. It is the driver's responsibility to read the system clock date and time and reformat the data into a form acceptable by the calibrator.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

4.2.8 SetTempStandard()

```
HRESULT SetTempStandard( [in]  long lSessionId,
                         [in]  int  nTemperatureStd,
                         [out] FCISTATUS *pStatus );
```

Description

This function sets the temperature standard the calibrator will use, IPTS-68 or ITS-90. The client needs to call this function prior to calling StartDownloading. The driver is responsible for insuring the calibrator uses the requested temperature standard for all subsequent operations (e.g. downloading).

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
nTemperatureStd	IPTS_68, ITS_90
Pstatus	Receives status to return to calling function.

4.2.9 GetTestResultsCount()

```
HRESULT GetTestResultsCount( [in]  long lSessionId,
                             [out] int  *pnCount,
                             [out] FCISTATUS *pStatus );
```

Description

This function queries the calibrator for the number of unique tags with test results. It is the driver's responsibility to insure that tags with multiple as-found or as-left results are counted only once.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pnCount	Receives number of unique tags with test results available.
pStatus	Receives status to return to calling function.

4.2.10 ClearMemory()

```
HRESULT ClearMemory( [in]  long lSessionId,
                     [out] FCISTATUS *pStatus );
```

Description

This function erases all downloaded procedures and all test results from the calibrator's memory.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

Comment

The calling application should warn the user of the impending memory deletion and provide an opportunity to cancel the action.

4.2.11 ValidateTag()

```
HRESULT ValidateTag( [in]  FCICOMMDEF  *pCommDef,
                     [in]  FCIDEVIN  *pInputDef,
                     [in]  FCIDEVOUT  *pOutputDef,
                     [in]  FCIDEVRELATION *pRelationDef,
                     [in]  FCITEST  *pTestDef,
                     [out] FCISTATUS  *pStatus );
```

Description

This function determines if an instrument definition will download properly to the calibrator. It does not actually download anything. If the data would not download properly, pStatus should indicate the reason.

Parameters

Parameters	Description
pCommDef	Pointer to a FCICOMMDEF struct.
pInputDef	Pointer to a FCIDEVIN struct.
pOutputDef	Pointer to a FCIDEVOUT struct.
pRelationDef	Pointer to a FCIDEVRELATION struct.
pTestDef	Pointer to a FCITEST struct.
pStatus	Receives status to return to calling function.

4.3 ICalibratorDownload

As its name implies, this interface controls the process of downloading test procedures to a calibrator.

4.3.1 StartDownloading()

```
HRESULT StartDownloading( [in]  long lSessionId,
                          [in, string] LPCWSTR szSessionName,
                          [out] FCISTATUS *pStatus );
```

Description

StartDownloading must be called prior to any other downloading functions.

This function verifies communication with the calibrator and ensures the calibrator doesn't contain any results. FCI_MEM_HAS_DATA is returned if results are found. If no results are found, the calibrator memory is completely erased

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
szSessionName	Name optionally assigned to a session. e.g. A route name.
pStatus	Receives status to return to calling function.

Comment

The szSessionName may or may not be used by the FCINTF driver. If appropriate, it is simply a convenient name used to identify the group of instrument test procedures being downloaded to the calibrator.

4.3.2 DownloadTag()

```
HRESULT DownloadTag( [in]  long   lSessionId,
                     [in, string] LPCWSTR szwDeviceTag,
                     [in, string] LPCWSTR szwDeviceSerialNum,
                     [in]  FCICOMMDEF *pCommDef,
                     [in]  FCIDEVIN *pInputDef,
                     [in]  FCIDEVOUT *pOutputDef,
                     [in]  FCIDEVRELATION *pRelationDef,
                     [in]  FCITEST *pTestDef,
                     [in, string] LPCWSTR szwSetupInstructions,
                     [in, string] LPCWSTR szwCleanupInstructions,
                     [out, string] LPWSTR *pszwLocationInCalibrator,
                     [out] FCISTATUS *pStatus );
```

Description

After initiating a download, the client must call this function repeatedly, once for each instrument tag in the group of instruments being downloaded. The passed data is validated and then reformatted by the driver into a form required by the calibrator.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
szwDeviceTag	Device tag name.
szwDeviceSerialNum	Device serial number.
pCommDef	Identifies the communication protocol, if any.
pInputDef	Points to input definition that includes range data, units, etc.
pOutputDef	Points to output definition that includes range data, units, etc.
pRelationDef	Identifies the relation between input and output values.
pTestDef	Points to a test definition.
szwSetupInstructions	Setup instructions
szwCleanupInstructions	Cleanup instructions
pszwLocationInCalibrator	Uniquely identifies the tag in the calibrator. This can be any driver-created value. The driver will assign the same value to pszwLocationInCalibrator in the UploadNextTag() function (see 4.4.2).
pStatus	Receives status to return to calling function.

Comment

pszwLocationInCalibrator is an [out] string, so the driver and client must observe the proper memory allocation and deletion procedures (see 2.7). The driver must initialize this pointer to NULL, then use CoTaskMemAlloc() to allocate memory. The client must call CoTaskMemFree() on any non-null returned value.

4.3.3 FinishDownloading()

```
HRESULT FinishDownloading( [in]  long lSessionId,
                           [out] FCISTATUS *pStatus );
```

Description

The client must call this function after downloading all test procedures to close the download process. The driver uses this function to do any required cleanup or final actions such as deleting temporary files or freeing dynamically allocated memory.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

4.3.4 AbortDownload()

```
HRESULT AbortDownload( [in]  long lSessionId,
                       [out] FCISTATUS *pStatus );
```

Description

The client can call this function to abort the download process (for example in response to a user request to cancel the download). The driver will clean up the download process and reset the calibrator, including erasing the calibrator memory if anything has been sent to the calibrator.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

4.4 ICalibratorUpload

This interface controls the process of uploading test results from a calibrator.

4.4.1 StartUploading()

```
HRESULT StartUploading( [in]  long lSessionId,
                        [out] FCISTATUS *pStatus );
```

Description

The client must call StartUploading prior to any other uploading functions. The driver verifies communication with the calibrator and performs any actions required to begin the uploading process.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

4.4.2 UploadNextTag()

```
HRESULT UploadNextTag( [in] long lSessionId,
                       [out, string] LPWSTR *pszwLocationInCalibrator,
                       [out, string] LPWSTR *pszwDeviceTag,
                       [out, string] LPWSTR *pszwDeviceSerialNum,
                       [out, string] LPWSTR *pszwTechnician,
                       [out, string] LPWSTR *pszwServiceNote,
                       [out] int  *pnTemperatureStd,
                       [out] FCICOMMDEF *pCommDef,
                       [out] FCIRESULT *pAsFound,
                       [out] FCIRESULT *pAsLeft,
                       [out] FCISTATUS *pStatus );
```

Description

UploadNextTag uploads the test results for a single tag from the calibrator memory. To ensure all available results are uploaded, the client must repeatedly call this function until the pStatus indicates that there are no more tags to upload (FCI_END_OF_UPLOAD).

There is no implied tag order to the uploaded data, and the driver does not keep a persistent history of which tags were uploaded. In other words, the driver is responsible for keeping track of what it has already uploaded during a session so that the same tag does not appear multiple times in the same session. However, the driver does not keep track of what it has uploaded from one session to the next. If the client calls the sequence

"StartUploading, UploadNextTag (multiple times), FinishUploading" more than once, it should expect that the driver will return the same data each time, but not necessarily in the same order.

This places the burden of keeping track of the test result data on the client CMS application. This can be significant in the case of partial uploads. For example, ten test procedures are downloaded to a calibrator, but the technician only has time in one day to execute five of them. An upload sequence between the client and driver would retrieve the five test results. If the technician finishes the last five on the next day without re-downloading the calibrator, then the next upload procedure would yield ten sets of test results. It is the client CMS application's responsibility to recognize that some of the data has been uploaded more than once.

Parameters

Parameters	**Description**
lSessionId	Driver assigned number that identifies the current session.
pszwLocationInCalibrator	Identifies the tag in the calibrator. This can be any value, but it must be the same value returned by the driver during the download for this tag.
pszwDeviceTag	Receives the device tag name or NULL.
pszwDeviceSerialNum	Receives the device serial number or NULL.
pszwTechnician	Receives the technician's name or NULL.
pszwServiceNote	Receives the service note or NULL.
pnTemperatureStd	IPTS-68 or ITS-90
pCommDef	Points to communication definition.
pAsFound	Points to as found results
pAsLeft	Points to as left results.
pStatus	Receives status to return to calling function.

Comment

There are several [out] string parameters in UploadNextTag(), so the driver and client must observe the proper memory allocation and deletion procedures (see 2.7). The driver should initialize these pointers to NULL, then use CoTaskMemAlloc() to allocate memory before assigning values. The client must call CoTaskMemFree() on any non-null returned values.

In addition, the pAsFound and pAsLeft parameters each point to an FCIRESULT structure that contains strings. Since the outer level structure is an [out] parameter, the szwAuxEquipManufacturer; szwAuxEquipModel; and szwAuxEquipSerialNum also require dynamic memory allocation and deallocation.

This function returns FCI_END_OF_UPLOAD when there are no more tags to upload.

4.4.3 FinishUploading()

```
HRESULT FinishUploading( [in]  long lSessionId,
                         [out] FCISTATUS   *pStatus );
```

Description

The client must call this function after it receives the FCI_END_OF_UPLOAD status from its last call to UploadNextTag().

FinishUploading allows the driver to perform any actions to complete the uploading process including deleting temporary files and freeing dynamically allocated memory.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

4.4.4 AbortUpload()

```
HRESULT AbortUpload( [in]  long lSessionId,
                     [out] FCISTATUS *pStatus );
```

Description

This function is called to abort the upload process, for example in response to a user request to cancel the upload. The driver must delete any temporary files and free any dynamically allocated memory.

Parameters

Parameters	Description
lSessionId	Driver assigned number that identifies the current session.
pStatus	Receives status to return to calling function.

5. Appendix A - FCINTF.IDL File

```
/*****************************************************************************
 * SourceSafe Header
 *
 * $Workfile: fcintf.idl $
 * $Revision: 39 $
 * $Date: 3/25/99 11:21a $
 *
 *****************************************************************************/

/*****************************************************************************
 * Copyright 1998-1999, Southwest Research Institute.
 * Release Version 1.0b
 *****************************************************************************/

/*****************************************************************************
 * File Description:
 *  This is the Interface Description Language (IDL) file that defines the
 * Field Calibrator Interface (FCI) for a COM object that controls the
 * exchange of information with an intelligent field calibrator.
 * From a practical standpoint, this set of interfaces should only be
 * implemented in an INPROC server (DLL).
 *
 * Each component built to this specification should also register itself
 * as part of the "FieldCalibrators" component category
 *
 * The first part of this file contains the data structures and definitions
 * required to exchange information.
 * The second part of the file contains the COM Interface definitions.
 *
 * The build line for this file must be:
 *    MIDL /ms_ext /c_ext /app_config /h ..\include\fcintf.h /iid ..\include\fciguids.c fcintf.idl
 *
 * The principal outputs from this are: fcintf.h, fciguids.c
 *****************************************************************************/

import "oaidl.idl";

/*****************************************************************************
 * GUID and strings for the "FieldCalibrators" component category.
 *****************************************************************************/

cpp_quote( "DEFINE_GUID( CATID_FieldCalibrators, 0xff5efd40, 0x7b15, 0x11d1, 0xb3, 0x26, 0x00, 0x00,
0x1c, 0xbe, 0x02, 0xaa);" )
cpp_quote( "#define CATIDFIELDCAL  {0xff5efd40, 0x7b15, 0x11d1, 0xb3, 0x26, 0x00, 0x00, 0x1c, 0xbe,
0x02, 0xaa}" )
cpp_quote( "#define OLEFCICATEGORY L\"Field Calibrators\"")
```

```
/*****************************************************************************
 * Status Codes
 * All calibrator functions return a standard HRESULT that indicates whether
 * the function call succeeded from a COM standpoint.
 * Function specific status is placed in the pStatus variable associated with
 * each function.  The possible values are enumerated here.
 *****************************************************************************/

typedef enum tagFCISTATUS
{
    FCI_OK               =  0, // No error.
    FCI_FAILED           =  1, // The function failed for an unspecified reason.
    FCI_COMM_ERROR       =  2, // Unable to communicate with the calibrator.
    FCI_REVISION_ERROR   =  3, // The calibrator driver revision is incompatible
                               //  with this calibrator.
    FCI_CANCELED         =  4, // The operation was canceled.
    FCI_NOT_SUPPORTED    =  5, // This calibrator does not support this function.
    FCI_ALREADY_OPEN     =  6, // There is already a calibrator open.
    FCI_WRONG_SESSION    =  7, // The session identifier is incorrect.
    FCI_WRONG_CALIBRATOR =  8, // The calibrator currently attached is not the one associated
                               //  with this session.
    FCI_ERR_IN_RANGE     =  9, // The input range is not supported by this calibrator.
    FCI_ERR_IN_UNITS     = 10, // The input engineering unit is not supported by this calibrator.
    FCI_ERR_IN_TYPE      = 11, // The input type is not supported by this calibrator.
    FCI_ERR_IN_MANUAL    = 12, // This calibrator does not support manual entry for input.
    FCI_ERR_OUT_RANGE    = 13, // The output range is not supported by this calibrator.
    FCI_ERR_OUT_UNITS    = 14, // The output engineering unit is not supported by this calibrator.
    FCI_ERR_OUT_TYPE     = 15, // The input type is not supported by this calibrator.
    FCI_ERR_OUT_MANUAL   = 16, // This calibrator does not support manual entry for output.
    FCI_ERR_INOUT        = 17, // Invalid combination of input and output types for this calibrator.
    FCI_ERR_RELATION     = 18, // The input to output relation is not supported by this calibrator.
    FCI_ERR_PROBE        = 19, // The temperature probe is not supported by this calibrator.
    FCI_ERR_CJC          = 20, // The cold junction compensation type is not supported.
    FCI_ERR_ENUM         = 21, // A given enumeration is not a valid value.
    FCI_ERR_RESOURCE     = 22, // Driver was unable to load a required resource.
    FCI_ERR_TESTPOINT    = 23, // Error in one or more test points.
                               // Out of range, not stable, or not valid.
    FCI_ERR_TESTTYPE     = 24, // The test type does not match the instrument or is not supported.
    FCI_ERR_POWER        = 25, // The loop power configuration is inconsistent or not supported.
    FCI_ERR_TAG          = 26, // A tag is required for this operation.
    FCI_MEM_FULL         = 27, // The calibrator memory is full.
    FCI_MEM_EMPTY        = 28,
    FCI_MEM_HAS_DATA     = 29, // The calibrator memory contains test data.
    FCI_END_OF_UPLOAD    = 30, // End of upload list has been reached.
    FCI_ERR_DOWNLOAD     = 31, // Error encountered when trying to finalize download.
    FCI_ERR_TAGLENGTH    = 32, // The tag length sent exceeds the length supported by this driver.
    FCI_ERR_SNLENGTH     = 33, // The serial number is longer than the max supported by this driver.
    FCI_ERR_MAXPOINTS    = 34, // There are more AF or AL points than can be uploaded in a packet.
    FCI_END
} FCISTATUS;

/*****************************************************************************
 * Calibrator Capabilities
 * These are bitfield definitions returned in the lCapabilities parameter.
```

```
 * If a bit is set, then the calibrator can support the associated activity.
 ****************************************************************************/

cpp_quote("#define FCI_CAP_ID          0x01L") // Read calibrator manufact, model, & serial
cpp_quote("#define FCI_CAP_CALINFO     0x02L") // Read calibrator last cal and next cal due
cpp_quote("#define FCI_CAP_DATESET     0x04L") // Set the calibrator date & time
cpp_quote("#define FCI_CAP_TEMPSTD     0x08L") // Set the calibrator temperature standard
cpp_quote("#define FCI_CAP_MEMCLEAR    0x10L") // Clear calibrator memory
cpp_quote("#define FCI_CAP_DOWNLOAD    0x20L") // Download test setups to the calibrator
cpp_quote("#define FCI_CAP_DLSETUP     0x40L") // Send text prompt to show before test
cpp_quote("#define FCI_CAP_DLCLEANUP   0x80L") // Send text prompt to show after test
cpp_quote("#define FCI_CAP_TECHNAME   0x100L") // Supplies technician name with results
cpp_quote("#define FCI_CAP_UPLOAD     0x200L") // Upload test data from the calibrator

/****************************************************************************
 * Definitions for various discreet values.
 ****************************************************************************/

cpp_quote("#define FCI_MAX_TEST_POINTS  21")

// Temperature standard indicators
cpp_quote("#define STD_IPTS68           1")
cpp_quote("#define STD_ITS90            2")

// Contact definitions for switches
cpp_quote("#define CONTACT_TYPE_WET     0")
cpp_quote("#define CONTACT_TYPE_DRY     1")

cpp_quote("#define CONTACT_FORM_NO      0")
cpp_quote("#define CONTACT_FORM_NC      1")
cpp_quote("#define CONTACT_FORM_C       2")

cpp_quote("#define CONTACT_TRIP_INC     0")
cpp_quote("#define CONTACT_TRIP_DEC     1")

//Manual entry vs calibrator readings for testing
cpp_quote("#define BY_USER              1")
cpp_quote("#define BY_CALIBRATOR        2")

//Power for device under test
cpp_quote("#define PWR_EXTERNAL         1")
cpp_quote("#define PWR_CALIBRATOR       2")
cpp_quote("#define PWR_CALIBRATOR_HIV   3")

//Float value to represent invalid floating point number.
//This number is equivalent to FLT_MAX
cpp_quote("#define INVALID_FLOAT        3.402823466e+38F")


////////////////////////////////////////////////////////////////////////////
/****************************************************************************
 * This first major section provides a method of defining an instrument for
 * a calibrator.
 * Each instrument definition is composed of the following blocks.
```

```
 *   Communication - defines any comm capabilities
 *   Input         - defines the sensor input to the instrument
 *   Output        - defines the output generated by the instrument
 *   Relation      - defines the expected relation between input and output.
 ***************************************************************************/
////////////////////////////////////////////////////////////////////////////


/***************************************************************************
 * Communication Block Definitions
 ***************************************************************************/
/*
 * Communication block types
 */
typedef enum tagFCICOMMTYPE
{
    FCI_COMMTYPE_NONE = 1,   // No digital communication
    FCI_COMMTYPE_HART = 2    // Uses HART protocol
} FCICOMMTYPE;

/*
 * Communication block structures
 */
typedef struct tagFCI_COMMDEF_HART
{
     BYTE cBlockId;           // The transmitter variable associated with this block.
                              // ASCII P, S, T, Q.  Or 0 - 79 Decimal if not mapped to
                              // analog output.
     BYTE cURev;              // HART Universal Revision
     BYTE cPollAddr;          // Poll Address
                              // The following arrays are in HART message format,
                              //   they are not NULL terminated strings.
     BYTE acUid[5];           // Five Byte Unique Identifier. Set to 0 for URev = 4
                              //   This is the mfid, devid, and uid from HART cmd 0 or 11
                              // The next three are from command 13,
     BYTE acTag[6];           //   The packed ASCII tag field in the instrument.
     BYTE acDescriptor[12];   //   The packed ASCII descriptor field
     BYTE acDate[3];          //   The date field
                              // From command 12,
     BYTE acMessage[24];      //   The packed ASCII message field
} FCI_COMMDEF_HART;


typedef struct tagFCICOMMDEF
{
     FCICOMMTYPE CommType;

     [switch_type( FCICOMMTYPE ), switch_is( CommType )]
     union
     {
          [case(FCI_COMMTYPE_HART)] FCI_COMMDEF_HART HartData;
          [default];
     };
} FCICOMMDEF;
```

```
/*************************************************************************
 * Instrument Input/Output block types defined
 * Instrument Input  = sourced or generated by the calibrator.
 * Instrument Output = measured by the calibrator.
 *************************************************************************/

/* I/O Block Types */
typedef enum tagFCIBLOCKTYPE
{                                   // Blocks may be used for (relative to device under test)
     FCI_BLKTYPE_GENERIC      = 1,  // input or output
     FCI_BLKTYPE_TEMP_RTD     = 2,  // input: calibrator simulates an RTD
     FCI_BLKTYPE_TEMP_TC      = 3,  // input: calibrator simulates a thermocouple (TC)
     FCI_BLKTYPE_FREQUENCY    = 4,  // input or output
     FCI_BLKTYPE_PRESSURE     = 5,  // input or output
     FCI_BLKTYPE_TEMP_MEASRTD = 6,  // input or output: use calibrator to measure temperature
     FCI_BLKTYPE_TEMP_MEASTC  = 7,  // input or output: use calibrator to measure temperature
     FCI_BLKTYPE_HART         = 8,  // input or output: set or measured via HART commands
     FCI_BLKTYPE_SWITCH       = 9,  // output
     FCI_BLKTYPE_DIFFTEMP     = 10  // differential temperature
} FCIBLOCKTYPE;

typedef struct tagFCI_BLKDEF_PRESSURE
{
     FLOAT     rURV;          // Upper range value
     FLOAT     rLRV;          // Lower range value
     FLOAT     rSettling;     // Settling time (how many seconds to wait for stable reading)
     WORD      wUnits;        // Unit Code. See HART table 2.
     WORD      wPressureType; // Type of pressure measurement. See HART table 19.
}FCI_BLKDEF_PRESSURE;

typedef struct tagFCI_BLKDEF_RTD
{
     FLOAT     rURV;          // Upper range value
     FLOAT     rLRV;          // Lower range value
     FLOAT     rSettling;     // Settling time (how many seconds to wait for stable reading)
     WORD      wUnits;        // Unit Code. See HART table 2.
     WORD      wProbeType;    // Temperature probe type.  See HART table 16.
     WORD      wNumWires;     // Number of wires used by the RTD. Legal values are 2, 3, or 4.
     WORD      wRESERVED;
}FCI_BLKDEF_RTD;

typedef struct tagFCI_BLKDEF_TC
{
     FLOAT     rURV;          // Upper range value
     FLOAT     rLRV;          // Lower range value
     FLOAT     rSettling;     // Settling time (how many seconds to wait for stable reading)
     FLOAT     rManualCJC;    // Value for manual cold junction compensation (NAN if not used)
                              //    Must be in same units as rURV and rLRV.
     WORD      wUnits;        // Unit Code. See HART table 2.
     WORD      wProbeType;    // Temperature probe type.  See HART table 16.
     WORD      wCJC;          // Cold Junction Compensation Type. See HART table 18.
     WORD      wProbeConnect; // How probe is connected.  See HART table 17.
```

```
}FCI_BLKDEF_TC;

typedef struct tagFCI_BLKDEF_FREQUENCY
{
     FLOAT     rURV;          // Upper range value
     FLOAT     rLRV;          // Lower range value
     FLOAT     rSettling;     // Settling time (how many seconds to wait for stable reading)
     FLOAT     rAmplitude;    // Expected amplitude in volts (peak to peak)
     WORD      wUnits;        // Unit Code. See HART table 2.
     WORD      wWaveForm;     // Expected waveform. See HART table 20.
}FCI_BLKDEF_FREQUENCY;

typedef struct tagFCI_BLKDEF_GENERIC
{
     FLOAT     rURV;          // Upper range value
     FLOAT     rLRV;          // Lower range value
     FLOAT     rSettling;     // Settling time (how many seconds to wait for stable reading)
     WORD      wUnits;        // Unit Code. See HART table 2.
}FCI_BLKDEF_GENERIC;

typedef struct tagFCI_BLKDEF_SWITCH
{
     WORD      wContactType;       // 0=Wet(voltage present), 1=Dry(contacts only)
     WORD      wForm;              // 0=Normally Open(NO), 1=Normally Closed(NC), 2=Form C
     WORD      wTripDirection;     // 0=Trip on Increase, 1=Trip on Decrease
     FLOAT     rWetContactVoltage; // expected contact voltage in Volts for wet contact type.
}FCI_BLKDEF_SWITCH;

/* useful only if calibrator has HART ability */
typedef struct tagFCI_BLKDEF_HART
{
     FLOAT     rUSL;          // Upper sensor limit
     FLOAT     rLSL;          // Lower sensor limit
     FLOAT     rSettling;     // Settling time (how many seconds to wait for stable reading)
     WORD      wUnits;        // Unit code.  See HART table 2.
}FCI_BLKDEF_HART;

/* useful only if calibrator can communicate with the instrument */
typedef struct tagFCI_BLKDEF_DIFFTEMP    //Differential Temperature
{
     FLOAT     rURV;             // Upper range value
     FLOAT     rLRV;             // Lower range value
     FLOAT     rSettling;        // Settling time (how many seconds to wait for stable reading)
     WORD      wUnits;           // Unit Code. See HART table 2.
     WORD      wDeviceVariable1; // Diff Temp = DeviceVariable1 - DeviceVariable2
     WORD      wDeviceVariable2; //
}FCI_BLKDEF_DIFFTEMP;

/*************************************************************************
 * Instrument Input Block Definition (Calibrator Generated, or Sourced)
 *************************************************************************/

typedef struct tagFCIDEVIN
{
```

```
     FCIBLOCKTYPE DevType;

     [switch_type( FCIBLOCKTYPE ), switch_is( DevType )]
     union
     {
          [case(FCI_BLKTYPE_PRESSURE)]     FCI_BLKDEF_PRESSURE  PressureData;
          [case(FCI_BLKTYPE_TEMP_RTD)]     FCI_BLKDEF_RTD       TempRtdData;
          [case(FCI_BLKTYPE_TEMP_TC)]      FCI_BLKDEF_TC        TempTcData;
          [case(FCI_BLKTYPE_TEMP_MEASRTD)] FCI_BLKDEF_RTD       TempMeasRtdData;
          [case(FCI_BLKTYPE_TEMP_MEASTC)]  FCI_BLKDEF_TC        TempMeasTcData;
          [case(FCI_BLKTYPE_FREQUENCY)]    FCI_BLKDEF_FREQUENCY FrequencyData;
          [case(FCI_BLKTYPE_HART)]         FCI_BLKDEF_HART      HartData;
          [case(FCI_BLKTYPE_DIFFTEMP)]     FCI_BLKDEF_DIFFTEMP  DiffTempData;
          [default]                        FCI_BLKDEF_GENERIC   GenericData;
     };
} FCIDEVIN;


/*************************************************************************
 * Instrument Output Block Definition (Calibrator Measurement)
 *************************************************************************/

typedef struct tagFCIDEVOUT
{
     FCIBLOCKTYPE DevType;

     [switch_type( FCIBLOCKTYPE ), switch_is( DevType )]
     union
     {
          [case(FCI_BLKTYPE_PRESSURE)]     FCI_BLKDEF_PRESSURE  PressureData;
          [case(FCI_BLKTYPE_TEMP_MEASRTD)] FCI_BLKDEF_RTD       TempMeasRtdData;
          [case(FCI_BLKTYPE_TEMP_MEASTC)]  FCI_BLKDEF_TC        TempMeasTcData;
          [case(FCI_BLKTYPE_FREQUENCY)]    FCI_BLKDEF_FREQUENCY FrequencyData;
          [case(FCI_BLKTYPE_HART)]         FCI_BLKDEF_HART      HartData;
          [case(FCI_BLKTYPE_SWITCH)]       FCI_BLKDEF_SWITCH    Switch;
          [default]                        FCI_BLKDEF_GENERIC   GenericData;
     };
} FCIDEVOUT;

/*************************************************************************
 * Instrument Relation Block Definition
 *************************************************************************/

/* These are a subset of HART Table 3 */
typedef enum tagFCIRELATIONTYPE
{
     FCI_RELTYPE_LINEAR = 0,
     FCI_RELTYPE_SQRT   = 1,
     FCI_RELTYPE_TABLE  = 4,   // break point table
     FCI_RELTYPE_SWITCH = 230
} FCIRELATIONTYPE;

typedef struct tagFCI_RELDEF_SQRT
{
```

```
     FLOAT     rBreakPoint;   // Transition point from sqrt to linear, %input.
} FCI_RELDEF_SQRT;

typedef struct tagFCI_RELDEF_TABLE
{
     WORD      wNumPoints;    // Number of valid points in the table.  Max of 30.
     WORD      wInterpolate;  // 1 = linear, 2 = spline
     FLOAT     rInput[30];    // Array of input points. Pct of input span.
     FLOAT     rOutput[30];   // Array of corresponding output points. Pct output span.
} FCI_RELDEF_TABLE;

typedef struct tagFCIDEVRELATION
{
     FCIRELATIONTYPE RelType;

     [switch_type( FCIRELATIONTYPE ), switch_is( RelType )]
     union
     {
          [case(FCI_RELTYPE_SQRT)]      FCI_RELDEF_SQRT    SqrtData;
          [case(FCI_RELTYPE_TABLE)]     FCI_RELDEF_TABLE   TableData;
          [default];
     };
} FCIDEVRELATION;

////////////////////////////////////////////////////////////////////////////
/**************************************************************************
 * This second major section provides a means of sending a test
 * procedure for an instrument to a calibrator, and for reading the test
 * results from the calibrator.
 **************************************************************************/
////////////////////////////////////////////////////////////////////////////

/**************************************************************************
 * Test Definitions for proportional and switch instruments
 **************************************************************************/

typedef enum tagFCITESTTTYPE
{
     FCI_TYPE_PROPORTIONAL = 1,
     FCI_TYPE_SWITCH       = 2
} FCITESTTYPE;

typedef struct tagFCI_TEST_PROPORTIONAL
{
     WORD  wReserved;           // DUT = Device Under Test, i.e. the Instrument.
     WORD  wPowerSource;        // For DUT. PWR_EXTERNAL, PWR_CALIBRATOR, PWR_CALIBRATOR_HIV
     WORD  wCalSourceInput;     // Input to DUT generated: BY_USER, BY_CALIBRATOR
     WORD  wCalReadInput;       // Input to DUT measured: BY_USER, BY_CALIBRATOR
     WORD  wCalMeasureOutput;   // Output from DUT measured: BY_USER, BY_CALIBRATOR
     WORD  wNumTestPoints;      // Number of points to use when testing
     FLOAT arTestPoint[21];     // All point values are percent of input span
     FLOAT rTestPointTolerance; // % of span allowed off of test point
     FLOAT rAdjustmentLimit;    // If adjusted, errors must fall below this limit
     FLOAT rMaxErrorLimit;      // Max allowed error (fail if >) in percent of range
```

```
} FCI_TEST_PROPORTIONAL;

/*
 * Unless otherwise noted, all of the values in this structure are expressed
 * in the Engineering Units specified by the input block.
 * It is expected that the calibrator will start at the input block rLRV
 * and ramp its output toward the rURV over the rRampTime indicated.
 * After a trip is detected, if bTestReset is true, then the calibrator should
 * reverse the direction of the sensor input and record the reset point.
 */
typedef struct tagFCI_TEST_SWITCH
{
     FLOAT     rTripSetPoint;      // Expected trip point in Eng. units of the input.
     FLOAT     rTripTolerance;     // +/- allowed deviation in Eng units from trip
     FLOAT     rResetDeadband;     // expected difference between the trip and reset points.
     FLOAT     rDeadbandTolerance; // +/- deviation allowed in the deadband value
     FLOAT     rRampTime;          // in seconds.
     BOOL      bTestReset;         // set to TRUE to test both Trip and Reset points.
} FCI_TEST_SWITCH;

typedef struct tagFCITEST
{
     FCITESTTYPE TestType;

     [switch_type( FCITESTTYPE ), switch_is( TestType )]
     union
     {
          [case(FCI_TYPE_PROPORTIONAL)]      FCI_TEST_PROPORTIONAL    Test;
          [case(FCI_TYPE_SWITCH)]            FCI_TEST_SWITCH          SwitchTest;
          [default];
     };
} FCITEST;

/*****************************************************************************
 * Test Results for proportional and switch instruments.
 *****************************************************************************/

typedef struct tagFCI_RESULT_PROPORTIONAL
{
     double    dInputLowerRangeValue;         // Input and output are from the DUT perspective.
     double    dInputUpperRangeValue;         // Input is TO the instrument FROM Calibrator.
     int       nInputRangeUnits;
     double    dOutputLowerRangeValue;        // Output is FROM the instrument TO Calibrator.
     double    dOutputUpperRangeValue;
     int       nOutputRangeUnits;
     int       nRelationship;                 // Transfer Function - relation of input to output
     int       nNumberOfTestPoints;           // Actual number of test points used
     double    adInput[21];                   // Test Point Input (Engineering Units)
     double    adOutput[21];                  // Associated Output (Engineering Units)
     [string] LPWSTR szwAuxEquipManufacturer; // Auxilliary equipment used in test.
     [string] LPWSTR szwAuxEquipModel;        // May be null if not used.
     [string] LPWSTR szwAuxEquipSerialNum;
     double    dAmbientTemperature;           // Set to INVALID_FLOAT if calibrator can't supply.
     int       nAmbientTemperatureUnits;      // Set to unknown if calibrator can't supply.
```

```
     SYSTEMTIME TestDate;                    // Date and time of test
} FCI_RESULT_PROPORTIONAL;


/*
 * In the switch result structure, use INVALID_FLOAT in either rTrip or rResetPoint to indicate
 * that the point failed (did not change state within the test parameters).
 */
typedef struct tagFCI_RESULT_SWITCH
{
    WORD       wUnits;             // Unit code associated with the switch
    float      rTripPoint;         // Engineering unit value where switch transitions.
    float      rResetPoint;        // Engineering unit value where switch resets.
    WORD       bResetTested;       // Set to true if the reset point was tested.
    SYSTEMTIME TestDate;           // Date and time of test
}FCI_RESULT_SWITCH;

typedef struct tagFCIRESULT
{
     FCITESTTYPE ResultType;

     [switch_type( FCITESTTYPE ), switch_is( ResultType )]
     union
     {
          [case(FCI_TYPE_PROPORTIONAL)]      FCI_RESULT_PROPORTIONAL    Results;
          [case(FCI_TYPE_SWITCH)]            FCI_RESULT_SWITCH          SwitchResults;
          [default];
     };
} FCIRESULT;

//////////////////////////////////////////////////////////////////////////
/************************************************************************
 * This final section defines the actual COM Interfaces
 ************************************************************************/
//////////////////////////////////////////////////////////////////////////

/************************************************************************
 * Interface ICalibratorInfo
 *   Primarily used to get information about the calibrator and to provide
 *   housekeeping functions.
 ************************************************************************/

[
     object,
     uuid(ff5efd41-7b15-11d1-b326-00001cbe02aa),
     version(1.0),
     helpstring("ICalibratorInfo Interface"),
     pointer_default(unique)
]
interface ICalibratorInfo : IUnknown
{
     HRESULT DriverProperties( [out]  FCISTATUS   *pStatus );

     HRESULT Open(             [in]   int         nPortNumber,
```

```
                              [out]  long        *plSessionId,
                              [out]  long        *plCapabilities,
                              [out]  FCISTATUS   *pStatus );

      HRESULT Close(          [in]   long        lSessionId,
                              [out]  FCISTATUS   *pStatus );

      HRESULT Properties(     [in]   long        lSessionId,
                              [out]  FCISTATUS   *pStatus );

      HRESULT GetId(           [in]  long        lSessionId,
                      [out, string]  LPWSTR      *pszwCalManufacturer,
                      [out, string]  LPWSTR      *pszwCalModel,
                      [out, string]  LPWSTR      *pszwCalSerialNum,
                              [out]  FCISTATUS   *pStatus );

      HRESULT GetCalDates(    [in]   long        lSessionId,
                              [out]  SYSTEMTIME  *pLastCalDate,
                              [out]  SYSTEMTIME  *pNextCalDueDate,
                              [out]  FCISTATUS   *pStatus );

      HRESULT SetDateAndTime( [in]   long        lSessionId,
                              [out]  FCISTATUS   *pStatus );

      HRESULT SetTempStandard(  [in] long        lSessionId,
                                [in] int         nTemperatureStd, // IPTS_68, ITS_90
                              [out]  FCISTATUS   *pStatus );

      HRESULT GetTestResultsCount( [in]  long        lSessionId,
                                   [out] int         *pnCount,
                                   [out] FCISTATUS   *pStatus );

      HRESULT ClearMemory(    [in]   long        lSessionId,
                              [out]  FCISTATUS   *pStatus );

      HRESULT ValidateTag(    [in]   FCICOMMDEF     *pCommDef,
                              [in]   FCIDEVIN       *pInputDef,
                              [in]   FCIDEVOUT      *pOutputDef,
                              [in]   FCIDEVRELATION *pRelationDef,
                              [in]   FCITEST        *pTestDef,
                              [out]  FCISTATUS      *pStatus );
};

/*****************************************************************************
 * Interface ICalibratorDownload
 *   Used to send instrument and test definitions to a calibrator.
 *****************************************************************************/

[
      object,
      uuid(ff5efd42-7b15-11d1-b326-00001cbe02aa),
      version(1.0),
      helpstring("ICalibratorDownload Interface"),
      pointer_default(unique)
```

```
]
interface ICalibratorDownload : IUnknown
{
     HRESULT StartDownloading(  [in]          long        lSessionId,
                                [in, string]  LPCWSTR     szSessionName,  // an id for this session
                                [out]         FCISTATUS   *pStatus );

     HRESULT DownloadTag( [in]          long           lSessionId,
                          [in, string]  LPCWSTR        szwDeviceTag,
                          [in, string]  LPCWSTR        szwDeviceSerialNum,
                          [in]          FCICOMMDEF     *pCommDef,
                          [in]          FCIDEVIN       *pInputDef,
                          [in]          FCIDEVOUT      *pOutputDef,
                          [in]          FCIDEVRELATION *pRelationDef,
                          [in]          FCITEST        *pTestDef,
                          [in, string]  LPCWSTR        szwSetupInstructions,
                          [in, string]  LPCWSTR        szwCleanupInstructions,
                          [out, string] LPWSTR         *pszwLocationInCalibrator,
                          [out]         FCISTATUS      *pStatus );

     HRESULT FinishDownloading( [in]   long        lSessionId,
                                [out]  FCISTATUS   *pStatus );

     HRESULT AbortDownload(     [in]   long        lSessionId,
                                [out]  FCISTATUS   *pStatus );
};

/****************************************************************************
 * Interface ICalibratorUpload
 *   Used to retrieve test results from a calibrator.
 * If the calibrator cannot supply the information in a given block
 * (e.g, pCommDef, pAsFound, pAsLeft),
 * then the Type variable in the block will be set to 0.
 * If a string cannot be supplied (pszDeviceTag, pszDeviceSerialNum, pszTechnician,
 * pszServiceNote), then the pointer will be set to NULL, and the client does not
 * have to call CoTaskMemFree.
 ****************************************************************************/

[
     object,
     uuid(ff5efd43-7b15-11d1-b326-00001cbe02aa),
     version(1.0),
     helpstring("ICalibratorUpload Interface"),
     pointer_default(unique)
]
interface ICalibratorUpload : IUnknown
{
     HRESULT StartUploading(  [in]   long        lSessionId,
                              [out]  FCISTATUS   *pStatus );

     HRESULT UploadNextTag( [in] long        lSessionId,
                    [out, string] LPWSTR      *pszwLocationInCalibrator,
                    [out, string] LPWSTR      *pszwDeviceTag,
                    [out, string] LPWSTR      *pszwDeviceSerialNum,
```

```
                    [out, string] LPWSTR      *pszwTechnician,
                    [out, string] LPWSTR      *pszwServiceNote,
                            [out] int         *pnTemperatureStd, // IPTS_68 or ITS_90
                            [out] FCICOMMDEF  *pCommDef,
                            [out] FCIRESULT   *pAsFound,
                            [out] FCIRESULT   *pAsLeft,
                            [out] FCISTATUS   *pStatus );

    HRESULT FinishUploading( [in]   long        lSessionId,
                             [out]  FCISTATUS   *pStatus );

    HRESULT AbortUpload(     [in]   long        lSessionId,
                             [out]  FCISTATUS   *pStatus );
};
```

6. Appendix B - Interoperability Tables

```
/*****************************************************************************
 * SourceSafe Header
 *
 * $Workfile: interop.h $
 * $Revision: 5 $
 * $Date: 3/25/99 11:12a $
 *
 *****************************************************************************/

/*****************************************************************************
 * Copyright 1998, Southwest Research Institute.
 *****************************************************************************/

/*****************************************************************************
 * File Description:
 *  This file contains definitions from the HART common tables that are
 * designed to enhance the interoperability of HART devices.
 *
 *****************************************************************************/

/*****************************************************************************
 * $History: interop.h $
 *
 * *****************  Version 5  *****************
 * User: Kholladay    Date: 3/25/99    Time: 11:12a
 * Updated in $/FieldCalibrators/fcintf/include
 * Added 3 new units.
 *
 * *****************  Version 4  *****************
 * User: Kholladay    Date: 12/01/98   Time: 1:07p
 * Updated in $/FieldCalibrators/fcintf/include
 *
 * *****************  Version 3  *****************
 * User: Kholladay    Date: 11/23/98   Time: 4:10p
 * Updated in $/FieldCalibrators/fcintf/include
 *
 *
 *****************************************************************************/


#ifndef INTEROP_H
#define INTEROP_H
```

```
/*****************************************************************************
 * HART Table 2 - Unit Codes
 *****************************************************************************/

#define UNIT_INH2O              1   // inches of water @ 68degF
#define UNIT_INHG               2   // inches of mercury @ 0 degC
#define UNIT_FTH2O              3   // feet of water @ 68 degF
#define UNIT_MMH2O              4   // millimeters of water @ 68 degF
#define UNIT_MMHG               5   // millimeters of mercury @ 0 degC
#define UNIT_PSI                6   // pounds per square inch
#define UNIT_BAR                7   // bars
#define UNIT_MBAR               8   // millibars
#define UNIT_G_PER_SQCM         9   // grams per square centimeter
#define UNIT_KG_PER_SQCM       10   // kilograms per square centimeter
#define UNIT_PA                11   // pascals
#define UNIT_KPA               12   // kilopascals
#define UNIT_TORR              13   // torr
#define UNIT_ATM               14   // atmospheres
#define UNIT_CUFT_PER_MIN      15   // cubic feet per minute
#define UNIT_GAL_PER_MIN       16   // gallons per minute
#define UNIT_LITERS_PER_MIN    17   // liters per minute
#define UNIT_IMPGAL_PER_MIN    18   // imperial gallons per minute
#define UNIT_CUMTR_PER_HR      19   // cubic meters per hour
#define UNIT_FT_PER_S          20   // feet per second
#define UNIT_METER_PER_S       21   // meters per second
#define UNIT_GAL_PER_S         22   // gallons per second
#define UNIT_MILGAL_PER_DAY    23   // million gallons per day
#define UNIT_LITERS_PER_S      24   // liters per second
#define UNIT_MIL_L_PER_DAY     25   // million liters per day
#define UNIT_CUFT_PER_S        26   // cubic feet per second
#define UNIT_CUFT_PER_DAY      27   // cubic feet per day
#define UNIT_CUMTR_PER_S       28   // cubic meters per second
#define UNIT_CUMTR_PER_DAY     29   // cubic meters per day
#define UNIT_IMPGAL_PER_HR     30   // imperial gallons per hour
#define UNIT_IMPGAL_PER_DAY    31   // imperial gallons per day
#define UNIT_DEGC              32   // degrees Cesius
#define UNIT_DEGF              33   // degrees Fahrenheit
#define UNIT_DEGR              34   // degrees Rankine
#define UNIT_KELVIN            35   // degrees Kelvin
#define UNIT_MV                36   // millivolts
#define UNIT_OHM               37   // ohms
#define UNIT_HZ                38   // hertz
#define UNIT_MA                39   // milliamps
#define UNIT_GAL               40   // gallons
#define UNIT_LITER             41   // liters
#define UNIT_IMPGAL            42   // imperial gallons
#define UNIT_CUMTR             43   // cubic meters
#define UNIT_FT                44   // feet
#define UNIT_METER             45   // meters
#define UNIT_BBL               46   // barrels (42 gallons)
#define UNIT_IN                47   // inches
#define UNIT_CM                48   // centimeters
#define UNIT_MM                49   // millimeters
#define UNIT_MIN               50   // minutes
```

```
#define UNIT_SEC              51   // seconds
#define UNIT_HR               52   // hours
#define UNIT_DAY              53   // days
#define UNIT_CSTOKE           54   // centistokes
#define UNIT_CPOISE           55   // centipoise
#define UNIT_UMHO             56   // microsiemens
#define UNIT_PCT              57   // percent
#define UNIT_V                58   // volts
#define UNIT_PH               59   // pH
#define UNIT_GRAM             60   // grams
#define UNIT_KG               61   // kilograms
#define UNIT_METTON           62   // metric tons
#define UNIT_LB               63   // pounds
#define UNIT_SHTON            64   // short tons
#define UNIT_LTON             65   // long tons
#define UNIT_MSIEMEN_PER_CM   66   // millisiemens per centimeter
#define UNIT_USIEMEN_PER_CM   67   // microsiemens per centimeter
#define UNIT_NEWTON           68   // Newton
#define UNIT_NEWTMETER        69   // newton meter
#define UNIT_G_PER_S          70   // grams per second
#define UNIT_G_PER_MIN        71   // grams per minute
#define UNIT_G_PER_HR         72   // grams per hour
#define UNIT_KG_PER_S         73   // kilograms per second
#define UNIT_KG_PER_MIN       74   // kilograms per minute
#define UNIT_KG_PER_HR        75   // kilograms per hour
#define UNIT_KG_PER_DAY       76   // kilograms per day
#define UNIT_METTON_PER_MIN   77   // metric tons per minute
#define UNIT_METTON_PER_HR    78   // metric tons per hour
#define UNIT_METTON_PER_DAY   79   // metric tons per day
#define UNIT_LB_PER_S         80   // pounds per second
#define UNIT_LB_PER_MIN       81   // pounds per minute
#define UNIT_LB_PER_HR        82   // pounds per hour
#define UNIT_LB_PER_DAY       83   // pounds per day
#define UNIT_SHTON_PER_MIN    84   // short tons per minute
#define UNIT_SHTON_PER_HR     85   // short tons per hour
#define UNIT_SHTON_PER_DAY    86   // short tons per day
#define UNIT_LTON_PER_HR      87   // long tons per hour
#define UNIT_LTON_PER_DAY     88   // long tons per day
#define UNIT_DATHERM          89   // deka therm
#define UNIT_SGU              90   // specific gravity unit
#define UNIT_G_PER_CUCM       91   // grams per cubic centimeter
#define UNIT_KG_PER_CUMTR     92   // kilograms per cubic centimeter
#define UNIT_LB_PER_GAL       93   // pounds per gallon
#define UNIT_LB_PER_CUFT      94   // pounds per cubic foot
#define UNIT_G_PER_ML         95   // grams per milliliter
#define UNIT_KG_PER_L         96   // kilograms per liter
#define UNIT_G_PER_L          97   // grams per liter
#define UNIT_LB_PER_CUIN      98   // pounds per cubic inch
#define UNIT_SHTON_PER_CUYD   99   // short tons per cubic yard
#define UNIT_DEGTWAD         100   // degrees twaddell
#define UNIT_DEGBRIX         101   // degree brix
#define UNIT_DEGBAUM_HV      102   // degrees baume heavy
#define UNIT_DEGBAUM_LT      103   // degrees baume light
#define UNIT_DEGAPI          104   // degrees API (Am Petroleum Inst)
```

```
#define UNIT_PCTSOL_WT        105   // percent solids by weight
#define UNIT_PCTSOL_VOL       106   // percent solids by volume
#define UNIT_DEGBALL          107   // degrees balling
#define UNIT_PROOF_PER_VOL    108   // proof per volume
#define UNIT_PROOF_PER_MASS   109   // proof per mass
#define UNIT_BUSH             110   // bushels
#define UNIT_CUYD             111   // cubic yards
#define UNIT_CUFT             112   // cubic feet
#define UNIT_CUIN             113   // cubic inches
#define UNIT_IN_PER_SEC       114   // inches per second
#define UNIT_IN_PER_MIN       115   // inches per minute
#define UNIT_FT_PER_MIN       116   // feet per minute
#define UNIT_DEG_PER_SEC      117   // degrees per second
#define UNIT_REV_PER_SEC      118   // revolutions per second
#define UNIT_RPM              119   // revolutions per minute
#define UNIT_MTR_PER_HR       120   // meters per hour
#define UNIT_NMLCUMTR_PER_HR  121   // normal cubic meter per hour
#define UNIT_NMLLITER_PER_HR  122   // normal liter per hour
#define UNIT_STDCUFT_PER_MIN  123   // standard cubic feet per minute
#define UNIT_BARRELS_LIQ      124   // liquid barrels (31.5 gal)
#define UNIT_OUNCE            125   // ounces
#define UNIT_FT_LBF           126   // foot pound force
#define UNIT_KW               127   // kilowatt
#define UNIT_KWH              128   // kilowatt hour
#define UNIT_HP               129   // horsepower
#define UNIT_CUFT_PER_HR      130   // cubic feet per hour
#define UNIT_CUMTR_PER_MIN    131   // cubic meters per minute
#define UNIT_BBL_PER_SEC      132   // barrels per second
#define UNIT_BBL_PER_MIN      133   // barrels per minute
#define UNIT_BBL_PER_HR       134   // barrels per hour
#define UNIT_BBL_PER_DAY      135   // barrels per day
#define UNIT_GAL_PER_HR       136   // gallons per hour
#define UNIT_IMPGAL_PER_SEC   137   // imperial gallons per second
#define UNIT_L_PER_HR         138   // liters per hour
#define UNIT_PPM              139   // parts per million
#define UNIT_MCAL_PER_HR      140   // mega calories per hour
#define UNIT_MJOULE_PER_HR    141   // mega joules per hour
#define UNIT_BTU_PER_HR       142   // british thermal units per hour
#define UNIT_DEG              143   // degrees
#define UNIT_RAD              144   // radians
#define UNIT_INH2O_60DEGF     145   // inches of water @ 60 degF
#define UNIT_PCT_STM_QUAL     150   // percent steam quality
#define UNIT_FT_IN_16THS      151   // Feet in sixteenths
#define UNIT_CUFT_PER_LB      152   // cubic feet per pound
#define UNIT_PF               153   // picofarads
#define UNIT_PCT_PLATO        160   // percent plato
#define UNIT_PCT_LEL          161   // % lower explosion limit
#define UNIT_MCAL             162   // mega calories
#define UNIT_KOHM             163   // kiloOhm
#define UNIT_MJOULES          164   // mega joule
#define UNIT_BTU              165   // British thermal unit
#define UNIT_NMLCUMTR         166   // normal cubic meter
#define UNIT_NMLLITER         167   // normal liter
#define UNIT_STDCUFT          168   // standard cubic feet
```

```
#define UNIT_PPB              169   // parts per billion
#define UNIT_GAL_PER_DAY      235   // gallons per day
#define UNIT_HLITER           236   // hectoliter
#define UNIT_MPA              237   // megapascals
#define UNIT_INH2O_4DEGC      238   // inches of water @ 4degC
#define UNIT_MMH2O_4DEGC      239   // millimeters water @ 4degC
#define UNIT_NOT_USED         250
#define UNIT_NONE             251
#define UNIT_UNKNOWN          252
#define UNIT_SPECIAL          253

/*****************************************************************************
 * HART Table 3 - Transfer Function (Relationship) Codes
 *****************************************************************************/

/* Equation comments are x as %input vs.  y as %output */
#define RELATION_LINEAR                   0  // y = x
#define RELATION_SQRT                     1  // y = sqrt( x ) * 10.0
#define RELATION_SQRT_3RD_PWR             2  // y = sqrt(x^3) * 0.1
#define RELATION_SQRT_5TH_PWR             3  // y = sqrt(x^5) * 0.001
#define RELATION_TABLE                    4  // special curve
#define RELATION_SQUARE                   5  // y = x^2 * 0.01
#define RELATION_SWITCH                 230  // discreet output
#define RELATION_SQRT_SPECIAL           231  // + table
#define RELATION_SQRT_THIRD_SPECIAL     232  // + table
#define RELATION_SQRT_FIFTH_SPECIAL     233  // + table
#define RELATION_NOT_USED               250
#define RELATION_NONE                   251
#define RELATION_UNKNOWN                252

/*****************************************************************************
 * HART Table 16 - Temperature Probe Definitions
 *****************************************************************************/

#define PROBE_OHMS                    1  // Ohms
#define PROBE_KOHMS                   2  // kiloOhms
#define PROBE_RTD_CALIBRATED          3  // must supply Calendar-VanDeusen parameters.

#define PROBE_RTD_PT50_385           11  // RTD Pt   50    a=0.003850  (IEC751)
#define PROBE_RTD_PT100_385          12  // RTD Pt  100    a=0.003850
#define PROBE_RTD_PT200_385          13  // RTD Pt  200    a=0.003850
#define PROBE_RTD_PT500_385          14  // RTD Pt  500    a=0.003850
#define PROBE_RTD_PT1000_385         15  // RTD Pt 1000    a=0.003850

#define PROBE_RTD_PT50_3916          21  // RTD Pt  50     a=0.003916  (JIS C1604-81)
#define PROBE_RTD_PT100_3916         22  // RTD Pt  100    a=0.003916

#define PROBE_RTD_PT50_392           31  // RTD Pt   50    a=0.003920  (MIL-T-24388)
#define PROBE_RTD_PT100_392          32  // RTD Pt  100    a=0.003920
#define PROBE_RTD_PT200_392          33  // RTD Pt  200    a=0.003920
#define PROBE_RTD_PT500_392          34  // RTD Pt  500    a=0.003920
#define PROBE_RTD_PT1000_392         35  // RTD Pt 1000    a=0.003920

#define PROBE_RTD_PT10_3923          40  // RTD Pt   10    a=0.003923  (SAMA RC21-4-1966)
```

```
#define PROBE_RTD_PT100_3923        41  // RTD Pt  100    a=0.003923
#define PROBE_RTD_PT200_3923        42  // RTD Pt  200    a=0.003923

#define PROBE_RTD_PT100_3926        51  // RTD Pt  100    a=0.003926  (IPTS-68)

#define PROBE_RTD_NI50_672          61  // RTD Ni   50    a=0.006720  (Edison curve #7)
#define PROBE_RTD_NI100_672         62  // RTD Ni  100    a=0.006720
#define PROBE_RTD_NI120_672         63  // RTD Ni  120    a=0.006720
#define PROBE_RTD_NI1000_672        64  // RTD Ni 1000    a=0.006720

#define PROBE_RTD_NI50_618          71  // RTD Ni   50    a=0.006180  (DIN 43760)
#define PROBE_RTD_NI100_618         72  // RTD Ni  100    a=0.006180
#define PROBE_RTD_NI120_618         73  // RTD Ni  120    a=0.006180
#define PROBE_RTD_NI1000_618        74  // RTD Ni 1000    a=0.006180

#define PROBE_RTD_CU10_427          80  // RTD Cu   10    a=0.004270  (??Standard ??)
#define PROBE_RTD_CU100_427         81  // RTD Cu  100    a=0.004270

#define PROBE_MICROVOLT            128  // microVolts
#define PROBE_MILLIVOLT            129  // milliVolts
#define PROBE_VOLT                 130  // Volts

                                        // Thermocouples.
                                        // Unless otherwise noted, thermocouple references are:
                                        // IEC 584, NIST MN 175, DIN 43710, BS 4937,
                                        // ANSI MC96.1, JIS C1602 and NF C42-321)
#define PROBE_TC_B                 131  // TC Type B (Pt30Rh-Pt6Rh) (IEC 584 etc.)
#define PROBE_TC_C_W5              132  // TC Type W5, Omega type C (W5-W26Rh)(ASTM E 988)
#define PROBE_TC_D_W3              133  // TC Type W3, Omega type D (W3-W25Rh)(ASTM E 988 )
#define PROBE_TC_E                 134  // TC Type E (Ni10Cr-Cu45Ni) (IEC 584 etc.)
#define PROBE_TC_G_W               135  // TC Type W, Omega type G (W-W26Rh)(ASTM E 988)
#define PROBE_TC_J                 136  // TC Type J (Fe-Cu45Ni) (IEC 584 etc.)
#define PROBE_TC_K                 137  // TC Type K (Ni10Cr-Ni5) (IEC 584 etc.)
#define PROBE_TC_N                 138  // TC Type N (Ni14CrSi-NiSi) (IEC 584 etc.)
#define PROBE_TC_R                 139  // TC Type R (Pt13Rh-Pt) (IEC 584 etc.)
#define PROBE_TC_S                 140  // TC Type S (Pt10Rh-Pt) (IEC 584 etc.)
#define PROBE_TC_T                 141  // TC Type T (Cu-Cu45Ni) (IEC 584 etc.)
#define PROBE_TC_L                 142  // TC Type L (Fe-CuNi) (DIN 43710)
#define PROBE_TC_U                 143  // TC Type U (Cu-CuNi) (DIN 43710)
#define PROBE_TC_PT20RH            144  // TC Pt20Rh-Pt40Rh (ASTM E 1751)
#define PROBE_TC_IR40              145  // TC Ir-Ir40Rh (ASTM E 1751)
#define PROBE_TC_PLATINEL          146  // TC Platinel II
#define PROBE_TC_NI_NIMO           147  // TC Ni-NiMo

#define PROBE_BIMETALLIC           220  // Bi-metallic strip
#define PROBE_VP_BULB              221  // Vapor pressure bulb
#define PROBE_LIQUID_EXP           222  // Liquid expansion
#define PROBE_MERCURY_BULB         223  // Mercury bulb

#define PROBE_NOT_USED             250  // Not used
#define PROBE_NONE                 251  // None
#define PROBE_UNKNOWN              252  // Unknown
#define PROBE_SPECIAL              253  // Special
```

```
/*****************************************************************************
 * HART Table 17 - Temperature Probe Connections
 *****************************************************************************/

#define CNCT_SINGLE                  1  // Single.  One probe is used to measure temperature.
#define CNCT_DIFFERENTIAL            2  // Differential.  Two probes are connected in series
                                        //   to read differential temperature.
#define CNCT_NOT_USED              250  // Not used
#define CNCT_NONE                  251  // None
#define CNCT_UNKNOWN               252  // Unknown
#define CNCT_SPECIAL               253  // Special

/*****************************************************************************
 * HART Table 18 - Thermocouple Cold Junction Compensation (CJC) Options
 *****************************************************************************/
#define TC_CJC_INTERNAL              1  // A sensor built into the device provides CJC
#define TC_CJC_EXTERNAL              2  // An external sensor must be used
#define TC_CJC_MANUAL                3  // A fixed value is supplied

#define TC_CJC_NOT_USED            250  // Not used
#define TC_CJC_NONE                251  // None
#define TC_CJC_UNKNOWN             252  // Unknown
#define TC_CJC_SPECIAL             253  // Special

/*****************************************************************************
 * HART Table 19 - Pressure Measurement Types
 *****************************************************************************/
#define PRESSURE_TYPE_DIFFERENTIAL      0  // Differential pressure measurement
#define PRESSURE_TYPE_GAUGE             1  // Gauge pressure measurement
#define PRESSURE_TYPE_ABSOLUTE          2  // Absolute pressure measurement
#define PRESSURE_TYPE_HIGH_LINE         3  // High line pressure measurement
#define PRESSURE_TYPE_LIQUID_LEVEL      4  // Liquid level pressure measurement
#define PRESSURE_TYPE_DRAFT_RANGE       5  // Draft range pressure measurement
#define PRESSURE_TYPE_HTG               6  // Hydrostatic Tank Gauge

#define PRESSURE_TYPE_NOT_USED        250  // Not used
#define PRESSURE_TYPE_NONE            251  // None
#define PRESSURE_TYPE_UNKNOWN         252  // Unknown
#define PRESSURE_TYPE_SPECIAL         253  // Special

/*****************************************************************************
 * HART Table 20 - Frequency Wave Forms
 *****************************************************************************/
#define WAVEFORM_SQUARE              0  // Square wave
#define WAVEFORM_SINE                1  // Sine wave
#define WAVEFORM_TRIANGLE            2  // Triangle wave
#define WAVEFORM_SAWTOOTH            3  // Sawtooth wave
#define WAVEFORM_NOT_USED          250  // Not used
#define WAVEFORM_NONE              251  // None
#define WAVEFORM_UNKNOWN           252  // Unknown
#define WAVEFORM_SPECIAL           253  // Special

#endif /* INTEROP_H */
```