Irisys Win32/C++API for IRC3xxx Series (Blackfin-based) Devices

API Version 3.0.40504.01

Contents

1 Irisys Win32/C++ 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.

1.2 Release notes

1.2.1 Version 3.0.40504.01 (May 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

- Added new Irisys::Blackfin::GetRegisterLabels() function and the associated accessor function Irisys::Blackfin::RegisterLabels() to download the new register labels
- Allows the setting of device ID strings up to 160 bytes in length, supported by DSP firmware version 475 and newer
- Added new Irisys::Blackfin.GetNetworkChecksum() function and the associated accessor function Irisys::Blackfin.NetworkChecksum() to create a checksum of the settings for all devices on the connected CAN network

1.2.2 Version 2.0.30801.1 (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

- Moved global functions, structures and the Irisys::Blackfin class into the Irisys namespace
- Added support for Irisys time zones
- Added support for getting & setting a new 'user string' setting
- Renamed global function StartupAPI() to Irisys::StartupBlackfinAPI()
- Renamed global function ShutdownAPI() to Irisys::ShutdownBlackfinAPI()
- Renamed global function GetAPIVersion() to Irisys::GetBlackfinAPIVersion()
- Renamed Blackfin::GetTime() to Irisys::Blackfin::GetUnitTime() for consistency
- Renamed Blackfin::SetTime() to Irisys::Blackfin::SetUnitTime() for consistency
- Renamed Blackfin::ClientConfigPortNumber() to Irisys::Blackfin::ClientConfigPort() for consistency
- Renamed Blackfin::GetClientConfigPortNumber() to Irisys::Blackfin::GetClientConfigPort() for consistency
- Renamed Blackfin::SetClientConfigPortNumber() to Irisys::Blackfin::SetClientConfigPort() for consistency
- Added BlackfinAPIServer example application
- Added UnitTimeToLocalTime example application to demonstrate use of Irisys time zones
- No longer need to specify port number in connection string (uses 4505 by default)
- Added Irisys::ConvertBlackfinSerialNumber utility function to convert from numeric to alphanumeric serial number representations

1.3 Included files

- BlackfinAPI.dll
- BlackfinAPI.lib
- BlackfinAPI.h
- BlackfinAPIOMF.lib
- BlackfinSystem.h
- IrisysTimeZone.h

1.4 Dependencies

- Microsoft Visual C++ Runtime 10.0 (msvcr100.dll)
- Microsoft Foundation Classes Runtime 10.0 (mfc100.dll)

To build the example projects you will need Visual Studio 2008 or 2010. The freely available Visual Studio Express editions will be able to compile and run the Console and Server applications, but the BlackfinDemo application uses MFC for the GUI which the Express editions do not support.

2 Basic usage

2.1 Using the API

This section provides an introduction to the basic usage of the Irisys API (Win32/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.

- Include the file BlackfinAPI.lib in your compiler library search path
- Instruct your linker to link to the BlackfinAPI.lib library.
- Include BlackfinAPI.h in files which need to use the Blackfin API.
- Make sure that BlackfinAPI.dll is avaliable in the DLL search path for your compiled executable

2.2 Initializing the API

Your application must call the Irisys::StartupBlackfinAPI() global function before attempting to use any functions or classes provided with this API. The one exception to this rule is the global function Irisys::GetBlackfinAPIVersion(), which can safely be called at any time.

Upon shutdown your application should call the Irisys::ShutdownBlackfinAPI() global function to cleanly release any resources currently held by the Blackfin API. After calling this function you must not use any other functions or classes provided with this API.

2.3 Connecting to a device

The Irisys::Blackfin class represents an Irisys IRC3xxx series master counting device. To connect to a device your application should create an instance of this class using the default constructor and then call one of the two overloaded Irisys::Blackfin::Connect() functions to provide a port address for the API to create a connection to (either an IP address or COM port name) or a pre-connected SOCKET handle to be used by the API.

2.4 Working with device configuration

Once an Irisys::Blackfin object is connected to a device there are many API functions that can be used to interact with the device. Many of these functions relate to getting and setting device configuration items such as the device id and device name. The usage of these functions are all broadly similar to the example below demonstrating how to get and set the site name configuration item on a connected device.

```
*     // In this code bfin is a connected instance of the \bfclass class
*
*     // Getting the site name from the device
*     if ( bfin.GetSiteName() )
*       printf ( "Device site name is %s\n", bfin.SiteName().c_str() );
*     else
*       printf ( "Failed to get site name\n" );
*
*     // Setting the site name on the device
*     if ( bfin.SetSiteName("My Site 1") )
*       printf ( "Successfully set new site name on the device\n" );
*     else
*       printf ( "Failed to set site name\n" );
*
```

The Irisys::Blackfin::GetSiteName() function initiates a blocking communications call to the connected device to query its configured site name string, returning true upon success or false if a timeout or other failure occurs. If successful the site name string configured on the device is available via the Irisys::Blackfin::SiteName() accessor function.

Similarly the Irisys::Blackfin::SetSiteName() initiates a blocking communications call to the connected device to write the new site name string to its configuration, returning true upon success or false after a timeout or any other failure.

2.5 Setting IP board configuration

Whilst it is possible to call the SetClientConfig* and SetDNS* families of functions whilst connected to a device by TCP/IP it is **NOT** recommended as this will cause the IP board to be reset and drop the connection. If you provided an IP address to the API and allowed it to create the connection then it will attempt to re-establish the connection for you, otherwise you must discard the Irisys::Blackfin object and create a new connection yourself. If a connection to a device connected in client connection mode is dropped in this way you may be unable to connect to that device again until the TCP/IP timeout occurs, which may take up to 10 minutes.

It is recommended that you connect to a device via serial if you wish to call any of these functions. Note that the GetClientConfig* and GetDNS* families families of functions are not affected by this.

2.6 System functions

Note that the system functions and structures defined in the BlackfinSystem.h header file are not intended for direct use and are subject to change without notice. Their functionality is encapsulated by the Irisys::Blackfin class which is defined in the BlackfinAPI.h header file and fully documented for your reference.

3 Application notes

3.1 Using Irisys timezones

As of API version 2.0.30801.1, 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::BlackfinInterfaces::IRISYS_TIMEZONES array enumerates the set of time zones available. Each Irisys::BlackfinInterfaces::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::BlackfinInterfaces::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::BlackfinInterfaces::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() function allows you to query the diagnostic log of the connected device, the results of which will be available via the Irisys::Blackfin::DeviceStatus() accessor function. 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 Packet timeouts

Communications calls made by an Irisys::Blackfin instance are marked as having failed if no response is received within the configured timeout period, which defaults to 5 seconds. You can query the current timeout period using the Irisys::Blackfin::PacketTimeout() accessor function. For the majority of API functions this timeout is the maximum time the function will block for before returning a result, with the exceptions to this rule being;

- Irisys::Blackfin::GetCounts()
- Irisys::Blackfin::GetLastNCounts()
- Irisys::Blackfin::GetDeviceStatus()

If you are connecting to a device on a high latency network it may be useful to increase the timeout period using the Irisys::Blackfin::SetPacketTimeout() function to allow the API more time to wait for responses from that device, however this has the side effect of increasing the delay before a blocking API function will return a failure when the connection has been lost. One potential strategy 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 latency issues.

4 Sample applications

This section provides a brief description of the sample applications included with the Win32 version of the API. If you have received this document without the sample applications listed below please contact your supplier.

4.1 BlackfinAPIConsole

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 or COM port number 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 is passed to the Irisys::Blackfin::Connect function of the previously created Irisys::Blackfin instance, which will return a positive number upon success or a negative numerical error code upon failure, in which case the 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 BlackfinAPIServer

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 Application.cpp initializes the Blackfin API and creates a new thread, which in turn creates an instance of the BlackfinServer class and executes its BlackfinServer::StartServer() function. The main thread then waits for the user to type the letter q (for quit), at which point it instructs the server created by the child thread to stop and shuts down the application.

Until it gets shut down the child thread executes a while loop within the BlackfinServer::StartServer() function, which listens for and accepts incoming connections on port 5000, the default port number for client connection mode on IRC3xxx series devices.

Upon receiving a connection it spawns a new thread to handle that connection, which begins execution in the Connect() located within the BlackfinServer.cpp file. This function creates an instance of the Irisys::Blackfin class and connects it with socket accepted by the server thread. If connected successfully it then calls the BlackfinServer::BlackfinConnected() function which executes a series of API calls upon the device and logs the number of connections upon which it successfully executed all of the API calls it attempted. After this has completed the Irisys::Blackfin object goes out of scope and is destroyed and the socket is closed.

4.3 BlackfinAPITest

An MFC based application which allows a user to connect to an IRC3xxx series device via IP or serial and execute a variety of API commands upon it.

Like the BlackfinAPIConsole sample application this provides another way of exploring the functionality available via the API with the added ease of a GUI in place of the command line interface. This application can easily be used to configure some basic settings on a device if desired.

To connect to a device type its IP address or COM# serial port address into the text box next to the Connect button in the Connection section of the GUI and click Connect. If successful the red X icon is replaced with a green tick, otherwise a dialog box will be displayed with an error code. To connect to another device click on the disconnect button, enter the address of the new device and click Connect again.

Once connected the other buttons in the application can be used to get and set all of the configuration items which are available via the API and perform other functions available via the API, such as downloading count log entries from the device. The success or failure or each function call is indicated by a green tick or red cross next to the button and, if getting information, the GUI will be updated with the information retrieved.

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 and the associated accessor function Irisys::Blackfin::CountLogPeriod() 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.

Regardless of which function you call, the retrieved count log entries are available via the Irisys::Blackfin::Counts() accessor function, which returns a std::vector containing an Irisys::Blackfin::Interfaces::Count instance for each count log entry downloaded from the device. This struct consists of a UTC timestamp representing the time the log entry was recorded and a std::vector of unsigned long (32 bits) values containing the count value of each enabled register enabled at the time the log entry was recorded. The number of count values in the vector may not be consistent between all of the count logs downloaded if the device configuration was modified during the log period retrived.

Note that subsequent calls to any of the Irisys::Blackfin::GetCounts(), Irisys::Blackfin::GetLastNCounts() or Irisys::Blackfin::GetCurrentCount() functions will overwrite the downloaded count data with the result of that function call.

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

- static bool Irisys::BlackfinInterfaces::GetTimeZoneIDFromDescription (std::string description, unsigned int &index)

Variables

- static const unsigned int Irisys::BlackfinInterfaces::IRISYS_TIMEZONES_SIZE = 90
- static const IrisysTimeZone Irisys::BlackfinInterfaces::IRISYS_TIMEZONES [ ]

7.1.1 Detailed description

7.1.2 Function documentation

static bool Irisys::BlackfinInterfaces::GetTimeZoneIDFromDescription (std::string *description,* unsigned int & *index*) `[static]`

Get the ID of an Irisys time zone given its description
Parameters

in	*description*	Description of the Irisys time zone to find
out	*index*	The ID of the Irisys time zone, if found

Returns

True if the time zone is found, false otherwise

7.1.3 Variable documentation

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

Total number of Irisys Time Zones currently defined

const IrisysTimeZone Irisys::BlackfinInterfaces::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 Namespace documentation

8.1 Irisys namespace reference

All API functionality is provided in the Irisys namespace.

Namespaces

- BlackfinInterfaces

Classes

- class Blackfin

  A representation of a IRC3xxx series counting network.

Functions

- bool StartupBlackfinAPI ()
- bool ShutdownBlackfinAPI ()
- void GetBlackfinAPIVersion (char *buf, int bufSize)
- void ConvertBlackfinSerialNumber (unsigned long serialNumber, char *buf, int bufSize)

Variables

- static const int MAX_VERSION_STRING = 120

  max string length for version strings (including a null terminating character)
- static const int MAX_IPADDRESS_STRING = 20

  max string length for IP related strings (including a null terminating character)
- static const int MAX_HOSTNAME_STRING = 120

  max string length for SetClientConfigHostname (including a null terminating character)
- static const int MAX_PROPERTY_STRING = 64

  max string length for flash property set functions (including a null terminating character)
- static const int MAX_DEVICEID_STRING = 160

  Maximum string length for the Device ID field (including a null terminator)
- static const int MIN_COUNT_LOG_PERIOD = 60

  min count log period (seconds) allowed by SetCountLogPeriod function
- static const int MAX_COUNT_LOG_PERIOD = 3600

  max count log period (seconds) allowed by SetCountLogPeriod function
- static const int MIN_CLIENT_PORT = 1

  min allowed value for SetClientConfigPort
- static const int MAX_CLIENT_PORT = 10000

  max allowed value for SetClientConfigPort
- static const int MIN_CLIENT_TIMEOUT = 1

  min allowed value for SetClientConfigTimeout
- static const int MAX_CLIENT_TIMEOUT = 172800

  max allowed value for SetClientConfigTimeout (2 days in seconds)
- static const short MAX_REGISTERS = 32

  Maximum number of registers that can be defined on a device.
- static const short MAX_REGISTER_LABEL = 24

  Maximum length of a register label, including the null terminator.

8.1.1 Function documentation

bool Irisys::StartupBlackfinAPI ( )

Initializes the API for first use. No Irisys::Blackfin objects should be created before this method is called

bool Irisys::ShutdownBlackfinAPI ( )

Releases all resources acquired by the API during the lifetime of the process All Irisys::-Blackfin objects should be destroyed before calling this function

void Irisys::GetBlackfinAPIVersion (char * *buf,* int *bufSize*)

Writes the version number of the API into the provided character array, which will only be NULL terminated if the length of the version string is less than the size of the buffer. The maximum size of the version string, including a terminating NULL character, is specified by the MAX_VERSION_STRING constant.

Parameters

out	*buf*	A character array to write the version string into
in	*bufSize*	The size of the character array

void Irisys::ConvertBlackfinSerialNumber (unsigned long *serialNumber,* char * *buf,* int *bufSize*)

Converts a numerical serial number obtained from a device into the corresponding alphanumeric serial number as used in Irisys applications and printed on IRC3xxx series devices.

This command was introduced in the 2.0.x series of APIs.

Parameters

in	*serialNumber*	A numerical serial number obtained from an Irisys::Blackfin instance
out	*buf*	A character array to write the alphanumeric serial number string into
in	*bufSize*	The size of the character array

See also

Blackfin::SerialNumber()

8.1.2 Variable documentation

const int Irisys::MAX_VERSION_STRING = 120 [static]

const int Irisys::MAX_IPADDRESS_STRING = 20 [static]

const int Irisys::MAX_HOSTNAME_STRING = 120 [static]

const int Irisys::MAX_PROPERTY_STRING = 64 [static]

const int Irisys::MAX_DEVICEID_STRING = 160 [static]

const int Irisys::MIN_COUNT_LOG_PERIOD = 60 [static]

const int Irisys::MAX_COUNT_LOG_PERIOD = 3600 [static]

const int Irisys::MIN_CLIENT_PORT = 1 [static]

const int Irisys::MAX_CLIENT_PORT = 10000 [static]

const int Irisys::MIN_CLIENT_TIMEOUT = 1 [static]

const int Irisys::MAX_CLIENT_TIMEOUT = 172800 [static]

const short Irisys::MAX_REGISTERS = 32 [static]

const short Irisys::MAX_REGISTER_LABEL = 24 [static]

8.2 Irisys::BlackfinInterfaces namespace reference

Classes

- struct Count
- struct DeviceLogEntry
- struct DeviceStatus
- struct IrisysTimeZone

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

Functions

- static bool GetTimeZoneIDFromDescription (std::string description, unsigned int &index)

Variables

- static const unsigned int IRISYS_TIMEZONES_SIZE = 90
- static const IrisysTimeZone IRISYS_TIMEZONES [ ]

8.2.1 Detailed description

This namespace contains structures to represent Blackfin device data structures

9 Class documentation

9.1 Irisys::Blackfin class reference

A representation of a IRC3xxx series counting network.

`#include <BlackfinAPI.h>`

Public Member Functions

- Blackfin ()
- ~Blackfin ()
- int Connect (std::string strPortAddress)
- int Connect (SOCKET socketHandle)
- bool ResetCountLogs ()
- bool ResetCurrentCount ()
- bool ResetDeviceStatus ()

Accessors

- int CommsID ()
- BlackfinInterfaces::DeviceStatus DeviceStatus ()
- const std::vector < BlackfinInterfaces::Count > Counts ()
- std::string DeviceID ()
- std::string DeviceName ()
- std::string SiteID ()
- std::string SiteName ()
- std::string LocaleString ()
- std::string UserString ()
- std::vector< std::string > RegisterLabels ()
- time_t UnitTime ()
- std::string MACAddress ()
- unsigned long SerialNumber ()
- unsigned short CountLogPeriod ()
- std::string ClientConfigIP ()
- std::string ClientConfigHostname ()
- unsigned short ClientConfigPort ()
- unsigned int ClientConfigTimeout ()
- bool ClientConfigEnable ()
- bool DHCPEnabled ()
- std::string IPAddress ()
- std::string SubnetMask ()
- std::string Gateway ()
- std::string IPFirmwareVersion ()
- std::string MonitorFirmwareVersion ()
- std::string DNS1 ()
- std::string DNS2 ()
- std::string DNS3 ()
- unsigned int PacketTimeout ()
- unsigned long long UpTime ()
- BlackfinInterfaces::IrisysTimeZone UnitTimeZone ()
- bool ApplyDST ()
- std::string NetworkChecksum ()

Getters

- bool GetCounts (time_t start, time_t end)
- bool GetCurrentCount ()
- bool GetLastNCounts (unsigned long n)
- bool GetDeviceID ()
- bool GetDeviceName ()
- bool GetSiteID ()
- bool GetSiteName ()
- bool GetUserString ()
- bool GetRegisterLabels ()
- bool GetDeviceStatus ()
- bool GetLocaleString ()
- bool GetUnitTime ()
- bool GetMACAddress ()
- bool GetSerialNumber ()
- bool GetCountLogPeriod ()
- bool GetDHCPEnabled ()
- bool GetIPAddress ()
- bool GetSubnetMask ()
- bool GetGateway ()
- bool GetIPFirmwareVersion ()
- bool GetMonitorFirmwareVersion ()
- bool GetDNS1 ()
- bool GetDNS2 ()
- bool GetDNS3 ()
- bool GetClientConfigIP ()
- bool GetClientConfigHostname ()
- bool GetClientConfigPort ()
- bool GetClientConfigTimeout ()
- bool GetClientConfigEnable ()
- bool GetUpTime ()
- bool GetUnitTimeZone ()
- bool GetNetworkChecksum ()

Setters

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

9.1.1 Detailed description

This class represents an Irisys IRC3xxx 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.

9.1.2 Constructor & destructor documentation

Irisys::Blackfin::Blackfin ( )

Create a new Blackfin instance ready to be connected to a device by calling either of the two connect functions. Your application must call StartupIrisysBlackfinAPI() before creating any Blackfin objects.

See also

Connect(std::string strPortAddress)
Connect(SOCKET socketHandle)

Irisys::Blackfin::~Blackfin ( )

Disconnects from any active connection and cleans up any allocated resources

9.1.3 Member function documentation

int Irisys::Blackfin::Connect (std::string *strPortAddress*)

Attempts to connect to a device at the specified IP address or COM port number. If a successful connection is made you can begin interacting with the device using any of the functions available in this class.

Parameters

in	*strPort-Address*	IP address or COM port number to create a connection on. IP addresses should be specified using the dotted notation (123.123.123.123) and may optionally be suffixed with a port number (:4505). Serial COM ports should be provided in the format COM#, for example COM1, COM5, COM14.

Returns

Upon success, a positive integer representing the Comms ID of the connected device. Upon failure, a negative integer representing one of the following error codes;

Code	Meaning
-1	Port in use
-2	Failed port test
-3	Invalid port address
-4	Failed to capture port
-5	Failed device discovery
-6	Invalid comms ID
-7	Connection aborted

int Irisys::Blackfin::Connect (SOCKET *socketHandle*)

Connect this Blackfin object to the specified Windows socket handle, which should be already be connected to a device. This should be used in client connection mode.

Parameters

in	*socketHandle*	An open Windows socket handle connected to a device

Returns

Upon success, a positive integer representing the Comms ID of the connected device. Upon failure, a negative integer representing one of the following error codes;

Code	Meaning
-1	Port in use
-2	Failed port test
-3	Invalid port address
-4	Failed to capture port
-5	Failed device discovery
-6	Invalid comms ID
-7	Connection aborted

int Irisys::Blackfin::CommsID ( )

Get the Comms ID of the device this object is connected to, which will be an integer value between 1 and 120. Each device on a CAN network must have a unique CAN ID to allow them to function correctly.

The Comms ID of a device can be configured using the People Counter Setup Tool. The values 121 - 126 are reserved and the value 127 is used to indicate an unconfigured unit.

Returns

Comms ID of the connected device

BlackfinInterfaces::DeviceStatus Irisys::Blackfin::DeviceStatus ( )

Returns

Device status logs retrieved from the connected device by the last successful call to GetDeviceStatus()

See also

GetDeviceStatus()

const std::vector<BlackfinInterfaces::Count> Irisys::Blackfin::Counts ( )

Returns

The count log entries downloaded from the device during the last successful call to either GetCounts() or GetLastNCounts() or live count values retrieved by the last successful call to GetCurrentCount().

See also

GetCounts()
GetLastNCounts()
GetCurrentCount()

std::string Irisys::Blackfin::DeviceID ( )

Returns

Device ID string retrieved from the device by the last successful call to GetDeviceID()

See also

GetDeviceID()

std::string Irisys::Blackfin::DeviceName ( )

Returns

Device name string retrieved from the device by the last successful call to GetDeviceName()

See also

GetDeviceName()

std::string Irisys::Blackfin::SiteID ( )

Returns

Site ID string retrieved from the device by the last successful call to GetSiteID()

See also

GetSiteID()

std::string Irisys::Blackfin::SiteName ( )

Returns

Site name string retrieved from the device by the last successful call to GetSiteName()

See also

GetSiteName()

std::string Irisys::Blackfin::LocaleString ( )

Returns

Locale string retrieved from the device by the last successful call to GetLocaleString()

See also

GetLocaleString()

std::string Irisys::Blackfin::UserString ( )

Returns

User string retrieved from the device by the last successful call to GetUserString()

This command was introduced in the 2.0.x series of APIs.

See also

GetUserString()

std::vector<std::string> Irisys::Blackfin::RegisterLabels ( )

Returns

Vector of labels for each of the enabled registers on the device, as retrieved by the last successful call to GetRegisterLabels(). These labels will be in the same order as the count values returned by the Counts() function, 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.

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

See also

GetRegisterLabels()

time_t Irisys::Blackfin::UnitTime ( )

Returns

The UTC time of the devices internal clock when the last successful call to GetUnitTime() was made.

Note

This time will always be in UTC regardless of any time zone which may be configured on the device.

See also

GetUnitTime()

std::string Irisys::Blackfin::MACAddress ( )

Returns

MAC address retrieved from the device by the last successful call to GetMACAddress()

Note

This is only applicable to devices which feature an IP board

See also

GetMACAddress()

unsigned long Irisys::Blackfin::SerialNumber ( )

Returns

Unique device serial number retrieved by the last successful call to GetSerialNumber()

Note

The value returned from this string can be converted into the alphanumeric format printed on the device and used within Irisys applications using the provided converter function.

See also

GetSerialNumber()
ConvertBlackfinSerialNumber()

unsigned short Irisys::Blackfin::CountLogPeriod ( )

Returns

Count log period retrieved from the device by the last successful call to GetCountLogPeriod()

See also

GetCountLogPeriod()

std::string Irisys::Blackfin::ClientConfigIP ( )

Returns

Client connection IP address retrieved from the device by the last successful call to GetClientConfigIP()

Note

This is only applicable to devices which feature an IP board

See also

GetClientConfigIP()

std::string Irisys::Blackfin::ClientConfigHostname ( )

Returns

Client connection hostname retrieved from the device by the last successful call to GetClientConfigHostname()

Note

This is only applicable to devices which feature an IP board

See also

GetClientConfigHostname()

unsigned short Irisys::Blackfin::ClientConfigPort ( )

Returns

Client connection port retrieved from the device by the last successful call to GetClientConfigPort()

Note

This is only applicable to devices which feature an IP board

See also

GetClientConfigPort()

unsigned int Irisys::Blackfin::ClientConfigTimeout ( )

Returns

Client connection timeout retrieved from the device by the last successful call to GetClientConfigTimeout()

Note

This is only applicable to devices which feature an IP board

See also

GetClientConfigTimeout()

bool Irisys::Blackfin::ClientConfigEnable ( )

Returns

Client connection enabled state retrieved from the device by the last successful call to GetClientConfigEnable()

Note

This is only applicable to devices which feature an IP board

See also

GetClientConfigEnable()

bool Irisys::Blackfin::DHCPEnabled ( )

Returns

DHCP enabled state retrieved from the device by the last successful call to GetDHCPEnabled()

Note

This is only applicable to devices which feature an IP board

See also

GetDHCPEnabled()

std::string Irisys::Blackfin::IPAddress ( )

Returns

Statically configured IP address retrieved from the device by the last successful call to GetIPAddress()

Note

This is only applicable to devices which feature an IP board

See also

GetIPAddress()

std::string Irisys::Blackfin::SubnetMask ( )

Returns

Statically configured subnet mask retrieved from the device by the last successful call to GetSubnetMask()

Note

This is only applicable to devices which feature an IP board

See also

GetSubnetMask()

std::string Irisys::Blackfin::Gateway ( )

Returns

Statically configured gateway retrieved from the device by the last successful call to GetGateway()

Note

This is only applicable to devices which feature an IP board

See also

GetGateway()

std::string Irisys::Blackfin::IPFirmwareVersion ( )

Returns

IP firmware version retrieved from the device by the last successful call to GetIP-FirmwareVersion()

Note

This is only applicable to devices which feature an IP board

See also

GetIPFirmwareVersion()

std::string Irisys::Blackfin::MonitorFirmwareVersion ( )

Returns

DSP firmware version retrieved from the device by the last successful call to Get-MonitorFirmwareVersion()

See also

GetMonitorFirmwareVersion()

std::string Irisys::Blackfin::DNS1 ( )

Returns

Primary DNS server retrieved from the device by the last successful call to GetDNS1()

Note

This is only applicable to devices which feature an IP board

See also

GetDNS1()

std::string Irisys::Blackfin::DNS2 ( )

Returns

Secondary DNS server retrieved from the device by the last successful call to GetDNS2()

Note

This is only applicable to devices which feature an IP board

See also

GetDNS2()

std::string Irisys::Blackfin::DNS3 ( )

Returns

Tertiary DNS server retrieved from the device by the last successful call to GetDNS3()

Note

This is only applicable to devices which feature an IP board

See also

GetDNS3()

unsigned int Irisys::Blackfin::PacketTimeout ( )

Get the packet timeout value currently used by the API when communicating with the connected device, which specifies how long it will wait for a response before returning a failure. This can be configured using the SetPacketTimeout() function.

Returns

Currently configured packet timeout in milliseconds

See also

SetPacketTimeout()

unsigned long long Irisys::Blackfin::UpTime ( )

Returns

Up time, in sconds, of the connected device at the time of the last successful call to GetUpTime()

See also

GetUpTime()

BlackfinInterfaces::IrisysTimeZone Irisys::Blackfin::UnitTimeZone ( )

Returns

Time zone configured on the connected device at the time of the last successful call to GetUnitTimeZone()

This command was introduced in the 2.0.x series of APIs.

See also

GetUnitTimeZone()

bool Irisys::Blackfin::ApplyDST ( )

Returns

True if daylight savings was enabled on the connected device at the time of the last successful call to GetUnitTimeZone()

This command was introduced in the 2.0.x series of APIs.

See also

GetUnitTimeZone()

std::string Irisys::Blackfin::NetworkChecksum ( )

Returns

The network settings checksum generated by the last successful call to GetNetwork-Checksum()

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

See also

GetNetworkChecksum()

bool Irisys::Blackfin::GetCounts (time_t *start,* time_t *end*)

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

See also

Counts()

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

Returns

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

bool Irisys::Blackfin::GetCurrentCount ( )

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. Upon successful completion the count line values will be available via the Counts() accessor function.

See also

Counts()

Returns

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

bool Irisys::Blackfin::GetLastNCounts (unsigned long *n*)

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

See also

Counts()

Parameters

in	*n*	Number of count log entries to retrieve

Returns

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

bool Irisys::Blackfin::GetDeviceID ( )

Get the device ID string from the device. Upon successful completion the device ID will be available via the DeviceID() accessor function.

See also

DeviceID()

Returns

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

bool Irisys::Blackfin::GetDeviceName ( )

Get the device name string from the device. Upon successful completion the device name will be available via the DeviceName() accessor function.

See also

DeviceName()

Returns

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

bool Irisys::Blackfin::GetSiteID ( )

Get the site ID string from the device. Upon successful completion the site ID name will be available via the SiteID() accessor function.

See also

SiteID()

Returns

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

bool Irisys::Blackfin::GetSiteName ( )

Get the site name string from the device. Upon successful completion the site name will be available via the SiteName() accessor function.

See also

SiteName()

Returns

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

bool Irisys::Blackfin::GetUserString ( )

Get the user string from the device. Upon successful completion the user string will be available via the UserString() accessor function.
This command was introduced in the 2.0.x series of APIs.

See also

UserString()

Returns

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

bool Irisys::Blackfin::GetRegisterLabels ( )

Retrieve the labels for the enabled 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

See also

RegisterLabels()

Returns

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

bool Irisys::Blackfin::GetDeviceStatus ( )

Get the device status logs since the last boot up or device log reset. Upon successful completion the device status logs will be available via the DeviceStatus() accessor function.

Note

This call may take longer than the configured packet timeout

See also

DeviceStatus()

Returns

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

bool Irisys::Blackfin::GetLocaleString ( )

Get the locale string from the device. Upon successful completion the locale string will be available via the LocaleString() accessor function.

Note

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

See also

LocaleString()

Returns

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

bool Irisys::Blackfin::GetUnitTime ( )

Get the current UTC time of the devices internal clock, which is not affected by the configured timezone. Upon successful completion the devices UTC time will be available via the UnitTime() accessor function.

See also

UnitTime()

Returns

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

bool Irisys::Blackfin::GetMACAddress ( )

Get the MAC address of the device, which must feature an IP board. Upon successful completion the devices MAC address will be available via the MACAddress() accessor function.

See also

MACAddress()

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 ( )

Get the unique serial number from the device. Upon successful completion the serial number will be available via the SerialNumber() accessor function.

See also

SerialNumber()

Returns

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

bool Irisys::Blackfin::GetCountLogPeriod ( )

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. Upon successful completion the configured count log period will be available via the CountLogPeriod() accessor function.

See also

CountLogPeriod()

Returns

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

bool Irisys::Blackfin::GetDHCPEnabled ( )

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. Upon successful completion the DHCP state of the device will be available via the DHCPEnabled() accessor function.

See also

DHCPEnabled()

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 ( )

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. Upon successful completion the IP address of the device will be available via the IPAddress() accessor function.

See also

IPAddress()

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 ( )

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. Upon successful completion the subnet mask of the device will be available via the SubnetMask() accessor function.

See also

SubnetMask()

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 ( )

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. Upon successful completion the gateway of the device will be available via the Gateway() accessor function.

See also

Gateway()

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 ( )

Get the version of IP firmware installed on the device, which must feature an IP board. Upon successful completion the IP firmware version of the device will be available via the IPFirmwareVersion() accessor function.

See also

IPFirmwareVersion()

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 ( )

Get the version of firmware installed on the device. Upon successful completion the firmware version of the device will be available via the MonitorFirmwareVersion() accessor function.

See also

MonitorFirmwareVersion()

Returns

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

bool Irisys::Blackfin::GetDNS1 ( )

Get the primary DNS server for the device, which must feature an IP board. Upon successful completion the primary DNS server for the device will be available via the DNS1() accessor function.

See also

DNS1()

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 ( )

Get the secondary DNS server for the device, which must feature an IP board. Upon successful completion the secondary DNS server for the device will be available via the DNS2() accessor function.

See also

DNS2()

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 ( )

Get the tertiary DNS server for the device, which must feature an IP board. Upon successful completion the tertiary DNS server for the device will be available via the DNS3() accessor function.

See also

DNS3()

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 ( )

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. Upon successful completion the client connection IP address configured on the device will be available via the ClientConfigIP() accessor function.

Note

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

See also

ClientConfigIP()

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 ( )

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. Upon successful completion the client connection hostname configured on the device will be available via the ClientConfigHostname() accessor function.

Note

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

See also

ClientConfigHostname()

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 ( )

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. Upon successful completion the client connection port configured on the device will be available via the ClientConfigPort() accessor function.

See also

ClientConfigPort()

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 ( )

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. Upon successful completion the client connection timeout configured on the device will be available via the ClientConfigTimeout() accessor function.

See also

ClientConfigTimeout()

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::GetClientConfigEnable ( )

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. Upon successful completion the client connection state configured on the device will be available via the ClientConfigEnable() accessor function.

See also

ClientConfigEnable()

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::GetUpTime ( )

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

See also

UpTime()

Returns

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

bool Irisys::Blackfin::GetUnitTimeZone ( )

Get the timezone configured on the device. Note that this has no functional impact on the device itself, which will always use UTC time. Upon successful completion the timezone configured on the device will be available via the UnitTimeZone() accessor function and the DST state will be available via the ApplyDST() function.

Note

The timezone can be configured using the People Counter Setup Tool

This command was introduced in the 2.0.x series of APIs.

See also

UnitTimeZone()
ApplyDST()

Returns

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

bool Irisys::Blackfin::GetNetworkChecksum ( )

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.

See also

NetworkChecksum()

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::SetDeviceID (std::string *deviceID*)

Set the device ID string on the device to the specified value, which has a maximum length of 63 characters

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 (std::string *deviceName*)

Set the device name string on the device to the specified value, which has a maximum length of 63 characters

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 (std::string *siteID*)

Set the site ID string on the device to the specified value, which has a maximum length of 63 characters

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 (std::string *siteName*)

Set the site name string on the device to the specified value, which has a maximum length of 63 characters

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 (std::string *localeString*)

Set the locale string on the device to the specified value, which has a maximum length of 63 characters

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 (std::string *userString*)

Set the user string on the device to the specified value, which has a maximum length of 63 characters

This command was introduced in the 2.0.x series of APIs.

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 *unitTime*)

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::SetCountLogPeriod (unsigned short *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	*nCountLog-Period*	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::SetClientConfigIP (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	*clientConfigI-P*	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::SetClientConfigHostname (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_HOSTNAME_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	*clientConfig-Port*	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	*clientConfig-Timeout*	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::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::SetDNS1 (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 (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 (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::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

bool Irisys::Blackfin::SetUnitTimeZone (BlackfinInterfaces::IrisysTimeZone *itz,* 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

This command was introduced in the 2.0.x series of APIs.

Parameters

in	*itz*	Irisys time zone 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

9.2 Irisys::BlackfinInterfaces::Count struct reference

`#include <BlackfinAPI.h>`

Public Member Functions

- Count ()

Public Attributes

- std::vector< unsigned long > countLines
- time_t countTime

9.2.1 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().

9.2.2 Constructor & destructor documentation

Irisys::BlackfinInterfaces::Count::Count ( )

9.2.3 Member data documentation

std::vector<unsigned long> Irisys::BlackfinInterfaces::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::BlackfinInterfaces::Count::countTime

The UTC device time when this count log entry was created

9.3 Irisys::BlackfinInterfaces::DeviceLogEntry struct reference

`#include <BlackfinAPI.h>`

Public Member Functions

- DeviceLogEntry ()

Public Attributes

- std::string errorDescription
- time_t timestamp

9.3.1 Detailed description

A structure containing a single device status log entry downloaded from a device

9.3.2 Constructor & destructor documentation

Irisys::BlackfinInterfaces::DeviceLogEntry::DeviceLogEntry ( )

9.3.3 Member data documentation

std::string Irisys::BlackfinInterfaces::DeviceLogEntry::errorDescription

A string describing the status of the device

time_t Irisys::BlackfinInterfaces::DeviceLogEntry::timestamp

The UTC device time when this device log entry was created

9.4 Irisys::BlackfinInterfaces::DeviceStatus struct reference

`#include <BlackfinAPI.h>`

Public Attributes

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

9.4.1 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.

9.4.2 Member data documentation

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

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

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

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

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

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

9.5 Irisys::BlackfinInterfaces::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

9.5.1 Constructor & destructor documentation

Irisys::BlackfinInterfaces::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.

9.5.2 Member data documentation

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

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

const bool Irisys::BlackfinInterfaces::IrisysTimeZone::m_bDSTSupported

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

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

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

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 40301
Issue 3