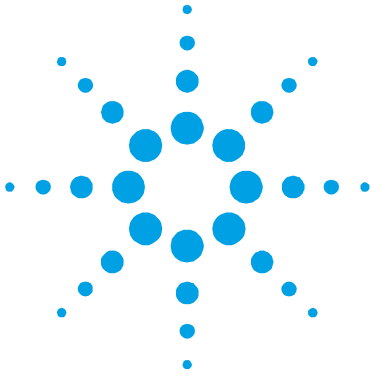


Agilent FTIR Instrument Interface

Application Programming Manual

Agilent Technologies

Notices

Manual Part Number

0020-412

Edition

Second edition, April 2011

Printed in USA

Agilent Technologies, Inc.

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

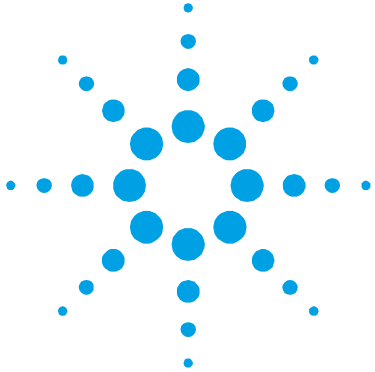
The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Contents

This page is intentionally left blank.



1. Overview

The main component for interfacing to the Agilent MicroLab instruments is through the FTIRInst.DLL interface DLL. This DLL provides a set of C-Callable, high-level language interface calls for communicating with the Agilent instrumentation products. The DLL interface can be invoked from VB, C, and C# modules, and is compatible with Microsoft® Windows®, Microsoft .NET Framework, and Microsoft Windows CE .NET Compact Framework platforms. The interface described in this document details the functional interface for use by a C# InteropService client wrapper class.

The interface DLL will encapsulate and wrap all details of the Driver interface layer below the DLL level. A simulation DLL (FTIRInst_sim.DLL) provides a software-only test solution, to provide an interface plug-in replacement that works seamlessly with applications built for the FTIRInst target interface.

NOTE The simulation DLL may not provide all of the functionality described in this document, and the return values may differ from those that the live instrument DLL provides.

The wrapper class for C# provides a number of interfaces designed for ease of use in interfacing with C# client applications. These routines generally wrap the DLL routines, provide data type transformations, and or combine a number of DLL routines into one call.

For network configuration of network-capable FTIR devices, see the separate document, Agilent Instrument Interface: Network Supplement. Access remains through the same DLLs.

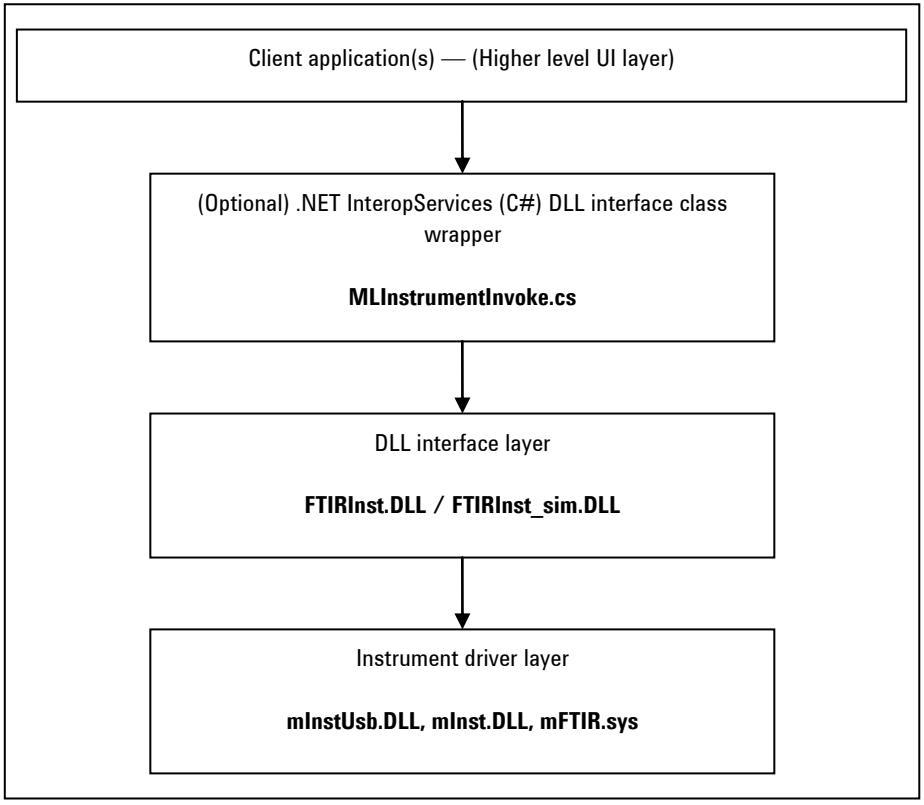
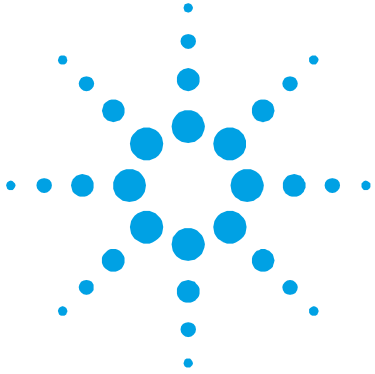


Figure 1. Architectural block diagram of interface



2. Data Types and Notation

The data types used in this document are the types as defined in the Microsoft .NET Framework.

Standard data types

- int: 32-bit integer (typically a long or DWORD in older style languages)
- short: 16-bit integer
- float: 32-bit floating point value (typically a float or single precision value in older style languages)
- double: 64-bit floating point value (typically a double precision value in older style languages)
- ref and or array[ ]: This is the .NET notation convention for a reference to a data type or a reference to an array of values (typically a pointer or ByRef value in older style languages)
- public and private: Protection and accessibility level of a variable or function

Enumerations

Specific sets of values stored as 32-bit integer values as follows:

```
PHASETYPE    { PT_MERTZ = 1, PT_FORMAN = 2, PT_FORMANRES = 3 };
APODTYPE     { APOD_NONE = 0, APOD_BOXCAR = APOD_NONE,
             APOD_TRIANGULAR = 1,
             APOD_WEAKNORTONBEER = 2, APOD_MEDIUMNORTONBEER = 3,
             APOD_STRONGNORTONBEER = 4, APOD_HAPPGENZEL = 5,
             APOD_BESSEL = 6,
             APOD_COSINE = 7, APOD_HANNING = APOD_COSINE, };
FTIR_STATE   { FTIR_Init = 0, FTIR_Collecting = 1, FTIR_DataReady
             = 2,
             FTIR_Aborting = 3, FTIR_Error = 4, };
PHASEPOINTS { PP_128 = 128, PP_256 = 256, PP_512 = 512, PP_1024
             = 1024 };
OFFSETCORRECTTYPE { OT_NONE = 0, OT_ALL = 1, OT_ENDS = 2 };
ZFFTYPE      { ZFF_NONE = 0, ZFF_2 = 1, ZFF_4 = 2, ZFF_8 = 3,
             ZFF_16 = 4 };
SAMPLINGTECHNOLOGYTYPE { ST_NONE = 0, ST_ATRSINGLE = -1,
             ST_ATRTRIPLE = -3,
             ST_ATRNINEBOUNCE = -9, ST_TRANSMISSIONCELL = 1,
             ST_GASCELL = 2,
             ST_REFLECTANCE = 3, };
ML_INSTRUMENT_TYPE { eInstrumentType_Undefined = 0,
             eInstrumentType_ML = 1, eInstrumentType_MLP = 2,
             eInstrumentType_MLX = 3, eInstrumentType_Exoscan =
             4, };
DATAXTYPE    { XT_ARB = 0, XT_WN = 1, XT_uM = 2, XT_nM = 3,
             XT_Seconds = 4,
             XT_Minutes = 5, XT_MassCharge = 9, XT_RAMSHFT = 13,
             XT_Points = 22, XT_Hours = 30, XT_AMU = 50,
             XT_Custom = 51 };
DATAYTYPE    { YT_ARB = 0, YT_IGRAM = 1, YT_Abs = 2, YT_Percent =
             11,
             YT_Intensity = 12, YT_RelAbundance = 13, YT_Trans =
             128,
             YT_Refl = 129, YT_Custom = 51, YT_Abundance = 52, };
REJECTREASON { RR_GOOD = 0, RR_20PCT = 0x00010004,
             RR_CENTERBURST = 0x00010005, RR_HW_UNSTABLE =
             0x00010010 };
```

Data structures

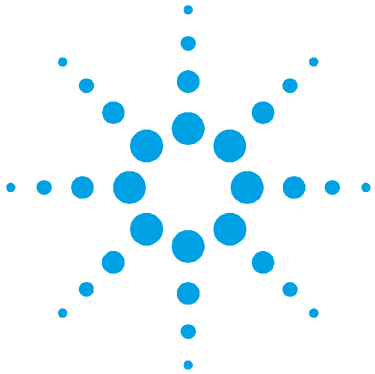
Structures of information, usually passed by reference, as follows:

```
struct _instrumentMLDiag
{
    public int nVersion;            // struct version (100 to
102)
    public int nEnergyStatus;       // Height of Center burst
    public int nLaserStatus;
    public int numTemps;
    public int nBatteryMinutes;
    public int nBatteryPct;
    public int nBatteryState;       // bits: 1=connected, 2=ac
connected,
                                    // 4=charging, 16=fully
charged
    public float fSourceCurrentStatus;
    public float fSourceVoltageStatus;
    public float fSpare;
    public float fTempCPU;          // Cpu board temperature
    public float fTempPower;        // Power board temperature
    public float fTempIR;           // IR board temperature
    public float fTempDetector;     // Detector temperature

    // MLDiag 102+
    public Int32 nSystemStatus;     // System Status
    public Int32 nShutdownReason;   // System Shutdown Reason
};

struct _instrumentMLVersion
{
    public int nVersion;            // struct version (100 to
103)
    public int fwRev;               // firmware rev
    public int dllRev;              // dll rev
    public int nReserved0;          // reserved; return value is
undefined
    public int instrType;           // ML_INSTRUMENT_TYPE enum
value
    public int sampleTechType;      // negative ==> ATR
    public int atrType;             // 1, 3, 9 (for ATR type
sampleTechs)
```

```
    public int spare;                // always returned as 0
    public double dLaserWN;          // in WN
    public double dBasePathLength;   // Transmission/gascell
sampleTechs in mm
    public double dAdjPathLength;    // Transmission/gascell
sampleTechs in mm

    // MLVersion 101+
    // Serial number (in WCHAR-compatible format)
    public short serialNo01;
    public short serialNo02;
    public short serialNo03;
    public short serialNo04;
    public short serialNo05;
    public short serialNo06;
    public short serialNo07;
    public short serialNo08;
    public short serialNo09;
    public short serialNo10;
    public short serialNo11;
    public short serialNo12;
    public short serialNo13;
    public short serialNo14;
    public short serialNo15;
    public short serialNo16;
    public short serialNo17;

    // MLVersion 102+
    public int nCpuBrdRev;
    public int nPwrBrdRev;
    public int nIrBrdRev;
    public int nLasBrdRev;

    // MLVersion 103+
    public int nUpdFwRev;
    public int nBootloaderFwRev;
    public int nFpgaRev;
};

struct _progress
{
    public int nStructSize;   // size bytes of struct (initially
= 28)
```

```
    public FTIR_STATE state;
    public int currentUnits;
    public int totalUnits;
    public int recentRejected;
    public int rejectReason;    // reason, or Good if the last
scan good
    public int numRejectsSame;  // num consec rejects w same
rejectReason
};
```

This page is intentionally left blank.



3. FTIRInst DLL

Summary of function interfaces

int **FTIRInst_SetTargetDeviceUsb**(wchar_t *pDeviceName);

int **FTIRInst_ SetTargetDeviceSerialPort**(long nPort);

int **FTIRInst_ SetTargetDeviceNetwork**(wchar_t *pDeviceName);

int **FTIRInst_Init**();

int **FTIRInst_Deinit**();

int **FTIRInst_SetComputeParams**(PHASEPOINTS ppoints, PHASETYPE ptype, APODTYPE papod, APODTYPE iapod, ZFFTYPE zff, OFFSETCORRECTTYPE offset);

int **FTIRInst_dptrStartSingleBeam**(int numScans, ref double from, ref double to, int res, int bAutoSetBkg, int bAutoSetClean);

int **FTIRInst_dptrStartSpectrum**(int numScans, ref double from, ref double to, int res, DATAXTYPE xtype, DATAYTYPE ytype, int bAutoSetUnknown);

int **FTIRInst_dptrGetLiveSpectrum**(ref double from, ref double to, int res, DATAXTYPE xtype, DATAYTYPE ytype, double[ ] array, int size, ref double actualFrom, ref double actualTo, ref int actualRes);

int **FTIRInst_dptrGetSingleBeam**(double[ ] array, int size, ref double actualFrom, ref double actualTo, ref int actualRes);

int **FTIRInst_dptrGetBackground**(double[ ] array, int size, ref double actualFrom, ref double actualTo, ref int actualRes);

int **FTIRInst_dptrGetClean**(double[ ] array, int size, ref double actualFrom, ref double actualTo, ref int actualRes);

int **FTIRInst_dptrGetSpectrum**(double[ ] array, int size, ref double actualFrom, ref double actualTo, ref int actualRes);

int **FTIRInst_dptrGetRatioSpectrum**(double[ ] bkgarray, double[ ] smparray, double[ ] outarray, int size, DATAYTYPE ytype);

int **FTIRInst_KillCollection**();

int **FTIRInst_SoftReset**();

int **FTIRInst_SetLaserWaveNumber**(ref float newLaser);

int **FTIRInst_SetPathlen**(ref float newPathlength);

int **FTIRInst_GetLaserWaveNumber**(ref float curLaser);

int **FTIRInst_GetPathlenEx**(ref _instrumentMLVersion, ref float curPathlength);

int **FTIRInst_GetVersion**(ref int fwRev, ref int dllRev, ref int serialNo);

int **FTIRInst_GetVersionEx**(ref _instrumentMLVersion _vInfo);

int **FTIRInst_GetStatus**(ref int nEnergyStatus, ref float fBatteryStatus, ref float fSourceCurrentStatus, ref float fSourceVoltageStatus, ref int nLaserStatus, ref float fDetectorStatus);

int **FTIRInst_GetStatusEx**(ref _instrumentMLDiag _dStatus);

FTIR_STATE **FTIRInst_CheckProgress**(ref int currentUnits, ref int totalUnits);

FTIR_STATE **FTIRInst_CheckProgressEx**(ref int currentUnits, ref int totalUnits,ref int rejectedScans);

int **FTIRInst_CheckProgressStruct**(ref _progress pProgress);

int **FTIRInst_StartCoaddedIGram**(int numScans, int nRes, int nPhasePts);

int **FTIRInst_dptrGetCoaddedIGram**(double[ ] pArray, int nArraySize);

int **FTIRInst_dptrSetBackground**(double[ ] pArray, int nSize, double from, double to, int nRes)

int **FTIRInst_dptrGetLiveSingleBeam**(int res, double[ ] pArray, long size, ref double actualFrom,ref double actualTo, ref int actualRes);

int **FTIRInst_dptrGetLiveIGram**(int res, double[ ] array, int size, ref int pActualFrom, ref int pActualTo, ref int pActualRes);

int **FTIRInst_GetIrGain** (ref int nVal);

int **FTIRInst_SetIrGain** (int res, uint flags);

int **FTIRInst_RegisterStatus (**IntPtr whandle int wm_MessageID);

int **FTIRInst_SetAppLedState** (int nLedState);

Typical usage patterns

1. Call one of the **SetTargetDeviceXxx( )** functions to choose the interface to connect over, and to provide any additional identifying information that will allow a connection to a device. If none of these functions are called, the DLL interface will default to connecting to the first USB device that is found.
2. Call **FTIRInst_Init** and check the return value to make sure the instrument connects properly.
3. (Optional) Check version numbers.
4. Call **FTIRInst_SetComputeParams** with the required settings.

5 Call **FTIRInst_dptrStartSingleBeam** to start the data collection sequence.

6 Monitor the instrument status with **FTIRInst_CheckProgressStruct.** When the instrument state changes to FTIR_DataReady, a single beam is ready.

7 Call **FTIRInst_dptrGetSingleBeam** to determine the size of the memory array that will be needed for the returned single beam. This is done by setting the 'array' parameter to zero (and all other parameters to valid values).

8 Allocate a memory array of sufficient size to receive the single beam result and call **FTIRInst_dptrGetSingleBeam** again but this time with the 'array' parameter set to point to the memory array.

9 If you want to get another single beam, go back to Step 4.

10 If you want to collect a spectrum, collect a single beam with the 'setAsBackground' flag on, then call the same sequence (4,5,6,7) but substitute **FTIRInst_dptrStartSpectrum** and **FTIRInst_dptrGetSpectrum** for the single beam calls.

11 You can collect and monitor spectra by calling **FTIRInst_dptrGetLiveSpectrum.** This automatically (and temporarily) switches to numCoadds == 1 and returns when the next spectrum is available. This would typically be used to display a live spectrum for the user or to monitor for sample contact, and so on. Each time **FTIRInst_dptrGetLiveSpectrum** is called, a new spectrum is returned. If you call it before the next is ready, the function will not return until a fresh spectrum is available. Terminate collection of the live spectra by calling **FTIRInst_KillCollection.**

12 To change collection parameters, go back to Step 3.

13 Before exiting the application, be sure to call **FTIRInst_Deinit** for an efficient shutdown.

FTIRInst_SetTargetDeviceUsb

The FTIRInst_SetTargetDeviceUsb function should be called before calling FTIRInst_Init if it is desired to connect to an FTIR device over the USB interface.

C# declaration

int FTIRInst_SetTargetDeviceUsb(wchar_t *pDeviceName);

C++ declaration

long FTIRInst_SetTargetDeviceUsb(wchar_t *pDeviceName);

Parameters

pDeviceName

[in] A placeholder for a pointer to a wide-character string that gives an identifying name to the FTIR device to connect to. **This is not currently used, and should be set to 0 (null).**

Return values

This function returns 0 if successful, otherwise an error code is returned.

-1 == General Error

Remarks

USB is the default connection method. Currently, the first USB device that is found is the one that a connection is made to; multiple USB FTIR devices are not currently supported. In the future, this function may allow a caller to choose among multiple USB FTIR devices. For the time being, the pDeviceName pointer is ignored, and it is recommended that a 0 (null) value be passed in.

FTIRInst_SetTargetDeviceSerialPort

The FTIRInst_SetTargetDeviceSerialPort function should be called before calling FTIRInst_Init if it is desired to connect to an FTIR device over a serial port (or virtual serial port) interface; Bluetooth connections are made using a virtual serial port.

C# declaration

int FTIRInst_SetTargetDeviceSerialPort(int nPort);

C++ declaration

long FTIRInst_SetTargetDeviceSerialPort(long nPort);

Parameters

nPort

[in] Serial port number to connect over; may be virtualized.

Return values

This function returns 0 if successful, otherwise an error code is returned.

-1 == General Error

Remarks

Some FTIR devices are configured with a Bluetooth interface, and can be connected to using a virtual serial port (also known as virtual COM port). The pairing between the host device (for example, PC) and FTIR device must be done outside of the purview of the FTIR DLL, and a virtual serial port assigned. Subsequently the device can be connected to by calling this function with the serial port number that was assigned.

FTIRInst_SetTargetDeviceNetwork

The FTIRInst_SetTargetDeviceNetwork function should be called before calling FTIRInst_Init if it is desired to connect to an FTIR device over the network interface, either wired Ethernet or wireless.

C# declaration

int FTIRInst_SetTargetDeviceNetwork(wchar_t *pDeviceName);

C++ declaration

long FTIRInst_ SetTargetDeviceNetwork(wchar_t *pDeviceName);

Parameters

pDeviceName

[in] A pointer to a wide-character string that gives the network name of the FTIR device to connect to. This may be an IP address in dotted-notation, or a hostname if such a name can be resolved on the local network.

Return values

This function returns 0 if successful, otherwise an error code is returned.

-1 == General Error

Remarks

It is recommended that the device name (hostname) string is a string representation of the IP address of the target FTIR device; for example, ‘192.168.1.2’. Care must be taken that the target FTIR network device is reachable by the client machine — the system routing tables and firewall must be configured to provide a path to the device; pinging the device can assist in ensuring that it is reachable.

It is possible to pass a network device name (hostname) as the input string, but the name must be resolvable by the client machine. Early development should avoid using hostnames and use IP addresses.

This function is called to connect over wired Ethernet or wireless — the local network configuration will dictate whether and how the FTIR device is reached. The FTIR device must have its network configuration set appropriately to appear reachable on the network.

Although the FTIR DLL provides a manner of accessing network devices, it is not able to provide any network support functionality — see your network administrator for more assistance.

FTIRInst_Init

The FTIRInst_Init function should be called before using any other functions in this component DLL.

C# declaration

int FTIRInst_Init();

C++ declaration

long FTIRInst_Init();

Parameters

None.

Return values

This function returns 0 if successful, otherwise an error code is returned. Any other value indicates that the instrument failed to initialize properly.

-1 == Internal Error

-2 == Cannot connect to the instrument. The instrument is probably off or disconnected.

Remarks

Be sure to check the return value. No other FTIRInst functions will succeed if this fails.

FTIRInst_Deinit

The FTIRInst_Deinit function should be called at the conclusion of use of the DLL. If this is not called before exiting, the DLL cannot clean up before terminating, resulting in a very slow application shutdown.

C# declaration

int FTIRInst_Deinit();

C++ declaration

long FTIRInst_Deinit();

Parameters

None.

Return values

This function always returns 0.

Remarks

None.

FTIRInst_SetComputeParams

The FTIRInst_SetComputeParams function should be called before using any of the data collection calls, in order to set the interferogram compute parameters. Default values will be used if no changes are made to the instrument through this call.

C# declaration

```
int FTIRInst_SetComputeParams(
        PHASEPOINTS ppoints,
        PHASETYPE ptype,
        APODTYPE papod,
        APODTYPE iapod,
        ZFFTYPE zff,
        OFFSETCORRECTTYPE offset
        );
```

C++ declaration

```
long FTIRInst_SetComputeParams(
        PHASEPOINTS ppoints,
        PHASETYPE ptype,
        APODTYPE papod,
        APODTYPE iapod,
        ZFFTYPE zff,
        OFFSETCORRECTTYPE offset
        );
```

Parameters

ppoints

[in] The number of phase points to be used for the COMPUTE algorithm.

ptype

[in] Phase correction type. (Currently only Mertz phase correction is supported).

papod

[in] The phase apodization type.

iapod

[in] The interferogram apodization type.

zfftype

[in] The type of zero fill factor to be used in the COMPUTE.

offset

[in] The type of offset correction to be used in the COMPUTE.

Return values

If successful, this function returns a 1 value if successful. A value of 0 is returned for failure.

Remarks

Default values are: 512 Phasepoints, Mertz Correction, Triangular phase apodization, HappGenzel interferogram apodization, NO zero fill factor, NO offset correction.

FTIRInst_dptrStartSingleBeam

The FTIRInst_dptrStartSingleBeam function is called to start the single beam collection of data.

C# declaration

```
int FTIRInst_dptrStartSingleBeam (
        int numScans,
        ref double from,
        ref double to,
        int res,
        int bAutoSetBkg,
        int bAutoSetClean
        );
```

C++ declaration

long **FTIRInst_dptrStartSingleBeam** (
long numScans,
double * from,
double * to,
long res,
long bAutoSetBkg,
long bAutoSetClean
);

Parameters

numScans

[in] The number of scans to be completed.

from

[in] The starting wave number in the spectral range.

to

[in] The ending wave number in the spectral range.

res

[in] The resolution.

bAutoSetBkg

[in] When this is non-zero, the next single beam collected will be kept as the new 'background' reference single beam. This will be used in the calculation of a spectrum.

bAutoSetClean

[in] When this is non-zero, the next single beam collected will be kept as the new 'clean' reference single beam.

Return values

Negative return values are error codes.

-1 == the instrument is not connected (or FTIRInst_Init has not been called).

-2 == specified resolution value is not valid.

-3 == instrument is not in a valid state to start data collection. State must not be FTIR_Collecting or FTIR_Aborting.

If successful, this function returns a positive value. This value corresponds to the number of data points that will be returned later when calling **FTIRInst_dptrGetSingleBeam**. The array size can also be obtained from **FTIRInst_dptrGetSingleBeam**.

Remarks

A single beam in a non-background corrected spectrum. You must acquire a single beam and tag it as the background before you can use **FTIRInst_dptrStartSpectrum.** The background single beam is used in the calculation of the spectrum.

FTIRInst_dptrStartSpectrum

The FTIRInst_dptrStartSpectrum function is called to start the collection of a spectrum. There must be a valid tagged single beam before a spectrum can be calculated.

C# declaration

```
int FTIRInst_dptrStartSpectrum(
        int numScans,
        ref double from,
        ref double to,
        int res,
        DATAXTYPE xtype,
        DATAYTYPE ytype,
        int bAutoSetUnknown
        );
```

C++ declaration

long **FTIRInst_dptrStartSpectrum**(
long numScans,
double * from,
double * to,
long res,
DATAXTYPE xtype,
DATAYTYPE ytype,
int bAutoSetUnknown
);

Parameters

numScans

[in] The number of scans to be completed.

from

[in] The starting wave number in the spectral range.

to

[in] The ending wave number in the spectral range.

res

[in] The resolution.

xtype

[in] Units of the X-axis of the spectrum.

ytype

[in] Units of the Y-axis of the spectrum.

bAutoSetUnknown

[in] (Reserved for future use. Set to zero.)

Return values

If successful, this function returns the number of points in each scan. If no background data is present, a -3 is returned. If background data is available but is incompatible, a -4 is returned. -1 is returned if the scan cannot be started for any other reason.

Remarks

None.

FTIRInst_dptrGetLiveSpectrum

The FTIRInst_dptrGetLiveSpectrum function is called to get the data from the last good collected spectrum.

C# declaration

```
int FTIRInst_dptrGetLiveSpectrum(
        ref double from,
        ref double to,
        int res,
        DATAXTYPE xtype,
        DATAYTYPE ytype,
        double[ ] array,
        int size,
        ref double actualFrom,
        ref double actualTo,
        ref int actualRes
        );
```

C++ declaration

long **FTIRInst_dptrGetLiveSpectrum**(
double* from,
double* to,
long res,
DATAXTYPE xtype,
DATAYTYPE ytype,
double* array,
long size,
double* actualFrom,
double* actualTo,
long* actualRes
);

Parameters

from

[in] The starting wave number in the spectral range.

to

[in] The ending wave number in the spectral range.

res

[in] The resolution.

xtype

[in] The unit type of the X-axis of the interferogram.

ytype

[in] The unit type of the Y-axis of the interferogram.

array

[out] The array of doubles containing the spectral data.

size

[out] The length of the array.

actualFrom

[out] The actual starting wave number from the spectral range.

actualTo

[out] The actual ending wave number from the spectral range.

actualRes

[out] The actual resolution.

Return values

If successful, this function returns the length of the data array. If no background data is present, a -3 is returned. If background data is available but is incompatible, a -4 is returned. -1 is returned if the scan cannot be started for any other reason.

Remarks

This initiates the collection of a sequence of live spectra for monitoring. A background single beam must be available. The first time this is called, the system automatically (and temporarily) switches numScans to 1, and then waits for the completion of one scan, computes a spectrum and returns. Subsequent calls wait for the next spectrum, then return. If scan has been completed before the second call, this function will return that spectrum immediately, but it will not return the same spectrum on consecutive calls.

FTIRInst_dptrGetSingleBeam

The FTIRInst_dptrGetSingleBeam function is called to get a completed single beam.

C# declaration

int **FTIRInst_dptrGetSingleBeam**(
double[ ] array,
int size,
ref double actualFrom,
ref double actualTo,
ref int actualRes);

C++ declaration

long **FTIRInst_dptrGetSingleBeam**(
double* array,
long size,
double* actualFrom,
double* actualTo,
long* actualRes);

Parameters

array

[out] The array of data.

size

[out] The length of the array.

actualFrom

[out] The actual starting wave number from the spectral range.

actualTo

[out] The actual ending wave number from the spectral range.

actualRes

[out]. The actual resolution.

Return values

If successful, this function returns the length of the data array. On error, a zero or negative value is returned.

Remarks

After initiating the collection of a single beam (using **FTIRInst_dptrStartSingleBeam**) and monitoring for FTIR_DataReady (using **FTIRInst_CheckProgressStruct**), the completed single beam may be retrieved using this call.

If you call **FTIRInst_dptrGetSingleBeam** with the array parameter set to zero and all other parameters set properly, the return value will be the length (in number of data points) of the output array. You can then allocate an appropriately-sized array to receive the result data.

FTIRInst_dptrGetBackground

The FTIRInst_dptrGetBackground function retrieves the instrument's stored background spectrum.

C# declaration

int **FTIRInst_dptrGetBackground**(
double[ ] array,
int size,
ref double actualFrom,
ref double actualTo,
ref int actualRes);

C++ declaration

long **FTIRInst_dptrGetBackground**(
double* array,
long size,
double* actualFrom,
double* actualTo,
long* actualRes);

Parameters

array

[out] The array of data.

size

[out] The length of the array.

actualFrom

[out] The actual starting wavenumber from the spectral range.

actualTo

[out] The actual ending wavenumber from the spectral range.

actualRes

[out] The actual resolution.

Return values

If successful, this function returns the length of the data array. On error, a zero or negative value is returned.

Remarks

If you call **FTIRInst_dptrGetBackground** with the array parameter set to zero and all other parameters set properly, the return value will be the length (in number of data points) of the output array. You can then allocate an appropriately-sized array to receive the result data.

FTIRInst_dptrGetClean

The FTIRInst_dptrGetClean function retrieves the instrument's stored Clean spectrum.

C# declaration

int **FTIRInst_dptrGetClean**(
double[ ] array,
int size,
ref double actualFrom,
ref double actualTo,
ref int actualRes);

C++ declaration

long **FTIRInst_dptrGetClean**(
double* array,
long size,
double* actualFrom,
double* actualTo,
long* actualRes);

Parameters

array

[out] The array of data.

size

[out] The length of the array.

actualFrom

[out] The actual starting wavenumber from the spectral range.

actualTo

[out] The actual ending wavenumber from the spectral range.

actualRes

[out] The actual resolution.

Return values

If successful, this function returns the length of the data array. On error, a zero or negative value is returned.

Remarks

If you call **FTIRInst_dptrGetClean** with the array parameter set to zero and all other parameters set properly, the return value will be the length (in number of data points) of the output array. You can then allocate an appropriately-sized array to receive the result data.

FTIRInst_dptrGetSpectrum

The FTIRInst_dptrGetSpectrum function retrieves the instrument's stored spectrum.

C# declaration

```
int FTIRInst_dptrGetSpectrum(
        double[ ] array,
        int size,
        ref double actualFrom,
        ref double actualTo,
        ref int actualRes);
```

C++ declaration

long **FTIRInst_dptrGetSpectrum**(
double* array,
long size,
double* actualFrom,
double* actualTo,
long* actualRes);

Parameters

array

[out] The array of data.

size

[out] The length of the array.

actualFrom

[out] The actual starting wavenumber from the spectral range.

actualTo

[out] The actual ending wavenumber from the spectral range.

actualRes

[out] The actual resolution.

Return values

If successful, this function returns the length of the data array. On error, a zero or negative value is returned. If the array that is passed in is too small, then a value of -9 is returned.

Remarks

Null can be passed in for the array parameter and the length of the data will be returned.

FTIRInst_dptrGetRatioSpectrum

The FTIRInst_dptrGetRatioSpectrum function returns the ratio of the background spectrum with the sample spectrum.

C# declaration

```
int FTIRInst_dptrGetRatioSpectrum(
        double[ ] bkgarray,
        double[ ] smparray,
        double[ ] outarray,
         int size,
        DATAYTYPE ytype);
```

C++ declaration

```
long FTIRInst_dptrGetRatioSpectrum(
        double* bkgarray,
        double* smparray,
        double* outarray,
         long size,
        DATAYTYPE ytype);
```

Parameters

bkarray

[in] The array of doubles containing the background spectrum data.

smparray

[in] The array of doubles containing the sample spectrum data.

outarray

[out] The array of doubles containing the return spectrum.

size

[out] The length of the outarray.

ytype

[in] The unit type of the Y-axis of the spectrum.

Return values

The function always returns size, as passed in, if successful. A value of 0 is returned for failure.

Remarks

None.

FTIRInst_KillCollection

The FTIRInst_KillCollection function stops the collection of data within the instrument.

C# declaration

int **FTIRInst_KillCollection**();

C++ declaration

long **FTIRInst_KillCollection**();

Parameters

None.

Return values

This function returns 0 if successful, otherwise an error code is returned. *Currently no errors are defined.*

Remarks

None.

FTIRInst_SetLaserWaveNumber

The FTIRInst_SetLaserWaveNumber sets the value of the Laser Wavenumber that is stored in the instrument's EEPROM and used in calculation of spectra.

C# declaration

int **FTIRInst_SetLaserWaveNumber**(
ref float newLaser
);

C++ declaration

long **FTIRInst_SetLaserWaveNumber**(
float* newLaser
);

Parameters

newLaser

[in] The value of the wavenumber.

Return values

This function returns 0 if successful, otherwise an error code is returned. *Currently no errors are defined.*

Remarks

The actual wavenumber of the instrument laser is used in the calculation of spectra. This value, after determination by a calibration procedure, is stored in EEPROM in the instrument by calling this function. This function does not implement the calibration procedure; it simply supplies the value to the instrument for use and storage.

FTIRInst_SetPathLen

The FTIRInst_SetPathLen sets the value of the PathLength number that is stored within the instrument's EEPROM.

C# declaration

int **FTIRInst_SetPathlen**(
ref float newPathlength
);

C++ declaration

long **FTIRInst_SetPathlen**(
float* newPathlength
);

Parameters

newPathlength

[in] The value of the new pathlength.

Return values

This function returns 0 if successful, otherwise an error code is returned. *Currently no errors are defined.*

Remarks

None.

FTIRInst_GetLaserWaveNumber

The FTIRInst_GetLaserWaveNumber gets the current value of the laser wavenumber that is stored the instrument's EEPROM.

C# declaration

int **FTIRInst_GetLaserWaveNumber**(
ref float curLaser
);

C++ declaration

long **FTIRInst_GetLaserWaveNumber**(
float* curLaser
);

Parameters

curLaser

[out] The value of the laser wavenumber.

Return values

This function returns 0 if successful, otherwise an error code is returned. *Currently no errors are defined.*

Remarks

None.

FTIRInst_GetPathlenEx

The FTIRInst_GetPathlenEx gets the current value of the pathlength that is stored within the instrument's version information.

C# declaration

int **FTIRInst_GetPathlenEx**(
ref _instrumentMLVersion mlvers,
ref double pathlen
);

C++ declaration

long **FTIRInst_GetPathlenEx**(
_instrumentMLVersion *mlvers,
double *pPathlen
);

Parameters

mlvers

[in] The instrument version information.

pathlen

[out] The value of the current pathlength.

Return values

This function returns 0 if successful, otherwise an error code is returned. *Currently no errors are defined.*

Remarks

None.

FTIRInst_GetVersion

The FTIRInst_GetVersion gets the current value of the firmware version, the DLL version, and the serial number of the instrument.

C# declaration

int **FTIRInst_GetVersion**(
ref int fwRev,
ref int dllRev,
ref int serialNo);

C++ declaration

long **FTIRInst_GetVersion**(
long* fwRev,
long* dllRev,
long* serialNo);

Parameters

fwRev

[out] The current version of the instrument firmware.

dllRev

[out] The current version number of the instrument interface DLL.

serialNo

[out] The serial number of the instrument.

Return values

This function returns 1 if successful, otherwise an error code is returned. *Currently no errors are defined.*

Remarks

This method has been made obsolete by the FTIRInst_GetVersionEx function.

FTIRInst_GetVersionEx

The FTIRInst_GetVersionEx gets the current version information from the instrument.

C# declaration

int **FTIRInst_GetVersionEx**(
ref _instrumentMLVersion _vInfo
);

C++ declaration

long **FTIRInst_GetVersionEx**(
instrumentMLVersion* _vInfo
);

Parameters

vinfo

[out] Current version information from the instrument. The _instrumentMLVersion struct contains the same information as provided by the FTIRInst_GetVersion function and also contains info about the instrument type.

Return values

This function returns 0 if successful, otherwise an error code is returned. A value of -1 is returned if the instrument is not connected, and a value of -2 is returned if the _vInfo pointer is null.

Remarks

The structure that is pointed to by _vInfo must be allocated by the caller, and the nVersion field must be filled in with the appropriate version of the structure. The version much match the size of the structure, since the function will fill in as many fields as the nVersion value dictates.

NOTE There is no longer an integer serial number field in the structure — the serial number is always stored in the wide-character serialNoXX array, allowing any alphanumeric character to appear in a serial number.

To access the serialNoXX array, the caller may need to translate from wide characters (Unicode) to 'multi-byte' (generally 8-bit ASCII); this can be done using the WideCharToMultiByte( ) Windows function. For those callers already using Unicode, they can reference the string directly, starting at the first character in the structure; it may be easier to reference the serialNoXX array as wchar_t serialNo[17] instead of individual characters, depending on the development environment.

FTIRInst_GetStatus

The FTIRInst_GetStatus gets the current status information from the instrument.

C# declaration

```
int FTIRInst_GetStatus(
        ref int nEnergyStatus,
        ref float fBatteryStatus,
        ref float fSourceCurrentStatus,
        ref float fSourceVoltageStatus,
        ref int nLaserStatus,
        ref float fDetectorStatus);
```

C++ declaration

```
long FTIRInst_GetStatus(
        int* nEnergyStatus,
        float* fBatteryStatus,
        float* fSourceCurrentStatus,
        float* fSourceVoltageStatus,
        int* nLaserStatus,
        float* fDetectorStatus);
```

Parameters

nEnergyStatus

[out] The current energy status from the instrument.

fBatteryStatus

[out] The current battery status.

fSourceCurrentStatus

[out] The source current.

fSourceVoltageStatus

[out] The source voltage.

nLaserStatus

[out] The current status of the laser.

nDetectorStatus

[out] The current status of the detector.

Return values

This function returns 1 if successful, otherwise a 0 is returned for failure.

Remarks

This function has been made obsolete by the FTIRInst_GetStatusEx function.

FTIRInst_GetStatusEx

The FTIRInst_GetStatusEx gets the extended status information from the instrument.

C# declaration

int **FTIRInst_GetStatusEx**(
ref _instrumentMLDiag _dStatus
);

C++ declaration

long **FTIRInst_GetStatusEx**(
instrumentMLDiag* _dStatus
);

Parameters

_dStatus

[out] An _instrumentMLDiag struct containing instrument status.

Return values

This function returns 1 if successful, otherwise an error code is returned. A -1 is returned if the instrument is not connected, and a -2 is returned if the _dStatus pointer is null.

Remarks

The structure that is pointed to by _dStatus must be allocated by the caller, and the nVersion field must be filled in with the appropriate version of the structure. The version much match the size of the structure, since the function will fill in as many fields as the nVersion value dictates.

System status values

SYSSTAT_UNCONNECTED 0x00000001 // std startup state

SYSSTAT_CONNECTED 0x00000002

SYSSTAT_CONNECTION_LOST 0x00000100 //was connected, but lost connection

SYSSTAT_SHUTTINGDOWN 0x00001000

SYSSTAT_SHUTDOWN 0x00002000 // implies now unconnected

System shutdown values

SHDOWN_NOTVALID 0x00000000

SHDOWN_USERBUTTON 0x00000001

SHDOWN_BATTERYCRITICAL 0x00000010

FTIRInst_CheckProgress

The FTIRInst_CheckProgress function returns the current state of the instrument.

C# declaration

```
FTIR_STATE FTIRInst_CheckProgress(
        ref int currentUnits,
        ref int totalUnits
        );
```

C++ declaration

```
FTIR_STATE FTIRInst_CheckProgress(
        long* currentUnits,
        long* totalUnits
        );
```

Parameters

currentUnits

[out] The current number of successful scans completed since the FTIRInst_dptrStartSpectrum was called.

totalUnits

[out] The total number of scans to be completed before the spectrum can be computed.

Return values

The function returns one of these values:

- FTIR_Init = 0,
- FTIR_Collecting = 1,
- FTIR_DataReady = 2,
- FTIR_Aborting = 3,
- FTIR_Error = 4

Remarks

This function has been made obsolete by the CheckProgressStruct function.

FTIRInst_CheckProgressEx

The FTIRInst_CheckProgressEx function returns the current state information from the instrument.

C# declaration

```
FTIR_STATE FTIRInst_CheckProgressEx(
        ref int currentUnits,
        ref int totalUnits,
        ref int rejectedScans);
```

C++ declaration

```
FTIR_STATE FTIRInst_CheckProgressEx(
        long* currentUnits,
        long* totalUnits,
        long* rejectedScans);
```

Parameters

currentUnits

[out] The current number of successful scans completed since the FTIRInst_dptrStartSpectrum was called.

totalUnits

[out] The total number of scans to be completed before the spectrum can be computed.

rejectedScans

[out] The total number of rejected scans out of the last 10 scans to be completed.

Return values

The function returns one of the following values:

- FTIR_Init = 0,
- FTIR_Collecting = 1,
- FTIR_DataReady = 2,
- FTIR_Aborting = 3,
- FTIR_Error = 4

Remarks

This function has been made obsolete by the CheckProgressStruct function.

FTIRInst_CheckProgressStruct

The FTIRInst_CheckProgressStruct function returns the current state information from the instrument in the form of a _progress Struct.

C# declaration

int **FTIRInst_CheckProgressStruct**(
ref _progress pProgress
);

C++ declaration

long **FTIRInst_CheckProgressStruct**(
progress* pProgress
);

Parameters

pProgress

[out] The _progress struct containing information about the progress of the current run.

Return values

This function returns the size of the _progress structure that is returned; this value is returned both when a valid pointer is passed in, as well as if a null pointer is passed in.

Remarks

None.

FTIRInst_StartCoaddedIGram

The start call takes the number of scans to coadd, the resolution (in wavenumber: 2, 4, 8, 16), and the number of phase points to the left of the centerburst; valid values for nPhasePts are the standard power-of-two values (128, 256, 512, 1024).

C# declaration

int **FTIRInst_StartCoaddedIGram**(
int numScans,
int nRes,
int nPhasePts
);

C++ declaration

long **FTIRInst_StartCoaddedIGram**(long numScans, long nRes, nPhasePts);

Parameters

numScans

[in] The number of scans to run before returning the completed coadded IGram.

nRes

[in] The resolution (in wavenumber: 2, 4, 8, 16).

nPhasePts

[in] The number of phase points to the left of the centerburst.

Return values

The return value will be 0 for success, and a negative number for failure.

Remarks

None.

FTIRInst_StartCoaddedIGramNotify

The start call takes the number of scans to coadd, the resolution (in wavenumber: 2, 4, 8, 16), and the number of phase points to the left of the centerburst; valid values for nPhasePts are the standard power-of-two values (128, 256, 512, 1024). The client is 'notified' of completion by signal of the passed in Operation System event handle.

C# declaration

int **FTIRInst_StartCoaddedIGramNotify**(
int numScans,
int nRes,
int nPhasePts
IntPtr eventHandle);

C++ declaration

long **FTIRInst_StartCoaddedIGramNotify**(long numScans, long nRes, nPhasePts, HANDLE hReadyEvent);

Parameters

numScans

[in] The number of scans to run before returning the completed coadded IGram.

nRes

[in] The resolution (in wavenumber: 2, 4, 8, 16).

nPhasePts

[in] The number of phase points to the left of the centerburst.

hReadyEvent

[in] Handle of a Windows Event to be set by the DLL when the next IGram is available.

Return values

The return value will be 0 for success, and a negative number for failure.

Remarks

If a valid handle to an Operating System event is passed in, the DLL will call SetEvent on that handle when the requested igram is available. The caller is responsible for creating the event, ensuring that it is not set when the call is made and destroying the event when no longer needed. The system will only set the event once per call to StartCoaddedIGramNotify. After being notified, the application must call StartCoaddedIGramNotify again to request another igram and another notification.

FTIRInst_dptrGetCoaddedIGram

This function fills in the array pointed to by pArray with Igram data. Calling this function with (pArray==0) will return the number of entries in the current array. If (pArray!=0), the nArraySize value MUST be set to the size of the pArray buffer that is being passed in, in elements. This will be used to verify that the array is large enough.

C# declaration

```
int FTIRInst_dptrGetCoaddedIGram(
        double[ ] pArray,
        int nArraySize
        );
```

C++ declaration

long **FTIRInst_dptrGetCoaddedIGram** (
double *pArray,
long nArraySize
);

Parameters

pArray

[out] The array that is to be populated the the coadded IGram data.

nArraySize

[in] The length of the pArray array that is being passed.

Return values

The return value is the number of elements in the array, which may be less (and should be) than the nArraySize value that is passed in. A value of -9 is returned if the size of the passed-in array is too small, per the nArraySize argument.

NOTE It is not guaranteed that the returned array size will always be the same from coadd to coadd. If there is a shift in the position of the centerburst, it may be possible to get more or less points. See remarks below.

Remarks

Note that the number of elements in an Igram array is NOT equal to the number of points dictated by nRes and nPhasePts in the StartCoaddedIgram( ) call. There are a number of padding points that may be added both to the left and right of the Igram, to ensure that a full set of data is provided. These padding points will always be returned in the GetCoaddedIgram( ) call, and can be dealt with as desired by the calling application.

The array size checking is required because there is a rare chance that the size might change, due to receiving a new coadded Igram, between a Get call with (pArray==0) and a subsequent call with (pArray!=0). By design, the system will return a size that is approximately 100 data points larger than necessary. Then, if a small fluctuation happens in the size of the igram, the buffer will have adequate capacity to handle it. The return value from this function will reflect the actual number of data points filled into the array.

FTIRInst_RegisterButton1

This function is called to register the Windows message that is returned when the instrument's trigger is pulled.

C# declaration

```
int FTIRInst_ResigerButton1(
        IntPtr whandle
        ref int wm_MessageID
        );
```

C++ declaration

```
long FTIRInst_ ResigerButton1 (
        HWND whandle,
        long* wm_MessageID
        );
```

Parameters

whandle

[in] The handle of the form that is going to handle the message.

wm_MessageID

[out] The ID of the Windows message that is posted when the trigger on the instrument is pulled.

Return values

This method always returns a 0.

Remarks

None.

FTIRInst_RegisterButton2

This function is called to register the Windows message that is returned when the instrument's navigation buttons are pushed.

C# declaration

```
int FTIRInst_ResigerButton1(
        IntPtr whandle
        ref int wm_MessageID
        );
```

C++ declaration

```
long FTIRInst_ ResigerButton1 (
        HWND whandle,
        long* wm_MessageID
        );
```

Parameters

whandle

[in] The handle of the form that is going to handle the message.

wm_MessageID

[out] The ID of the Windows message that is posted when either of the side buttons on the Agilent 4100 ExoScan FTIR are pressed.

Return values

This method always returns a 0.

Remarks

None.

FTIRInst_dptrSetBackground

The FTIRInst_dptrSetBackground sends background data to the instrument. This background data is then used when the instrument creates spectrums when either calling FTIRInst_dptrGetLiveSpectrum or FTIRInst_dptrStartSpectrum.

C# declaration

int **FTIRInst_SetBackground** (
    double[ ] pArray,
    int nSize,
    double from,
    double to,
    int nRes);

C++ declaration

long **FTIRInst_SetBackground** (
    double* pArray,
    long nSize,
    double* from,
    double* to,
    long nRes);

Parameters

pArray

[in] The array of data containing the background information.

nSize

[in] The size of the data being passed in.

from

[in] The starting X value.

to

[in] The ending X value.

nRes

[in] The resolution.

Return values

This function returns 0 if successful, otherwise an error code is returned:

- -2: Could not access target object buffer.
- -1: Memory allocation error.

Remarks

None.

FTIRInst_dptrGetLiveSingleBeam

The FTIRInst_dptrGetLiveSingleBeam function is called to get the data from the last good collected single beam.

C# declaration

```
int FTIRInst_dptrGetLiveSingleBeam (
        int res,
        double[ ] pArray,
        int size,
        ref double actualFrom,
        ref double actualTo,
        ref int actualRes);
```

C++ declaration

long **FTIRInst_dptrGetLiveSingleBeam** (
long res,
double* pArray,
long size,
double* actualFrom,
double* actualTo,
long actualRes);

Parameters

res

[in] The resolution of the single beam.

pArray

[out] The array that is to be filled with the single beam data.

size

[in] The size of the array being passed in.

actualFrom

[out] The actual From value.

actualTo

[out] The actual To value.

actualRes

[out] The actual resolution.

Return values

This function returns the size of the array if successful. Otherwise, an error code is returned:

- -4: Could not acquire the single beam.

- -9: pArray is too small to hold data.
- -11: Device is no longer connected.

Remarks

To get the size of the data, a null should be passed in as the pArray argument. This will trigger the function to return the size of the array. An array of the correct size should then be allocated and the function should be called a second time while passing the allocated array in as the pArray argument.

FTIRInst_dptrGetLiveIGram

The FTIRInst_dptrGetLiveIGram function is called to get the data from the last good collected IGram.

C# declaration

```
int FTIRInst_dptrGetLiveIGram (
        int res,
        double[ ] pArray,
        int size,
        ref  int actualFrom,
        ref int actualTo,
        ref int actualRes);
```

C++ declaration

```
long FTIRInst_dptrGetLiveIGram (
        long res,
        double* pArray,
        long size,
        long* actualFrom,
        long* actualTo,
        long actualRes);
```

Parameters

res

[in] The resolution of the IGram.

pArray

[out] The array that is to be filled with the IGram data.

size

[in] The size of the array being passed in.

actualFrom

[out] The actual From value.

actualTo

[out] The actual To value.

actualRes

[out] The actual resolution.

Return values

This function returns the size of the array if successful. Otherwise, an error code is returned:

- -4: Could not acquire the interferogram.
- -9: Returned if the pArray parameter is too small for the data. Could also return this value if the size parameter is too small.
- -11: Device is no longer connected.

Remarks

To get the size of the data, a null should be passed in as the pArray argument. This will trigger the function to return the size of the array. An array of the correct size should then be allocated and the function should be called a second time while passing the allocated array in as the pArray argument.

FTIRInst_GetIrGain

The FTIRInst_GetIrGain function is called to get the current Ir Gain value from the instrument.

C# declaration

int **FTIRInst_GetIrGain**(
    ref int nVal);

C++ declaration

long **FTIRInst_GetIrGain**(
    long* nVal);

Parameters

nVal

[out] The value of the Ir Gain variable in the instrument.

Return values

This function returns a 0 if successful. On any error, it will return a -1.

Remarks

None.

FTIRInst_SetIrGain

The FTIRInst_SetIrGain function is called to set the current Ir Gain value in the instrument.

C# declaration

int **FTIRInst_SetIrGain**(
    int nVal,
    uint flags);

C++ declaration

long **FTIRInst_SetIrGain**(
    long nVal,
    unsigned long flags);

Parameters

nVal

[in] The value of the Ir Gain variable that the instrument will use when scanning.

flags

[in] A flag to tell the instrument to give the instrument additional commands. A '0' value will do nothing. '1' will set the value to non-volatile memory.

Return values

This function returns a 0 if successful. On any error, it will return a -1.

Remarks

Passing in a -1 as the nVal argument will make the instrument use the factory-set default value as the gain.

FTIRInst_RegisterStatus

This function is called to register the Windows message that is returned when the instrument's progress changes.

C# declaration

int **FTIRInst_RegisterStatus**(
IntPtr whandle
int wm_MessageID
);

C++ declaration

long **FTIRInst_ RegisterStatus** (
HWND whandle,
long wm_MessageID
);

Parameters

whandle

[in] The handle of the form that is going to handle the message.

wm_MessageID

[out] The ID of the Windows message that is posted when the instrument progress or state changes.

Return values

This method always returns a 0.

Remarks

The wParam parameter of the message that is posted will contain the current status of the instrument. The available values are:

- SYSSTAT_UNCONNECTED 0x00000001 // std startup state
- SYSSTAT_CONNECTED 0x00000002
- SYSSTAT_CONNECTION_LOST 0x00000100
- SYSSTAT_SHUTTINGDOWN 0x00001000
- SYSSTAT_SHUTDOWN 0x00002000

The lParam parameter contains the current progress of the instrument. The upper 16 bits contains the current progress value and the lower 16 bits contains the total progress.

If the instrument status is currently SYSSTAT_CONNECTION_LOST, all get live signal calls will return -11. Most other calls will return a -4 error message. In general, all calls to the FTIRInst assembly should cease while the instrument has any status but SYSSTAT_CONNECTED.

FTIRInst_RegisterStatusEvents

This function is called to register for events to be triggered when the instrument's progress or status changes.

C# declaration

```
int FTIRInst_RegisterStatusEvents(
        IntPtr hEvent
        );
```

C++ declaration

```
long FTIRInst_ RegisterStatusEvents(
        HANDLE hEvent
        );
```

Parameters

hEvent

[in] The handle of the event that will be set by the framework when the instrument progress of state has changed.

Return values

This method always returns a 0.

Remarks

The hEvent parameter gives the handle of an event that will be set by the framework when the instrument progress or state changes. The progress will change as scans are coadded, and the state will change if the instrument is shut down or loses its physical connection.

The client that calls this function should wait for the event to be set, typically using WaitForSingleObject( ) or one of its relatives. After the client is done processing the event, it is responsible for calling ResetEvent( ) so that a new event may be sent.

As part of processing this event, the client will need to ascertain what progress and/or state has changed; this information may be garnered by calling FTIRInst_CheckProgressStruct( ) or FTIRInst_GetStatusEx( ).

FTIRInst_SetAppLedState

This function is called to set the state of the 'Application' LED; possible states are Off, Red, and Amber.

C# declaration

int **FTIRInst_SetAppLedState**(
    int nLedState);

C++ declaration

long **FTIRInst_SetAppLedState**(
    long nLedState);

Parameters

nLedState

[in] The state of the application LED.

Possible states, and their integer values, are:

- Off 1
- Amber 2
- Red 3

Return values

This method returns a 0 if the call was successful, and a -1 if there was an error.

Remarks

The application LED is set to Amber when the FTIR device is first started — this is to signify that the instrument is not yet communicating with a host. After the host software has connected to the device, it may control the LED as it deems appropriate; one example would be to set the LED to the Off state to denote that the host software is communicating with the device. It may also be desirable to set the LED to the Red state when the application has found that an error has occurred.

FTIRInst_I2cAdc_GetReadings

This function is called to retrieve the readings from the external I2C analog-to-digital converter (ADC).

C# declaration

int **FTIRInst_I2cAdc_GetReadings**(
    int *pArray);

C++ declaration

long **FTIRInst_I2cAdc_GetReadings**(
long *pArray);

Parameters

pArray

[out] A pointer to a caller-allocated array of 8 32-bit values; the values will be filled in by this function if successful. Only the least significant 12 bits of each value are relevant, as the supported ADC provides only 12 bits of data.

Return values

This method returns a 0 if the call was successful, and a -1 if there was an error.

Remarks

This function is specific to the Burr-Brown ADS7828 ADC, which is an I2C device that is externally attached to the FTIR device I2C bus, and configured with an address of 0x94 (0x95 with the Read bit set). This devices provides 8 channels of analog-to-digital conversion.

The FTIR device will occasionally — about once every ten seconds — sample the values in the ADC, and cache them for return through this function; calling this function does not initiate an ADC conversion.

Only the least significant 12 bits of each 32-bit value are relevant, since the ADS7828 is a 12-bit ADC that provides only 12 bits of resolution. It is the responsibility of the caller to interpret the 12 bits of information, converting the bits into a voltage, and subsequently understanding what that voltage means.

FTIRInst_I2cIo_SetPinDirs

This function is called to set the directions of the PCA9555 I/O lines.

C# declaration

int **FTIRInst_I2cIo_SetPinDirs**(
    int vals);

C++ declaration

long **FTIRInst_I2cIo_SetPinDirs**(
    long vals);

Parameters

vals

[in] Each of the least significant 16 bits dictates the signal direction for a pin.

Possible bit states, and their integer values, are:

- Output 0
- Input 1

Return values

This method returns a 0 if the call was successful, and a -1 if there was an error.

Remarks

This function is specific to the Philips PCA9555 16-bit port expander. All 16 bits of the port expander are used, and positioned in the least significant bits of the 32-bit values used in this API. Port 0 is in the LSB, while Port 1 is in the next most significant byte.

MSB			LSB
0	0	Port 1	Port 0

By default, the PCA9555 defines all pins as inputs upon power-up or reset. Hardware must deal with such startup conditions appropriately. The PCA9555 has a weak (100 kilohm) pullup resistor on each pin that pull each input pin high when undriven.

FTIRInst_I2cIo_SetPinVals

This function is called to set the drives of the PCA9555 I/O lines.

C# declaration

int **FTIRInst_I2cIo_SetPinVals**(
    int vals);

C++ declaration

long **FTIRInst_I2cIo_SetPinVals**(
    long vals);

Parameters

vals

[in] Each of the 16 least significant bits dictates the drive for a pin defined as an output.

Possible bit states, and their integer values, are:

- Drive Low 0
- Drive High 1

Return values

This method returns a 0 if the call was successful, and a -1 if there was an error.

Remarks

A value written to a pin defined as an input is not relevant as long as the direction remains set to input. The caller is responsible for managing any electrical issues related to driving pins and changing their directions.

See FTIRInst_I2cIo_SetPinDirs for additional information.

FTIRInst_I2cIo_GetPinVals

This function is called to get the values read from the PCA9555 I/O lines.

C# declaration

int **FTIRInst_I2cIo_GetPinVals**(
long *pVals);

C++ declaration

long **FTIRInst_I2cIo_GetPinVals**(
long *pVals);

Parameters

pVals

[out] A pointer to a caller-allocated 32-bit value that will receive the input values sensed by the PCA9555. Each of the 16 least significant bits corresponds with a pin, and is set to describe its read state.

Possible states, and their integer values, are:

- Read Low 0
- Read High 1

Return values

This method returns a 0 if the call was successful, and a -1 if there was an error.

Remarks

The FTIR device will occasionally — about once every ten seconds — sample the values in the PCA9555, and cache them for return through this function. Calling this function does not initiate a capture of data; this makes looking for momentary events (for example, button presses) infeasible.

The bit values for pins that are defined as inputs are more relevant for this function than those defined as outputs. Note that the value read for a pin may differ from what is driven, even for pins defined as outputs, if external circuitry is driving the pin more strongly than the PCA9555 is.

See FTIRInst_I2cIo_SetPinDirs for additional information.

FTIRInst_GetExtTemps

This function is called to retrieve the temperatures measured by up to four temperature sensors attached to the external I2C bus.

C# declaration

int **FTIRInst_GetExtTemps**(
    float *pfTemp1, float *pfTemp2, float *pfTemp3, *pfTemp4);

C++ declaration

long **FTIRInst_GetExtTemps**(
    float *pfTemp1, float *pfTemp2, float *pfTemp3, *pfTemp4);

Parameters

pfTemp1

[out] A pointer to a caller-allocated 32-bit float value that will be filled with the temperature of sensor 1 (I2C address 0x98); in case of an error, the value remains as it was when passed in.

pfTemp2

[out] A pointer to a caller-allocated 32-bit float value that will be filled with the temperature of sensor 2 (I2C address 0x9A); in case of an error, the value remains as it was when passed in.

pfTemp3

[out] A pointer to a caller-allocated 32-bit float value that will be filled with the temperature of sensor 3 (I2C address 0x9C); in case of an error, the value remains as it was when passed in.

pfTemp4

[out] A pointer to a caller-allocated 32-bit float value that will be filled with the temperature of sensor 4 (I2C address 0x9E); in case of an error, the value remains as it was when passed in.

Return values

This method returns a bit-code denoting which temperature sensor values have valid data. If Temp1 is valid, then 0x01 (the LSbit) is ORed in; if Temp2 is valid, then 0x02 is ORed in; if Temp3 is valid then 0x04 is ORed in; if Temp4 is valid then 0x08 is ORed in.

A negative number (-1, -2) is returned in case of an error.

Remarks

This function is specific to LM75 (and compatible) temperature sensors, which are I2C devices that are attached to the FTIR device external I2C bus. Sensor 1 has an I2C address of 0x98, Sensor 2 has an I2C address of 0x9A, Sensor 3 has an I2C address of 0x9C, and Sensor 4 has an I2C address of 0x9E.

Temperature values are read from the temperature sensors no more frequently than every 5 seconds, so no change in values will be seen if this function is called more frequently.

FTIRInst_GetExtTemp

This function is called to retrieve the temperature measured by an OEM temperature sensor attached to the external I2C bus.

C# declaration

int **FTIRInst_GetExtTemp**(
    float *pfTemp);

C++ declaration

long **FTIRInst_GetExtTemp**(
    float *pfTemp);

Parameters

pfTemp

[out] A pointer to a caller-allocated 32-bit float value that will be filled with the temperature of sensor 1 (I2C address 0x96); in case of an error, the value remains as it was when passed in.

Return values

This method returns a 1 if the call was successful, and a negative value (-1, -2) if there was an error.

Remarks

This function is specific to the LM76 temperature sensor, which is an I2C device that can be attached externally to the FTIR device I2C bus by an OEM. The temperature sensor must be of type LM76 (or identical), and must have an I2C address of 0x96.

Temperature values are read from the temperature sensors no more frequently than every 5 seconds, so no change in value will be seen if this function is called more frequently.

FTIRInst_GetOemNvmemData

This function is called to retrieve the OEM data stored in the device's non-volatile memory.

C# declaration

int **FTIRInst_GetOemNvmemData**(
byte[ ] aData,
int nBufSize);

C++ declaration

long **FTIRInst_GetOemNvmemData**(
unsigned char *pData,
long nBufSize);

Parameters

pData

[out] A pointer to a caller-allocated buffer that will be filled with the OEM data that is stored inside of a device.

nBufSize

[in] An integer value that represents the size of the buffer that is being passed in.

Return values

This method returns the number of bytes retrieved if the call was successful; this value is always 128.

A negative value is returned if there was an error; -1 for a general error, -2 for nBufSize too small.

Remarks

Each FTIR device has non-volatile memory, a portion of which can be used to store OEM data; this memory will persist across power losses to the device.

The OEM is solely responsible for managing this data. When an FTIR device is first provided to an OEM, the contents of the OEM NVMEM should contain bytes of all 0xFF. The OEM must choose how to organize his data, must fill the NVMEM with such data, and must interpret the data after reading it from the device. Data is always handled as a single, atomic, 128 B package, for both get and set.

The OEM NVMEM data is fixed at 128 bytes. The nBufSize value that is passed in to this function must be at least 128 B; a larger value will allow the function to continue, while a smaller value will return an error. In general, nBufSize should always be 128.

FTIRInst_SetOemNvmemData

This function is called to set the data that is stored in the device's non-volatile memory.

C# declaration

int **FTIRInst_SetOemNvmemData** (
    byte[ ] aData,
    int nBufSize);

C++ declaration

long **FTIRInst_SetOemNvmemData**(
    unsigned char *pData,
    long nBufSize);

Parameters

pData

[out] A pointer to a caller-allocated buffer that contains the OEM data that is to be stored inside of a device's non-volatile memory.

nBufSize

[in] An integer value that represents the size of the buffer that is being passed in.

Return values

This method returns the number of bytes written if the call was successful; this value is always 128.

A negative value is returned if there was an error; -1 for a general error, -2 for nBufSize too small.

Remarks

Each FTIR device has non-volatile memory, a portion of which can be used to store OEM data; this memory will persist across power losses to the device. See the GetOemNvmemData() documentation for more details.

The OEM NVMEM data is fixed at 128 bytes. The nBufSize value that is passed in to this function should be 128 (bytes); a larger value will allow the function to continue, but only the first 128 bytes will be stored in non-volatile memory; a smaller value will return an error. In general, nBufSize should always be 128.

This page is intentionally left blank.