Irisys LinuxAPI for IRC3xxx Series (Blackfin-based) Devices

API Version 3.0.40504.01

Contents

1 Irisys Linux API for IRC3xxx series (Blackfin-based) devices

1.1 Introduction

The Irisys IRC3xxx series of People Counters provide TCP/IP connectivity via an IP master device. Irisys provides four Application Programming Interfaces (APIs) that developers can use to connect to and download count data from IRC3xxx series devices, in addition to setting and reading a number of user configurable identification strings on the device. The available APIs are;

- .NET (3.5)
- Java (1.6)
- Win32 (C++)
- Linux (C++)

Third parties can use these APIs to integrate count data from IRC3xxx series devices into their own custom applications. It is not intended to be used for the setup of counter networks or the retrieval of advanced data such as path maps or video, thus no functionality is provided for these tasks.

This documentation is intended for use by developers with a working knowledge of application programming and it is recommended to be read alongside the accompanying Sample Applications. Note that this documentation assumes devices have been installed in accordance with our recommended practices, as outlined in the accompanying Installation Guide IPU40182 and Applications Guide IPU40184. Failure to follow these practices may affect the accuracy of the count data provided by IRC3xxx series devices.

The IRC3xxx series of devices use a Blackfin processing chip. Blackfin is a Registered Trademark of Analog Devices Inc.

The Linux/C++ API supports three platforms:

- x86 32 bit Intel architecture
- x86 64 bit architecture
- AVR32 Atmel architecture

The Linux API was built and compiled using gcc version 4.3.3. The library is built on top of the pthread library and this library must be included in the linker build path. Note that for the AVR32 platform, pthread support must be included in the kernel. For example, using gcc, use the -lpthread switch. Similarly to include the API library use the -lBlackfinAPI switch.

Because several platforms are supported under one API, for clarity we have redefined the integral data types (short, int, long and long long). These can be found in the user header file PlatformDataTypes.h. We have used a simple naming scheme, for example i_uint32 is equivalent to a 32 bit unsigned integer, and i_sint64 is equivalent to a 64 bit integer (or long long). If you are developing for a single platform it is safe to cast between our types and an equivalent length integral type. For example i_sint32 can be safely cast to an int on a 32 bit system.

1.2 Release notes

1.2.1 Version 3.0.40504.01 (April 2012)

This release of the API adds support for new counter features in the latest firmware for IRC3xxx series devices, such as the ability to support up to 32 count registers.

Changelog

- Serial numbers are now represented more accurately using the uint32_t data type rather than as an unsigned long to resolve portability issues
- Added new Irisys::Blackfin.GetRegisterLabels() function to allow the new count register labels to be downloaded
- Now allows the setting of device ID strings up to 160 bytes in length, supported by DSP firmware version 475
- Added new Irisys::Blackfin.GetNetworkChecksum() function to create a checksum of the settings for all devices on the connected CAN network

1.2.2 Version 2.0.30816.01 (August 2011)

In this version of the API the Irisys namespace has been introduced to avoid any potential for function or object name collisions with other libraries. Additionally some functions have been renamed to provide consistency across all Irisys APIs for Blackfin-based devices. This is a breaking change from previous versions of the API and therefore any code based on an older version of the API will need to be updated to compile with this version. Other significant changes in this version include support for multiple (>2) count lines, Irisys time zones and improved documentation.

Changelog

- Added new user header file BlackfinAPI.h, which should be used instead of Blackfin.h
- Moved classes and structs into the Irisys namespace
- Added support for Irisys time zones
- Added support for getting & setting a new 'user string' setting
- Renamed the following functions for consistency with other APIs;
  - Irisys::Blackfin::GetTime() to Irisys::Blackfin::GetUnitTime()
  - Irisys::Blackfin::SetTime() to Irisys::Blackfin::SetUnitTime()
  - bfclass::GetClientConfigPortNumber() to Irisys::Blackfin::GetClientConfigPort()
  - Irisys::Blackfin::SetClientConfigPortNumber() to Irisys::Blackfin::SetClientConfigPort()
  - Irisys::Blackfin::SetTimeout() to Irisys::Blackfin::SetPacketTimeout()
- Changed out parameter type of Irisys::Blackfin::GetSerialNumber() from std::string to unsigned long
- Added Irisys::ConvertBlackfinSerialNumber() utility function to convert from numeric to alphanumeric serial number representations

2 Basic usage

This section provides an introduction to the basic usage of the Irisys API (Linux/C++) to connect to a device and perform some simple tasks with it. For more detailed information about the functions named in this section and other functions available in the API refer to the included class documentation.

2.1 Initializing the API

Before attempting to use any of the functions or classes provided with this API you must first initialise it by creating an object of type Irisys::BlackfinEngine and calling its Irisys::BlackfinEngine::StartEngine() function. This object will be used by your application to create and destroy connections to devices.

Upon shutdown your application should clean up this object by calling its Irisys::BlackfinEngine::ShutdownEngine() function. Note that any connected Irisys::Blackfin objects should first be disconnected by passing them to the engines Irisys::BlackfinEngine::RemoveBlackfinEndPoint() function. After shutting down the engine your application must not use any other functions or classes provided with this API.

2.2 Connecting to a device

Your application can connect to a device in one of two ways - directly or indirectly via client connection mode. To start a direct connection you should create a socket and connect it to the IP address of the deviceyou wish to communicate with. Once the socket is connected your application should create an Irisys::Blackfin object and pass both it and the connected socket to the Irisys::BlackfinEngine::AddBlackfinEndPoint() function of your initialized engine object.

To use an indirect connection you must configure the client connection mode on the device to connect to the IP address or hostname of the machine which will run your application. The application should create a socket and bind it to the network interface and port number upon which the connection(s) will be received and wait to accept a connection. Upon accepting a connection the application should create an instance of Irisys::Blackfin and pass it with the accepted socket to the Irisys::BlackfinEngine::AddBlackfinEndPoint() function of your engine object.

2.3 Disconnecting from a device

To disconnect from a device once you have finished communicating with it or after a comms error has occured you should pass the Irisys::Blackfin object to the Irisys::BlackfinEngine::RemoveBlackfinEndPoint() function of engine object you originally used to connect to the device. Once this is done the object should not be used further and can be safely destroyed.

2.4 Setting IP board configuration

Calling any of the SetClientConfig* or SetDNS* families of functions will cause the IP board on the device to be reset and your connection to the device will be lost. After calling any of these functions your Irisys::Blackfin will be unable to communicate with the device and you should disconnect it and destroy the object before, if necessary, creating a new connection as outlined above. Note that the related GetClientConfig* and GetDNS* families of functions are not affected by this.

It is recommended that these functions are only used when connected directly to the device, however they can also be used in client connection mode with the caveat that you

may have to wait for a TCP/IP timeout to occur before you can communicate with the device again, which may take up to 10 minutes.

3 Application notes

3.1 Using Irisys timezones

As of API version 2.0.30816.01, it is now possible to get and set the time zone in which a device resides - the same setting which is available in the People Counter Setup Tool and Validation Tool. This setting does not affect the Irisys::Blackfin::GetUnitTime() function which always returns a value in UTC time.

The time zone setting is restricted to a (large) range of time zones which correspond to Windows time zones ID's. The Irisys::IRISYS_TIMEZONES array enumerates the set of time zones available. Each Irisys::IrisysTimeZone object has a base UTC offset and a Boolean value indicating whether or not the time zone supports Daylight Savings Time (DST). When setting the time zone, the appropriate Irisys::IrisysTimeZone object must be passed along with a boolean indicating whether or not daylight savings is to be applied or not. Note the distinction between DST being supported, as indicated by Irisys::IrisysTimeZone.m_bDSTSupported and DST being applied, as indicated by the applyDST parameter as passed to the Irisys::Blackfin::SetUnitTimeZone() function.

If DST is to be taken into account when converting a UTC time to a local time then it is necessary to have more information than the API provides - the base UTC offset must be adjusted by a set of rules dependant on the particular time zone. These rules are non-trivial for many zones and are typically represented in a database. This information is available in the Windows registry, or by using the System.TimeZoneInfo class.

General procedure for calculating local time:

- Retrieve UTC time from device using Irisys::Blackfin::GetUnitTime()
- Retrieve time zone from device using Irisys::Blackfin::GetUnitTimeZone()
- If not applying daylight savings rules, add the time zone base UTC offset to the UTC time.
- If applying daylight savings rules:
  - Query database (e.g. the Windows Registry) for daylight savings rules
  - Apply rules to the UTC time to generate appropriate UTC offset for that point in time
  - Add UTC offset to the UTC time to get adjusted local time

3.1.1 Missing timezones

Some of the time zone IDs contained within the Irisys API do not exist on some versions of Microsoft Windows when the latest time zone updates have not been installed, therefore you are advised not to assume the registry keys containing the time zone information exist on a given target machine.

You can add the missing time zones on Microsoft Windows NT 5.0 (XP, Server 2000) and newer by installing the latest time zone update via Windows Update or by downloading the latest time zone update package from the Microsoft Support website.

3.2 Using device status for troubleshooting

The Irisys::Blackfin::GetDeviceStatus() method allows you to query the diagnostic log of the connected device. This can be used to determine if the device is in an error state or is issuing any warning messages, as well as informative messages such as the last reset time.

Note that IRC3xxx series devices will reset a device every minute in the event that a fatal error condition (such as array failure) has occurred. This ensures the device never becomes unresponsive, however it will cause any communications calls to be more likely to fail, especially those which take a long time to complete such as downloading the count log.

3.3 Setting packet timeout

The Irisys::Blackfin::SetPacketTimeout() method allows you to set the maximum return trip time for a data request from the remote device, which defaults to 5 seconds. Whilst TCP/IP guarantees ordered packet delivery up to a specified duration, this may be an unacceptably long period to find out that a packet has not been delivered. On a poor network- due to a faulty connection or hardware, packet congestion, or a bad IP stack packet timeouts may occur frequently, causing many or all communications calls made by this API to fail. The aforementioned function call allows you to raise the timeout period up to a maximum of 30 seconds if necessary.

One potential way to use this mechanism is to count the number of failed API calls and, upon reaching a set threshold, raise the timeout value to try and work around network issues. This has the side effect of causing any subsequent API calls take longer to return failure in the event that the connection has been lost completely.

4 Sample applications

This section provides a brief description of the sample applications included with the Linux version of the API. If you have received this document without the sample applications listed below please contact your supplier.

4.1 SampleConsoleApp

A simple console application which demonstrates the correct procedure for connecting directly to a device using a TCP/IP socket and allows the user to test the various functions available through the API.

This application is a useful way of exploring the functionality available through the API and performing simple tasks on a device and could easily be modified to work with batch or powershell scripts if desired, allowing the API to be used indirectly with almost any programming language.

The entry point for the application starts a while loop which will persist until the application is instructed to quit. This loop first creates a new instance of the Irisys::Blackfin class and then prompts the user to enter an IP address to which it should try to connect. Alternatively the command EXIT can be entered to break out of the while loop and quit the application.

Once an address to connect to has been entered it creates a socket and connects it to the provided address. If the socket was connected successfully it is associated with the previously created Irisys::Blackfin instance by passing both to the Irisys::BlackfinEngine::AddBlackfinEndPoint() function. This will return true if an IRC3xxx series device was found on the connection or false otherwise, in which case the Irisys::Blackfin object is cleaned up and execution returns to the start of the while loop.

Upon a successful connection the code enters another while loop to process commands upon the connected device from the user. The HELP command can be used to list all of the available commands and gives a short description of each, whilst the DISCONNECT command can be used to disconnect from the device and return to the outer while loop to connect to another device.

4.2 SampleServerApp

A simple console application which demonstrates how to listen for and accept client connections from IRC3xxx series devices and execute a set of API commands upon each device that connects.

The application entry point, in SampleServerApp.cpp creates a BlackfinServer object and starts it running before entering an infinite loop which calls a function on the server every 10 milliseconds to handle incoming connections.

Upon its creation the BlackfinServer object creates an instance of the Irisys::BlackfinEngine class, which is required to connect to IRC3xxx series devices using the API. It also starts a worker thread which will execute a series of API calls upon each connected device every 30 seconds.

For each connection that it recieves it creates an Irisys::Blackfin instance and attaches it to the socket by passing both to the Irisys::BlackfinEngine::AddBlackfinEndPoint() function of the previously created engine object. If successfully attached it adds it to a list of connected devices, which will be processed by the worker thread.

You can modify this sample application to disconnect from each device once it has executed the API calls upon it by creating a DISCONNECT_MODE preprocessor definition. If this is not defined connections will be held open indefinitely.

5 Count logs

5.1 Introduction

This section describes how Irisys IRC3xxx series devices handle count logs internally and how you can interact with them using the API. A count log is a record of the count values for each configured register at the time the log was written to the device's non-volatile memory.

5.2 Logging period

The frequency with which count logs are created is determined by the CountLogPeriod setting which specifies, in seconds, the interval between each count log. The interval is based on the number of seconds since the start of the day to provide predictable logging times, hence the default logging interval of 900 seconds (15 minutes) will create count logs at 0000, 0015, 0030, etc.

You can use the Irisys::Blackfin::GetCountLogPeriod() API function to read the current value of this setting from the device. The Irisys::Blackfin::SetCountLogPeriod() API function or the People Counter Setup Tool can be used to change the value of this setting. Any changes to this setting will take effect immediately without any further action required.

5.3 Accessing count logs

There are two functions in the API for retrieving count logs from a device, Irisys::Blackfin::GetCounts() and Irisys::Blackfin::GetLastNCounts(). The first of these allows you to retrieve all count logs which fall into a specified time period, whilst the second retrieves a specified number of the most recent log entries.

Each of these functions takes a std::vector<Irisys::BlackfinInterfaces::Count> reference parameter into which it will add the count log entries downloaded from the device. Each of these entries in turn contain a UTC timestamp representing the time the log entry was recorded and a std::vector<unsigned long> containing the 32 bit count value for each configured count register. The size of this vector depends on the number of registers configured at the time the log entry was recorded and may not be consistent between all of the log entries returned if the device configuration was changed during the log period retrieved.

5.4 Storage

Count logs are written to the device's non-volatile flash memory and therefore are not lost even if the power to the device is removed. The number of count log entries which can be stored on the device depends on the number of count registers which have been configured.

Due to the nature of the flash memory used on the device the only way to erase data is to erase the entire sector. Therefore the count logs are split between two sectors, with the oldest sector being erased when the newer sector is full. Assuming you do not reset the count logs at any point this means that the minimum number of stored count logs (after both flash sectors have been filled at least once) is half of the maximum theoretical capacity of the device and the number of counts available via the API will be any amount between the minimum and maximum.

The maximum and minimum storage capacities for a range of configured registers are given in the table below, along with the minimum number of days worth of logging this represents at 5, 15 and 60 minute logging intervals.

Registers	Minimum Capacity	Maximum Capacity	5 Minute Interval (Min)	15 Minute Interval (Min)	60 Minute Interval (Min)
2	4680	9361	16 Days	**48 Days**	195 Days
4	2978	5957	10 Days	31 Days	124 Days
8	1724	3448	5 Days	17 Days	71 Days
16	936	1872	3 Days	9 Days	39 Days
32	489	978	1 Day	5 Days	20 Days

The default logging period of 15 minutes with 2 enabled count registers will have a minimum capacity of 48 days worth of count log entries.

5.5 Resetting count logs

You can use the API to reset the count log on an IRC3xxx series device by calling the Irisys::Blackfin::ResetCountLogs() function, which will permanently erase all count log entries from the device. Note that flash memory has a finite lifespan and excessive use of this function could reduce the lifespan of your devices. Rather than resetting the count logs your application should keep track of the most recent count log entry retrieved from the device and use this as the starting point when requesting new data.

6 Frequently asked questions

- **What is the baud rate (speed) of an IP connection?**

  An Irisys IRC3xxx device can have a link speed of 10Mbps or 100Mbps, however the maximum data baud rate is limited to 1Mbps

- **What is the baud rate (speed) of a serial connection?**

  The baud rate of a serial connection to an Irisys IRC3xxx device is 115200bps

- **Which TCP/IP ports are used to communicate with an Irisys IRC3xxx device?**

  Port 4505 is used to establish direct IP connections to the device.

  Port 80 is used to connect to the web interface on the device and configure it using the built in People Counter Setup Tool (PCST) software.

  By default port 5000 is used by devices for outgoing connections when client connection mode is enabled, however this can be configured via the web interface.

- **What is the default IP address of a device?**

  The factory default IP address is 192.168.0.10, using a subnet mask of 255.255.255.0

- **What is the maximum value of a count line?**

  The count registers on the device are 32 bits, giving a maximum value of over 4.2 billion, which is reset to 0 each time the device is powered off and on again.

- **What is the lifespan of the flash memory storage on the device?**

  The flash memory used on the device has a rated minimum lifespan of 100,000 erase / write cycles, which equates to roughly 26,000 years of logging at a 15 minute interval. Resetting the count log files will cause additional erase / write cycles to occur and will reduce the useful life of the device, therefore you should avoid doing this unless absolutely necessary. For more information about the count log files on the device see the Count Logs section.

7 Module documentation

7.1 Irisys timezone definitions

Functions

- Irisys::IrisysTimeZone::IrisysTimeZone (std::string strBaseUTCOffset, bool bDSTSupported, std::string strDescription)

Variables

- const std::string Irisys::IrisysTimeZone::m_strBaseUTCOffset
- const bool Irisys::IrisysTimeZone::m_bDSTSupported
- const std::string Irisys::IrisysTimeZone::m_strDescription
- static const unsigned int Irisys::IRISYS_TIMEZONES_SIZE = 90
- static const IrisysTimeZone Irisys::IRISYS_TIMEZONES [ ]

7.1.1 Detailed description

7.1.2 Function documentation

Irisys::IrisysTimeZone::IrisysTimeZone (std::string *strBaseUTCOffset,* bool *bDSTSupported,* std::string *strDescription*)

Creates a new immutable object based on the parameters specified. Note that on a Irisys-TimeZone object with a m_strDescription value which is equivalent of a value from the IRISYS_TIMEZONES can be used to set the value in the device. We do not support custom time zones. For this use the Irisys::Blackfin::LocaleString() field.

7.1.3 Variable documentation

const std::string Irisys::IrisysTimeZone::m_strBaseUTCOffset

The base UTC offset for the time zone, before any daylight savings are take into account

const bool Irisys::IrisysTimeZone::m_bDSTSupported

Whether or not this timezone has daylight savings rules which could be applied if desired

const std::string Irisys::IrisysTimeZone::m_strDescription

The string descriptor of the timezone. This corresponds to a Windows time zone ID key in the registry.

const unsigned int Irisys::IRISYS_TIMEZONES_SIZE = 90 `[static]`

Total number of Irisys Time Zones currently defined

const IrisysTimeZone Irisys::IRISYS_TIMEZONES[ ] `[static]`

A list of all of the time zones which are supported for IRC3000 devices. The order and size of this array is unspecified (may change). However, the descriptor strings are guaranteed to remain constant. These should be used if serializing time zone values to a database.

8 Class documentation

8.1 Irisys::Blackfin class reference

A representation of a IRC3000 series counting network.

`#include <Blackfin.h>`

Inherits Irisys::IStandardDeviceErrorHandler.

Classes

- struct Count
- struct DeviceLogEntry

  A single device status log entry.
- struct DeviceStatus

Public Member Functions

- bool ResetCountLogs ()
- bool ResetCurrentCount ()
- bool ResetDeviceStatus ()

Getters

- bool GetClientConfigEnable (bool &resultValue)
- bool GetClientConfigHostname (std::string &resultValue)
- bool GetClientConfigIP (std::string &resultValue)
- bool GetClientConfigPort (unsigned short &resultValue)
- bool GetClientConfigTimeout (unsigned int &resultValue)
- bool GetCountLogPeriod (unsigned int &resultValue)
- bool GetCounts (time_t startTime, time_t endTime, std::vector< Count > &result-Value)
- bool GetCurrentCount (Count &resultValue)
- bool GetLastNCounts (unsigned int n, std::vector< Count > &resultValue)
- bool GetDeviceID (std::string &resultValue)
- bool GetDeviceName (std::string &resultValue)
- bool GetSiteID (std::string &resultValue)
- bool GetSiteName (std::string &resultValue)
- bool GetUserString (std::string &resultValue)
- bool GetRegisterLabels (std::vector< std::string > &labels)
- bool GetDeviceStatus (DeviceStatus &resultValue)
- bool GetLocaleString (std::string &resultValue)
- bool GetMACAddress (std::string &resultValue)
- bool GetSerialNumber (uint32_t &resultValue)
- bool GetUnitTime (time_t &resultValue)
- bool GetUpTime (i_uint64 *upTime)
- bool GetDHCPEnabled (bool &resultValue)
- bool GetIPAddress (std::string &resultValue)
- bool GetSubnetMask (std::string &resultValue)
- bool GetGateway (std::string &resultValue)
- bool GetIPFirmwareVersion (std::string &resultValue)
- bool GetMonitorFirmwareVersion (std::string &resultValue)
- bool GetDNS1 (std::string &resultValue)
- bool GetDNS2 (std::string &resultValue)
- bool GetDNS3 (std::string &resultValue)
- bool GetUnitTimeZone (int &tz_id, bool &useDST)
- bool GetNetworkChecksum (std::string &checksum)

Setters

- bool SetClientConfigEnable (bool clientConfigEnable)
- bool SetClientConfigHostname (const std::string &clientConfigHostname)
- bool SetClientConfigIP (const std::string &clientConfigIP)
- bool SetClientConfigPort (unsigned short clientConfigPort)
- bool SetClientConfigTimeout (unsigned int clientConfigTimeout)
- bool SetCountLogPeriod (unsigned int nCountLogPeriod)
- bool SetDeviceID (const std::string &deviceID)
- bool SetDeviceName (const std::string &deviceName)
- bool SetSiteID (const std::string &siteID)
- bool SetSiteName (const std::string &siteName)
- bool SetLocaleString (const std::string &localeString)
- bool SetUserString (const std::string userString)
- bool SetUnitTime (time_t timestamp)
- bool SetDNS1 (const std::string &dns1)
- bool SetDNS2 (const std::string &dns2)
- bool SetDNS3 (const std::string &dns3)
- bool SetUnitTimeZone (const int tz_id, const bool bEnableDST)
- bool SetPacketTimeout (unsigned int timeoutMS)

8.1.1 Detailed description

A representation of a IRC3000 series counting network.

This class represents an Irisys IRC3000 series counting unit (a master device) and optionally a set of node devices. It is used for all communication calls to the device including connection and disconnection.

8.1.2 Member function documentation

bool Irisys::Blackfin::GetClientConfigEnable (bool & *resultValue*)

Get the client connection state configured on the device, which must feature an IP board. This setting specifies whether the device will attempt to establish an outgoing connection to the IP address or hostname configured on the device.

Parameters

out	*resultValue*	The client connection enabled state retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetClientConfigHostname (std::string & *resultValue*)

Get the client connection hostname configured on the device, which must feature an IP board. This setting specifies the hostname the device will attempt to connect to when client connection mode is enabled.

Note

If both an IP address and hostname are specified for client connection mode the IP address is used

Parameters

out	*resultValue*	The client connection hostname retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetClientConfigIP (std::string & *resultValue*)

Get the client connection IP address configured on the device, which must feature an IP board. This setting specifies the IP address the device will attempt to connect to when client connection mode is enabled.

Note

If both an IP address and hostname are specified for client connection mode the IP address is used

Parameters

out	*resultValue*	The client connection IP address retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetClientConfigPort (unsigned short & *resultValue*)

Get the client connection port configured on the device, which must feature an IP board. This setting specifies the port the device will attempt to connect to when client connection mode is enabled.

Parameters

out	*resultValue*	The client connection port retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetClientConfigTimeout (unsigned int & *resultValue*)

Get the client connection timeout configured on the device, which must feature an IP board. This setting specifies the timeout, in seconds, between client connection attempts by the device when client connection mode is enabled.

Parameters

out	*resultValue*	The client connection timeout retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetCountLogPeriod (unsigned int & *resultValue*)

Get the count log period currently configured on the device, which specifies the interval, in seconds, between count log entries being written to the devices memory.

Parameters

out	*resultValue*	The configured count log period retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetCounts (time_t *startTime,* time_t *endTime,* std::vector< Count > & *resultValue*)

Gets the count log entries from the device which fall within the specified time range.

Note

This call may take longer than the configured packet timeout

Parameters

in	*start*	UTC time of the earliest count log entry to retrieve
in	*end*	UTC time of the latest count log entry to retrieve
out	*resultValue*	A vector in which the count log entries retrieved from the device will be stored. Any existing data in the vector will be erased before new entries are added.

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetCurrentCount (Count & *resultValue*)

Get the live count line values from the device. These values are not written to the count log and therefore may not be the same as the most recent entry returned by the GetCounts() function.

Parameters

out	*resultValue*	A count object containing the current count values for each line

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetLastNCounts (unsigned int *n,* std::vector< Count > & *resultValue*)

Get the latest N count log entries from the device. Upon successful completion the count log entries will be available via the Counts() accessor function.

Note

This call may take longer than the configured packet timeout
The number of count log entries retrieved may be less than N if there are less than N entries in the device count log

Parameters

in	*n*	Number of count log entries to retrieve
out	*resultValue*	A vector in which the count log entries retrieved from the device will be stored. Any existing data in the vector will be erased before new entries are added.

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetDeviceID (std::string & *resultValue*)

Get the device ID string from the device.
Parameters

out	*resultValue*	The device ID string retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetDeviceName (std::string & *resultValue*)

Get the device name string from the device.
Parameters

out	*resultValue*	The device name string retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetSiteID (std::string & *resultValue*)

Get the site ID string from the device.
Parameters

out	*resultValue*	The site ID string retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetSiteName (std::string & *resultValue*)

Get the site name string from the device.
Parameters

out	*resultValue*	The site name string retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetUserString (std::string & *resultValue*)

Get the user string from the device.
This command was introduced in the 2.0.x series of APIs.

Parameters

out	*resultValue*	The user string retrieved from the device

Returns

True on success, false if the devices firmware is too old or a timeout or other comms error occurred

bool Irisys::Blackfin::GetRegisterLabels (std::vector< std::string > & *labels*)

Retrieve the labels for the enabled count registers from the device. Upon successful completion the retrieved labels will be available via the RegisterLabels() accessor function

This command was introduced in the 3.0.x series of APIs. **Minimum DSP Firmware Version:** 454

Parameters

out	*labels*	Vector of labels for each of the enabled registers on the device. These labels will be in the same order as the count values returned by the Irisys::Blackfin::GetCounts(), Irisys::Blackfin::GetLastNCounts() and Irisys::Blackfin::GetCurrentCounts() functions, unless the number of enabled register definitions has been changed on the device since the time at which the count values were recorded by the device.

Returns

True on success, false if the devices firmware is too old or a comms failure occurs

bool Irisys::Blackfin::GetDeviceStatus (DeviceStatus & *resultValue*)

Get the device status logs since the last boot up or device log reset.

Note

This call may take longer than the configured packet timeout

Parameters

out	*resultValue*	The device status logs retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetLocaleString (std::string & *resultValue*)

Get the locale string from the device.

Note

This value is not linked to the unit timezone and has no functional impact on the device

Parameters

out	*resultValue*	The locale string retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetMACAddress (std::string & *resultValue*)

Get the MAC address of the device, which must feature an IP board.
Parameters

out	*resultValue*	The MAC address retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetSerialNumber (uint32_t & *resultValue*)

Get the unique serial number from the device.
Parameters

out	*resultValue*	The unique serial number retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetUnitTime (time_t & *resultValue*)

Get the current UTC time of the devices internal clock, which is not affected by the configured timezone.
Parameters

out	*resultValue*	The UTC time retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetUpTime (i_uint64 * *upTime*)

Get the elapsed time, in seconds, since the device was booted. Upon successful completion the uptime will be available via the UpTime() accessor function.
Parameters

out	*upTime*	The up time retrieved from the device, in seconds

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetDHCPEnabled (bool & *resultValue*)

Get the DHCP state of the device, which must feature an IP board. This setting determines if the device obtains an IP address from a DHCP server or uses a statically configured address.

Parameters

out	*resultValue*	The DHCP state retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetIPAddress (std::string & *resultValue*)

Get the statically configured IP address of the device, which must feature an IP board. This setting determines the IP address of a device when it is not using DHCP.

Parameters

out	*resultValue*	The statically configured IP address retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetSubnetMask (std::string & *resultValue*)

Get the statically configured subnet mask of the device, which must feature an IP board. This setting determines the subnet mask of a device when it is not using DHCP.

Parameters

out	*resultValue*	The statically configured subnet mask retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetGateway (std::string & *resultValue*)

Get the statically configured gateway of the device, which must feature an IP board. This setting determines the gateway of a device when it is not using DHCP.

Parameters

out	*resultValue*	The statically configured gateway retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetIPFirmwareVersion (std::string & *resultValue*)

Get the version of IP firmware installed on the device, which must feature an IP board.

Parameters

out	*resultValue*	The IP firmware version retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetMonitorFirmwareVersion (std::string & *resultValue*)

Get the version of firmware installed on the device.
Parameters

out	*resultValue*	The firmware version retrieved from the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::GetDNS1 (std::string & *resultValue*)

Get the primary DNS server for the device, which must feature an IP board.
Parameters

out	*resultValue*	The primary DNS server retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetDNS2 (std::string & *resultValue*)

Get the secondary DNS server for the device, which must feature an IP board.
Parameters

out	*resultValue*	The secondary DNS server retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetDNS3 (std::string & *resultValue*)

Get the tertiary DNS server for the device, which must feature an IP board.
Parameters

out	*resultValue*	The tertiary DNS server retrieved from the device

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::GetUnitTimeZone (int & *tz_id,* bool & *useDST*)

Get the timezone configured on the device. Note that this has no functional impact on the device itself, which will always use UTC time.

Note

The timezone can be configured using the People Counter Setup Tool

Parameters

out	*tz_id*	The index of the Irisys timezone configured on the device in the IRISYS_TIMEZONES[] array
out	*useDST*	The DST enabled state configured on the device

Returns

True on success, false if the devices firmware is too old or a timeout or other comms error occurred

bool Irisys::Blackfin::GetNetworkChecksum (std::string & *checksum*)

Generates a checksum of the current configuration settings of all devices on the CAN network of the connected device, which can be used to detect whether any configuration changes have taken place on the network between two subsequent calls to this function. Upon successful completion the network checksum will be available via the NetworkChecksum() accessor function.

Note

This checksum may also be affected by firmware upgrades to any device on the CAN network

This command was introduced in the 3.0.x series of APIs.

Parameters

out	*checksum*	The computed checksum of the settings for all devices on the network

Returns

True on success, false if a timeout or other comms error occurred

bool Irisys::Blackfin::ResetCountLogs ( )

Empty the count log stored on the device. This action cannot be undone and will not affect the live count line values.

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::ResetCurrentCount ( )

Reset the live count line values on the device. This action cannot be undone and does not affect existing count log entries stored on the device.

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::ResetDeviceStatus ( )

Clear all entries in the device status logs. This action cannot be undone.

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetClientConfigEnable (bool *clientConfigEnable*)

Enable or disable client connection mode on the device, which must feature an IP board. If this setting is enabled the device will regularly attempt to establish an outgoing client connection to the IP address or hostname specified in the client connection settings. Please refer to the notes on Setting IP Board Configuration before using this function with a TCP/IP connection.

Parameters

in	*clientConfig-Enable*	Client connection enabled state to set on the device

See also

SetClientConfigIP()
SetClientConfigHostname()
SetClientConfigPort()
SetClientConfigTimeout()

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::SetClientConfigHostname (const std::string & *clientConfigHostname*)

Set the client connection hostname on the device, which must feature an IP board. This setting specifies the hostname the device will attempt to connect to when client connection mode is enabled. Please refer to the notes on Setting IP Board Configuration before using this function with a TCP/IP connection.

Note

If a client connection IP address other than 0.0.0.0 is specified this setting has no effect

Parameters

in	*clientConfig-Hostname*	The client connection hostname to set on the device, whose length must not exceed the value specified by the MAX_HO-STNAME_STRING constant

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::SetClientConfigIP (const std::string & *clientConfigIP*)

Set the client connection IP address on the device, which must feature an IP board. This setting specifies the IP address the device will attempt to connect to when client connection mode is enabled. Please refer to the notes on Setting IP Board Configuration before using this function with a TCP/IP connection.

Note

Set this value to 0.0.0.0 if you want to specify a hostname to connect to instead

Parameters

in	*clientConfigIP*	The client connection IP address to set on the device, in dotted notation, whose length must not exceed the value specified by the MAX_IPADDRESS_STRING constant

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::SetClientConfigPort (unsigned short *clientConfigPort*)

Set the client connection port number on the device, which must feature an IP board. This setting specifies the port the device will attempt to connect to when client connection mode is enabled. Please refer to the notes on Setting IP Board Configuration before using this function with a TCP/IP connection.

Parameters

in	*clientConfigPort*	The client connection port number to set on the device, which must be a value between the MIN_CLIENT_PORT and MAX_CLIENT_PORT constants.

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::SetClientConfigTimeout (unsigned int *clientConfigTimeout*)

Set the client connection timeout on the device, which must feature an IP board. This setting specifies the timeout, in seconds, between client connection attempts by the device when client connection mode is enabled. Please refer to the notes on Setting IP Board Configuration before using this function with a TCP/IP connection.

Parameters

in	*clientConfigTimeout*	The client connection timeout to set on the device, in seconds, which must be a value between the MIN_CLIENT_TIMEOUT and MAX_CLIENT_TIMEOUT constants.

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::SetCountLogPeriod (unsigned int *nCountLogPeriod*)

Set the count log period on the device which specifies the interval, in seconds, between count log entries being written to the devices memory.

Parameters

in	*nCountLogPeriod*	The count log period to set on the device, in seconds, which must be a value between the MIN_COUNT_LOG_PERIOD and MAX_COUNT_LOG_PERIOD constants

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetDeviceID (const std::string & *deviceID*)

Set the device ID string on the device to the specified value
Parameters

in	*deviceID*	Device ID string to set on the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetDeviceName (const std::string & *deviceName*)

Set the device name string on the device to the specified value
Parameters

in	*deviceName*	Device name string to set on the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetSiteID (const std::string & *siteID*)

Set the site ID string on the device to the specified value
Parameters

in	*siteID*	Site ID string to set on the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetSiteName (const std::string & *siteName*)

Set the site name string on the device to the specified value
Parameters

in	*siteName*	Site name string to set on the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetLocaleString (const std::string & *localeString*)

Set the locale string on the device to the specified value
Parameters

in	*localeString*	Locale string to set on the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetUserString (const std::string *userString*)

Set the user string on the device to the specified value

Parameters

in	*userString*	User string to set on the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetUnitTime (time_t *timestamp*)

Sets the internal clock on the device to the specified time, which should be in UTC
Parameters

in	*unitTime*	The UTC time to set on the device

Returns

Returns true on success, false on failure - timeout or comms error

bool Irisys::Blackfin::SetDNS1 (const std::string & *dns1*)

Set the primary DNS server for the device to use when resolving hostnames, which must feature an IP board. Please refer to the notes on Setting IP Board Configuration before using this function with a TCP/IP connection.
Parameters

in	*dns1*	Primary DNS server IP address, in dotted notation, to set on this device, whose length must not exceed the# value specified by the MAX_IPADDRESS_STRING constant

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::SetDNS2 (const std::string & *dns2*)

Set the secondary DNS server for the device to use when resolving hostnames, which must feature an IP board. Please refer to the notes on Setting IP Board Configuration before using this function with a TCP/IP connection.
Parameters

in	*dns2*	Secondary DNS server IP address, in dotted notation, to set on this device, whose length must not exceed the# value specified by the MAX_IPADDRESS_STRING constant

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::SetDNS3 (const std::string & *dns3*)

Set the tertiary DNS server for the device to use when resolving hostnames, which must feature an IP board. Please refer to the notes on Setting IP Board Configuration before using this function with a TCP/IP connection.

Parameters

in	*dns3*	Tertiary DNS server IP address, in dotted notation, to set on this device, whose length must not exceed the# value specified by the MAX_IPADDRESS_STRING constant

Returns

True on success, false if the device does not have an IP board or a timeout or other comms error occurred

bool Irisys::Blackfin::SetUnitTimeZone (const int *tz_id,* const bool *bEnableDST*)

Set the time zone and DST state on the device. Note that this has no functional impact on the device itself, which will always use UTC time internally.

Note

The time zone can also be configured using the People Counter Setup Tool

Parameters

in	*itz*	Index of the Irisys time zone within the IRISYS_TIMEZONES[] array to set on the device
in	*bEnableDST*	DST state to set on the device, specifies whether a DST offset should be applied where appropriate

Returns

True on success, false if the devices firmware is too old or a timeout or other comms error occurred

bool Irisys::Blackfin::SetPacketTimeout (unsigned int *timeoutMS*)

Set the packet timeout value for the API to use when communicating with the connected device, which specifies how long it will wait for a response before returning a failure. Longer values can help reduce failures on high latency connections but will also cause Get∗ and Set∗ function calls to block for longer whilst they wait for a response when the connection has been lost.

Parameters

in	*timeoutMS*	Timeout value, in milliseconds, to use during communications with the device

See also

PacketTimeout()

Returns

True on success, false on failure

8.2 Irisys::BlackfinEngine class reference

This class connects Blackfin objects to SOCKET handles.

```
#include <BlackfinEngine.h>
```

Public Member Functions

- BlackfinEngine ()
- void StartEngine ()
- void ShutdownEngine ()
- bool AddBlackfinEndPoint (Blackfin *endPoint, SOCKET socketHandle, IBlackfinErrorHandler *errorHandler)
- void RemoveBlackfinEndPoint (Blackfin *endPoint)

Static Public Member Functions

- static std::string GetAPIVersion ()

8.2.1 Detailed description

This class connects Blackfin objects to SOCKET handles.

8.2.2 Constructor & destructor documentation

Irisys::BlackfinEngine::BlackfinEngine ( )

Constructs a new BlackfinEngine object

8.2.3 Member function documentation

static std::string Irisys::BlackfinEngine::GetAPIVersion ( ) [static]

Gets the version number of the executing API assembly

Returns

A string value representing a version number

void Irisys::BlackfinEngine::StartEngine ( )

Intializes the BlackfinEngine object. This method is required to be called before AddBlackfinEndPoint() is called

void Irisys::BlackfinEngine::ShutdownEngine ( )

Releases allocated resources and closes. This should not be called until all Blackfin objects have been destroyed. No further method calls should be invoked.

bool Irisys::BlackfinEngine::AddBlackfinEndPoint (Blackfin * *endPoint,* SOCKET *socketHandle,* IBlackfinErrorHandler * *errorHandler*)

Connects the specified blackfin object to the specified connected SOCKET handler. Errors in the connection will be reported to the specified error handler. Do not attempt to connect the same Blackfin object more than once.

Parameters

in	*endPoint*	The Blackfin object to connect.
in	*socketHandle*	An established socket connection descriptor
in	*errorHandler*	An object to callback in case of errors

void Irisys::BlackfinEngine::RemoveBlackfinEndPoint (Blackfin * *endPoint*)

Removes the Blackfin object from the engine and disconnects it. The Blackfin object will not be deleted, this is the responsibility of the caller. The SOCKET handle will be closed by this method.

Parameters

in	*endPoint*	The Blackfin object to disconnect

8.3 Irisys::Blackfin::Count struct reference

`#include <Blackfin.h>`

Public Attributes

- std::vector< i_uint32 > countLines
- time_t countTime

8.3.1 Detailed description

A structure containing a single count log entry downloaded from a device, or the live count values for the count lines when used with Blackfin::GetCurrentCounts().

8.3.2 Member data documentation

std::vector<i_uint32> Irisys::Blackfin::Count::countLines

A vector containing the count values for each count line configured on the device at the time this count log entry was created.

time_t Irisys::Blackfin::Count::countTime

The UTC device time when this count log entry was created

8.4 Irisys::Blackfin::DeviceLogEntry struct reference

A single device status log entry.

`#include <Blackfin.h>`

Public Attributes

- std::string errorDescription
- time_t timestamp

8.4.1 Detailed description

A single device status log entry.

This structure contains a string which describes a condition that the device was in at a specific time stamp.

8.4.2 Member data documentation

std::string Irisys::Blackfin::DeviceLogEntry::errorDescription

A string describing the status of the device

time_t Irisys::Blackfin::DeviceLogEntry::timestamp

The UTC device time when this device log entry was created

8.5 Irisys::Blackfin::DeviceStatus struct reference

`#include <Blackfin.h>`

Public Attributes

- std::vector< DeviceLogEntry > m_infoList
- std::vector< DeviceLogEntry > m_warnList
- std::vector< DeviceLogEntry > m_errorList

8.5.1 Detailed description

A structure containing all of the device status log entries downloaded from a device, which are grouped into one of three categories which are, in order of increasing severity: information, warnings and errors. Information entries provide general messages about the devices operation, such as reset events. Warning messages indicate a potential problem which may need to be addresses whilst error messages indicate a fault with the device or its configuration which must be corrected.

8.5.2 Member data documentation

std::vector<DeviceLogEntry> Irisys::Blackfin::DeviceStatus::m_infoList

Vector of device status log entries which fall into the information category

std::vector<DeviceLogEntry> Irisys::Blackfin::DeviceStatus::m_warnList

Vector of device status logs entries which fall into the warnings category

std::vector<DeviceLogEntry> Irisys::Blackfin::DeviceStatus::m_errorList

Vector of device status logs entries which fall into the errors category

8.6 Irisys::IrisysTimeZone struct reference

Immutable class representing one of a set of supported time zones.

`#include <IrisysTimeZone.h>`

Public Member Functions

- IrisysTimeZone (std::string strBaseUTCOffset, bool bDSTSupported, std::string strDescription)

Public Attributes

- const std::string m_strBaseUTCOffset
- const bool m_bDSTSupported
- const std::string m_strDescription

8.6.1 Detailed description

Immutable class representing one of a set of supported time zones.

InfraRed Integrated Systems Limited

Park Circle Tithe Barn Way
Swan Valley
Northampton NN4 9BG UK
Tel: **+44 (0) 1604 594 200**
Fax: **+44 (0) 1604 594 210**
Email: **support@irisys.co.uk**
sales@irisys.co.uk

Web site: **www.irisys.co.uk**

Irisys Americas

One Glenlake Parkway
Suite 700
Atlanta GA 30328 USA
Tel: **+1 678 638 6248**
Email: **support@irisys.net**
sales@irisys.net

Web site: **www.irisys.net**

March 2014
IPU 40304
Issue 3