CIOBase

Overview

Summary

CIOBase is a C++ object that encapsulates key WinPLC runtime functions. Its primary function is simplification of backplane I/O access, although it does provide a few other functions. It is not intended to replace the WinPLC API, but rather to simplify the more complicated parts of it.

CIOBase is provided as source code. While the actual WinPLC API can be called from any language, because CIOBase is a C++ class, class, it must be used with the C++ language. The source is provided for you to use as you wish. Feel free to modify or enhance it.

I/O Driver

Before discussing the usage of CIOBase, some background about the WinPLC I/O interface would be useful. To provide a single point of control of the backplane and image register, the WinPLC I/O interface is implemented as a CE device driver. This provides a mechanism for sharing the I/O state across different applications and allows for synchronization and output lockout by the primary controller, if desired. The I/O driver is loaded by STARTUP.EXE at system initialization. As device drivers are not known to be very user friendly, the WinPLC API is wrapped around it, eliminating the need for the application to interface directly to the driver.

The I/O driver creates a worker thread during its initialization. All analog reads occur from this thread. Discrete reads and all I/O writes are done during the appropriate driver calls. Only analog reads, due to timing requirements, are performed by the I/O thread.

By default, the read thread is started at idle priority with no scan-to-scan sleep time. This means that the I/O driver will only run when other threads are sleeping – it will never pre-empt another thread. But it also means that the I/O thread will run continuously while other threads sleep. If the primary control app sleeps at least 1 or 2 ms at the end of each scan, the I/O thread will run fine. If not, you won't get analog inputs.

The advantage to this is that we don't waste a lot of time scanning analog inputs that generally change slowly. It also eliminates the need for the application to expressly service the driver. This disadvantage to this is that we don't have a very good feel of how many channels per scan we are getting, and, if the logic thread doesn't provide enough sleep time, we may not get anything at all. So what's the solution?

Since everyone has a different opinion about how this should work, we made it configurable. There are two critical settings in the I/O thread: 1) thread priority, and 2) sleep time. These two settings provide for three basic methods of operation: 1) continuous scanning at idle priority, 2) interval scanning at non-idle priority, and 3) polling. The

WinPLC API for setting up the I/O scan is WPLCConfigureScan. When configured for polling, each poll is triggered by a call to WPLCPollIO.

To prevent control of outputs by more than one application, the primary control app should lock writes to the outputs. While locked, only that client can change the output state. The function for locking writes is WPLCLockIOWrites. To unlock, call WPLCUnlockIOWrites.

Since reconfiguration of the I/O scan thread could have significant impact on I/O behavior, only the lock owner can call WPLCConfigureScan and WPLCPollIO. So in order to change the scan configuration, you must lock the outputs, thereby becoming the "I/O master". If the scan configuration has been changed, upon unlocking the outputs, the default scan config will be restored.

CIOBase Usage

So what's all that driver stuff got to do with CIOBase. In a word? Nothing - at least in the sense that CIOBase has anything to do with the implementation of the I/O. It is significant in the sense that CIOBase's primary function is to interface to the WinPLC API that interfaces to the driver. So you still need to understand it…

CIOBase, like virtually every other interface I have written, has a pretty lean interface. Keep it simple – somebody might find it useful. So, here's how you use it:

1. Instantiate a CIOBase object. Your choice – stack, heap, or static.

2. Initialize the newly created object by calling its Init method. During the Init method, the object initializes the runtime, reads the I/O base configuration, parses it and creates CModule objects for each module found, calcs standard PLC I/O config info, and does a read of the I/O to intialize its internal image register.

3. At the top of each PLC scan, you should read the I/O state by calling ReadIO. This reads the inputs, parses the input data, and stores the input state in the internal image register.

4. Do logic. There are two sets of methods for accessing I/O data (naturally) depending on whether you want to use SLOT:TYPE:POINT addressing or PLC style (slotless) addressing. For SLOT:TYPE:POINT use the Get/Set DI/DO/WI/WO/DWI/DWO methods where D stands for discrete, W for word, and DW double word; I is for input, and O is for output. For PLC style addressing use the Get/Set X/Y/WX/WY/DWX/DWY..

5. At the bottom of each scan, write the new I/O state by calling WriteIO. The SetXXX methods set the objects internal image register, but **do not** update the actual I/O. The I/O doesn't get updated until you call WriteIO. Remember this one, it can play with your head if you forget.

6. After writing, call the Idle function. It provides the sleep that the I/O driver needs and can do the polling also, if so configured.

7. When the CIOBase object is no longer needed, call Exit to shut down.

8. After calling Exit, the object may be destroyed.

There are several other methods in CIOBase but the previous steps cover the basics of I/O scanning and usage.

TIP: Consider using macros, variables, or some form of indirection to prevent hardcoding I/O addresses into your program. It's much easier to change 1 macro, than 27 hardcoded references. I'm personally partial to macros like the following:

```
#define MyInput              (IOB.GetDI(slot1, point1))
#define MyOutput             (IOB.GetDO(slot2, point2))
#define SetMyOutput(Val)     (IOB.SetDO(slot2, point2, Val))
```

The code then would look like this, for instance:

```
if(!MyInput) SetMyOutput(!MyInput);
```

The macros are put into an IO config header and everyone includes them. Slotx and pointx are replaced with the appropriate slot and point number and <poof> symbolic programming. If you require remapping at runtime (as opposed to compile time) you can make slotx and pointx variables and assign them at startup.

API Reference

The API reference is listed in header file order, which, with a few exceptions, is most closely tied to the order that you would need to call the functions. The alternative was alphabetical, which I liked less because is didn't preserve the logical groups that were in the header file. Sorry if this has caused any confusion.

CIOBase::CIOBase

Constructs a CIOBase object. Performs basic variable initialization, but initialization must be completed by calling Init() before using the object.

CIOBase();

Parameters:

None.

Return Value:

None.

CIOBase::Init

Performs initialization of newly created CIOBase object. During the Init method, the object initializes the WinPLC runtime, reads the I/O base configuration, parses the base config and creates CModule objects for each module found, calcs standard PLC I/O config info, and does a read of the I/O to intialize its internal image register.

int Init();

Parameters:

None.

Return Value:

Returns TRUE on success.

CIOBase::Exit

Performs shutdown of CIOBase object. Log off of WinPLC runtime and frees allocated memory

```
int Exit();
```

Parameters:

None.

Return Value:

None

CIOBase::OnIdle

Provides single call to manage sleep time and when enabled, manual polling. Both functions can be performed without calling OnIdle, this is provided for convenience. OnIdle also calls GetScanCount(TRUE) and returns the result.

```
int OnIdle(int SleepTime=1, int PollCount=0);
```

Parameters:

SleepTime
Amount if time in milliseconds to sleep. Default is 1 ms.

PollCount
Number of times to call PollIO when I/O scan is configured for polling. Default is 0 since the default scan configuration does not poll.

Return Value:

Returns the result of GetScanCount(TRUE). See GetScanCount for more info.

CIOBase::LockIOWrites

Locks outputs from being written by other I/O clients. When running more than one I/O client, it is best for the primary control app to be the only app controlling the outputs. Locking I/O writes guarantees that only the caller can control them.

```
BOOL LockIOWrites();
```

Parameters:

None.

Return Value:

None-zero if lock succeeds, zero on failure.

CIOBase::UnlockIOWrites

Unlocks previously locked IO writes. If the scan configuration was modified, the default scan config is restored.

```
BOOL UnlockIOWrites();
```

Parameters:

None.

Return Value:

Non-zero on success, zero on failure.

CIOBase::ConfigIOScan

Sets configuration of the I/O scanning thread. The I/O scanning thread is used to read analog inputs only. Discrete inputs are read during the call to ReadIO. Note: Only the current write lock owner can configure the scan. See LockIOWrites for details on write locking.

The two parameters: ScanPriority and SleepTime, combine to provide three basic modes of operation. The default scan configuration is idle priority and no sleep. This provides a continuous background scanning of I/O that runs when other apps sleep. A second mode is to raise the priority (perhaps even to time critical) and specify some sleep time. This provides a timed interrupt approach to scanning, where the sleep time defines the interrupt rate. The third approach is to set the priority to time critical and set the sleep time to –1, which signals the thread to stop scanning and wait for a poll command. The poll command is then issued as often as desired by calling PollIO.

int ConfigIOScan(DWORD ScanPriority, DWORD SleepTime);

Parameters:

ScanPriority
Priority of I/O scanning thread. All standard WinCE thread priorities are valid. See Win32 SDK SetThreadPriority for values. If you set a value higher than THREAD_PRIORITY_IDLE, be sure to allow sleep time or setup for polling – not doing so could lock up your WinPLC.

SleepTime
Number of milliseconds to sleep between scans. A value of –1 puts the thread into poll mode.

Return Value:

Non-zero on succes, zero on faliure.

CIOBase::PollIO

Sends a poll event to the I/O scanning thread, causing it to execute a single scan. Only valid if the thread is configured for polling.

int PollIO();

Parameters:

None.

Return Value:

Non-zero on success, zero on failure.

CIOBase::ReadIO

Reads I/O status from the driver, parses the data, and stores it in CIOBase's internal image register. Discrete inputs are read from the base, analog inputs are read from the driver's cache copy serviced by the I/O scanning thread.

```
int ReadIO();
```

Parameters:

None.

Return Value:

Zero on success, non-zero value is warning or error code.

CIOBase::WriteIO

Writes current state of CIOBase's internal image register to the I/O driver.

```
int WriteIO();
```

Parameters:

None.

Return Value:

Zero on success, non-zero is warning or error code.

CIOBase::GetDI

Reads a single discrete input from CIOBase's internal image register, using Slot:Point addressing.

BOOL GetDI(short Slot, short Point);

Parameters:

Slot
Slot number of desired module.

Point
Offset of desired point.

Return Value:

Value of specified I/O point or –1 on addressing failure.

CIOBase::GetDO

Reads a single discrete output from CIOBase's internal image register, using Slot:Point addressing.

BOOL GetDO(short Slot, short Point);

Parameters:

Slot
Slot number of desired module.

Point
Offset of desired point.

Return Value:

Value of specified I/O point or –1 on addressing failure.

CIOBase::SetDO

Writes a single discrete output to CIOBase's internal image register, using Slot:Point addressing. This does not write the value to the actual output. The actual output is updated when WriteIO is called.

void SetDO(short Slot, short Point, BOOL NewVal);

Parameters:

Slot
Slot of target module.

Point
Offset of target point.

NewVal
New value for target point.

Return Value:

None.

CIOBase::GetWI

Reads a single word input from CIOBase's internal image register, using Slot:Point addressing.

WORD GetWI(short Slot, short Point);

Parameters:

Slot
Slot number of desired module.

Point
Offset of desired point.

Return Value:

Value of specified I/O point or –1 on addressing failure.

CIOBase::GetWO

Reads a single word output from CIOBase's internal image register, using Slot:Point addressing.

WORD GetWO(short Slot, short Point);

Parameters:

Slot
Slot number of desired module.

Point
Offset of desired point.

Return Value:

Value of specified I/O point or –1 on addressing failure.

CIOBase::SetWO

Writes a single word output to CIOBase's internal image register, using Slot:Point addressing. This does not write the value to the actual output. The actual output is updated when WriteIO is called.

void SetWO(short Slot, short Point, WORD NewVal);

Parameters:

Slot
Slot of target module.

Point
Offset of target point.

NewVal
New value for target point.

Return Value:

None.

CIOBase::GetDWI

Reads a single dword input from CIOBase's internal image register, using Slot:Point addressing.

DWORD GetDWI(short Slot, short Point);

Parameters:

Slot
Slot number of desired module.

Point
Offset of desired point.

Return Value:

Value of specified I/O point or –1 on addressing failure.

CIOBase::GetDWO

Reads a single dword output from CIOBase's internal image register, using Slot:Point addressing.

DWORD GetDWO(short Slot, short Point);

Parameters:

Slot
Slot number of desired module.

Point
Offset of desired point.

Return Value:

Value of specified I/O point or –1 on addressing failure.

CIOBase::SetDWO

Writes a single dword output to CIOBase's internal image register, using Slot:Point addressing. This does not write the value to the actual output. The actual output is updated when WriteIO is called.

void SetDWO(short Slot, short Point, DWORD NewVal);

Parameters:

Slot
Slot of target module.

Point
Offset of target point.

NewVal
New value for target point.

Return Value:

None.

CIOBase::GetX

Reads a single discrete input from CIOBase's internal image register, using PLC style mapped addressing.

BOOL GetX(short Point);

Parameters:

Point
Offset of desired point within all points of the same type installed in the base. For instance, the first point of the second 8 point input module in the base would be 8 in decimal, or 010 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

Return Value:

Value of specified I/O point, or –1 for addressing error.

CIOBase::GetY

Reads a single discrete output from CIOBase's internal image register, using PLC style mapped addressing.

BOOL GetY(short Point);

Parameters:

Point
Offset of desired point within all points of the same type installed in the base. For instance, the first point of the second 8 point input module in the base would be 8 in decimal, or 010 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

Return Value:

Value of specified I/O point, or –1 for addressing error.

CIOBase::SetY

Writes a single discrete output to CIOBase's internal image register, using PLC style mapped addressing. This does not write the value to the actual output. The actual output is updated when WriteIO is called.

BOOL SetY(short Point, BOOL NewVal);

Parameters:

Point
Offset of target point within all points of the same type installed in the base. For instance, the first point of the third 8 point output module in the base would be 16 in decimal, or 020 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

NewVal
New value for target point.

Return Value:

Zero on success, -1 on addressing error.

CIOBase::GetWX

Reads a single word input from CIOBase's internal image register, using PLC style mapped addressing.

```
WORD GetWX(short Point);
```

Parameters:

Point
Offset of desired point within all points of the same type installed in the base. For instance, the first point of the second 8 point input module in the base would be 8 in decimal, or 010 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

Return Value:

Value of specified I/O point, or –1 for addressing error.

CIOBase::GetWY

Reads a single word output from CIOBase's internal image register, using PLC style mapped addressing.

```
WORD GetWY(short Point);
```

Parameters:

Point
Offset of desired point within all points of the same type installed in the base. For instance, the first point of the second 8 point input module in the base would be 8 in decimal, or 010 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

Return Value:

Value of specified I/O point, or –1 for addressing error.

CIOBase::SetWY

Writes a single word output to CIOBase's internal image register, using PLC style mapped addressing. This does not write the value to the actual output. The actual output is updated when WriteIO is called.

BOOL SetWY(short Point, WORD NewVal);

Parameters:

Point
Offset of target point within all points of the same type installed in the base. For instance, the first point of the third 8 point output module in the base would be 16 in decimal, or 020 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

NewVal
New value for target point.

Return Value:

Zero on success, -1 on addressing error.

CIOBase::GetDWX

Reads a single dword input from CIOBase's internal image register, using PLC style mapped addressing.

DWORD GetDWX(short Point);

Parameters:

Point
Offset of desired point within all points of the same type installed in the base. For instance, the first point of the second 8 point input module in the base would be 8 in decimal, or 010 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

Return Value:

Value of specified I/O point, or –1 for addressing error.

CIOBase::GetDWY

Reads a single dword output from CIOBase's internal image register, using PLC style mapped addressing.

DWORD GetDWY(short Point);

Parameters:

Point
Offset of desired point within all points of the same type installed in the base. For instance, the first point of the second 8 point input module in the base would be 8 in decimal, or 010 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

Return Value:

Value of specified I/O point, or –1 for addressing error.

CIOBase::SetDWY

Writes a single dword output to CIOBase's internal image register, using PLC style mapped addressing. This does not write the value to the actual output. The actual output is updated when WriteIO is called.

BOOL SetDWY(short Point, DWORD NewVal);

Parameters:

Point
Offset of target point within all points of the same type installed in the base. For instance, the first point of the third 8 point output module in the base would be 16 in decimal, or 020 in octal. It may be easier to use octal addressing since most modules use 8 point alignment.

NewVal
New value for target point.

Return Value:

Zero on success, -1 on addressing error.

CIOBase::ClearOutputs

Writes zero to all outputs in CIOBase's internal image register. The actual I/O isn't updated until WriteIO is called. This is useful for shutdown routines, calling ClearOutputs and then WriteIO.

```
void ClearOutputs();
```

Parameters:

None.

Return Value:

None.

CIOBase::OutputDebugString

Provides a convenient way to send debugging output to Viewer.

```
void OutputDebugString(LPCSTR pStr);
```

Parameters:

pStr
Wide string to send to the Viewer window.

Return Value:

None.

CIOBase::TurnOnRunLED

Provides convenient way to turn on the WinPLC's run LED. It is generally considered good form to turn on the run LED when your application is executing logic and controlling I/O.

```
void TurnOnRunLED();
```

Parameters:

None.

Return Value:

None.

CIOBase::TurnOffRunLED

Provides convenient way to turn off the WinPLC's run LED. It is generally considered good form to turn off the run LED when your application is no longer executing logic and controlling I/O.

```
void TurnOffRunLED();
```

Parameter:

None.

Return Value:

None.