



**JTECH**MEDICAL  
MEASURING HUMAN PERFORMANCE

# IRIS™

Instrument Registration and Interface Software

## MANUAL

(800) 985-8324 or (385) 695-5000

Fax: (385) 695-5001

[www.jtechmedical.com](http://www.jtechmedical.com)

This Page Intentionally Left Blank

# Contents

Publication Information .....	2
Manufacturer .....	2
Graphic Symbol Definitions .....	3
Command-Line Parameters .....	4
Interprocess Communication .....	5
Window Messages .....	6
Iris Open .....	7
Iris Close .....	7
Iris Reset .....	7
Iris Error .....	8
Enable Device .....	8
Disable Device .....	8
Disable All Devices .....	8
Request Calibration .....	8
Device Data Raw .....	9
Device Data Converted .....	10
Device Event .....	10
Device Status .....	11
Device Idle .....	11
Device Calibration .....	12
Device Calibration (Inclinometer) .....	13
Iris-Shared Software Library .....	13
Device IDs and Device Types .....	13
Function Reference .....	14
IsIrisInternalMessage .....	15
IsIrisExternalMessage .....	15
IsValidDeviceType .....	15
IsValidDeviceID .....	15
IsRemoteDeviceType .....	16
IsRemoteDeviceID .....	16
IsCalibrated .....	16
IsInclinometer .....	16
GetInterfaceType .....	17
GetDeviceType .....	17
GetDeviceName .....	17
GetErrorText .....	17
Index .....	18

JTECH Medical IRIS™ Manual  
**Publication Information**




JTECH Medical Industries, Inc.  
7633 S Main  
Midvale, UT 84047  
United States of America

(800) 985-8324  
(385) 695-5000  
Fax (385) 695-5001  
[www.jtechmedical.com](http://www.jtechmedical.com)

© 2022 JTECH Medical. All rights reserved. This manual is provided as a service to our customers, any other use is strictly prohibited. This manual may not be reproduced by any means, physically or electronically, without the express written consent of JTECH Medical.

IRIS™ is a trademark of JTECH Medical. Microsoft, Word, Notepad and Windows are trademarks or registered trademarks of Microsoft Corporation.

## Graphic Symbol Definitions

Symbol		Meaning
	An EN 980 symbol	Identifying the Manufacturer

## Command-Line Parameters

In order to communicate with a partner application, IRIS must be initially supplied with some basic information. Command-line parameters provide a convenient way to initialize IRIS with this data. Additional configuration settings should also be supplied as needed.

Key Parameters	Format	Options	Description
Partner	-Px	x = Integer	The parameter 'x' should be the HWND of the window in the partner application that IRIS is supposed to communicate with. The specified partner window object must be capable of processing the custom IRIS messages defined in this document in order for communication to work. If this parameter is omitted, IRIS will operate in solo mode.
Partner Type	-Ox	x = Integer	This parameter specifies the type of partner attempting to communicate with Iris (eg. Tracker 5, Northstar, a customer app). This also determines which location in the Windows registry is used to load and save Iris's application settings. This allows Iris to sync its settings with JTECH partners as needed while still maintaining separate solo mode settings, for example. Refer to the <i>IrisPartnerType</i> enumerated values in the Iris code to see which values are possible.
Interface Type	-Ix	x = Integer x = 1 x = 2 x = 3	Refer to the <i>InterfaceType</i> enumerated values in the Iris code to see possible values. Iris runs in Legacy mode, communicating with devices via the Tracker Legacy interface box. Iris runs in Freedom mode, communicating with devices via the Tracker Freedom USB receiver. Iris runs in Echo mode, communicating with Echo devices.
Serial Port	-Sx	x = Integer	(Only applies when <i>InterfaceType</i> is Legacy) The parameter 'x' specifies the COM port to be used for serial port communication.

Some additional parameters are also available for use but are generally only required by JTECH software. Thus most third-party partner applications will not need to use the following parameters

Other Parameters	Format	Options	Description
Echo Device Mode	-Ex	x = Integer  x = 1 x = 2 x = 3	(Only applies when <i>InterfaceType</i> is Echo) Refer to the <i>EchoDeviceType</i> enumerated values in the Iris code to see possible values. Iris will attempt to connect to an Echo receiver connected via USB. Iris will attempt to connect to an Echo device connected via USB. Iris will attempt to connect to an Echo Commander console connected via USB.
Warning Mode	-W		Turns on Iris's warning mode, which causes non-critical warning messages to be reported alongside critical error messages (which are always displayed).
Verbose Mode	-V		Turns on Iris's verbose mode, which causes much more detail to be output to Iris's text log file. This is often helpful when trying to test and debug Iris.
History Mode	-H		Turns on Iris's history mode, which results in Iris appending new entries to its primary log file instead of overwriting it each time it starts up.

## Interprocess Communication

IRIS uses application-specific [Windows messages](#) to pass information back and forth with a partner application. The [PostMessage](#) Windows API function is all that is needed to transmit these messages. The proper way to receive these messages depends on the development environment used to create the partner. For example, a .NET [Windows Forms](#) application should have a form class that overrides the [Form.WndProc](#) method. Refer to the sample Iris-Partner application and its source code for an illustration of this functionality.

Communication between IRIS and its partner is relatively straightforward. The partner must first launch the IRIS process with appropriate command-line parameters. Upon startup, IRIS will initialize itself according to these parameters and then respond with its own message window handle. Once the partner has received this first message with IRIS's handle, it can then send device command messages to IRIS. After a device is enabled and connected, IRIS will send its partner a steady stream of device data and status messages for that device. Multiple devices can be simultaneously connected (up to a maximum of 4), and data messages are usually sent over at a rough rate of 10 per second per device.

## Window Messages

Message Description	Sender	Message ID	WParam (32 bits)	LParam (32 bits)
Iris Open	IRIS	WM_IRIS_OPEN	Window Handle	—
Iris Close	Both	WM_IRIS_CLOSE	Close Type	—
Iris Reset	IRIS	WM_IRIS_RESET	Partner Handle	Reset Options
Iris Error	IRIS	WM_IRIS_ERROR	Error ID	—
Enable Device	Partner	WM_IRIS_DEVICE_ENABLE	Device ID	—
Disable Device	Partner	WM_IRIS_DEVICE_DISABLE	Device ID	—
Disable All Devices	Partner	WM_IRIS_DEVICE_DISABLE_ALL	—	—
Request Calibration	Partner	WM_IRIS_DEVICE_GET_CALIBRATION	Device ID	—
Device Data Raw	IRIS	WM_IRIS_DEVICE_DATA_RAW	Device ID	Timestamp & Data
Device Data Converted	IRIS	WM_IRIS_DEVICE_DATA_CONVERTED	Device ID	Timestamp & Data
Device Event	IRIS	WM_IRIS_DEVICE_EVENT	Device ID	Timestamp & Data
Device Status	IRIS	WM_IRIS_DEVICE_STATUS	Device ID	Status & Power Level
Device Idle	IRIS	WM_IRIS_DEVICE_IDLE	Device ID	Idle Time
Device Calibration Date	IRIS	WM_IRIS_DEVICE_CALIB_DATE	Device ID	Calibration Data
Device Calibration Zero Count	IRIS	WM_IRIS_DEVICE_CALIB_ZERO_COUNT	Device ID	Calibration Data
Device Calibration Offset Count	IRIS	WM_IRIS_DEVICE_CALIB_OFFSET_COUNT	Device ID	Calibration Data
Device Calibration Offset	IRIS	WM_IRIS_DEVICE_CALIB_OFFSET	Device ID	Calibration Data
Device Calibration Inc Pos 1	IRIS	WM_IRIS_DEVICE_CALIB_INC_POS_1	Device ID	Calibration Data
Device Calibration Inc Pos 2	IRIS	WM_IRIS_DEVICE_CALIB_INC_POS_2	Device ID	Calibration Data
Device Calibration Inc Pos 3	IRIS	WM_IRIS_DEVICE_CALIB_INC_POS_3	Device ID	Calibration Data
Device Calibration Inc Pos 4	IRIS	WM_IRIS_DEVICE_CALIB_INC_POS_4	Device ID	Calibration Data

The various types of messages used for this interprocess communication are listed in the table above. Each message is identified by a unique ID (defined in the Iris-Shared library) and contains two additional integer parameters. The size of these parameters is always assumed to be 32 bits, even when running on 64-bit systems. The message type determines how these parameters are interpreted. Refer to a particular message description section to see more details about its parameters.



## Iris Open

Parameters: The sender's window handle.

This message is used to establish and maintain communication between IRIS and its partner. Upon startup and periodically thereafter, IRIS will send this message to its designated partner. This informs the partner application that IRIS is up and running and also provides the essential window handle (HWND) which allows the partner to send messages back to IRIS.

## Iris Close

Parameters: The desired *CloseType* value (defined below and in the Iris-Shared library).

The partner application should send IRIS this message whenever it wants IRIS to close (usually after device communication is no longer required). The close type parameter specifies additional information about whether IRIS should close. Upon receiving this message and determining it should close (depending on its status), IRIS disconnects any devices still connected, closes all open ports/handles, and exits.

IRIS will also send its partner this same message (with a "notify" parameter value) anytime it closes, regardless of what or who initiated its closing.

Name	Value	Description
CT_NORMAL	0x01	Standard close message, ignored if IRIS is currently visible.
CT_FORCE_CLOSE	0x02	Forceful close message, which closes IRIS even if it is visible.
CT_NOTIFY	0x03	Notification sent back to partner that IRIS is closing.

## Iris Reset

Parameters: The sender's partner handle (WParam) and other reset options (LParam).

This message is sent from a newly created instance of IRIS to any other previously running instances of IRIS. A Windows mutex object is used to make sure only one instance of IRIS is capable of communicating with devices at a time, so any additional instances of IRIS will use this message type to notify the original instance of their creation before terminating. The relevant command-line parameters used to start up each new instance are passed via this message to the original instance.

This paradigm makes life easier for partner applications. Rather than having to check for an existing IRIS instance every time a new device connection is required, partner applications can simply create a new instance of IRIS. This new instance of IRIS will then carry out the annoying work of looking for another instance of itself. If it finds one, it simply forwards its startup parameters via this message and then terminates. The original instance can then process the new parameters, resetting and reconfiguring itself as requested.

## **Iris Error**

Parameters: The error type.

These messages are sent to the partner whenever IRIS encounters an error. The error type corresponds to an enumerated ErrorID type. The Iris-Shared library defines all possible values and also provides a method for retrieving a brief text description for each one.

Not all error codes correspond to “critical” errors, and some of the more common ones should have tailored responses. For example, the *ErrorID.FreedomNoReceiver* code signifies that a Freedom USB receiver could not be found, so a helpful response might include a prompt instructing the user to connect their receiver to the computer.

## **Enable Device**

Parameters: The ID of the device to be enabled.

This command instructs IRIS to enable the specified device and commence data transfer. If IRIS can successfully connect to the device, then it will start forwarding device data messages along to the partner application. IRIS will also periodically send device status messages indicating the device’s connection state and power level.

## **Disable Device**

Parameters: The ID of the device to be disabled.

This command instructs IRIS to disable the specified device and stop data transfer. If IRIS determines that the specified device is currently connected, then this connection will be closed. Once IRIS confirms the device is disconnected, it will send a device status message indicating as much.

## **Disable All Devices**

Parameters: None

This command instructs IRIS to disable all currently connected devices. As a result, no more data messages from any device will be sent to the partner application until devices are enabled again.

## **Request Calibration**

Parameters: The ID of the device whose calibration data is requested.

This command instructs IRIS to retrieve the specified device’s factory calibration data (only applicable for instruments requiring calibration). This calibration data, once received from the device, will then be passed along to the partner application via multiple device calibration messages.

## Device Data Raw

Parameters: The device ID and a single sample of native (raw) data.

IRIS sends this message to its partner application whenever new data is received from an enabled device. Under normal operating conditions, connected devices are sampled every tenth of a second, so roughly 10 data messages will be sent every second.

Native device data is encoded in the lower 24 bits of the LParam. A data timestamp is also included, encoded in the upper 8 bits of the LParam. This is a circular timestamp which increments from 0 to 127 then cycles back to 0. This value can be used by the partner to ensure that data messages aren't lost or received in the wrong order.

In general, these raw data messages can generally be ignored by most users. However, researchers or data enthusiasts might want to monitor these values for their own purposes. Furthermore, if manual recalibration (or re-zeroing) is desired, then native instrument values become essential.

Older Freedom and Legacy inclinometers will not send these messages at all, but the newer Freedom and Echo accelerometer-based inclinometers will. These newer inclinometers natively send signed 12-bit X and Y positional values, which are then packaged together in this message as a single 24-bit value.

Native values for force gauges (e.g. grip strength gauge) and the goniometer are unsigned integral values that increase linearly as force (or angle) increases. Native values for these instruments usually range from 0 to 4095 (0x0 to 0xFFF), but future devices might use all available 24 bits. These native values must be converted before being interpreted as meaningful force (or angle) values.

Possible native values for the heart rate monitor are 0 (no heart beat detected) and 1 (heart beat detected).

## Device Data Converted

Parameters: The device ID and a single sample of converted data.

Like the corresponding raw device data message, IRIS sends this message to its partner application whenever new data is received from an enabled device. The timestamp is calculated and encoded the same way as it is for raw data messages.

For most JTECH instruments, converted data values are calculated using their stored factory calibration values. Thus if these values are missing or invalid, no converted data messages can be sent.

Converted data is encoded as a signed 24-bit integer which is equal to the actual decimal value multiplied by 1,000 and rounded to the nearest whole number. This allows us to send values as large as  $\pm 8,388$  with up to 3 decimal points of precision.

Converted data for angular devices (inclinometers and goniometers) will range from  $0^\circ$  to  $359^\circ$ . It should be noted that for the older Freedom and Legacy inclinometers, these angles are relative to an arbitrary  $0^\circ$  point established when the device is first powered on. Converted values for force gauges are always sent in Newtons. Converted values for heart rate monitors are calculated pulse rates and are sent in beats per minute (bpm). Some examples: A goniometer's calculated angle of  $234^\circ$  is encoded as 0x039210 (234,000). A static strength gauge's calculated push force (push values are sent as negative values) of 123.456 N is encoded as 0xFE1DC0 (-123,456). A heart rate monitor's calculated pulse rate of 88 bpm is encoded as 0x0157C0 (88,000).

## Device Event

Parameters: The device ID and the event type.

IRIS sends this to its partner application whenever a defined event is triggered by an enabled device. Currently the primary inclinometer, the goniometer, and the hand/foot switches are the only devices that can raise these events, which occur whenever an Enter or Next button is pressed on the corresponding device.

Much like the device data message, the lower 24 bits of LParam indicate which event occurred, with the following possible values (defined in the Iris-Shared library):

Name	Value	Description
DE_ENTER	0x01	Enter button pressed
DE_NEXT	0x02	Next button pressed

A data timestamp is also included, encoded in the upper 8 bits of the LParam. This timestamp is based on the same timestamp format established for device data messages, so it can be used to synchronize each event to the data which was sampled at that same time.

## Device Status

Parameters: The device ID and its current status, namely its connection state and current power level.

IRIS sends these messages to its partner application whenever it detects changes to a device's status (determined from various internal messages received from the Tracker Freedom wireless receiver). These messages contain information regarding the device's connection status and its power level.

The connection state is encoded in the upper 16 bits of LParam, with the following possible values (defined in the Iris-Shared library):

Name	Value	Description
DS_UNKNOWN	0x00	Device state unknown (yet to be determined)
DS_DISABLED	0x01	Device disconnected (sent after a disable device request succeeds)
DS_ENABLED	0x02	Device connected (sent after an enable device request succeeds)
DS_CONNECTING	0x03	Device connecting (sent after a device is enabled but not yet connected)
DS_INTERRUPTED	0x04	Device interrupted (sent after a device is enabled but loses connection with Iris for a few seconds).
DS_LOST	0x05	Device lost (sent after a device connection is interrupted for several seconds)
DS_ERROR	0x06	Device is in an error state

The device's power level is encoded in the lower 16 bits of LParam, with the following possible values (defined in the Iris-Shared library):

Name	Value	Description
PL_UNKNOWN	0x00	Power level unknown or not applicable
PL_VERY_LOW	0x01	Very low power level (recharge immediately)
PL_LOW	0x02	Low power level (roughly 5-20 minutes of power left)
PL_NORMAL	0x03	Normal power level

## Device Idle

Parameters: The device ID and the idle duration (in seconds).

Note: This message is currently sent for Echo instruments only.

Iris sends these messages to its partner application whenever the idle state of a connected device changes. This allows partners to respond appropriately, perhaps notifying the user or even disabling the idle device.

Iris detects idleness (inactivity) by monitoring the incoming data stream of each connected device. If enough

time passes with only minor fluctuations in data, the corresponding device is considered to be idle, and this message is generated. The idle time threshold is set at 5 minutes (300 seconds).

Idle messages will be generated for every multiple of the idle time interval (i.e. every 5 minutes) and will continue as long as the device remains idle.

Once an idle device becomes active again (indicated by non-negligible changes in its data stream), this same idle message will be sent, but with its idle duration parameter (LParam) set to 0. Thus an idle duration of 0 effectively signifies the device is not idle anymore.

For example, suppose a grip gauge is connected but is left lying untouched on a table for 18 minutes. After 5 minutes have passed, Iris will send this message with an idle duration parameter of 300 (300 s = 5 min). Another message will be sent at 10 minutes with an idle duration of 600, and a third at 15 minutes with a value of 900. After 18 minutes the grip gauge is picked up again and a test is started. This will generate another idle message, but this time with an idle duration of 0. If the grip gauge is continuously used thereafter, then no more idle messages will be sent.

### Device Calibration

Parameters: The device ID and various factory calibration data.

IRIS sends these messages to its partner application whenever it receives factory calibration data from a connected instrument. These messages will result from a request for factory calibration data but can also arrive unsolicited (usually right after a device first connects). Because the calibration data can't all fit within the 32-bit LParam field at once, multiple device calibration messages must be sent.

The calibration date is sent as an integer date, defined as the number of days elapsed between the factory calibration date and January 1, 2005. Zero and offset counts are already integer values and can therefore be sent directly without any conversion.

Offsets are encoded the same way as converted data values, i.e. by converting the original value to the appropriate units, multiplying the result by 1000, and rounding to the nearest integer. For force devices the offset is the weight used to calibrate the device (converted to Newtons). For goniometers, the offset should always be 180° since they are calibrated using 180° as the offset position.

For example, suppose a pinch device was factory calibrated on February 29, 2012 using an offset weight of 20 lb. Its calibration date would be encoded as 0xA37 (2615 day difference) and its offset would be 0x15B84

**(88964 = 88.964 N \* 1000 and 20 lb ≈ 88.964432 N)**

## Device Calibration (Inclinometer)

Parameters: The device ID and its factory calibration data.

IRIS sends these messages to its partner application whenever it receives factory calibration data from a connected inclinometer. These messages will result from a request for factory calibration data but can also arrive unsolicited (usually right after a device first connects). Because the calibration data can't all fit within the 32-bit LParam field at once, multiple device calibration messages must be sent.

The inclinometer's calibration data is sent using the standard calibration data message, while the four pairs of inclinometer calibration positions are each encoded in their own message. The x and y values are all signed 12-bit integers, so each one is first converted to a signed 16-bit integer. The converted x value is then stored in the upper 16 bits of LParam, and the y value is stored in the lower 16 bits.

## Iris-Shared Software Library

Iris-Shared is a C++/CLI library intended for use by both IRIS and its partners. As a .NET assembly it can be added and used by other .NET projects (regardless of programming language) relatively easily. It also contains a pair of C++ source files (IrisShared.h and IrisShared.cpp) that can be directly embedded and compiled within other C++ projects (obviating the need to link to and deploy the Iris-Shared library).

The primary purpose of Iris-Shared is to provide the various constants and enumerated values required for IRIS's interprocess communication, including the message IDs which identify each IRIS-specific Windows message. Most of these are described in more detail in the "Window Messages" section of this document. Some additional functions are also provided, most of which help classify IRIS messages and JTECH devices, and these are detailed below.

## Device IDs and Device Types

Several of the Iris-Shared functions deal with device IDs and device types, which bear some additional explanation. Each JTECH device is identified by a unique, hex integer device ID (serial number), and its device type is actually embedded within that device ID. Iris-Shared provides functions for determining a device's type from its ID, which can then be used in other Iris-Shared functions to retrieve additional information concerning that device type.

In many cases knowing a particular device's type is important, especially when determining how its data messages should be interpreted and displayed. The currently defined device types are displayed below.

Device	Device Type (Hex)
Wireless USB Receiver	0x00
Inclinometer (Primary)	0x01
Inclinometer (Secondary)	0x02
Muscle Tester	0x03
Isometric Muscle Tester (IsoTrack)	0x04
Static Strength Gauge	0x05
Goniometer	0x06
Algometer	0x07
Grip Strength Gauge	0x08
Pinch Strength Gauge	0x09
Heart Monitor	0x0A
Foot Switches	0x0B
Hand Switch	0x0C
Commander Console	0x0D
Dualer (Primary)	0x0E
Dualer (Secondary)	0x0F

## Function Reference

This section provides detailed information regarding each available function and its possible parameters. Only the C++/CLI syntax of each function is provided since the corresponding C++ versions contained in the IrisShared.\* files are virtually identical.

### IsIrisMessage

<b>Syntax</b>	<code>static bool IsIrisMessage(int messageID)</code>
<b>Parameters</b>	messageID: The ID of the Windows message to check.
<b>Result</b>	True if the message ID falls within the range used for IRIS messages, false otherwise.

This function is most useful when first processing incoming Windows messages (usually in the partner message window's *WndProc* method or its equivalent). It allows client code to quickly distinguish between IRIS and non-IRIS messages.



## IsIrisInternalMessage

<b>Syntax</b>	<code>static bool IsIrisInternalMessage(int messageID)</code>
<b>Parameters</b>	messageID: The ID of the Windows message to check.
<b>Result</b>	True if the message ID falls within the range used for “internal” IRIS messages, false otherwise.

This function is almost identical to *IsIrisMessage*, except that only the subset of IRIS messages that are categorized as “internal” is considered. Internal messages are only used within the IRIS application, so partner code shouldn’t ever need to use this function.

## IsIrisExternalMessage

<b>Syntax</b>	<code>static bool IsIrisExternalMessage(int messageID)</code>
<b>Parameters</b>	messageID: The ID of the Windows message to check.
<b>Result</b>	True if the message ID falls within the range used for “external” IRIS messages, false otherwise.

This function is almost identical to *IsIrisMessage*, except that only the subset of IRIS messages that are categorized as “external” is considered. External messages are intended to be passed between IRIS and its partner. All possible external messages are listed and detailed elsewhere in this document.

## IsValidDeviceType

<b>Syntax</b>	<code>static bool IsValidDeviceType(DeviceType device)</code>
<b>Parameters</b>	device: The DeviceType value to check for validity.
<b>Result</b>	True if ‘device’ matches any of our valid device types, false if Unknown or an undefined value.

This function determines whether the specified device type matches one of our predefined enumerated values.

## IsValidDeviceID

<b>Syntax</b>	<code>static bool IsValidDeviceID(int deviceID, InterfaceType interfacetype)</code>
<b>Parameters</b>	deviceID: The device ID to check for validity. interfaceType: The InterfaceType corresponding to the device ID being checked.
<b>Result</b>	True if the device ID is valid for a device within the InterfaceType device family, false otherwise.

This function determines whether the specified device ID is valid, which is especially handy when verifying a device ID manually entered by a user. Because different device families (represented by the various *InterfaceType* enumerated values) have different device ID formats, this additional parameter is needed.

## IsRemoteDeviceType

<b>Syntax</b>	<code>static bool IsRemoteDeviceType(DeviceType device)</code>
<b>Parameters</b>	device: The DeviceType value to check.
<b>Result</b>	True if 'device' matches any of our valid remote device types, false otherwise.

This function determines whether the specified device type matches one of our predefined enumerated values for remote devices. Remote device types can communicate wirelessly with a separate wireless receiver, while the other device types must always be directly plugged in. Sometimes it can be useful to quickly make this distinction.

## IsRemoteDeviceID

<b>Syntax</b>	<code>static bool IsRemoteDeviceID(int deviceID, InterfaceType interfacetype)</code>
<b>Parameters</b>	deviceID: The device ID to check. interfaceType: The InterfaceType corresponding to the device ID being checked.
<b>Result</b>	True if the device ID is valid for a remote device within the InterfaceType device family, false otherwise.

This function determines whether the specified device ID is a remote device type. It is almost identical to the *IsValidDeviceID* function, but only remote device types are considered valid here (see the *IsRemoteDeviceType* function as well).

## IsCalibrated

<b>Syntax</b>	<code>static bool IsCalibrated(DeviceType device)</code>
<b>Parameters</b>	device: The DeviceType value to check.
<b>Result</b>	True if 'device' is a device type that requires calibration.

This function determines whether the specified device type corresponds to a device that requires calibration (i.e. test instruments that measure force or angles). Sometimes it can be useful to quickly classify devices based on this distinction.

## IsInclinometer

<b>Syntax</b>	<code>static bool IsInclinometer(DeviceType device)</code>
<b>Parameters</b>	device: The DeviceType value to check.
<b>Result</b>	True if 'device' is an inclinometer.

This function determines whether the specified device is an inclinometer. Sometimes it can be useful to quickly classify devices based on this distinction.

## GetInterfaceType

<b>Syntax</b>	<code>static InterfaceType GetInterfaceType(int deviceID)</code>
<b>Parameters</b>	deviceID: The device ID to check.
<b>Result</b>	The InterfaceType (device family) matching the specified 'deviceID', or Unknown if this cannot be determined.

This function attempts to determine which device family a given device ID belongs to. This can usually be deduced from most valid device IDs (the number of bits used in the ID is the most telling indicator). However, other device IDs might be valid in multiple families (or none), in which case this function returns the result *Unknown*.

## GetDeviceType

<b>Syntax</b>	<code>static DeviceType GetDeviceType(int deviceID, InterfaceType interfaceType)</code>
<b>Parameters</b>	deviceID: The device ID to evaluate. interfaceType: The InterfaceType corresponding to the device ID being checked.
<b>Result</b>	The DeviceType matching the specified 'deviceID', or Unknown if this cannot be determined.

This function attempts to determine which device type a given device ID corresponds to. This is possible because the device type is embedded within the device ID, but it can only be determined with certainty if the device family (interface type) is also known.

## GetDeviceName

<b>Syntax</b>	<code>static String^ GetDeviceName(DeviceType device)</code>
<b>Parameters</b>	device: A valid DeviceType value.
<b>Result</b>	A short text description matching the specified 'device', or an empty (but not null) string if no match is found.

This function retrieves the short text "name" of the specified device type (e.g. "Goniometer").

## GetErrorText

<b>Syntax</b>	<code>static String^ GetErrorText(ErrorID errorID)</code>
<b>Parameters</b>	errorID: A valid ErrorID value.
<b>Result</b>	A short text description matching the specified 'errorID', or "Unknown IRIS Error" if no match is found.

This function retrieves the short text description for the specified error ID. This is most useful when an "Iris Error" message is received and client code wishes to display a simple error message.

# Index

Command-Line Parameters	4
Device Calibration	12
Device Calibration (Inclinometer)	13
Device Data Converted	10
Device Data Raw	9
Device Event	10
Device Idle	11
Device IDs and Device Types	13
Device Status	11
Disable All Devices	8
Disable Device	8
Enable Device	8
Function Reference	14
GetDeviceName	17
GetDeviceType	17
GetErrorText	17
GetInterfaceType	17
Graphic Symbol Definitions	3
Interprocess Communication	5
Iris Close	7
Iris Error	8
Iris Open	7
Iris Reset	7
Iris-Shared Software Library	13
IsCalibrated	16
IsInclinometer	16
IsIrisExternalMessage	15
IsIrisInternalMessage	15
IsRemoteDeviceID	16
IsRemoteDeviceType	16
IsValidDeviceID	15
IsValidDeviceType	15
Manufacturer	2
Publication Information	2
Request Calibration	8
Window Messages	6