

SeaIO API Reference Manual

2.0

Generated by Doxygen 1.4.6

Fri Aug 18 13:07:33 2006

Contents

1	SeaIO API Class Documentation	1
1.1	DigitalIoDevice Class Reference	1
1.2	tagADAPTER_INFO Struct Reference	11
1.3	tagADAPTER_STATE Struct Reference	14
2	SeaIO API File Documentation	17
2.1	include/digitalIoDevice.h File Reference	17
2.2	include/seaio.h File Reference	18
2.3	include/seaiostruct.h File Reference	29
3	SeaIO API Example Documentation	35
3.1	example.c	35
3.2	example.cpp	39

Chapter 1

SeaIO API Class Documentation

1.1 DigitalIoDevice Class Reference

A SeaIO API wrapper class.

```
#include <digitalIoDevice.h>
```

Public Member Functions

- [DigitalIoDevice](#) (unsigned short int deviceNumber=0)
This is the [DigitalIoDevice](#) constructor.
- int [open](#) (unsigned short int deviceNumber=0)
Open a different SeaIO Device.
- int [close](#) ()
*Close this device, **MUST** be called before program termination.*
- int [readByte](#) (unsigned long port, unsigned char &data, enum [AccessMode](#) mode)
Read a byte from a particular port.
- int [writeByte](#) (unsigned long port, const unsigned char data, enum [AccessMode](#) mode)
Write a byte to a particular port.

- int [readBit](#) (unsigned long bitNumber, BOOLEAN &bitState, enum [Access-Mode](#) mode)
Read a single bit.
- int [writeBit](#) (unsigned long bitNumber, const BOOLEAN bitState, enum [Access-Mode](#) mode)
Write a single bit.
- int [readAnalog](#) (unsigned long channel, unsigned long &data)
Read an A/D channel value.
- int [writeAnalog](#) (unsigned long channel, unsigned long data)
Write to a DAC.
- int [getInfo](#) ([ADAPTER_INFO](#) &adapterInfo)
Get adapter information.
- int [getState](#) ([ADAPTER_STATE](#) &adapterState)
Get adapter state information (only valid on PIO and A/D models).
- int [setState](#) ([ADAPTER_STATE](#) &adapterState)
Set an adapter's state (only valid on PIO and A/D models).
- int [getType](#) ([SeaIo_CardType](#) &cardType)
Return card classification.
- int [getHandle](#) ()
Handle to the current device.
- int [readPort](#) (unsigned char portNumber, unsigned char &value)
OBSOLETE.
- int [writePort](#) (unsigned char portNumber, unsigned char value)
OBSOLETE.
- int [readRelative](#) (unsigned char portNumber, unsigned char &value)
OBSOLETE.
- int [readAllInputs](#) (unsigned char &buffer, unsigned long &bufferLength)
OBSOLETE.
- int [writeRelative](#) (unsigned char portNumber, unsigned char value)
OBSOLETE.

- int [closeRelay](#) (unsigned short int relayNumber)
OBSOLETE.
- int [openRelay](#) (unsigned short int relayNumber)
OBSOLETE.

Public Attributes

- int [m_openStatus](#)
Used internally to see if a device is already open.

1.1.1 Detailed Description

A SeaIO API wrapper class.

This class is simply a wrapper for the existing SeaIO API. This class is explicitly used in SeaIOTst utility found in the examples directory.

Examples:

[example.cpp](#).

1.1.2 Constructor & Destructor Documentation

1.1.2.1 DigitalIoDevice::DigitalIoDevice (unsigned short int *deviceNumber* = 0)

This is the [DigitalIoDevice](#) constructor.

It takes one argument, the device number that the class should attach itself to initially. Default is 0.

Parameters:

deviceNumber integer number of device

1.1.3 Member Function Documentation

1.1.3.1 int DigitalIoDevice::open (unsigned short int *deviceNumber* = 0)

Open a different SeaIO Device.

Devices are numbered in order of model number starting with 0. If more than one of a particular model are present, they will enumerate in the order they are found.

Parameters:

deviceNumber integer number of device

Returns:

int SEAIO_ERROR_NONE on success; error code otherwise

Examples:

[example.cpp](#).

1.1.3.2 int DigitalIoDevice::close ()

Close this device, **MUST** be called before program termination.

Lets go of the current device handle. If you don't close the device, the driver will believe the device is still in use.

Returns:

int SEAIO_ERROR_NONE on success; error code otherwise

Examples:

[example.cpp](#).

1.1.3.3 int DigitalIoDevice::readByte (unsigned long *port*, unsigned char & *data*, enum [AccessMode](#) *mode*)

Read a byte from a particular port.

Relative will read the nth input. Absolute will read the nth port. Note: there's no protection, since reading is a safe operation, to prevent reading write only registers; you just won't receive any useful data.

Parameters:

port the port or input number you want to read

data a byte in which to store the read data

mode Absolute or Relative(inputs only)

See also:

[AccessMode](#)

Returns:

SEAIO_ERROR_NONE on success; error code otherwise

Examples:

[example.cpp](#).

1.1.3.4 int DigitalIoDevice::writeByte (unsigned long *port*, const unsigned char *data*, enum [AccessMode](#) *mode*)

Write a byte to a particular port.

Relative will write the nth output. Absolute will write the nth register (if it can). Note: built-in port protection causes this to fail if a port is not configured for output.

Parameters:

port the port or output to write to
data the data byte to write
mode Absolute or Relative(outputs only)

See also:

[AccessMode](#)

Returns:

SEAI0_ERROR_NONE on success; error code otherwise

Examples:

[example.cpp](#).

1.1.3.5 int DigitalIoDevice::readBit (unsigned long *bitNumber*, BOOLEAN & *bitState*, enum [AccessMode](#) *mode*)

Read a single bit.

In Absolute [AccessMode](#) the *bitNumber* will be *dataBit* + (*Port* * 8); in Relative [AccessMode](#) the *bitNumber* will be *dataBit* + (*input* * 8). *Port* refers to the register layout described in the device manual. The *input* number refers to the nth port configured as an input.

Parameters:

bitNumber the nth bit to read
bitState BOOLEAN into which the current bit state is read
mode Absolute or Relative(inputs only)

See also:

[AccessMode](#)

Returns:

int SEAI0_ERROR_NONE on success; error code of failure

Examples:

[example.cpp](#).

1.1.3.6 `int DigitalIoDevice::writeBit (unsigned long bitNumber, const BOOLEAN bitState, enum AccessMode mode)`

Write a single bit.

In Absolute `AccessMode` the `bitNumber` will be `dataBit + (Port * 8)`; in Relative `AccessMode` the `bitNumber` will be `dataBit + (output * 8)`. Port refers to the register layout described in the device manual. The output number refers to the `n`th port configured as an output.

Parameters:

bitNumber the `n`th bit to write

bitState BOOLEAN value to write (TRUE or FALSE)

mode Absolute or Relative(outputs only)

See also:

[AccessMode](#)

Returns:

int SEAIO_ERROR_NONE on success; error code of failure

Examples:

[example.cpp](#).

1.1.3.7 `int DigitalIoDevice::readAnalog (unsigned long channel, unsigned long & data)`

Read an A/D channel value.

12 bits accuracy. Only works on A/D capable cards. Calling on incompatible cards returns no data.

Parameters:

channel the channel number to read

data the read data will go here

Returns:

int SEAIO_ERROR_NONE on success; error code of failure

Examples:

[example.cpp](#).

1.1.3.8 int DigitalIoDevice::writeAnalog (unsigned long *channel*, unsigned long *data*)

Write to a DAC.

Write 12 bits to jumper set DAC. Only works on D/A capable cards. Calling on incompatible cards does not write any data.

Parameters:

channel the channel to write to

data the data to write to the channel (0x000-0xFFFF)

Examples:

[example.cpp](#).

1.1.3.9 int DigitalIoDevice::getInfo ([ADAPTER_INFO](#) & *adapterInfo*)

Get adapter information.

This function reads the current ADAPTER_INFO structure from the driver. It is also now possible to see which ports are configured to input and output.

Parameters:

adapterInfo struct into which all the device data is stored

See also:

[ADAPTER_INFO](#)

Returns:

int SEAIO_ERROR_NONE on success; error code of failure

Examples:

[example.cpp](#).

1.1.3.10 int DigitalIoDevice::getState ([ADAPTER_STATE](#) & *adapterState*)

Get adapter state information (only valid on PIO and A/D models).

Fills in the current Control Words used and A/D ranges. If this function is called in conjunction with a non PIO or A/D card the data will not reflect the physical state of your device.

Parameters:

adapterState the state struct to fill

See also:[ADAPTER_STATE](#)**Returns:**

int SEAIO_ERROR_NONE on success; error code of failure

Examples:[example.cpp](#).**1.1.3.11 int DigitalIoDevice::setState ([ADAPTER_STATE](#) & *adapterState*)**

Set an adapter's state (only valid on PIO and A/D models).

This is used to set configuration data. Both Control Words and A/D ranges can be set. NOTE: If your card isn't configurable or doesn't have an A/D this call will not physically affect your card.

Parameters:*adapterState* a configuration struct that you want to load**Returns:**

int SEAIO_ERROR_NONE on success; error code of failure

Examples:[example.cpp](#).**1.1.3.12 int DigitalIoDevice::getType ([SeaIo_CardType](#) & *cardType*)**

Return card classification.

DIO = Digital / relay I/O; PIO_8255 = cards which use or mimic 8255-style control words; (4030, 8205..) PIO_Non8255 = cards which use linear control words; (8001, 8010..)

Parameters:*cardType* will be assigned the card type value**See also:**[SeaIo_CardType](#)**Returns:**

int SEAIO_ERROR_NONE on success; error code of failure

1.1.3.13 int DigitalIoDevice::getHandle ()

Handle to the current device.

This is useful if you want to open another device, but want to remember what the previous one was.

Returns:

int hDevice the handle to the current device

1.1.3.14 int DigitalIoDevice::readPort (unsigned char *portNumber*, unsigned char & *value*)

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

1.1.3.15 int DigitalIoDevice::writePort (unsigned char *portNumber*, unsigned char *value*)

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

1.1.3.16 int DigitalIoDevice::readRelative (unsigned char *portNumber*, unsigned char & *value*)

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

1.1.3.17 int DigitalIoDevice::readAllInputs (unsigned char & *buffer*, unsigned long & *bufferLength*)

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

1.1.3.18 `int DigitalIoDevice::writeRelative (unsigned char portNumber, unsigned char value)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

1.1.3.19 `int DigitalIoDevice::closeRelay (unsigned short int relayNumber)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

1.1.3.20 `int DigitalIoDevice::openRelay (unsigned short int relayNumber)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

1.1.4 Member Data Documentation**1.1.4.1** `int DigitalIoDevice::m\_openStatus`

Used internally to see if a device is already open.

The documentation for this class was generated from the following file:

- `include/digitalIoDevice.h`

1.2 tagADAPTER_INFO Struct Reference

Info struct driver sends to API.

```
#include <seaiostream.h>
```

Public Attributes

- WORD [wCardNumber](#)
Model as 0x[MODEL_NUMBER].
- WORD [wBaseIO](#)
The base IO address used.
- UCHAR [ucControlWords](#)
The number of control words.
- UCHAR [ucPortInCount](#)
Current number of inputs.
- UCHAR [ucPortOutCount](#)
Current number of outputs.
- UCHAR [ucPortCount](#)
Total ports available.
- UCHAR [ucInput](#) [MAXINPUTS]
Array of ports set as inputs.
- UCHAR [ucOutput](#) [MAXOUTPUTS]
Array of ports set as outputs.
- UCHAR [ucChannelsAtoD](#)
The number of A/D channels.
- UCHAR [ucChannelsDtoA](#)
The number of DAC channels.
- UCHAR [ucEnabled](#)
Enabled status.

1.2.1 Detailed Description

Info struct driver sends to API.

Contains information about the current device configuration. The only non static members of this struct are the ucInput[] and ucOutput[] arrays which will change when a new control word is written or a new ADAPTER_STATE is set.

See also:

[ADAPTER_STATE](#)

Examples:

[example.c](#), and [example.cpp](#).

1.2.2 Member Data Documentation

1.2.2.1 WORD [tagADAPTER_INFO::wCardNumber](#)

Model as 0x[MODEL_NUMBER].

1.2.2.2 WORD [tagADAPTER_INFO::wBaseIO](#)

The base IO address used.

1.2.2.3 UCHAR [tagADAPTER_INFO::ucControlWords](#)

The number of control words.

1.2.2.4 UCHAR [tagADAPTER_INFO::ucPortInCount](#)

Current number of inputs.

Examples:

[example.c](#), and [example.cpp](#).

1.2.2.5 UCHAR [tagADAPTER_INFO::ucPortOutCount](#)

Current number of outputs.

Examples:

[example.c](#), and [example.cpp](#).

1.2.2.6 UCHAR [tagADAPTER_INFO::ucPortCount](#)

Total ports available.

1.2.2.7 UCHAR [tagADAPTER_INFO::ucInput](#)[MAXINPUTS]

Array of ports set as inputs.

Examples:

[example.c](#), and [example.cpp](#).

1.2.2.8 UCHAR [tagADAPTER_INFO::ucOutput](#)[MAXOUTPUTS]

Array of ports set as outputs.

Examples:

[example.c](#), and [example.cpp](#).

1.2.2.9 UCHAR [tagADAPTER_INFO::ucChannelsAtoD](#)

The number of A/D channels.

1.2.2.10 UCHAR [tagADAPTER_INFO::ucChannelsDtoA](#)

The number of DAC channels.

1.2.2.11 UCHAR [tagADAPTER_INFO::ucEnabled](#)

Enabled status.

The documentation for this struct was generated from the following file:

- [include/seaiestruct.h](#)

1.3 tagADAPTER_STATE Struct Reference

State structure sent back and forth.

```
#include <seaiestruct.h>
```

Public Attributes

- UCHAR [ucModeCW](#) [MAXCONTROL]
Mode Control Words 0-(n-1).
- UCHAR [ucAIRange](#) [MAXAICHANNELS]
Input voltage ranges for A/D.

1.3.1 Detailed Description

State structure sent back and forth.

Used to pass each Port's state parameters between the application and the driver. It is unnecessary to provide configuration parameters, if your device is incapable of the specified feature.

Examples:

[example.c](#), and [example.cpp](#).

1.3.2 Member Data Documentation

1.3.2.1 UCHAR [tagADAPTER_STATE::ucModeCW](#)[MAXCONTROL]

Mode Control Words 0-(n-1).

Examples:

[example.c](#), and [example.cpp](#).

1.3.2.2 UCHAR [tagADAPTER_STATE::ucAIRange](#)[MAXAICHANNELS]

Input voltage ranges for A/D.

Examples:

[example.c](#), and [example.cpp](#).

The documentation for this struct was generated from the following file:

- include/[seaiestruct.h](#)

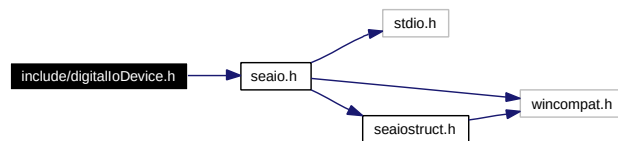
Chapter 2

SeaIO API File Documentation

2.1 include/digitalIoDevice.h File Reference

```
#include "seaio.h"
```

Include dependency graph for digitalIoDevice.h:



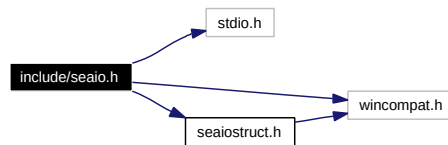
Classes

- class [DigitalIoDevice](#)
A SeaIO API wrapper class.

2.2 include/seaio.h File Reference

```
#include <stdio.h>
#include "wincompat.h"
#include "seaiostruct.h"
```

Include dependency graph for seaio.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum [Direction](#) { [Input](#), [Output](#), [Split](#), [NotExist](#) }
Direction Identifier.
- enum [AccessMode](#) { [Relative](#), [Absolute](#) }
Use this to determine access orientation.
- enum [SeaIo_CardType](#) { [DIO](#) = 0, [PIO_8255](#) = 1, [PIO_Non8255](#) = 2, [PIO_Bit-Level](#) = 3 }
The device classification.

Functions

- int [Seaio_Error](#) (int errornumber)
Error decode switch.
- int [SeaIo_OpenDevice](#) (unsigned char DeviceNumber, int *phDevice)
Opens nth SeaIO device.
- int [SeaIo_CloseDevice](#) (int hDevice)

Close the file handle for a SeaIO device.

- int [SeaIo_ReadByte](#) (int hDevice, unsigned long Port, unsigned char *Byte, enum [AccessMode](#) Mode)

Read a single byte from a device.

- int [SeaIo_WriteByte](#) (int hDevice, unsigned long Port, const unsigned char Data, enum [AccessMode](#) Mode)

Write a single byte to a device.

- int [SeaIo_ReadBit](#) (int hDevice, unsigned long BitNumber, BOOLEAN *BitState, enum [AccessMode](#) Mode)

Read the state of a single bit / relay on a device.

- int [SeaIo_WriteBit](#) (int hDevice, unsigned long BitNumber, const BOOLEAN BitState, enum [AccessMode](#) Mode)

Set or clear the state of a single bit / relay on a device.

- int [SeaIo_ReadAnalog](#) (int hDevice, unsigned long Channel, unsigned long *Data)

Reads a value from an analog to digital channel on a device.

- int [SeaIo_WriteAnalog](#) (int hDevice, unsigned long Channel, unsigned long Data)

Write a value to a digital to analog channel on a device.

- int [SeaIo_GetAdapterInfo](#) (int hDevice, [pADAPTER_INFO](#) pAdapter_Info)

Get device info.

- int [SeaIo_GetAdapterState](#) (int hDevice, [pADAPTER_STATE](#) pAdapterState)

Get an adapter's state.

- int [SeaIo_SetAdapterState](#) (int hDevice, [pADAPTER_STATE](#) pAdapterState)

Set a device's state.

- [SeaIo_CardType](#) [SeaIo_GetCardType](#) (int hDevice)

Return card classification.

- int [SeaIo_ReadReg](#) (int hDevice, unsigned char bRegNumber, unsigned char *pBuffer, unsigned long dwLength, unsigned long *pDwRetLength)

OBSOLETE.

- int [SeaIo_WriteReg](#) (int hDevice, unsigned char bRegNumber, unsigned char bMask)
OBSOLETE.
- int [SeaIo_RelayClose](#) (int hDevice, unsigned char uRelayNum)
OBSOLETE.
- int [SeaIo_RelayOpen](#) (int hDevice, unsigned char uRelayNum)
OBSOLETE.
- int [SeaIo_GetData](#) (int hDevice, unsigned char *pBuffer, unsigned long dwLength, unsigned long *pDwRetLength)
OBSOLETE.
- int [SeaIo_PutData](#) (int hDevice, unsigned char *pBuffer)
OBSOLETE.

2.2.1 Enumeration Type Documentation

2.2.1.1 enum [Direction](#)

Direction Identifier.

Currently unused.

Enumerator:

Input Indicates a port configured for input.

Output Indicates a port configured for output.

Split Indicates a port in SplitC configuration.

NotExist Indicates that there is no port here.

2.2.1.2 enum [AccessMode](#)

Use this to determine access orientation.

Relative access indicates either reading inputs only or writing outputs only. Absolute mode indicates writing directly to the register layout in your device manual.

Enumerator:

Relative Read only inputs, Write only outputs.

Absolute Register layout.

2.2.1.3 enum [SeaIo_CardType](#)

The device classification.

Mainly used to determine control word styles.

Enumerator:

DIO Not a configurable card.

PIO_8255 8255 style control words

PIO_Non8255 Linear style control words.

PIO_BitLevel Each bit is configurable.

2.2.2 Function Documentation

2.2.2.1 int Seaio_Error (int *errornumber*)

Error decode switch.

Decodes IOCTL return errors into SEAIO_ERROR_CODES

Parameters:

errornumber this will be a kernel error code like EINVAL

Returns:

int SEAIO_ERROR_CODES

2.2.2.2 int SeaIo_OpenDevice (unsigned char *DeviceNumber*, int * *phDevice*)

Opens nth SeaIO device.

Also obtains file handle.

Parameters:

DeviceNumber the nth SeaIO device

**phDevice* pointer to device handle

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.3 `int SeaIo_CloseDevice (int hDevice)`

Close the file handle for a SeaIO device.

Parameters:

hDevice device file handle

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.4 `int SeaIo_ReadByte (int hDevice, unsigned long Port, unsigned char *Byte, enum AccessMode Mode)`

Read a single byte from a device.

There is no built in read port protection, since this is a safe operation, but there is no guarantee what you read from write only ports is accurate.

Parameters:

hDevice the device file handle

Port the port or input number to read

**Byte* pointer to a byte worth of space to stick input

Mode Absolute(register layout) or Relative(inputs only)

See also:

[AccessMode](#)

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.5 `int SeaIo_WriteByte (int hDevice, unsigned long Port, const unsigned char Data, enum AccessMode Mode)`

Write a single byte to a device.

This function will only write to output configured ports or (new) configuration ports. In Relative mode you can only write to configured output ports. NOTICE: You can only write to configuration ports in Absolute AccessMode.

Parameters:

hDevice the device file handle
Port port or output number to write to
Data the data byte to attempt to write
Mode Relative(output only) or Absolute(register layout)

See also:

[AccessMode](#)

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.6 int SeaIo_ReadBit (int *hDevice*, unsigned long *BitNumber*, BOOLEAN **BitState*, enum [AccessMode](#) *Mode*)

Read the state of a single bit / relay on a device.

In Absolute [AccessMode](#) the bitNumber will be dataBit + (Port * 8); in Relative [AccessMode](#) the bitNumber will be dataBit + (input * 8). Port refers to the register layout described in the device manual. The input number refers to the nth port configured as an input.

Parameters:

hDevice the device file handle
BitNumber the bitNumber you wish to read
****BitState*** pointer to the BOOLEAN in which the state will be stored
Mode Absolute(register layout) or Relative(inputs only)

See also:

[AccessMode](#)

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.7 `int SeaIo_WriteBit (int hDevice, unsigned long BitNumber, const BOOLEAN BitState, enum AccessMode Mode)`

Set or clear the state of a single bit / relay on a device.

In Absolute [AccessMode](#) the *bitNumber* will be *dataBit* + (*Port* * 8); in Relative [AccessMode](#) the *bitNumber* will be *dataBit* + (*output* * 8). *Port* refers to the register layout described in the device manual. The *output* number refers to the *n*th port configured as an output.

Parameters:

hDevice the device file handle

BitNumber the *bitNumber* you wish to read

BitState state you wish to set the bit

Mode Absolute(register layout) or Relative(inputs only)

See also:

[AccessMode](#)

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.8 `int SeaIo_ReadAnalog (int hDevice, unsigned long Channel, unsigned long * Data)`

Reads a value from an analog to digital channel on a device.

12 bits accuracy. Only valid on A/D capable devices. If you do call it on other devices, no data will be read.

Parameters:

hDevice the device file handle

Channel the A/D channel you wish to read

**Data* pointer to location to store the read data

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.9 int SeaIo_WriteAnalog (int *hDevice*, unsigned long *Channel*, unsigned long *Data*)

Write a value to a digital to analog channel on a device.

Writes an analog output on a particular DAC. Only valid on D/A capable devices. If you do call it on other devices, no data will be written.

Parameters:

hDevice the device file handle

Channel the DAC channel you wish to write

Data the data you wish to write [0x000-0xFFFF]

Returns:

SEADIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.10 int SeaIo_GetAdapterInfo (int *hDevice*, [pADAPTER_INFO](#) *pAdapter_Info*)

Get device info.

Generic device information is now contained in the ADAPTER_INFO struct. Everything that is returned is static, however, except for the ucInput[] and ucOutput[] arrays which will be updated when a control word is written or the state is changed.

Parameters:

hDevice the device file handle

pAdapter_Info a pointer to an ADAPTER_INFO struct to fill up

See also:

[ADAPTER_INFO](#)

Returns:

SEADIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.11 `int SeaIo_GetAdapterState (int hDevice, pADAPTER\_STATE pAdapterState)`

Get an adapter's state.

The current control word configuration and A/D ranges are put in the ADAPTER_STATE struct passed.

Parameters:

hDevice the device file handle

pAdapterState pointer to the adapter state struct to fill up

See also:

[ADAPTER_STATE](#)

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.12 `int SeaIo_SetAdapterState (int hDevice, pADAPTER\_STATE pAdapterState)`

Set a device's state.

This can be used to set up all of an adapters control words and A/D ranges at once.

NOTE: There is no point in trying to set attributes your device isn't capable of.

Parameters:

hDevice the device file handle

pAdapterState pointer to your configured ADAPTER_STATE struct

See also:

[ADAPTER_STATE](#)

Returns:

SEAIO_ERROR_NONE on success; error code on failure.

Examples:

[example.c](#).

2.2.2.13 [SeaIo_CardType](#) SeaIo_GetCardType (int *hDevice*)

Return card classification.

DIO = Digital / relay I/O; PIO_8255 = cards which use or mimic 8255-style control words; (4030, 8205..) PIO_Non8255 = cards which use linear control words; (8001, 8010..)

Parameters:

hDevice the device file handle

Returns:

SeaIo_CardType

See also:

[SeaIo_CardType](#)

2.2.2.14 `int SeaIo_ReadReg (int hDevice, unsigned char bRegNumber, unsigned char * pBuffer, unsigned long dwLength, unsigned long * pDwRetLength)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

2.2.2.15 `int SeaIo_WriteReg (int hDevice, unsigned char bRegNumber, unsigned char bMask)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

2.2.2.16 `int SeaIo_RelayClose (int hDevice, unsigned char uRelayNum)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

2.2.2.17 `int SeaIo_RelayOpen (int hDevice, unsigned char uRelayNum)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

2.2.2.18 `int SeaIo_GetData (int hDevice, unsigned char * pBuffer, unsigned long dwLength, unsigned long * pDwRetLength)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

2.2.2.19 `int SeaIo_PutData (int hDevice, unsigned char * pBuffer)`

OBSOLETE.

Deprecated

Remains for backwards compatibility, but no longer supported.

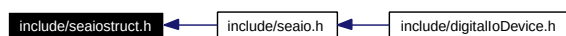
2.3 include/seaiostream.h File Reference

```
#include "wincompat.h"
```

Include dependency graph for seaiostream.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [tagADAPTER_INFO](#)
Info struct driver sends to API.
- struct [tagADAPTER_STATE](#)
State structure sent back and forth.

Defines

- #define [MAXINPUTS](#) 16
- #define [MAXOUTPUTS](#) 16
- #define [MAXCONTROL](#) 5
- #define [MAXICP](#) 4
- #define [MAXAICHANNELS](#) 8
- #define [RANGE_0_TO_5](#) 0x00
- #define [RANGE_MINUS_5_TO_5](#) 0x01
- #define [RANGE_0_TO_10](#) 0x02
- #define [RANGE_MINUS_10_TO_10](#) 0x03

Typedefs

- typedef [tagADAPTER_INFO](#) [ADAPTER_INFO](#)
Info struct driver sends to API.
- typedef [tagADAPTER_INFO](#) * [PADAPTER_INFO](#)
Info struct driver sends to API.

- typedef [tagADAPTER_STATE](#) ADAPTER_STATE
State structure sent back and forth.
- typedef [tagADAPTER_STATE](#) * pADAPTER_STATE
State structure sent back and forth.

Enumerations

- enum [SEAIO_ERROR_CODES](#) {
 [SEAIO_ERROR_NONE](#),
 [SEAIO_ERROR_PARAM](#),
 [SEAIO_ERROR_NOT_SUPPORTED](#),
 [SEAIO_ERROR_PORT_NOT_OPEN](#),
 [SEAIO_ERROR_INVALID_NAME](#),
 [SEAIO_ERROR_SHARING_VIOLATION](#),
 [SEAIO_ERROR_DRIVER_TYPE](#),
 [SEAIO_ERROR_BUFFER](#),
 [SEAIO_ERROR_INCOMPLETE](#),
 [SEAIO_ERROR_GENERAL](#),
 [SEAIO_ERROR_IO_PENDING](#),
 [SEAIO_ERROR_NO_TXBUFFER](#),
 [SEAIO_ERROR_INVALID_DEVICE_REQUEST](#),
 [SEAIO_ERROR_INVALID_HANDLE](#),
 [SEAIO_ERROR_CANCELED](#),
 [SEAIO_ERROR_LAST](#)
}

Return status codes from API.

2.3.1 Define Documentation

2.3.1.1 `#define MAXINPUTS 16`

2.3.1.2 `#define MAXOUTPUTS 16`

2.3.1.3 `#define MAXCONTROL 5`

2.3.1.4 `#define MAXICP 4`

2.3.1.5 `#define MAXAICHANNELS 8`

2.3.1.6 `#define RANGE_0_TO_5 0x00`

2.3.1.7 `#define RANGE_MINUS_5_TO_5 0x01`

2.3.1.8 `#define RANGE_0_TO_10 0x02`

2.3.1.9 `#define RANGE_MINUS_10_TO_10 0x03`

2.3.2 Typedef Documentation

2.3.2.1 typedef struct [tagADAPTER_INFO](#) [ADAPTER_INFO](#)

Info struct driver sends to API.

Contains information about the current device configuration. The only non static members of this struct are the ucInput[] and ucOutput[] arrays which will change when a new control word is written or a new ADAPTER_STATE is set.

See also:

[ADAPTER_STATE](#)

2.3.2.2 typedef struct [tagADAPTER_INFO](#) * [pADAPTER_INFO](#)

Info struct driver sends to API.

Contains information about the current device configuration. The only non static members of this struct are the ucInput[] and ucOutput[] arrays which will change when a new control word is written or a new ADAPTER_STATE is set.

See also:

[ADAPTER_STATE](#)

2.3.2.3 typedef struct [tagADAPTER_STATE](#) ADAPTER_STATE

State structure sent back and forth.

Used to pass each Port's state parameters between the application and the driver. It is unnecessary to provide configuration parameters, if your device is incapable of the specified feature.

2.3.2.4 typedef struct [tagADAPTER_STATE](#) * [pADAPTER_STATE](#)

State structure sent back and forth.

Used to pass each Port's state parameters between the application and the driver. It is unnecessary to provide configuration parameters, if your device is incapable of the specified feature.

2.3.3 Enumeration Type Documentation

2.3.3.1 enum [SEAIO_ERROR_CODES](#)

Return status codes from API.

These codes are also used to report errors to the API interface.

Enumerator:

- SEAIO_ERROR_NONE* Operation successfully completed.
- SEAIO_ERROR_PARAM* One or more of the given parameters was invalid.0.
- SEAIO_ERROR_NOT_SUPPORTED* An unsupported operation was attempted.
- SEAIO_ERROR_PORT_NOT_OPEN* Unable to open port.
- SEAIO_ERROR_INVALID_NAME* The device name given was not found.
Try reloading the driver
- SEAIO_ERROR_SHARING_VIOLATION* The specified device is already in use.
- SEAIO_ERROR_DRIVER_TYPE* Fatal driver error.
- SEAIO_ERROR_BUFFER* Passed NULL buffer.
- SEAIO_ERROR_INCOMPLETE* Failure to complete specified operation.
- SEAIO_ERROR_GENERAL* Unknown error.
- SEAIO_ERROR_IO_PENDING* Device request not yet completed.
- SEAIO_ERROR_NO_TXBUFFER* Required transmit buffer not found.
- SEAIO_ERROR_INVALID_DEVICE_REQUEST* Requested operation in valid on device.

SEAIO_ERROR_INVALID_HANDLE Passed invalid device handle.

SEAIO_ERROR_CANCELED Operation has been canceled.

SEAIO_ERROR_LAST Fatal cascade error.

Chapter 3

SeaIO API Example Documentation

3.1 example.c

A C programming example. This is simply a demonstration of the use of the SeaIO API NOTE: There is no error checking in this code. Also not all of these calls will work properly for all SeaIO devices. In this example it is assumed that there is a 3701 and 3820 installed on the system. When the driver is loaded, either by the seaio-load script or some other means, the devices will be enumerated in numerical order. (dev 0 = 3701, dev 1 = 3820)

```
1 #include <YOUR_INCLUDES>
2
3 //Note that you will have to link the library at compile time:
4 //gcc -o MyTest example.c -lseaiio
5 //Also note that the default make install copies seaio.h to /usr/local/include
6 #include <seaio.h>
7
8 int main(void)
9 {
10     //My device file pointers
11     int myDev;
12
13     //Data structures used throughout this example code
14     SeaIo_CardType myCardType;
15     ADAPTER_INFO myCardInfo;
16     ADAPTER_STATE myCardState;
17     unsigned char dataByte;
18     BOOLEAN myBit;
19     unsigned long a2dData;
20
21
22     // -----
```

```

23 //Open device 0. In this case it's a 3701: 6 ports configurable as
24 //input or output, SplitC capability, 8255 style control words.
25 // -----
26 SeaIo_OpenDevice(0, myDev);
27
28 //Grab the current control word configuration.
29 SeaIo_GetAdapterState(myDev, &myCardState);
30
31 //Now change control words. (3701 has 2 8255 style control words)
32 //128 is A-C output
33 myCardState.ucModeCW[0] = 0x80;
34 //131 is A out, B in, C(upper) out, C(lower) in
35 myCardState.ucModeCW[1] = 0x83;
36
37 //Now write the modified configuration to the device.
38 SeaIo_SetAdapterState(myDev, &myCardState);
39
40 //This device is now configured and ready to write outputs on
41 //ports 0, 1, 2, 8, and 10(upper nibble). Verify with ADAPTER_INFO.
42 SeaIo_GetAdapterInfo(myDev, &myCardInfo);
43
44 printf("\nINPUTS: ");
45 for (i = 0; i < myCardInfo.ucPortInCount; i++)
46     printf("%d, ", myCardInfo.ucInput[i]);
47
48 //The device is also configured to read inputs on ports 9 and
49 //10(lower nibble). Verify with ADAPTER_INFO.
50 printf("\nOUTPUTS: ");
51 for (i = 0; i < myCardInfo.ucPortOutCount; i++)
52     printf("%d, ", myCardInfo.ucOutput[i]);
53
54 //To set all outputs to 0, use WriteByte in Relative AccessMode.
55 for (i = 0; i < myCardInfo.ucPortInCount; i++)
56     SeaIo_WriteByte(myDev, i, 0x00, Relative);
57
58 //To read just inputs use ReadByte in Realative AccessMode.
59 //This call will read port 9.
60 SeaIo_ReadByte(myDev, 0, &dataByte, Relative);
61 printf("read input 0: %02X", dataByte);
62
63 //If you prefer to use the register layout found in the device manual,
64 //you may use Absolute AccessMode to read and write.
65 //You will not be able to write to a port configured as an input, but
66 //if you would like you may write directly to control ports.
67
68 //This will read the data from port 10. Port 10 is SplitC, so the
69 //upper nibble will be data previously written to port 10, and the
70 //lower nibble will be input data.
71 SeaIo_ReadByte(myDev, 10, &dataByte, Absolute);
72 printf("read prt 10: %02X", dataByte);
73
74 //To reconfigure the first three ports as inputs write 255 to port 3.
75 SeaIo_WriteByte(myDev, 3, 0xFF, Absolute);
76
77 //You also have the option to set/clear or read a single bit.
78 //The bitNumber is equivalent to dataBit(0-7) + (n * 8); where
79 //n is the port number in Absolute AccessMode and n is the nth

```

```
80         //input or output (depending of Read or Write) in Relative AccessMode.
81
82         //Read dataBit 1 of input 3(fourth input or port 9)
83         SeaIo_ReadBit(myDev, 25, &myBit, Relative);
84
85         //Alternatively, you could read the same bit likeso
86         SeaIo_ReadBit(myDev, 73, &myBit, Absolute);
87
88         //To set dataBit 5 of output 0(first input or port 8)
89         SeaIo_WriteBit(myDev, 5, TRUE, Relative);
90
91         //Alternatively, you could write to the same bit likeso
92         SeaIo_WriteBit(myDev, 69, FALSE, Absolute);
93
94         //When done with this device, close it.
95         SeaIo_CloseDevice(myDev);
96
97
98         // -----
99         //Open device 1. In this case it's a 3820: 8 Channels A/D, 2 Channels
100        //D/A, 16 BitLevel configurable Digital IO.
101        // -----
102        SeaIo_OpenDevice(1, myDev);
103
104        //Grab the current A/D conversion ranges.
105        SeaIo_GetAdapterState(myDev, &myCardState);
106
107        //Setup channel 0 for 0-5V and channel 7 for 0-10V
108        //Not that although the 3820 has configurable IO,
109        //it does not use the ucModeCW array.
110        myCardState.ucAIRange[0] = 0x00;
111        myCardState.ucAIRange[1] = 0x02;
112
113        //Now write the modified configuration to the device.
114        SeaIo_SetAdapterState(myDev, &myCardState);
115
116        //Read A/D conversion on channel 0(first channel)
117        //There are only 12 bits of A/D accuracy.
118        SeaIo_ReadAnalog(myDev, 0, &a2dData);
119        printf("A/D Channel 0: %03X", a2dData);
120
121        //Read A/D conversion on channel 7(eighth channel)
122        SeaIo_ReadAnalog(myDev, 7, &a2dData);
123        printf("A/D Channel 7: %03X", a2dData);
124
125        //The 3820 also has two DAC channels. These D/A converters are
126        //set to a range of 0-2V or 0-4V by jumpers. For this example,
127        //it is assumed both are set to 0-2V.
128
129        //To get an output very close to 2V on channel 0(first channel),
130        //one can write a value of around 4000.
131        SeaIo_WriteAnalog(myDev, 0, 4000);
132
133        //To get an output close to 1V on channel 1(second channel),
134        //one can write a value of around 2000.
135        SeaIo_WriteAnalog(myDev, 1, 2000);
136
```

```
137         //When done with this device, close it.  
138         SeaIo_CloseDevice(myDev);  
139  
140         return 0;  
141     }
```

3.2 example.cpp

A C++ programming example. This is simply a demonstration of the use of the [DigitalIoDevice](#) class. NOTE: There is no error checking in this code. Also not all of these calls will work properly for all SeaIO devices. In this example it is assumed that there is a 3701 and 3820 installed on the system. When the driver is loaded, either by the seaio-load script or some other means, the devices will be enumerated in numerical order. (dev 0 = 3701, dev 1 = 3820)

```

1 #include <YOUR_INCLUDES> //Most notably stdlib.h
2
3 //Note that you will have to link the library at compile time:
4 //g++ -o MyTest example.cpp -lseaiio
5 //Also note that the default make install copies digitalIoDevice.h to /usr/local/include
6 #include <digitalIoDevice.h>
7
8 int main(void)
9 {
10     //My device wrapper. You may optionally supply the constructor with
11     //the device number to initially attach itself to: DigitalIoDevice myDev(1);
12     DigitalIoDevice myDev;
13
14     //Data structures used throughout this example code
15     SeaIo_CardType myCardType;
16     ADAPTER_INFO myCardInfo;
17     ADAPTER_STATE myCardState;
18     unsigned char dataByte;
19     BOOLEAN myBit;
20     unsigned long a2dData;
21
22
23     // -----
24     //Open device 0. In this case it's a 3701: 6 ports configurable as
25     //input or output, SplitC capability, 8255 style control words.
26     //This is not strictly necessary, but you may do it if you like.
27     // -----
28     myDev.open(0);
29
30     //Grab the current control word configuration.
31     myDev.getState(myCardState);
32
33     //Now change control words. (3701 has 2 8255 style control words)
34     //128 is A-C output
35     myCardState.ucModeCW[0] = 0x80;
36     //131 is A out, B in, C(upper) out, C(lower) in
37     myCardState.ucModeCW[1] = 0x83;
38
39     //Now write the modified configuration to the device.
40     myDev.setState(myCardState);
41
42     //This device is now configured and ready to write outputs on
43     //ports 0, 1, 2, 8, and 10(upper nibble). Verify with ADAPTER_INFO.
44     myDev.getInfo(myCardInfo);
45
46     printf("\nINPUTS: ");

```

```

47         for (i = 0; i < myCardInfo.ucPortInCount; i++)
48             printf("%d, ", myCardInfo.ucInput[i]);
49
50         //The device is also configured to read inputs on ports 9 and
51         //10(lower nibble). Verify with ADAPTER_INFO.
52         printf("\nOUTPUTS: ");
53         for (i = 0; i < myCardInfo.ucPortOutCount; i++)
54             printf("%d, ", myCardInfo.ucOutput[i]);
55
56         //To set all outputs to 0, use WriteByte in Relative AccessMode.
57         for (i = 0; i < myCardInfo.ucPortInCount; i++)
58             myDev.writeByte(i, 0x00, Relative);
59
60         //To read just inputs use ReadByte in Realative AccessMode.
61         //This call will read port 9.
62         myDev.readByte(0, dataByte, Relative);
63         printf("read input 0: %02X", dataByte);
64
65         //If you prefer to use the register layout found in the device manual,
66         //you may use Absolute AccessMode to read and write.
67         //You will not be able to write to a port configured as an input, but
68         //if you would like you may write directly to control ports.
69
70         //This will read the data from port 10. Port 10 is SplitC, so the
71         //upper nibble will be data previously written to port 10, and the
72         //lower nibble will be input data.
73         myDev.readByte(myDev, 10, dataByte, Absolute);
74         printf("read prt 10: %02X", dataByte);
75
76         //To reconfigure the first three ports as inputs write 255 to port 3.
77         myDev.writeByte(3, 0xFF, Absolute);
78
79         //You also have the option to set/clear or read a single bit.
80         //The bitNumber is equivalent to dataBit(0-7) + (n * 8); where
81         //n is the port number in Absolute AccessMode and n is the nth
82         //input or output (depending of Read or Write) in Relative AccessMode.
83
84         //Read dataBit 1 of input 3(fourth input or port 9)
85         myDev.readBit(25, myBit, Relative);
86
87         //Alternatively, you could read the same bit likeso
88         myDev.readBit(73, myBit, Absolute);
89
90         //To set dataBit 5 of output 0(first input or port 8)
91         myDev.writeBit(5, TRUE, Relative);
92
93         //Alternatively, you could write to the same bit likeso
94         myDev.writeBit(69, FALSE, Absolute);
95
96         //This is automatically called by open() when you switch devices,
97         //but it is a good idead to go ahead and call it yourself.
98         myDev.close();
99
100
101         // -----
102         //Open device 1. In this case it's a 3820: 8 Channels A/D, 2 Channels
103         //D/A, 16 BitLevel configurable Digital IO.

```

```
104          // -----
105          myDev.open(1);
106
107          //Grab the current A/D conversion ranges.
108          myDev.getState(myCardState);
109
110          //Setup channel 0 for 0-5V and channel 7 for 0-10V
111          //Not that although the 3820 has configurable IO,
112          //it does not use the ucModeCW array.
113          myCardState.ucAIRange[0] = 0x00;
114          myCardState.ucAIRange[1] = 0x02;
115
116          //Now write the modified configuration to the device.
117          myDev.setState(myCardState);
118
119          //Read A/D conversion on channel 0(first channel)
120          //There are only 12 bits of A/D accuracy.
121          myDev.readAnalog(0, a2dData);
122          printf("A/D Channel 0: %03X", a2dData);
123
124          //Read A/D conversion on channel 7(eighth channel)
125          myDev.readAnalog(7, a2dData);
126          printf("A/D Channel 7: %03X", a2dData);
127
128          //The 3820 also has two DAC channels. These D/A converters are
129          //set to a range of 0-2V or 0-4V by jumpers. For this example,
130          //it is assumed both are set to 0-2V.
131
132          //To get an output very close to 2V on channel 0(first channel),
133          //one can write a value of around 4000.
134          myDev.writeAnalog(0, 4000);
135
136          //To get an output close to 1V on channel 1(second channel),
137          //one can write a value of around 2000.
138          myDev.writeAnalog(1, 2000);
139
140          //When done with this device, close it. This time it is necessary,
141          //otherwise the device will still be in use.
142          myDev.close();
143
144          return 0;
145 }
```

Index

- Absolute
 - seaio.h, [20](#)
- AccessMode
 - seaio.h, [20](#)
- ADAPTER_INFO
 - seaistruct.h, [31](#)
- ADAPTER_STATE
 - seaistruct.h, [31](#)
- close
 - DigitalIoDevice, [4](#)
- closeRelay
 - DigitalIoDevice, [10](#)
- DigitalIoDevice, [1](#)
 - DigitalIoDevice, [3](#)
- DigitalIoDevice
 - close, [4](#)
 - closeRelay, [10](#)
 - DigitalIoDevice, [3](#)
 - getHandle, [8](#)
 - getInfo, [7](#)
 - getState, [7](#)
 - getType, [8](#)
 - m_openStatus, [10](#)
 - open, [3](#)
 - openRelay, [10](#)
 - readAllInputs, [9](#)
 - readAnalog, [6](#)
 - readBit, [5](#)
 - readByte, [4](#)
 - readPort, [9](#)
 - readRelative, [9](#)
 - setState, [8](#)
 - writeAnalog, [6](#)
 - writeBit, [5](#)
 - writeByte, [4](#)
 - writePort, [9](#)
 - writeRelative, [9](#)
- DIO
 - seaio.h, [21](#)
- Direction
 - seaio.h, [20](#)
- getHandle
 - DigitalIoDevice, [8](#)
- getInfo
 - DigitalIoDevice, [7](#)
- getState
 - DigitalIoDevice, [7](#)
- getType
 - DigitalIoDevice, [8](#)
- include/digitalIoDevice.h, [17](#)
- include/seaio.h, [18](#)
- include/seaistruct.h, [29](#)
- Input
 - seaio.h, [20](#)
- m_openStatus
 - DigitalIoDevice, [10](#)
- MAXAICHANNELS
 - seaistruct.h, [31](#)
- MAXCONTROL
 - seaistruct.h, [31](#)
- MAXICP
 - seaistruct.h, [31](#)
- MAXINPUTS
 - seaistruct.h, [31](#)
- MAXOUTPUTS
 - seaistruct.h, [31](#)
- NotExist
 - seaio.h, [20](#)

- open
 - DigitalIoDevice, [3](#)
- openRelay
 - DigitalIoDevice, [10](#)
- Output
 - seaio.h, [20](#)
- pADAPTER_INFO
 - seaistruct.h, [31](#)
- pADAPTER_STATE
 - seaistruct.h, [32](#)
- PIO_8255
 - seaio.h, [21](#)
- PIO_BitLevel
 - seaio.h, [21](#)
- PIO_Non8255
 - seaio.h, [21](#)
- RANGE_0_TO_10
 - seaistruct.h, [31](#)
- RANGE_0_TO_5
 - seaistruct.h, [31](#)
- RANGE_MINUS_10_TO_10
 - seaistruct.h, [31](#)
- RANGE_MINUS_5_TO_5
 - seaistruct.h, [31](#)
- readAllInputs
 - DigitalIoDevice, [9](#)
- readAnalog
 - DigitalIoDevice, [6](#)
- readBit
 - DigitalIoDevice, [5](#)
- readByte
 - DigitalIoDevice, [4](#)
- readPort
 - DigitalIoDevice, [9](#)
- readRelative
 - DigitalIoDevice, [9](#)
- Relative
 - seaio.h, [20](#)
- seaio.h
 - Absolute, [20](#)
 - AccessMode, [20](#)
 - DIO, [21](#)
 - Direction, [20](#)
 - Input, [20](#)
 - NotExist, [20](#)
 - Output, [20](#)
 - PIO_8255, [21](#)
 - PIO_BitLevel, [21](#)
 - PIO_Non8255, [21](#)
 - Relative, [20](#)
 - SeaIo_CardType, [20](#)
 - SeaIo_CloseDevice, [21](#)
 - SeaIo_Error, [21](#)
 - SeaIo_GetAdapterInfo, [25](#)
 - SeaIo_GetAdapterState, [25](#)
 - SeaIo_GetCardType, [26](#)
 - SeaIo_GetData, [28](#)
 - SeaIo_OpenDevice, [21](#)
 - SeaIo_PutData, [28](#)
 - SeaIo_ReadAnalog, [24](#)
 - SeaIo_ReadBit, [23](#)
 - SeaIo_ReadByte, [22](#)
 - SeaIo_ReadReg, [27](#)
 - SeaIo_RelayClose, [27](#)
 - SeaIo_RelayOpen, [27](#)
 - SeaIo_SetAdapterState, [26](#)
 - SeaIo_WriteAnalog, [24](#)
 - SeaIo_WriteBit, [23](#)
 - SeaIo_WriteByte, [22](#)
 - SeaIo_WriteReg, [27](#)
 - Split, [20](#)
- SeaIo_CardType
 - seaio.h, [20](#)
- SeaIo_CloseDevice
 - seaio.h, [21](#)
- SeaIo_Error
 - seaio.h, [21](#)
- SEAIO_ERROR_BUFFER
 - seaistruct.h, [32](#)
- SEAIO_ERROR_CANCELED
 - seaistruct.h, [33](#)
- SEAIO_ERROR_CODES
 - seaistruct.h, [32](#)
- SEAIO_ERROR_DRIVER_TYPE
 - seaistruct.h, [32](#)
- SEAIO_ERROR_GENERAL
 - seaistruct.h, [32](#)
- SEAIO_ERROR_INCOMPLETE
 - seaistruct.h, [32](#)

- SEAI0_ERROR_INVALID_DEVICE_-
REQUEST
seai0struct.h, [32](#)
- SEAI0_ERROR_INVALID_HANDLE
seai0struct.h, [32](#)
- SEAI0_ERROR_INVALID_NAME
seai0struct.h, [32](#)
- SEAI0_ERROR_IO_PENDING
seai0struct.h, [32](#)
- SEAI0_ERROR_LAST
seai0struct.h, [33](#)
- SEAI0_ERROR_NO_TXBUFFER
seai0struct.h, [32](#)
- SEAI0_ERROR_NONE
seai0struct.h, [32](#)
- SEAI0_ERROR_NOT_SUPPORTED
seai0struct.h, [32](#)
- SEAI0_ERROR_PARAM
seai0struct.h, [32](#)
- SEAI0_ERROR_PORT_NOT_OPEN
seai0struct.h, [32](#)
- SEAI0_ERROR_SHARING_-
VIOLATION
seai0struct.h, [32](#)
- SeaIo_GetAdapterInfo
seai0.h, [25](#)
- SeaIo_GetAdapterState
seai0.h, [25](#)
- SeaIo_GetCardType
seai0.h, [26](#)
- SeaIo_GetData
seai0.h, [28](#)
- SeaIo_OpenDevice
seai0.h, [21](#)
- SeaIo_PutData
seai0.h, [28](#)
- SeaIo_ReadAnalog
seai0.h, [24](#)
- SeaIo_ReadBit
seai0.h, [23](#)
- SeaIo_ReadByte
seai0.h, [22](#)
- SeaIo_ReadReg
seai0.h, [27](#)
- SeaIo_RelayClose
seai0.h, [27](#)
- SeaIo_RelayOpen
seai0.h, [27](#)
- SeaIo_SetAdapterState
seai0.h, [26](#)
- SeaIo_WriteAnalog
seai0.h, [24](#)
- SeaIo_WriteBit
seai0.h, [23](#)
- SeaIo_WriteByte
seai0.h, [22](#)
- SeaIo_WriteReg
seai0.h, [27](#)
- seai0struct.h
 - ADAPTER_INFO, [31](#)
 - ADAPTER_STATE, [31](#)
 - MAXAICHANNELS, [31](#)
 - MAXCONTROL, [31](#)
 - MAXICP, [31](#)
 - MAXINPUTS, [31](#)
 - MAXOUTPUTS, [31](#)
 - pADAPTER_INFO, [31](#)
 - pADAPTER_STATE, [32](#)
 - RANGE_0_TO_10, [31](#)
 - RANGE_0_TO_5, [31](#)
 - RANGE_MINUS_10_TO_10, [31](#)
 - RANGE_MINUS_5_TO_5, [31](#)
 - SEAI0_ERROR_BUFFER, [32](#)
 - SEAI0_ERROR_CANCELED, [33](#)
 - SEAI0_ERROR_CODES, [32](#)
 - SEAI0_ERROR_DRIVER_TYPE,
[32](#)
 - SEAI0_ERROR_GENERAL, [32](#)
 - SEAI0_ERROR_INCOMPLETE,
[32](#)
 - SEAI0_ERROR_INVALID_-
DEVICE_REQUEST, [32](#)
 - SEAI0_ERROR_INVALID_-
HANDLE, [32](#)
 - SEAI0_ERROR_INVALID_-
NAME, [32](#)
 - SEAI0_ERROR_IO_PENDING, [32](#)
 - SEAI0_ERROR_LAST, [33](#)
 - SEAI0_ERROR_NO_TXBUFFER,
[32](#)
 - SEAI0_ERROR_NONE, [32](#)

SEAIO_ERROR_NOT_SUPPORTED, [32](#)
SEAIO_ERROR_PARAM, [32](#)
SEAIO_ERROR_PORT_NOT_OPEN, [32](#)
SEAIO_ERROR_SHARING_VIOLATION, [32](#)
setState
 DigitalIoDevice, [8](#)
Split
 seaio.h, [20](#)
tagADAPTER_INFO, [11](#)
tagADAPTER_INFO
 ucChannelsAtoD, [13](#)
 ucChannelsDtoA, [13](#)
 ucControlWords, [12](#)
 ucEnabled, [13](#)
 ucInput, [13](#)
 ucOutput, [13](#)
 ucPortCount, [12](#)
 ucPortInCount, [12](#)
 ucPortOutCount, [12](#)
 wBaseIO, [12](#)
 wCardNumber, [12](#)
tagADAPTER_STATE, [14](#)
tagADAPTER_STATE
 ucAIRange, [14](#)
 ucModeCW, [14](#)

ucAIRange
 tagADAPTER_STATE, [14](#)
ucChannelsAtoD
 tagADAPTER_INFO, [13](#)
ucChannelsDtoA
 tagADAPTER_INFO, [13](#)
ucControlWords
 tagADAPTER_INFO, [12](#)
ucEnabled
 tagADAPTER_INFO, [13](#)
ucInput
 tagADAPTER_INFO, [13](#)
ucModeCW
 tagADAPTER_STATE, [14](#)
ucOutput
 tagADAPTER_INFO, [13](#)

ucPortCount
 tagADAPTER_INFO, [12](#)
ucPortInCount
 tagADAPTER_INFO, [12](#)
ucPortOutCount
 tagADAPTER_INFO, [12](#)

wBaseIO
 tagADAPTER_INFO, [12](#)
wCardNumber
 tagADAPTER_INFO, [12](#)
writeAnalog
 DigitalIoDevice, [6](#)
writeBit
 DigitalIoDevice, [5](#)
writeByte
 DigitalIoDevice, [4](#)
writePort
 DigitalIoDevice, [9](#)
writeRelative
 DigitalIoDevice, [9](#)