Application Program Interface to Hytec Industry Packs

Supported on 9010 IOC and 5992/3331 VME

Definition Document

Hytec Electronics Ltd.

Reading

England

Tel: +44 (0)118 - 9757770

Fax: +44 (0)118 - 9757566

Email: sales@hytec-electronics.co.uk

Hytec Electronics Ltd

5 Cradock Road, Reading,

Berkshire RG2 0JT

United Kingdom

Phone: +44 (0) 118 9757770
Fax: +44 (0) 118 9757566

Title: **Application Program Interface (API) to Hytec Industry Packs on 9010 IOC and 5992/3331 VME**

Definition Document

Document Ref:	HYT/API 060713
Version:	V02.05
Date Issued:	Not Issued
Author:	Darrell Nineham / Jim Chen
Authorised:	Provisional
Pages:	88

Security Level:

- ❑ Public Domain
- ❑ Commercial Confidential
- ☑ Confidential - Internal Use Only
- ❑ Highly Confidential – Controlled Copy Named Recipients Only

Controlled Copy Number: *Not Required* **Recipient:** *Not Required*

Revision History

Version	Description of Revision	Author	Time	Issued:
V00.01	Initial Release	DAN	10/07/2006	Not Issued
V01.01	Updated for C++ API	JC	13/07/2006	Not Issued
V01.02	Adjusting error code Modified some parameters' type and settings Changed 8402 waveform function description that it is 32k words per channel for a normal 8402 IP card with 1M memory space by default Added C style code in the example	JC	18/08/2006	Not Issued
V01.03	Revise document errors	JC	25/08/2006	Not Issued
V01.04	Revise parameter definition errors	JC	11/09/2006	Not Issued
V02.01	Added Web Service Interface Definition	JC	21/09/2006	Not Issued
V02.02	Corrected data type to comply with wsdl spec	JC	20/10/2006	Not Issued
V02.03	Added Linux Driver and API installation for VME devices	JC	07/11/2006	Not Issued
V02.04	8506 signal invert only has one bit setting in DIR parameter. Not two as before. Signal inversion only applies to 8506.	JC	24/11/2006	Not Issued
V02.05	8512 support added.	DAN	26/09/2007	Not Issued

Contents

1 Introduction

This document defines the commands supported by Hytec Electronics Ltd's Application Program Interface (API) layer to support Hytec's Industry Pack (IP) cards on our 9010 IOC and 5331/3331 VME interface. This layer provides a series of hardware independent C++ interfaces in a shared library for Linux and a Dynamic Link Library (dll) for Windows.

The aim is to provide common interface (API) to Hytec's IP Cards for both of the following hardware platforms…

1. IOC Blade 9010.
2. PCI Card 5331 to VME Crate Controller 3331.

To use these interfaces in either in the C++ or C environments, include the following header files in the program:

HytecIPAPI.h

Hy8402.h

Hy8411.h

Hy85056.h

Hy8512.h

Hy85156.h

Hy8601.h

When you compile the source code, use C++ compiler instead of C compiler even if your code is written in C. This will link the C++ library properly. For example in Linux, use g++ instead of gcc. Please see an example in the example section.

The use of object orientat design and classes provides a more robust yet simpler interface with improved support for enchancment, additions and future maintenace.

The current list of Hytec IP card supported are:

8402	- 16 channel 16 bit Digital to Analogue Converter (DAC).
8411	- 16 channel 16 bit Analogue to Digital Converter (ADC).
8505	- 16 bit Digital Input / Output card.
8506	- 48 bit Digital Input / Output card.
8512	- 16 x 32 Bit Scaler card.
8515	- 8 channel RS232 card.
8516	- 8 chnnel RS422/RS485 card.
8601	- 4 axis (channel) Stepper Motor Controller

NOTE: The current implementation has no support for interrupts. This support will be implemented and will include callbacks and asynchronous support for the serial lines. A basic serial line support is provided in this release.

1.1 Reference

Generic error code:

-9010001 Carrier board fan number invalid
-9010002 Carrier board temperature sensor number invalid
-9010003 Carrier board push button number invalid
-9010004 IP slot invalid
-9010005 Carrier register address invalid
-9010006 IP card not present
-9010007 Channel number does not exist
-9010008 Device driver error

1.2 Glossary

Term	Description
ASCII	American Standard Code For Information Interchange
API	Application Program Interface
IOC	Input / Output Controller.
IP	Industry Pack.

2 8402 Analogue Output Class

The Intermediate Common Interface Layer provides the following commands/functions in support of the Hytec 8402 16 x 16 Bit DAC IP (Industry Pack) Card. The interface prototype is defined as below.

2.1 Public function -- Hy8402_Configure

Hy8402_Configure

Syntax

```
CHy8402::Hy8402_Configure(     int vmeslot,
                               char ipslot,
                               int vectornum,
                               int doram,
                               int clockSource,
                               int clockRate,
                               int inhibit): long;
```

Parameter

`vmeslot` - Which VME slot is the IP Card fitted onto (for VME only, ignored by IOC9010).

`Ipslot` - Which IP site is the Card fitted ('A','B','C','D', 'E' or 'F').

`vectornum` - Interrupt Vector for this IP Card (Interrupt Vector 0=find me one).

`doram` - 0 (Output via Registers) or 1 (Output via RAM & Set to passed ClockRate).

`clockSource` - 0 (internal) or 1(external).

`clockRate` - Sampling Clock Rate * See Below Table.

`inhibit` - 0 (inhibit disabled) or 1 (inhibit enabled).

*clockrate Mapping…

Clockrate Value	Frequency
0	1 Hz
1	2 Hz
2	5 Hz
3	10 Hz

4	20 Hz
5	50 Hz
6	100 Hz
7	200 Hz
8	500 Hz
9	1 KHz
10	2 KHz
11	5 KHz
12	10 KHz
13	20 KHz

Result

0 - is returned if the card is successfully configured.

Negative value - indicates the configure has failed.

Description

This function is used to Configure 8402 IP cards. All Hytec Electronics Ltd IP Cards are VITA4 compliant and have their identification stored in an ID PROM. This configure function checks that the required IP Card is in the requested location (VME Slot and IP Site), then it confirms that all requested parameters are in range. Failure of either of these checks will return a negative value. If these checks are successful, the card is configured using the passed parameters. During this configuration any IP Card access that fails will cause a negative return. Please check error code definition for details. On successfully writing all the parameters a return of 0 will be produced.

Error code

-8402001 IP card in the specified slot is not 8402

-8402010 VME setting invalid

-8402012 Clock rate setting invalid

Example

```
/* Configure Slot 3 in Site 'C' at 20KHz */
CHy8402 DAC1;                          //create the object
If (DAC1.Hy8402_Configure(3,'C',0,0,0,13,0))
{
     /* Display Error Message and Exit */
     printf("8402 Configure Failed !/n");
     return(-1);
}
```

2.2 Hy8402_Write

Hy8402_Write

Syntax

```
CHy8402::Hy8402_Write(    int DAC_ch,
                          unsigned short int rootvalue): long;
```

Parameter

`DAC_ch` - Which DAC Channel to update (0-15).

`value` - The new value for this DAC Channel.

Result

0 - is returned if the card value is successfully updated.

Negative value - indicates the update has failed.

Description

This function is used to update the DAC output values (directly via the registers) of the 8402 IP cards.

Error code

-8402013 Channel number invalid

-8402014 8402 not configured

Example: assume the DAC1 has been created during the configuration

```
/* Set Signal 0 to full scale */
If (DAC1.Hy8402_Write(0,65535))
{
     /* Display Error Message */
     printf("8402 Update Failed !/n");
}
```

2.3 Hy8402_WaveForm

Hy8402_WaveForm

Syntax

```
CHy8402::Hy8402_Write(    int DAC_ch,
                          unsigned short int *value): long;
```

Parameter

`DAC_ch` - Which DAC Channel to update (0-15).

`*value` - pointer to the waveform buffer.

Note: the waveform buffer size must be 32k word exactly.

Result

0 - is returned if the card value is successfully updated.

Negative value - indicates the update has failed.

Description

This function is used to update a DAC signal waveform memory values of the 8402 IP cards.

Error code

-8402014 waveform memory setting does not match 32k

Example: assume the DAC1 has been created during the configuration

```
/* Set channel 7 to a certain waveform data */
unsigned short int value[32768];
If (DAC1.Hy8402_WaveForm(7, value))
{
     /* Display Error Message */
     printf("Writing to waveform memory Failed !/n");
}
```

8411 Analogue Input Class

The Intermediate Common Interface Layer provides the following commands/functions in support of the Hytec 8411 16 x 16 Bit ADC IP (Industry Pack) Card. The interface prototype is defined as below.

2.4 Hy8411_Configure

Hy8411_Configure

Syntax

```
CHy8411::Hy8411_Configure(     int vmeslot,
                               char ipslot,
                               int clockRate): long;
```

Parameter

`vmeslot` - Which VME slot is the IP Card fitted onto (for VME only, ignored by IOC9010).

`Ipslot` - Which IP site is the Card fitted ('A','B','C','D', 'E' or 'F').

`clockRate` - Sampling Clock Rate * See Below Table.

*clockrate Mapping…

Clockrate Value	Frequency	Clockrate Value	Frequency
0	1 Hz	8	500 Hz
1	2 Hz	9	1 KHz
2	5 Hz	10	2 KHz
3	10 Hz	11	5 KHz
4	20 Hz	12	10 KHz
5	50 Hz	13	20 KHz
6	100 Hz	14	50 KHz
7	200 Hz	15	100 KHz

Result

0 - is returned if the card is successfully configured.

Negative value - indicates the configure has failed.

Description

This function is used to Configure 8411 IP cards. All Hytec Electronics Ltd IP Cards are VITA4 compliant and have their identification stored in an ID PROM. This configure function checks that the required IP Card is in the requested location (VME Slot and IP Site), then it confirms that all requested parameters are in range. Failure of either of these checks will cause a negative return. If these checks are successful, the card is configured using the passed parameters. During this configuration any IP Card access that fails will cause a negative return. On successfully writing all the parameters a return of 0 will be produced.

Error code

-8411001	IP card is not 8411
-8411010	VME slot setting invalid
-8411012	Clock rate setting invalid

Example

```
/* Configure Slot 3 in Site 'C' at 100KHz */
CHy8411 ADC1;
Long status;
Status = ADC1.Hy8411_Configure(3,'C',15);
If (status != 0)
{
     /* Display Error Message and Exit */
     printf("8411 Configure Failed !/n");
     return status;
}
```

2.5 Hy8411_Read

Hy8411_Read

Syntax

```
CHy8411::Hy8411_Read(     int ADC_ch,
                          USHORT *value): long;
```

Parameter

`ADC_ch` - Which ADC Channel to update (0-15).

`*value` - Pointer to the variable that receives the value of this ADC Channel.

Result

0 - is returned if the card value is successfully read.

Negative value - indicates the read has failed.

Description

This function is used to read the present ADC values of the 8411 IP cards.

Error code

-8411013 8411 channel number invalid

-8411014 8411 Not configured

Example: assume the ADC1 has been created during the configuration

```
/* Read ADC channel 0 */
unsigned short int value;
If (ADC1.Hy8411_Read(0,&value) == 0)
{
     printf("8411 ADC channel 0 value = 0x%X !/n", value);
}
```

---

3 8505 / 8506 Digital I/O Class

The Intermediate Common Interface Layer provides the following commands/functions in support of the Hytec 8505 (1 x 16 Bit Digital I/O) and 8506 (3 x 16 Bit Digital I/O) IP (Industry Pack) Card. The interface prototype is defined as below.

3.1 Hy85056_Configure

Hy85056_Configure

Syntax

```
Function for 8505.
CHy85056::Hy85056_Configure(  int vmeslot,
                              char ipslot,
                              int debrate,
                              unsigned short int debmask,
                              int scanrate,
                              int dir,
                              unsigned short int vectornum,
                              unsigned short int intmask,
                              int pwidth,
                              unsigned short int pmask,
                              int clock): long;

Function overload for 8506.
CHy85056::Hy85056_Configure(  int vmeslot,
                              char ipslot,
                              int portnum,
                              int debrate,
                              unsigned short int debmask,
                              int scanrate,
                              int dir,
                              unsigned short int vectornum,
                              unsigned short int intmask,
```

```
                                    int pwidth,
                                    unsigned short int pmask,
                                    int clock): long;
```

Parameter

`vmeslot` - Which VME slot is the IP Card fitted onto (for VME only, ignored by IOC9010).

`ipslot` - Which IP site is the Card fitted ('A','B','C','D', 'E' or 'F').

`portnum` - Which Digital Channel / Port (0-2 for 8506 only).

`debrate` - Debounce Rate (0 = 100Hz, 1 = 200Hz, 2 = 500Hz, 3 = 1kHz).

`debmask` - Debounce Bit Mask, select input bits to debounce (0x0000 - 0xFFFF).

`scanrate` - input scan rate (0 = 1kHz, 1 = 10kHz, 2 = 100kHz, 3 = 1MHz).

`dir` - The direction of the I/O…

0 = All 16 bits inputs.
1 = lower 8 bits outputs / high 8 bits inputs (8505 Only).
2 = lower 8 bits inputs / high 8 bits outputs (8505 Only).
3 = All 16 bits outputs.
Bits 2 & 3 (values 4 and 8, respectively) encode whether the input and output values should be inverted.
(dir & 4) == 4 => Invert all signals. Applies to input and output (only 8506).

`vectornum` - Interrupt Vector for this IP Card (Interrupt Vector 0=find me one).

`intmask` - Interrupt Bit Mask, select which bits to generate interrupts (0x0000 - 0xFFFF).

`pwidth` - pulse width
0 = 1 msec
1 = 10 msec
2 = 100 msec
3 = 1 sec
4 = 2 sec
5 = 5 sec
6 = 10 sec
7 = 20 sec
8 = 50 sec
9 = 100 sec

`pmask` - Pulse Output Bit Mask, select bits to pulse on output (0x0000 - 0xFFFF).

`clock` - 0 = internal, 1 = external.

Result

0 - is returned if the card is successfully configured.

Negative value - indicates the configuration has failed.

Description

This function is used to Configure the 8505 and 8506 IP cards. An 8506 is simply seen as three 8505s in the same location (VME and IP Slot) with each port being referenced as separate 'card'. All Hytec Electronics Ltd IP Cards are VITA4 compliant and have their identification stored in an ID PROM. This configure function checks that the required IP Card is in the requested location (VME Slot and IP Site), then it confirms that all requested parameters are in range. Failure of either of these checks will cause a negative return. If these checks are successful, the card is configured using the passed parameters. During this configuration any IP Card access that fails will cause a negative return. On successfully writing all the parameters a return of 0 will be produced.

Error code

-8505010	IP is not 8505 card
-8506010	IP is not 8506 card
-85056012	8505 or 8506 IP not configured
-85056013	Debounce rate setting invalid
-85056014	Scan rate setting invalid
-85056015	IO direction setting invalid
-85056016	Pause width setting invalid
-85056017	Clock setting invalid
-85056018	Port number invalid (for 8506 only)

Example

```
/* configure 8506 in VME slot 0, site 3 Port 1 As Outputs All Pulsed*/
CHy85056 DIO1;
long status;
status = DIO1.Hy85056_Configure(0,3,1,0,0,0,3,0,0,0,0,0);
if (status != 0)
{
     /* Display Error Message and Exit */
     printf("Configure 8506 Failed !/n");
     return status;
}
```

3.2 Hy85056_Write

Hy85056_Write

Syntax

```
Write a word (16 bits).
Function for 8505.
CHy85056::Hy85056_Write( unsigned short int mask,
                         unsigned short int value): long;

Function overload for 8506.
CHy85056::Hy85056_Write( int portnum,
                         unsigned short int mask,
                         unsigned short int value): long;
```

Parameter

mask	- mask bits. Only bits set to 1 are affected.
value	- The new value for this Digital Output Channel. For word write, its value could be 0 ~ 65535. For bit write, its value could be either 1 or 0.
`portnum`	- Which Digital Port (0-2 for 8506).

Result

0	- is returned if the card value is successfully updated.
Negative value	- indicates the update has failed.

Description

This function set is used to update the Digital Output of the 8505 or 8506 IP cards either by the whole word or a single bit.

Error code

-85056019	Write violation. Try to write to an input port.

Example: assume the DIO1 has been created during the configuration

```
/* Set 8505 outputs to Test Pattern */
If (DIO1.Hy85056_Write(0xFFFF, 0xCCCC))
{
     /* Display Error Message */
     printf("8505 Update Failed !/n");
}

/* Set 8506 port 1 (second port) outputs to Test Pattern */
If (DIO1.Hy85056_Write(1, 0xFFFF, 0xCCCC))
{
     /* Display Error Message */
     printf("8506 Update Failed !/n");
}
```

3.3 Hy85056_Write_Bit

Hy85056_Write_Bit

Syntax

```
Write a bit to 8505.
CHy85056::Hy85056_Write_Bit(  int Bit, unsigned short int value): long;

Function over load for 8506.
CHy85056::Hy85056_Write_Bit(  int portnum, int Bit, unsigned short int
value): long;
```

Parameter

Bit	- which bit to write (0 ~ 15).
value	- 1 sets the bit, 0 clears the bit.
portnum	- Which Digital Channel / Port (0-2 for 8506).

Result

0	- is returned if the card value is successfully updated.
Negative value	- indicates the update has failed.

Description

This function is used to update the Digital Output of the 8505 or 8506 IP cards either by a single bit.

Error code

-85056020	Bit number invalid

Example: assume the DIO1 has been created during the configuration

```
/* Set 8505 bit 4 to HIGH */
If (DIO1.Hy85056_Write_Bit(4, 1))
{
     /* Display Error Message */
     printf("8505 Update Failed !/n");
}
```

```
/* Clear 8506 port 1 (second port) bit 14 to 0 */
If (DIO1.Hy85056_Write(1, 14, 0))
{
     /* Display Error Message */
     printf("8506 Update Failed !/n");
}
```

3.4 Hy85056_Read

Hy85056_Read

Syntax

```
Read the whole word.
For 8505.
CHy85056::Hy85056_Read(  unsigned short int *value
                         unsigned short int mask): long;
For 8506. Function overload.
CHy85056::Hy85056_Read(  int portnum,
                         unsigned short int *value
                         unsigned short int mask): long;
```

Parameter

`*value`	- Pointer to the value that stores the Digital Input Register.
mask	- mask bits. Only bits set to 1 are affected. Any other bits will be read 0.
`portnum`	- Which Digital Channel / Port (0-2 for 8506).

Result

0	- is returned if the card value is successfully updated.
Negative value	- indicates the update has failed.

Description

This function is used to read the masked bits of the Digital Input of the 8505 or 8506 IP cards.

Note

If the channels have been set to output, it reads the output status back.

Example: assume the DIO1 has been created during the configuration

```
/* Read 8505 input register with mask 0xAA */
unsigned short int value;
If (DIO1.Hy85056_Read(&value, 0xAA))
```

```
{
     /* Display Error Message */
     printf("8505 Read Failed !/n");
}
else
{
     printf("8505 input = 0x%X !/n", value);
}

/* Read 8506 port 2 input register with bits define in mask 0x55*/
unsigned short int value;
If (DIO1.Hy85056_Read(2, &value, 0x55))
{
     /* Display Error Message */
     printf("8506 Read Failed !/n");
}
else
{
     printf("8506 port 2 output = 0x%X !/n", value);
}
```

3.5 Hy85056_Read_Bit

Hy85056_Read_Bit

Syntax

```
Read a single bit of the input card.
For 8505.
CHy85056::Hy85056_Read_Bit(   int Bit): bool;
For 8506. Function overload.
CHy85056::Hy85056_Read_Bit(   int portnum, int Bit): bool;
```

Parameter

Bit - bit number (0 ~ 15)

`portnum` - Which Digital Channel / Port (0-2 for 8506).

Result

false - bit = 0.

true - bit = 1.

If error occurs, error code is in m_ErrorCode property of the class and the return would be false. This error code can be retrieved by using DIO1.GetErrorCode function. Please see example below.

Description

This function is used to read a single bit of the Digital Input of the 8505 or 8506 IP cards input.

Note

If a bit has been set to output, it reads the output status back.

Example: assume the DIO1 has been created during the configuration

```
/* Read 8505 bit 3 */
If (DIO1.Hy85056_Read_Bit(3))
{
     /* bit is set, do something… */
}else
{
```

```
     If (DIO1.GetErrorCode != 0)
          printf("8505/6 read error! 0x%X \n", DIO1.GetErrorCode);
     else
          /* bit has been cleared… */
}

/* Read 8506 port 1 bit 12 */
If (DIO1.Hy85056_Read_Bit(1, 12))
{
     /* bit is set, do something… */
}else
{
     If (DIO1.GetErrorCode != 0)
          printf("8505/6 read error! 0x%X \n", DIO1.GetErrorCode);
     else
          /* bit has been cleared… */
}
```

4 8512 Scaler / Counter Class

The Intermediate Common Interface Layer provides the following commands/functions in support of the Hytec 8512 (16 x 32 bit Scaler / Counter) IP (Industry Pack) Card. The interface prototype is defined as below.

4.1 Hy8512_Configure

Hy8512_Configure

Syntax

```
CHy8512::Hy8512_Configure(    int vmeslot,
                              char ipslot,
                              USHORT scaler,
                              USHORT clock,
                              char vector,
                              USHORT overflow): long;
```

Parameter

`vmeslot` - Which VME slot is the IP Card fitted onto (for VME only, ignored by IOC9010).

`Ipslot` - Which IP site is the Card fitted (‘A’,’B’,’C’,’D’, ‘E’ or ‘F’).

`scaler` - Which scalers are being armed / set up (selection is bit wise, so 0xFFFF is all 16 Scalers).

`clock` - How each scaler is clocked, selection is bit wise 0 – external input or 1 is Internal 10MHz clock, 0xFFFF is all 16 Scalers off the 10MHz Internal Clock and 0x0000 is all driven from external inputs.

`Vector`[1] - The interrupt vector to assign this card. This forms the upper 8 bits of the interrupt vector register.

`Overflow`[1] - Which scalers when over flowing cause an interupt (bit wise selection, so 0xFFFF is all 16 Scalers).

[1] For Future Use.

Result

0 - is returned if the card is successfully configured.

Negative value - indicates the configuration has failed.

Description

This function is used to Configure 8512 IP cards. All Hytec Electronics Ltd IP Cards are VITA4 compliant and have their identification stored in an ID PROM. This configure function checks that the required IP Card is in the requested location (VME Slot and IP Site), then it confirms that all requested parameters are in range. Failure of either of these checks will cause a negative return. If these checks are successful, the card is configured using the passed parameters. During this configuration any IP Card access that fails will cause a negative return. On successfully writing all the parameters a return of 0 will be produced.

The 8512 is by default setup as 16 x 32 Bit Scalers.

Error code

-8512010 Not 8512 IP card

Example

```
/* Configure Slot 3 in Site 'C' to use scalers 0 and 3 */
/* scaler 0 to be clocked internal 10MHz and scaler 3 Externally */
CHy8512 SCALER1;
long status;
status = SCALER1.Hy8512_Configure(3,'C',0x0009,0x0001,0,0);
If (status != 0)
{
     /* Display Error Message and Exit */
     printf("8512 Configure Failed !/n");
     return status;
}
```

4.2 Hy8512_Read

Hy8512_Read

Syntax

```
CHy8512::Hy8512_Read(    int scaler,
                         ULONG *value,
                         boolean clear): long;
```

Parameter

`scaler` - Which Scaler to read (0-15).

`*value` - A pointer of an int to store the count read from the speicified scaler.

`clear` - Flag set to 1 to Clear Counter to Zero After Reading.

.

Result

0 - is returned if the scaler was successfully read.

Negative value - indicates the read has failed.

Description

This function is used to read the count from any single scaler on an 8512 IP cards. Due to the nature of the hardware a Hy8512_Read_All( ) is quicker than 3 individual Hy8512_Read ( ), also it is quicker clear all scalers than a single one.

Example: assume the SCALER1 has been created during the configuration

```
/* Read a count and reset to 0 */
#define CLEAR_WHEN_READ  1
#define DON'T_CLEAR      0
ULONG count;
if (SCALER1.Hy8512_Read(0, &count, CLEAR_WHEN_READ))
{
     /* Display Error Message */
     printf("8512: Scaler Can Not be Read\n");
}
```

4.3 Hy8512_Read_All

Hy8512_Read_All

Syntax

```
CHy8512::Hy8512_Read(    ULONG *value,
                         boolean clear): long;
```

Parameter

`*value` - A pointer of an array of int to store the count read from the speicified scaler.

`clear` - Flag set to 1 to Clear Counter to Zero After Reading.

.

Result

0 - is returned if the scaler was successfully read.

Negative value - indicates the read has failed.

Description

This function is used to read the count from all the scalers on an 8512 IP cards. Due to the nature of the hardware a Hy8512_Read reading All( ) is quicker than even just 3 individual Hy8512_Read ( ), also it is quicker clear all scalers than a single one.

Example: assume the SCALER1 has been created during the configuration

```
/* Read a count and reset to 0 */
#define CLEAR_WHEN_READ  1
#define DON'T_CLEAR      0
ULONG count[16];
if (SCALER1.Hy8512_Read ( &count[0], CLEAR_WHEN_READ))
{
     /* Display Error Message */
     printf("8512: Some or All of the Scalers Can Not be Read\n");
}
```

4.4 Hy8512_Write

Hy8512_Write

Syntax

```
CHy8512::Hy8512_Write(    int scaler,
                          ULONG value): long;
```

Parameter

`scaler` - Which scaler to write (0-15).

`value` - The value to write to this scaler.

Result

0 - is returned if the scaler's value is successfully updated.

Negative value - indicates the update has failed.

Description

This function is used to output character streams via the 8512 IP cards.

Example: assume the SCALER1 has been created during the configuration

```
/* write a string to port 3 */
ULONG startvalue = 12300;

If (SCALER1.Hy8512_Write(3, startvalue))
{
     /* Display Error Message */
     printf("8512 Write Failed !/n");
}
```

---

5 8515 / 8516 Serial Communication Class

The Intermediate Common Interface Layer provides the following commands/functions in support of the Hytec 8515 8 x RS232 and Hytec 8516 8 x RS422 / RS485 IP (Industry Pack) Card. The interface prototype is defined as below.

Note: These API's are very basic and temporary functions in order to get things going. We will implement more sophisticated functions include the call backs afterwards.

5.1 Hy85156_Configure

Hy8515_Configure

Syntax

```
CHy85156::Hy85156_Configure(  int vmeslot,
                              char ipslot,
                              char vector,
                              int delay,
                              int halfduplex,
                              int delay485): long;
```

Parameter

`vmeslot`	- Which VME slot is the IP Card fitted onto (for VME only, ignored by IOC9010).
`Ipslot`	- Which IP site is the Card fitted ('A','B','C','D', 'E' or 'F').
`Vector`	- The interrupt vector to assign this card. This forms the upper 8 bits of the interrupt vector register. The lower 8 bits connects to the interrupt hardware source.
`Delay`	- + Values interval between successive interrupts, - Values the depth of the interrupt buffer. Valid value is -64 ~ 32000.
`halfduplex`	- whether the card is to run in full or half duplex mode(0=Full Duplex, 1=Half Duplex with echo, 2=Half duplex without echo).
`delay485`	- The delay in bit length after the last character to switch RS485 mode (valid values are 0 ~ 15).

Result

0	- is returned if the card is successfully configured.
Negative value	- indicates the configuration has failed.

Description

This function is used to Configure 8515 and 8516 IP cards. All Hytec Electronics Ltd IP Cards are VITA4 compliant and have their identification stored in an ID PROM. This configure function checks that the required IP Card is in the requested location (VME Slot and IP Site), then it confirms that all requested parameters are in range. Failure of either of these checks will cause a negative return. If these checks are successful, the card is configured using the passed parameters. During this configuration any IP Card access that fails will cause a negative return. On successfully writing all the parameters a return of 0 will be produced.

Error code

-85156010 Neither 8515 nor 8516 IP card

Example

```
/* Configure Slot 3 in Site 'C' serial communication card as half duplex
with echo for all ports in VME system*/
CHy85156 SERIAL1;
long status;
status = SERIAL1.Hy85156_Configure(3,'C',0,0,1,0);
If (status != 0)
{
     /* Display Error Message and Exit */
     printf("8515 Configure Failed !/n");
     return status;
}
```

5.2 Hy85156_ComsSetup

Hy85156_ComsSetup

Syntax

```
CHy85156::Hy85156_ComsSetup(  int port,
                              unsigned long int baud,
                              char parity,
                              int numstopbits,
                              int numdatabits,
                              char flowctr,
                              ): long;
```

Parameter

`port` - Which port is being configured 0 ~ 7.

`baud` - The desired baud rate of the device. This setting can only be one of these values: 100, 400, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200, 153600, 230400, 460800, 921600. Any other value will generate an error.

`parity` - A character indicating the required parity. Can be either “N” for no parity, “O” for odd parity or “E” for even parity check.

`stopbits` - The number of stop bits required. The value can only be 0 for 1 stop bit and 4 for either 1-1/2 bit or 2 bits depending on the data bit. If data bit is 5, setting 4 represent 1-1/2 stop bit. Other data bit settings represent 2 stop bits.

`databits` - The number of data bits required. Can be only 5, 6, 7 and 8.

`flowctrl` - A character indicating the required flow control.

```
   Bit7 6 5 4 3 2 1 0
      0 0 X X X X X X
      [_] | | [_____]
       |  | |    |
 no used  | |    software flow control
          | |    0 0 0 0      no software flow control
          | |    1 1 1 1      transmit Xon1 and Xon2 / Xoff1 and Xoff2
          | |                 and receiver compares  Xon1 and Xon2 / Xoff1
          | |                 and Xoff2
          | =0 disable hardware flow control, =1 enable
          =0 use RTS/CTS, =1 use DTR/DSR pair
```

Result

0 - is returned if the card is successfully configured.

Negative value - indicates the configure has failed.

Description

This function is used to set up the Communication Protocols for 8515 and 8516 IP cards. This function confirms that all requested parameters are in range. Failure of either of these checks will cause a negative return. If these checks are successful, the card is configured using the passed parameters. During this configuration any IP Card access that fails will cause a negative return. On successfully writing all the parameters a return of 0 will be produced.

Error code

-85156011	85156 IP not configured
-85156012	Port number invalid
-85156013	Baud rate setting invalid
-85156014	Parity setting invalid
-85156015	Data bit setting invalid
-85156016	Stop bit setting invalid
-85156017	Flow control setting invalid

Example: assume the SERIAL1 has been created during the configuration

```
/* Setup Coms on Port 7 as 9600 Baud, Even Parity, 1 Stop Bit */
/* 8 data bit, No Flow Control */
long status;
status = SERIAL1.Hy85156_ ComsSetup (7, 9600,'E',1,8,0x00);
If (status != 0)
{
     /* Display Error Message and Exit */
     printf("8515 Coms Setup Failed !/n");
     return status;
}
```

5.3 Hy85156_Write

Hy85156_Write

Syntax

```
CHy85156::Hy85156_Write( int port,
                         Unsigned char *value,
                         int length): long;
```

Parameter

`port`	- Which Port to Transmit (0-7).
`*value`	- A pointer of a string to transmit via this port.
Length	- number of characters to write

Result

>0	- number of bytes written
Negative value	- indicates the update has failed.

Description

This function is used to output character streams via the 8515 and 8516 IP cards.

Example: assume the SERIAL1 has been created during the configuration

```
/* write a string to port 3 */
unsigned char startupstring[]="START 123";

If (SERIAL1.Hy85156_Write(3, (UCHAR*) startupstring,
     strlen(startupstring))
{
     /* Display Error Message */
     printf("8515 Write Failed !/n");
}
```

5.4 Hy85156_Read

Hy85156_Read

Syntax

```
CHy85156::Hy85156_Read(  int port,
                         Unsigned char *value): long;
```

Parameter

`port` - Which Port to receive (0-7).

`*value` - A pointer of a string to copy any data received via this port.

Result

>0 - number of characters read

0 - nothing received.

Negative value - indicates a read error.

Description

This function is used to read character streams received via the 8515 and 8516 IP cards.

Example: assume the SERIAL1 has been created during the configuration

```
/* Read a char from the receiving buffer */
unsigned char buffer[64];
unsigned char a;
long count;
int i=0, j;

do
{
     do
     {
          count = SERIAL1.Hy85156_Read(0, &a)
     }while(count<=0);
```

```
     If (a == '\n') break;     //out the loop when return is hit
     For(j=0;j<count;j++,i++)
          Buffer[i] = a;
}while(1);

/* Display Error Message */
printf("8515 Received string: %s/n", buffer);

}
```

5.5 Hy85156_Flush

Hy85156_Flush

Syntax

```
Individual port flush.
CHy85156::Hy85156_Flush( int port, bool TX, bool RX): long;

Function overloaded for multiple ports being flushed
CHy85156::Hy85156_Flush( unsigned short int mask): long;
```

Parameter

`port` - Which Port to receive (0-7).

`TX` - if true flush the TX buffer for the specified port.

`RX` - if true flush the RX buffer for the specified port.

`mask` - A bit mask of which port TX, RX buffers to flush. Bit 0 and 1 represent port 0, bit 2 and 3 represent port 1 and so forth. Mask bit define as below.

Define	Value	Description	Define	Value	Description
COM_0_TX_FLUSH	0x0001	Flush TX Buffer of Com 0	**COM_4_TX_FLUSH**	0x0100	Flush TX Buffer of Com 4
COM_0_RX_FLUSH	0x0002	Flush RX Buffer of Com 0	**COM_4_RX_FLUSH**	0x0200	Flush RX Buffer of Com 4
COM_1_TX_FLUSH	0x0004	Flush TX Buffer of Com 1	**COM_5_TX_FLUSH**	0x0400	Flush TX Buffer of Com 5
COM_1_RX_FLUSH	0x0008	Flush RX Buffer of Com 1	**COM_5_RX_FLUSH**	0x0800	Flush RX Buffer of Com 5
COM_2_TX_FLUSH	0x0010	Flush TX Buffer of Com 2	**COM_6_TX_FLUSH**	0x1000	Flush TX Buffer of Com 6
COM_2_RX_FLUSH	0x0020	Flush RX Buffer of Com 2	**COM_6_RX_FLUSH**	0x2000	Flush RX Buffer of Com 6
COM_3_TX_FLUSH	0x0040	Flush TX Buffer of Com 3	**COM_7_TX_FLUSH**	0x4000	Flush TX Buffer of Com 7
COM_3_RX_FLUSH	0x0080	Flush RX Buffer of Com 3	**COM_7_RX_FLUSH**	0x8000	Flush RX Buffer of Com 7

Result

0 - is returned if the buffer flush is successfully.

Negative value - indicates some aspect of the flash has failed.

Description

This function is used to flush / empty any combination of transmit and receive buffer of any / all channels. The bit masks are set up as follows…

Example: assume the SERIAL1 has been created during the configuration

```
/* Flush Port 5 Tx and Rx Buffers */
If (SERIAL1.Hy85156_Flush(5, true, true))
{
     /* Display Error Message */
     printf("8515 Flush Failed !/n");
}

/* Flush Port 0, 1 Tx and Rx Buffers */
If (SERIAL1.Hy85156_Flush(COM_0_TX_FLUSH ||
                          COM_0_RX_FLUSH ||
                          COM_1_TX_FLUSH ||
                          COM_1_RX_FLUSH))
{
     /* Display Error Message */
     printf("8515 Flush Failed !/n");
}
```

6 8601 Motor Controller Class

The Intermediate Common Interface Layer provides the following commands/functions in support of the Hytec 8601 4 x Stepper Motor IP (Industry Pack) Card. The interface prototype is defined as below.

6.1 Hy8601_Configure

Hy8601_Configure

Syntax

```
Hy8601::Hy8601_Configure(     int vmeslot,
                              char ipslot
                              ): long;
```

Parameter

`vmeslot` - Which VME slot is the IP Card fitted onto (for VME only, ignored by IOC9010).

`Ipslot` - Which IP site is the Card fitted ('A','B','C','D', 'E' or 'F').

Result

0 - is returned if the card is successfully configured.

Negative value - indicates the configure has failed.

Description

This function is used to Configure 8601 IP cards. All Hytec Electronics Ltd IP Cards are VITA4 compliant and have their identification stored in an ID PROM. This configure function checks that the required IP Card is in the requested location (VME Slot and IP Site), then it confirms that all requested parameters are in range. Failure of either of these checks will cause a negative return. On successfully writing all the parameters a return of 0 will be produced.

Example

```
/* Configure VME Slot 3 in Site 'C' as an 8601 */
CHy8601 MOTOR1;
If (MOTOR1.Hy8601_Configure(3,'C'))
{
     /* Display Error Message and Exit */
     printf("8601 Configure Failed !/n");
     return(-1);
}
```

6.2 Hy8601_MotorSetup

Hy8601_MotorSetup

Syntax

```
CHy8601::Hy8601_MotorSetup(    int axis,
                               int start,
                               int max,
                               int ramp ): long;
```

Parameter

`axis`	- Which motor to set up (0 ~ 3);
`start`	- The Start / Stop Speed of the motor (in Steps/Min).
`max`	- The Maximum Speed of the motor (in Steps/Min).
`ramp`	- The Acceleration / Ramp Rate of the motor (in Steps per Min per Second).

Result

0	- is returned if the card is successfully configured.
Negative value	- indicates the configure has failed.

Description

This function is used to set up the speed and acceleration parameter for a channel / motor on the 8601 IP cards. This confirms that all requested parameters are in range. Failure of either of these checks will cause a negative return. If these checks are successful, the card is configured using the passed parameters. During this configuration any IP Card access that fails will cause a negative return. On successfully writing all the parameters a return of 0 will be produced.

Error code

-8601010	IP card is not 8601
-8601011	8601 IP not configured
-8601012	axis number invalid
-8601013	start/stop speed setting invalid
-8601014	maximum speed setting invalid
-8601015	ramp rate setting invalid

Example: assume the MOTOR1 object has been created during the configuration

```
/* Configure Motor channel 3 start speed 100, max speed 400, ramp 64 */
If (MOTOR1.Hy8601_MotorSetup(3,100,400,64))
{
     printf("8601 Motor Setup Failed !/n");
     return(-1);
}
```

6.3 Hy8601_Move

Hy8601_Move

Syntax

```
CHy8601::Hy8601_Move(     int axis,
                          Long step): long;
```

Parameter

`axis` - Which motor channel to set up.

`step` - The movement of the motor (in Steps), signed for direction.

Result

0 - is returned if the card is successfully moved.

Negative value - indicates the move has failed, due to a drive fault or at relevant limit switch etc.

Description

This function is used to order a movement for any channel / motor on a 8601 IP cards. Failure to fully complete the requested number of steps (due to a drive fault or at relevant limit switch etc) will cause a negative return else a successful full move will return a 0.

Example: assume the MOTOR1 object has been created during the configuration

```
/* Command Motor channel 2 to move 100 steps in Negative Direction */
If (MOTOR1.Hy8601_Move(2,-100))
{
     /* Display Error Message and Exit */
     printf("8601 Motor Move Failed !/n");
     return(-1);
}
```

6.4 Hy8601_ReadEncoder

Hy8601_ReadEncoder

Syntax

```
CHy8601::Hy8601_ReadEncoder(  int axis,
                              unsigned long int *presentpos ): long;
```

Parameter

`axis` - Which motor channel to read

`presentpos` - The present absolute position of the motor (in Steps), signed for direction.

Result

0 - is returned if the card is successfully read.

Negative value - indicates the read has failed.

Description

This function is used to read the present position (in steps) for any channel / motor on the 8601 IP cards. The 8601 automatically switches this value to the data from an encoder if one is fitted. Failure to read will cause a negative return else a successful read will return a 0.

Example: assume the MOTOR1 object has been created during the configuration

```
/* Read Motor channel 0 Position */
unsigned long int position;
If (MOTOR1.Hy8601_Read(0,*position))
{
     /* Display Error Message and Exit */
     printf("8601 Position Read Failed !/n");
     return(-1);
}
```

6.5 Hy8601_Status

Hy8601_Status

Syntax

```
CHy8601::Hy8601_Status(  int axis,
                         unsigned short int mask,
                         unsigned short int *status): long;
```

Parameter

`axis` - Which motor channel to read.

*status - pointer to the returned status word.

mask - mask bits. Only bits set to 1 are affected. Any other bits will be read 0. Please see define below.

Define	Value	Description	Define	Value	Description
MOTOR_RESET	0x0001	reset	**MOTOR_USE_ENCODER**	0x0100	Flush TX Buffer of Com 4
MOTOR_NEGLIMIT	0x0002	Negative limit	**NOT USED**	0x0200	Reserved
MOTOR_POSILIMIT	0x0004	Positive limit	**NOT USED**	0x0400	Reserved
MOTOR_HOMELIMIT	0x0008	Home limit	**MOTOR_DIRECTION**	0x0800	Direction
MOTOR_DRIVEFAULT	0x0010	Driver fault	**MOTOR_ABORT**	0x1000	abort
MOTOR_GO	0x0020	Motor go	**MOTOR_DONE**	0x2000	Action done
MOTOR_JOG	0x0040	Free run	**MOTOR_INTERRUPT**	0x4000	Interrupt enable
MOTOR_ENCODER_DETECTED	0x0080	Encoder detected	**MOTOR_STOP_AT_HOME**	0x8000	Stop at home

Result

0 - is returned if the card is successfully read.

Negative value - indicates the read has failed.

Description

This function is used to read the present status word for any channel / motor on a 8601 IP cards. Only the bit set in the mask will be returned. Other bits are 0.

Example: assume the MOTOR1 object has been created during the configuration

```
/* Check motor channel 3 status: done bit, no driver fault*/
unsigned short int status;
unsigned short int data;
status = MOTOR1.Hy8601_Status(3, MOTOR_DONE ||
                          MOTOR_DRIVEFAULT, &data)
if(status == 0)
{
     /* Do something… */
}else
{
     /*Read error, do something else */
}
```

6.6 Hy8601 Individual Status Functions

Status functions return individual flag bit of the specified channel. These include: Done flag, Driver fault, Encoder detected, Home limit, Positive limit and Negative limit. Their prototypes are as below. All these functions have the same footprint.

Syntax

```
CHy8601::Hy8601_Done(    int axis): bool;
CHy8601::Hy8601_Driverfault(  int axis): bool;
CHy8601::Hy8601_EncoderDetected (  int axis): bool;
CHy8601::Hy8601_HomeLimit(    int axis): bool;
CHy8601::Hy8601_PosiLimit(    int axis): bool;
CHy8601::Hy8601_NegaLimit(    int axis): bool;
```

Parameter

`axis` - Which motor channel to read.

Result

false - the specified flag = 0.

true - the specified flag = 1.

If error occurs, return will be false and the error code is in the m_ErrorCode property of the class which can be retrieved by using Motor1.GetErrorCode() function. Please see example below.

Description

These functions are used to read the present status flags for any channel / motor on a 8601 IP cards. These basically give direct access to 8601 Control and Status Register.

Example: assume the MOTOR1 object has been created during the configuration

```
/* Check motor channel 3 negative limit is reached and the motor is
done*/
If (MOTOR1.Hy8601_NegaLimit(3) && MOTOR1.Hy8601_Done(3))
{
     /* Do something… */
}else
{
     If (MOTOR1.GetErrorCode() != 0)
```

```
          printf("Get status error! 0x%X \n", MOTOR1.GetErrorCode());
     else
          /* do something else */
}
```

6.7 Hy8601_Settings

Hy8601_Settings

Syntax

```
CHy8601::Hy8601_Settings(     int axis
                              unsigned short int mask,
                              unsigned short value): long;
```

Parameter

axis	- Which motor channel to set up.
value	- bit value to be set to the CSR.
mask	- mask bits. Only bits set to 1 are affected. Any other bits will be read 0. Please see CHy8601_Status for mask bit definition

Result

0	- is returned if the card is successfully read.
Negative value	- indicates the read has failed.

Description

This function is used to set special settings for any channel / motor on a 8601 IP card. The operation only affects those mask bits being set. Failure to set will cause a negative return else a successful write will return a 0.

Example: assume the MOTOR1 object has been created during the configuration

```
/* set motor axis 1 with use encoder, interrupt disable*/
If (MOTOR1.Hy8601_Settings(1, MOTOR_USE_ENCODER ||
                           MOTOR_INTERRUPT, 0x0100))
{
     /* Do something… */
}else
{
     /* do something else */
}
```

6.8 Hy8601_Settings

Some other settings can be set to the specified channel. These include: abort, reset, use encoder and stop at home. Their prototypes are as below. They all have the same footprint and are grouped here.

Syntax

```
CHy8601::Hy8601_Abort(   int axis, bool value): long;
CHy8601::Hy8601_StopAtHome(   int axis, bool value): long;
CHy8601::Hy8601_UseEncoder (  int axis, bool value): long;
CHy8601::Hy8601_Reset(   int axis, bool value): long;
CHy8601::Hy8601_Interrupt(    int axis, bool value): long;
```

Parameter

`axis`	- Which motor to set up.
Value	- true to set, false to clear

Result

0	- is returned if the card is successfully read.
Negative value	- indicates the read has failed.

Description

These functions are used to set special status for any channel / motor on a 8601 IP cards. These basically give direct access to 8601 Control and Status Register. Failure to set will cause a negative return else a successful write will return a 0.

Example: assume the MOTOR1 object has been created during the configuration

```
/* Use encoder for motor channel 3 */
If (MOTOR1.Hy8601_ UseEncoder(3, true))
{
     /* Do something… */
}else
{
     /* do something else */
}
```

7 Installation of libraries

7.1 Under Linux with IOC9010 Blade

- Create a directory some where such as /usr/local/my
- Copy the tar file (for example: HytecIPAPI-18-08-2006V001.tar) from the disk to that directory
- Extract the tar file from there by using the following command. This will create three (3) directory: include, lib and sample. Include directory contains all the header files. Lib directory contains shared libraries. Sample directory contains a sample code how to use the API functions.

  tar xvf HytecIPAPI-18-08-2006V001.tar

- Copy all files in this lib subdirectory to /usr/local/lib
- Add a line to the user .bash_profile like this:

  Export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH

  This will ensure the next boot to add the path in.
- Now we can use the sample code in the sample directory to test the hardware. Please see next section.

7.2 Under Windows

- Copy the HytecIPAPI.h, HytecIPAPI.lib and HytecIPAPI.dll to a directory on the C drive
- Start the Visual Studio development environment with the application program
- Set a new entry in the VC++ directories --- Include files to include this directory
- Do the same entry for VC++ directories --- library files
- In the project property pages, find C/C++ Command line settings. Add the HytecIPAPI library file for the link program.

  /new directory/HytecIPAPI.lib

- Now compile the application it should work

7.3 Under Linux with 5331 PCI card and VME64 Crate

Hytec5331 is a Hytec design PCI interface that can be installed on any machine that supports PCI devices. It enables a normal machine such as x86 PC, Alpha etc to communicate to the VME64 crate via VPI3331 interface.

Hytec has developed the Linux device driver for this 5331 PCI card and relative API functions to talk to the VME64 devices. The user interfaces, which have two forms of API functions and web services, are identical for either VME64

devices or IOC9010 blade. The only difference is that the vmeslot argument is ignored by IOC9010 but for VME64, it has to be the real slot where the VME carrier board is plugged in. Please note that the VME64 slot starts from number 1 and VME carrier board (Hytec developed) supports up to 4 industry pack slots, not 6 IPs in the IOC9010.

To install the 5331 Linux device driver and the API functions please follow the instructions below.

- Setup a Linux machine such as a normal x86 PC. Install the Scientific Linux 4 as the required operating system. Please refer to IOC9010_PS_iss1 rev 03-01.DOC document section Appendix A “Complete Guide to Installing Linux on the 9010 IOC Blade“ for details.
- Plug 5331 card into the Linux PC and connect the cable to the VME64 3331.
- Copy Hytec5331driver.tar file to /root directory and extract from there by using:

  tar xvf Hytec5331driver.tar

  This will create IOCBlade9010/pci directory under /root and put driver files into that directory
- Change to /root/IOCBlade9010/pci directory and do a “make”
- Load the driver by using command:

  ./IOC9010_load

  Alternatively can put this command in the .bash_profile file in order to load the driver automatically during boot
- Copy HytecIPAPI-09-11-2006V007.tar file to /usr/local directory
- Extract the tar file by using command:

  Tar xvf HytecIPAPI-18-08-2006V001.tar

  This will install all API shared libraries.
- Plug 8002 with IP cards into VME64 crate. Plug a transition card on the back of the crate and connect necessary cables.
- Try examples in the example directory as a start point. Again, these API interfaces are exactly the same as the ones used for IOC9010 except the vmeslot setting.

8 Example

Below is a simple test.cpp program which uses HytecIPAPI API functions to control a 8505 card. Obviously we need to connect the hardware to the right place (in this case, the slot C) first.

```
#include "HytecIPAPI.h"
#include "Hy8402.h"
#include "Hy85056.h"
#include "Hy8411.h"
#include "Hy8601.h"
#include "Hy85156.h"

int main()
{
	long status;

	printf("Test program started!\n");

	/*Create a 8505 object in C programming;
	CHy85056 *DIO1;
	DIO1 = CreateCHy85056();
	*/

	//Create a 8505 object in C++ version;
	CHy85056 DIO1;


	/*VME slot 0 site C As 1KHz Scan 1Khz Debounced Inputs for 8505 in C
	If(DIO1->Hy85056_Configure(0,"C",4,0xFFFF,0,0,0,0,0,0))
	*/

	//VME slot 0 site C As 1KHz Scan 1Khz Debounced Inputs for 8505 in C++
	If(DIO1.Hy85056_Configure(0,"C",4,0xFFFF,0,0,0,0,0,0))
	{
		printf("8505 Input Configure Failed !\n");
	}


	/*Read 8505 bit 3 in C
	If (DIO1->Hy85056_Read_Bit(3))
	*/

	//Read 8505 bit 3 in C++
	status = DIO1.Hy85056_Read_Bit(3);
	If (status)
	{
		printf("8505 bit 3 is high!\n");
	}else
	{
		If (DIO1.GetErrorCode == 0)
			printf("8505 bit 3 is low!\n");
```

```
            else
                  printf("Read 8505 failed! 0x%X\n", status);
      }

      /* destroy the object in C only. C++ will destroy its object automatically.
      DeleteCHy85056(DIO1);
      */
      return 0;
}
```

To complie and link the test program in Linux, the command line in the makefile looks like this (assume the shared library libHytecIPAPI.so.1.0 and libIOCGeneric.so.1.0 are in /usr/local/lib directory and this directory has been added to the LD_LIBRARY_PATH environmnet variable).

```
g++ -g –Wall –L/usr/local/lib test.c –lHytecIPAPI –lIOCGeneric –o test
```

For more detail examples, please see demo_use.c program come with the API function.

- Use an text editor to open the demo_use.c file in the sample subdirectory created during the library installation.
- There are 6 sections in the software each section is associated to one type of IP card.
- Uncomment the section you want to test with the particular IP card
- Check the IP card is plugged in the correct slot as it defined in its section. Otherwise you can modify the settings in the example to match the hardware.
- Ensure signal is connected or the device is connected.
- Save any changes
- run ./makedemo to make the file.
- run ./demo_use

It should run a test and display the results. If there are errors, check this manual for error code.

9 Hytec Industry Pack Web Service Interface

The Hytec Industry Pack web service interface is designed for applications who want to remote access the IP cards either on the IOC9010 or 5992/3331 VME system via internet over HTTP, SOAP and XML.

The Hytec IOC9010 web server uses this web service as the main interface talking to the IOC9010 hardware across the platforms.

The web service provides a series of interfaces that can get/set parameters from/to the carrier board and can also get/set parameters from/to each individual IP cards plugged in.

Because of the nature of the internet operation, it is not intended to use the web service for a real time control system which requires time critical controls.

9.1 Generic Data Class Definition

BasicData class:

This class define four generic data types that would be used by all members. This is the very base class. All the subsequent classes are ingerited from this.

Properties:

Name	Type	Description
uintValue	USHORT	This property stores a value which type is unsigned short integer.
boolValue	bool	This property stores a value which type is boolean.
stringValue	String	This property stores a value which type is string.
longValue	Long	This property stores a value which type is long.

Response class members:

This class inherits from the BasicData base class plus an error code property described below. It is used for all web services to store the response values of the call.

Properties:

Name	Type	Description
uintValue	USHORT	This property stores a returned value which type is unsigned short integer.
boolValue	bool	This property stores a returned value which type is boolean.
stringValue	String	This property stores a returned value which type is string.
longValue	Long	This property stores a value which type is long.
errorCode	Long	Operation error code. If success, returns 0. Otherwise returns a negative value. Please refer to API function error code definitions.

Request class members:

This object contains parameters passed to the web service. It also inherits from the BasicData base class plus other properties such as vmeslot defines the VME slot number; ipslot defines the IP card label; channel property defines which signal channel want to operate; port property defines which port is the channel belong to etc. Please see definition below. When pass output value to a web service, either uintValue, boolValue, stringValue or longValue will be used depend on the IP type. For example, if output to a DAC channel, use uintValue property; if use serial port to send, use stringValue property.

Properties:

Name	Type	Description
vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot ‘A’ to ‘F’

channel	int	The signal channel to operate.
port	int	The port number to operate.
uintValue	USHORT	This property stores value of unsigned short integer to be passed to the hardware.
boolValue	bool	This property stores value of boolean to be passed to the hardware.
stringValue	String	This property stores value of string to be passed to the hardware.
longValue	long	This property stores value of long to be passed to the hardware.

9.2 Search IP Cards Installed and Get Their ID PROM Information

---

Hy9010_SearchIP

Syntax

[WebMethod]

Public ResponseIDs Hy9010_SearchIP();

Parameter

None.

Result

ResponseIDs - This is a responseIDs object inherited from a generic Response class defined in previous section.

In this web service, only the error code property is used from the generic Response object.

The ResponseIDs object contains extra 6 elements of Identity objects apart from those defined in the generic Response object. Each identity object stores one IP card ID PROM information. The Identity class is also defined below.

ResponseIDs class members:

Because it inherits the generic Response class described above, it has all properties of the generic response object plus other properties defined below.

Properties: (only list used properties)

Name	Type	Description
IPAIdentity	Identity	This is an identity object stores slot 'A' IP card's ID PROM information defined below.
IPBIdentity	Identity	This is an identity object stores slot 'B' IP card's ID PROM information defined below.
IPCIdentity	Identity	This is an identity object stores slot 'C' IP card's ID PROM information defined below.
IPDIdentity	Identity	This is an identity object stores slot 'D' IP card's ID PROM information defined below.
IPEIdentity	Identity	This is an identity object stores slot 'E' IP card's ID PROM information defined below.
IPFIdentity	Identity	This is an identity object stores slot 'F' IP card's ID PROM information defined below.
errorCode	Long	Operation error code. If success, returns 0. Otherwise returns a negative value. Please refer to API function error code definitions.

Identity class members:

Properties:

Name	Type	Description
VITA4	String	12 chars. VITA4 spec header. The valid value should always be “564954413420”.
CompanyCode	String	8 chars. Company code. For Hytec the value is “00800300”.
ModelNumber	String	4 chars. IP card model such as “8402”, “8515” etc.
Revision	String	4 chars. Firmware revision.
Reserved	String	4 chars. For Hytec cards the value is “0000”.
DriverID	String	8 chars. Driver ID.
Flags	String	4 chars. Bit1=1, support 8MHz; Bit1=0 not support. Bit2=1 support 32MHz, bit2=0 not.
NofBytes	String	4 chars. Number of bytes used.
SerialNumber	String	4 chars. Product serial number.

Methods:

None

Description

This web service sends a query to the remote IOC9010 asking for ID PROM information about the IP cards installed. If success it returns a 6 element array of Identity object in the ResponseIDs object. If an error happens, the error code stores in the errorCode property. Each identity object contains the ID PROM information of each IP card. The first object is associated to slot “A” IP, the second to slot “B” and so forth. If no IP installed in one slot, it still returns the identity object but with all properties set to “0”.

Example:

9.3 Get Carrier Board Environment Parameters

Hy9010_ReadEnv

This web service only applies to IOC9010 blade.

Syntax

[WebMethod]

Public ResponseEnv Hy9010_ReadEnv();

Parameter

None.

Result

ResponseEnv - This is an ResponseEnv object inherited form the generic response class. It has generic response class properties plus 3 measurement objects (Fans, Temps and Switches) associating to the carrier board fan speeds, temperatures and front panel switch status. These object classes are defined as below. In the generic response object part, only errorCode property is used to indicate either the operation is successful or not.

ResponseEnv class members:

Properties: (only the used ones are listed)

Name	Type	Description
errorCode	long	If success, errorCode=0. Otherwise is an error please referring to API definition.
Fans	Fans object	This object contains 6 fan speed reading of the carrier board. See definition below.
Temps	Temps object	This object contains 5 temperature measurement of the carrier board. See below.
Switches	Switches object	This object contains 4 push button status of the carrier board front panel. See below.

Fans class members:

Properties:

Name	Type	Description
Fan1	Int	Number 1 fan speed measurement.
Fan2	Int	Number 2 fan speed measurement.

Fan3	int	Number 3 fan speed measurement.
Fan4	Int	Number 4 fan speed measurement.
Fan5	Int	Number 5 fan speed measurement.
Fan6	Int	Number 6 fan speed measurement.

Temps class members:

Properties:

Name	Type	Description
Temp1	Int	Number 1 temperature sensor reading.
Temp2	Int	Number 2 temperature sensor reading.
Temp3	int	Number 3 temperature sensor reading.
Temp4	Int	Number 4 temperature sensor reading.
Temp5	Int	Number 5 temperature sensor reading.

Swithes class members:

Properties:

Name	Type	Description
Switch1	Bool	UP push button.
Switch2	Bool	OK push button.
Switch3	Bool	DOWN push button.
Switch4	Bool	RESET push button.

Description

This web service asks current environment parameters of the IOC9010 carrier board. It returns 6 fan speeds, 5 temperatures and 4 switch status measurements if successful. The readings are stored in associated objects defined above.

Example:

9.4 Set Carrier Board Fan Speed

Hy9010_SetFanSpeed

This web service only applies to IOC9010 blade.

Syntax

[WebMethod]

Public Response setFanSpeed(Request req);

Parameter

req - is the generic Request object defined in section 8.1.

In this web service, the channel property specifies which fan to set. The valid fan number is 1 ~ 6. If the channel value is set to 0, it means to set automatic control for all of the fans back to the carrier board.

The speed setting is in the uintValue property. The value can either be 0, 1, 2. Value=0 means stop; Value=1 set the fan to high speed (about 7000 RPM); value=2 set to low speed (about 5000 RMP). If channel setting is 0 (to auto), this field is ignored.

Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
channel	int	Which fan to set. Valid value 0 ~ 6. If the channel value is 0, set fan control to auto.
uintValue	USHORT	This property stores fan speed setting.

Result

Response - a generic response object that contains the status. Only the error code property is used.

Description

This web service sets a fan speed specified by the channel property of the Request object. It sets one fan speed at a time. If the channel setting is set to 0, the service sets all the fan speed control to automatic.

Example:

9.5 Configure 8411

Hy8411_Configure

Syntax

[WebMethod]

Public Response Hy8411_Configure(Request8411 req);

Parameter

req - is a Request8411 object that inherits from the generic Request object (defined in 8.3 section)plus other properties defined below. The only properties used are listed below. Other properties are not used.

Request8411 object members:

Properties: (only the used ones are listed)

Name	Type	Description
vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
clockrate	Int	Sampling Clock Rate. See definition below or section 2.4 Hy8411_configure

Definition:

clockrate Mapping…

Clockrate Value	Frequency	Clockrate Value	Frequency
0	1 Hz	8	500 Hz
1	2 Hz	9	1 KHz
2	5 Hz	10	2 KHz
3	10 Hz	11	5 KHz
4	20 Hz	12	10 KHz
5	50 Hz	13	20 KHz
6	100 Hz	14	50 KHz
7	200 Hz	15	100 KHz

Result

Response - only the error code property is used.

Description

This web service is used to configure the 8411 IP card. This configuration has to be done before any IO operations.

Example:

9.6 Read a Channel Value From 8411

Hy8411_Read

Syntax

[WebMethod]

Public Response Hy8411_Read(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used (mandate) are listed below. Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
channel	int	The signal channel to read.

Result

Response - if success, ADC value stores in the uintValue property.

- if errors, error code in errorCode property.

Other properties have no meanings.

Description

This web service reads the specified ADC channel signal and stores the result in the uintValue property of the Response object.

Example:

9.7 Configure 8402

Hy8402_Configure

Syntax

[WebMethod]

Public Response Hy8402_Configure(Request8402 req);

Parameter

req - is a Request8402 object that is inherited from the Request8411 object (defined in 8.4 section) plus other properties defined below. The only properties used are listed below. Other properties are not used.

Request8402 object members:

Properties: (only the used ones are listed)

Name	Type	Description
vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
vectornum	int	Interrupt Vector for this IP Card (Interrupt Vector 0=find me one).
doram	int	0 (Output via Registers) or 1 (Output via RAM & Set to passed ClockRate).
clockSource	int	0 (internal) or 1(external).
clockRate	Int	Sampling Clock Rate. See definition below or see section 2.1 Hy8402_configure
inhibit	int	0 (inhibit disabled) or 1 (inhibit enabled).

Definition:

clockRate Mapping…

Clockrate Value	Frequency
0	1 Hz
1	2 Hz
2	5 Hz
3	10 Hz
4	20 Hz
5	50 Hz

6	100 Hz
7	200 Hz
8	500 Hz
9	1 KHz
10	2 KHz
11	5 KHz
12	10 KHz
13	20 KHz

Result

Response - only the error code property is used.

Description

This web service is used to configure the 8402 IP card. This configuration has to be done before any IO operations.

Example:

9.8 Update a DAC Channel Value of the 8402

Hy8402_Write

Syntax

[WebMethod]

Public Response Hy8402_Write(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used (mandate) are listed below. Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
channel	int	The signal channel to write.
uintValue	USHORT	This property stores value of unsigned short integer to be passed to the hardware.

Result

Response - if success, errorCode property is 0. If error happens, error code in errorCode property.

Other properties have no meanings.

Description

This web service updates the specified DAC channel to the specified value passed by the uintValue property of the Request object.

Example:

9.9 Configure 8505/8506

Hy85056_Configure

Syntax

[WebMethod]

Public Response Hy85056_Configure(Request85056 req);

Parameter

req - is a Request85056 object that is inherited from the generic Request object (defined in 8.1 section) plus other properties defined below. The only properties used are listed below. Other properties are not used.

Request85056 class members:

Properties: (only the used ones are listed)

Name	Type	Description
vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
debrate	int	Debounce Rate (0 = 100Hz, 1 = 200Hz, 2 = 500Hz, 3 = 1kHz).
debmask	USHORT	Debounce Bit Mask, select input bits to debounce (0x0000 - 0xFFFF).
scanrate	int	input scan rate (0 = 1kHz, 1 = 10kHz, 2 = 100kHz, 3 = 1MHz).
dir	int	The direction of the I/O. Please see definition below or section 3.1.
vectonum	USHORT	Interrupt Vector for this IP Card (Interrupt Vector 0=find me one).
intmask	USHORT	Interrupt Bit Mask, select which bits to generate interrupts (0x0000 - 0xFFFF).
pwidth	int	Pulse width. Please see definition below or section 3.1.
pmask	USHORT	Pulse Output Bit Mask, select bits to pulse on output (0x0000 - 0xFFFF).
clock	int	0 = internal, 1 = external.
portnum	int	Which Digital Channel / Port (0-2 for 8506 only). For 8505, it must be 100.

Definitions:

dir - The direction of the I/O…

0 = All 16 bits inputs.

1 = lower 8 bits outputs / high 8 bits inputs (8505 Only).

2 = lower 8 bits inputs / high 8 bits outputs (8505 Only).

3 = All 16 bits outputs.

Bits 2 & 3 (values 4 and 8, respectively) encode whether the input and output values should be inverted.

(dir & 4) == 4 => Invert all input bits

(dir & 8) == 8 => Invert all output bits

`pwidth` - pulse width

0 = 1 msec
1 = 10 msec
2 = 100 msec
3 = 1 sec
4 = 2 sec
5 = 5 sec
6 = 10 sec
7 = 20 sec
8 = 50 sec
9 = 100 sec

Result

Response - only the error code property is used.

Description

This web service is used to configure the 8505 or 8506 IP card. This configuration has to be done before any IO operations.

Example:

9.10 Update 8505/8506 Digital I/O Channels

Hy85056_Write

Syntax

[WebMethod]

Public Response Hy85056_Write(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used are listed below. Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
channel	int	The signal channel to write. This is the bit mask defines which bits will be written.
port	int	The port number to operate. For 8505, it must be 100, for 8506, it could be 0, 1 or 2.
uintValue	USHORT	This property stores value of unsigned short integer to be passed to the hardware.

Result

Response - if success, errorCode property is 0. If error happens, error code in errorCode property. Other properties have no meanings.

Description

This web service updates the bits defined by the bit mask on the specified port. If it is 8505 IP, the port number must be 0. If the IP is 8506, the port number could either be 0, 1 or 2.

Example:

9.11 Read 8505/8506 Digital I/O Channel Status

Hy85056_Read

Syntax

[WebMethod]

Public Response Hy85056_Read(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used are listed below.
Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
port	int	The port number to operate. For 8505, it must be 0, for 8506, it could be 0, 1 or 2.

Result

Response - The 16 bit values of the specified port are returned in uintValue property if success, and also the errorCode property is 0. If error happens, error code in errorCode property.
Other properties have no meanings.

Description

This web service reads the specified I/O port 16 bit status. If it is 8505 IP, the port number must be 0. If the IP is 8506, the port number could either be 0, 1 or 2.

Example:

9.12 Configure 8601

Hy8601_Configure

Syntax

[WebMethod]

Public Response Hy8601_Configure(Request8601 req);

Parameter

req - is a Request8601 object that is inherited from the generic Request object (defined in 8.1 section) plus other properties defined below. The only properties used are listed below. Other properties are not used.

Request8601 class members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
axis	int	Which motor to set up (0 ~ 3).
start	USHORT	The Start / Stop Speed of the motor (in Steps/Min).
max	int	The Maximum Speed of the motor (in Steps/Min).
ramp	int	The Maximum Speed of the motor (in Steps/Min).

Result

Response - only the error code property is used.

Description

This web service is used to configure the 8601 IP card. This configuration has to be done before any IO operations.

Example:

9.13 Command 8601 Axis To Move

Hy8601_Move

Syntax

[WebMethod]

Public Response Hy8601_Move(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used are listed below. Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
port	int	The axis number of the 8601 to operate. Value 0 ~ 3 represent the 4 motor channels.
longValue	long	The steps to move. Positive value moves one direction. Negative moves the opposite.

Result

Response - if success, errorCode property is 0. If error happens, error code in errorCode property. Other properties have no meanings.

Description

This web service commands the specified motor to move steps specified in the longValue property. If the longValue property has positive value, it moves to one direction as defined by the user. Negative value causes the motor move to the opposite direction.

Example:

9.14 Get 8601 Motor Status

Hy8601_Status

Syntax

[WebMethod]

Public Response Hy8601_Status(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used are listed below.

Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
port	int	The axis number of the 8601 to operate. Value 0 ~ 3 represent the 4 motor channels.

Result

Response8601 - This object returns all status of the specified axis plus its encoder position readings. Please see below definition. This object inherits from the generic response object. If success, the errorCode property is 0. If error happens, error code in errorCode property.

Other properties have no meanings.

Response8601 class members:

Properties: (only the used ones are listed)

Name	Type	Description
errorCode	long	If success, errorCode=0. Otherwise is an error please referring to API definition.
encoder	long	Returns the absolute position of the motor.
done	bool	True if the movement is done. False otherwise.
driverFault	bool	True if the driver has been detected faulty. False otherwise.
encoderDetected	bool	True if there is an encoder detected. False otherwise.
homeLimit	bool	True if the motor reaches the home limit. False otherwise.

posiLimit	bool	True if the motor reaches the positive limit. False otherwise.
negaLimit	bool	True the motor reaches the negative limit. False otherwise.

Description

This web service returns the specified motor readings including encoder value, driver faulty flag, movement done flag etc.

Example:

9.15 Set 8601

Hy8601_Settings

Syntax

[WebMethod]

Public Response Hy8601_Settings(Request8601Settings req);

Parameter

req - is inherited from the generic Request object as defined in 8.1 section. Properties used are listed below.

Other properties are not used.

Request8601Settings class members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	Int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
port	int	The axis number of the 8601 to operate. Value 0 ~ 3 represent the 4 motor channels.
abort	bool	True to command the motor to abort.
stopAtHome	bool	True to command the motor to stop at home.
useEncoder	bool	True to instruct the motor absolute position register to count encoder.
reset	bool	True to reset the motor.
interrupt	bool	True to enable interrupt.

Result

Response - If success, errorCode property is 0. If error happens, error code in errorCode property.

Other properties have no meanings.

Description

This web service commands the specified motor to do certain actions according to the property settings listed above.

Example:

9.16 Configure 85156

Hy85156_Configure

Syntax

[WebMethod]

Public Response Hy85156_Configure(Request85156 req);

Parameter

req - is a Request85156 object that is inherited from the generic Request object (defined in 8.1 section) plus other properties defined below. The only properties used are listed below. Other properties are not used.

Request85156 class members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
vector	Int	The interrupt vector to assign this card. Only the lower 8 bits are used as of the interrupt vector register.
delay	int	+ Values interval between successive interrupts, - Values the depth of the interrupt buffer. Valid value is -64 ~ 32000.
halfduplex	int	Whether the card is to run in full or half duplex mode(0=Full Duplex, 1=Half Duplex with echo, 2=Half duplex without echo).
delay485	int	The delay in bit length after the last character to switch RS485 mode (valid values are 0 ~ 15).
port	int	The port number to configure. Valid value: 0 ~ 7.
baud	ULONG	Baud rate of the serial port. Valid values are: 100, 400, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200, 153600, 230400, 460800, 921600.
parity	int	Parity setting. 0 for no parity check; 1 for odd parity check; 2 for even parity check.
stopbit	int	Stop bit. The valid value can be: 1 or 4. Please see definition below.
databits	int	Data bits. Valid value is 5 ~ 8.
flowControl	int	Flow control. Only lower 8 bits are used. Please see definition below.

Definitions:

`stopbits` - The number of stop bits required.

1	1 stop bit
4	1 1/2 stop bit if data bit = 5
4	2 stop bit if data bit = 6,7,8

`flowctrl` - A character indicating the required flow control.

```
   Bit7 6 5 4 3 2 1 0
      0 0 X X X X X X
      [_] | | [_____]
       |  | |    |
 no used  | |    software flow control
          | |    0 0 0 0      no software flow control
          | |    1 1 1 1      transmit Xon1 and Xon2 / Xoff1 and Xoff2
          | |                 and receiver compares  Xon1 and Xon2 / Xoff1
          | |                 and Xoff2
          | =0 disable hardware flow control, =1 enable

          =0 use RTS/CTS, =1 use DTR/DSR pair
```

Result

Response - only the error code property is used.

Description

This web service is used to configure the 8515/8516 IP cards. Please note, vector, delay, halfduplex and delay485 properties are applied to all of the 8 ports. This configuration has to be done before any IO operations.

Example:

9.17 Transmmit a String To the Specified Port on 8515/8516

Hy85156_Send

Syntax

[WebMethod]

Public Response Hy85156_Send(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used are listed below. Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
port	int	The serial port to send. Can only be 0 ~ 7 represent COM port 1 ~ COM port 8.
stringValue	string	The string to be sent.

Result

Response - If success, errorCode property is 0. If error happens, error code in errorCode property. Other properties have no meanings.

Description

This web service transmits a string to the specified port on the 8515/8516. In theory, there is no limit to the length of the string, but it depends on the operating system specification for string definition.

Example:

9.18 Receive a String From the Specified Port of 8515/8516

Hy85156_Receive

Syntax

[WebMethod]

Public Response Hy85156_Receive(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used are listed below. Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
port	int	The port from which the string is received. Valid value 0 ~ 7 represent 8 ports.

Result

Response - If success, received string is in stringValue property of the response object. If COM buffer is empty, return string is empty.

Description

This web service returns a string received from the specified serial port on the 8515/8516 IP card.

Example:

9.19 Flush 8515/8516 Serial Port Send/Receive Buffers

Hy85156_Flush

Syntax

[WebMethod]

Public Response Hy85156_Flush(Request req);

Parameter

req - is the generic Request object as defined in 8.1 section. Properties used are listed below. Other properties are not used.

Request object members:

Properties: (only the used ones are listed)

Name	Type	Description
Vmeslot	Int	VME slot number. Not applicable to carrier board settings, nor for the IOC9010 blade.
Ipslot	int	IP slot. Not applicable to carrier board settings. Value = 0~5 represent slot 'A' to 'F'
uintValue	USHORT	To specify the port buffers to flush. Please see definition below.

Definition:

Bit0: =1, do port0 TX buffer flush; =0, do not flush.

Bit1: =1, do port0 RX buffer flush; =0, do not flush.

Bit2: =1, do port1 TX buffer flush; =0, do not flush.

Bit3: =1, do port1 RX buffer flush; =0, do not flush.

……

Bit14: =1, do port7 TX buffer flush; =0, do not flush.

Bit15: =1, do port7 RX buffer flush; =0, do not flush.

Result

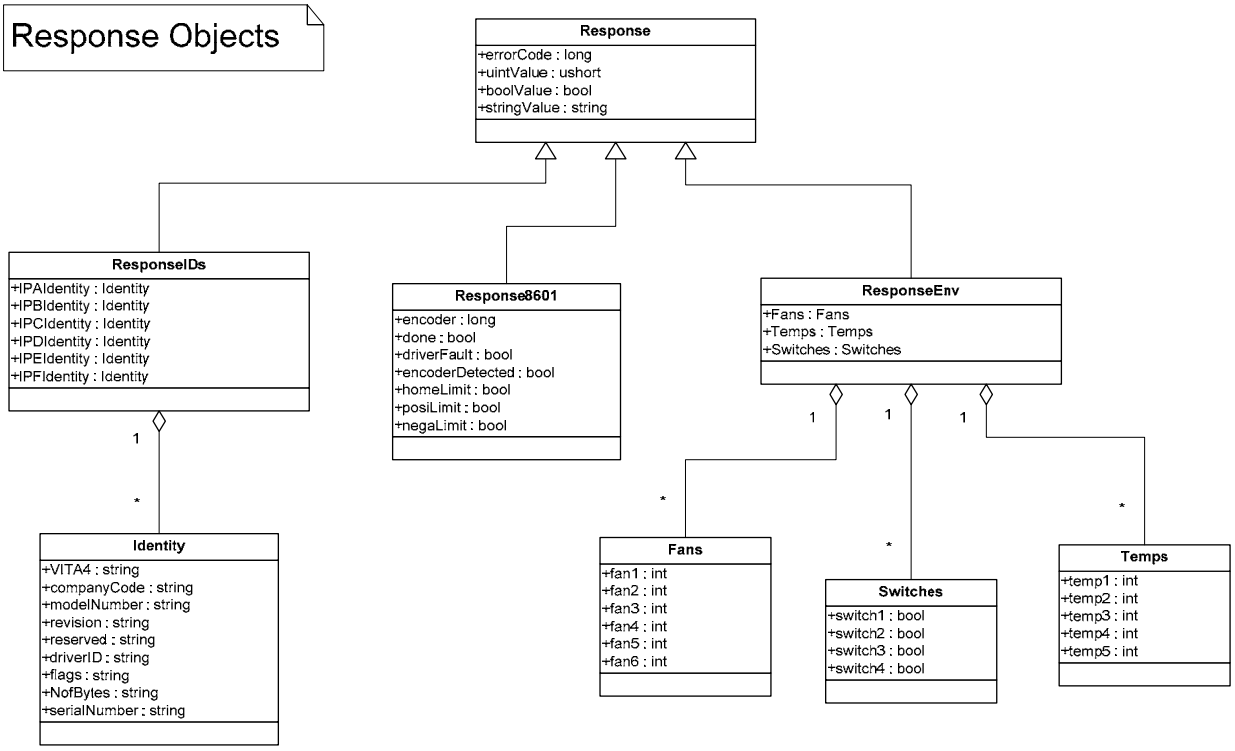
Response - If success, errorCode property is 0. If error happens, error code in errorCode property. Other properties have no meanings.

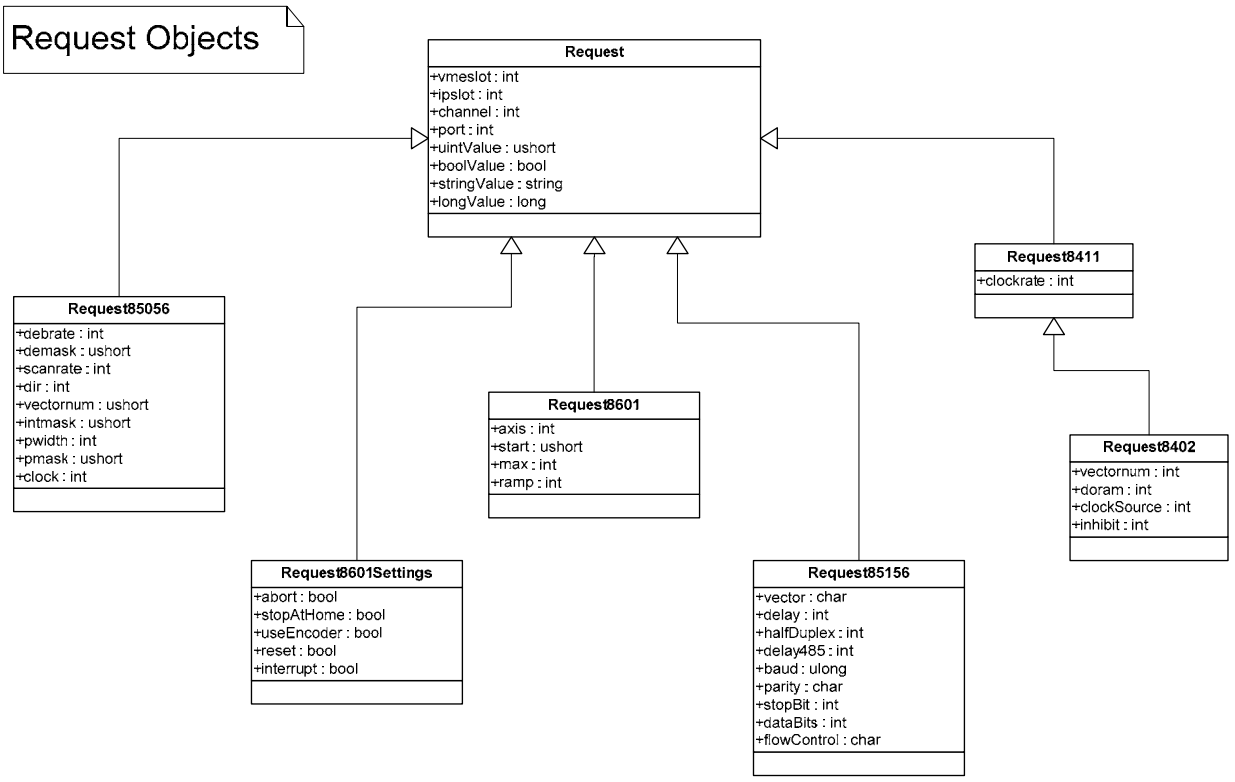
Description

This web service commands the specified motor to do certain actions according to the property settings listed above.

Example:

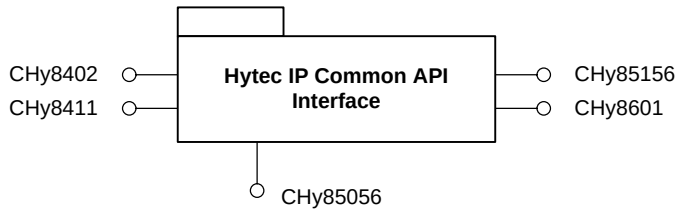
Web service domain models:





Appendix

The interface of the Hytec IP Common API function model summary.



Each individual class is defined as below.

CHy8402
-InitVMESlot : int = 100 -InitIPSlot : int = 100 -Handler : int
+Hy8402_Configure(in vmeslot : int, in ipslot : char, in vectornum : int, in doram : int, in clockSource : int, in clockRate : int, in inhibit : int) : long +Hy8402_Write(in signal : int, in value : int) : long +Hy8402_WaveForm(in signal : int, in *value : char, in length : int) : long +CHy8402() +~CHy8402()

CHy8411
-InitVMESlot : int = 100 -InitIPSlot : int = 100 -Handler : int
+Hy8411_Configure(in vmeslot : int, in ipslot : char, in clockRate : int) : long +Hy8411_Read(in signal : int, inout *value : int) : long +Hy8411() +~Hy8411()

CHy85056
-InitVMESlot : int = 100 -InitIPSlot : int = 100 -Handler : int
+Hy85056_Configure(in vmeslot : int, in ipslot : char, in debrate : int, in debmask : int, in dir : int, in vectornum : int, in intmask : int, in pwidth : int, in pmask : int, in clock : int) : long +Hy85056_Configure(in vmeslot : int, in ipslot : char, in portnum : int, in debrate : int, in debmask : int, in dir : int, in vectornum : int, in intmask : int, in pwidth : int, in pmask : int, in clock : int) : long +Hy85056_Write(in mask : unsigned int, in value : unsigned int) : long +Hy85056_Write(in portnum : int, in mask : unsigned int, in value : unsigned int) : long +Hy85056_Write_Bit(in Bit : unsigned int, in value : unsigned int) : long +Hy85056_Write_Bit(in portnum : int, in Bit : unsigned int, in value : unsigned int) : long +Hy85056_Read(in &value : unsigned int) : long +Hy85056_Read(in portnum : int, in &value : unsigned int) : long +Hy85056_Read_Bit(in Bit : int) : bool +Hy85056_Read_Bit(in portnum : int, in Bit : int) : bool +CHy85056() +~CHy85056()

CHy85156
-InitVMESlot : int = 100 -InitIPSlot : int = 100 -Handler : int
+Hy85156_Configure(in vmeslot : int, in ipslot : char, in vector : int, in delay : int, in halfduplex : int, in delay485 : int) : long +Hy85156_ComsSetup(in port : int, inout baud : int, in parity : int, in numstopbits : int, in numdatabits : int, in flowctr : char) : long +Hy85156_Write(in port : int, in *value : char, in length : int) : long +Hy85156_Read(in port : int, inout *value : char) : int +Hy85156_Flush(in port : int, in TX : bool, in RX : bool) : long +Hy85156_Flush(in mask : unsigned int) : long +Hy85156() +~Hy85156()

CHy8601
-InitVMESlot : int = 100 -InitIPSlot : int = 100 -Handler : int
+Hy8601_Configure(in vmeslot : int, in ipslot : char) : long +Hy8601_MotorSetup(in channel : int, inout start : int, in max : int, in ramp : int) : long +Hy8601_Move(in channel : int, inout step : long) : long +Hy8601_ReadEncoder(in channel : int, inout *position : long) : long +Hy8601_Status(in channel : int, in mask : unsigned int, in *status : unsigned int) : bool +Hy8601_Done(in channel : int) : bool +Hy8601_DriverFault(in channel : int) : bool +Hy8601_EncoderDetected(in channel : int) : bool +Hy8601_HomeLimit(in channel : int) : bool +Hy8601_PosiLimit(in channel : int) : bool +Hy8601_NegaLimit(in channel : int) : bool +Hy8601_Settings(in channel : int, in mask : unsigned int, in value : unsigned int) : bool +Hy8601_Abort(in channel : int, in value : bool) : long +Hy8601_StopAtHome(in channel : int, in value : bool) : long +Hy8601_UseEncoder(in channel : int, in value : bool) : long +Hy8601_Reset(in channel : int, in value : bool) : long +Hy8601_Interrupt(in channel : int, in value : bool) : long +Hy8601() +~Hy8601()

Note: CHy8402, CHy8411, CHy85056, CHy85156 and CHy8601 are class names

- in front of any variables or functions are private

+ in front of any variables or functions are public