U2C-11 API Programmer's Guide.

U2C-11 board initialization routines

```
BYTE U2C_GetDeviceCount();
```

The *U2C_GetDeviceCount* function checks how many I2CBridge devices are currently attached.

Parameters:

None.

Return value:

The function returns the number of the I2CBridge devices detected on current computer.

```
U2C_RESULT U2C_GetSerialNum(
      HANDLE hDevice,
      long* pSerialNum
);
```

The *U2C_GetSerialNum* function retrieves the Serial Number of the current device. This ID is unique for current I2CBridge device and can help to identify it when using a number of devices simultaneously.

Parameters:

hDevice

Handle to the I2CBridge device to retrieve the Serial Number from. The device has to be opened first, using *U2C_OpenDevice* or *U2C_OpenDeviceBySerialNum* function.

pSerialNum

Pointer to a long integer variable to be filled with the device Serial Number.

Return value:

U2C_SUCCESS

Serial Number was successfully obtained.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_IsHandleValid (
      HANDLE hDevice
);
```

The *U2C_IsHandleValid* function checks whether the device referenced by *hDevice* handle is currently present

Parameters:

hDevice

Handle to the I2CBridge device that will be checked.

Return value:

U2C_SUCCESS

The device referenced by *hDevice* handle is present.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
HANDLE U2C_OpenDevice(
      BYTE nDevice
);
```

The *U2C_OpenDevice* function opens the I2CBridge device.

Parameters:

nDevice

The device number to open.

Return value:

If the function succeeds, the return value is a valid handle to the specified device. If the function fails, the return value is INVALID_HANDLE_VALUE. This can happen if the specified device is not present.

```
HANDLE U2C_OpenDeviceBySerialNum(
      long nSerialNum
);
```

The *U2C_OpenDeviceBySerialNum* function opens the I2CBridge device with specified Serial Number.

Parameters:

nSerialNum

The Serial Number of the device to open.

Return value:

If the function succeeds, the return value is a valid handle to the specified device. If the function fails, the return value is INVALID_HANDLE_VALUE. This can happen if the specified device is not present.

```
U2C_RESULT U2C_CloseDevice(
      HANDLE hDevice
);
```

The *U2C_CloseDevice* function closes the open device handle.

Parameters:

hDevice

Handle to the I2CBridge device to close.

Return value:

U2C_SUCCESS

The device referenced by *hDevice* handle was successfully closed.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_GetFirmwareVersion(
      HANDLE hDevice,
      PU2C_VESION_INFO pVersion
);
```

The *U2C_GetFirmwareVersion* function retrieves the version of the firmware currently loaded into the I2CBridge device referenced by *hDevice* handle.

Parameters:

hDevice

Handle to the I2CBridge device to obtain firmware version from.

pVersion

Pointer to a U2C_VERSION_INFO structure to be filled with the firmware version number.

.

Return value:

U2C_SUCCESS
The firmware version was successfully retrieved.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_GetDriverVersion(
      HANDLE hDevice,
      PU2C_VERSION_INFO pVersion
);
```

The *U2C_GetDriverVersion* function retrieves the version of the driver used to communicate with I2CBridge device.

Parameters:

hDevice
Handle to the I2CBridge device to obtain the version of the driver used to communicate with.
pVersion
Pointer to a U2C_VERSION_INFO structure to be filled with the driver version number.

Return value:

U2C_SUCCESS
The driver version was successfully retrieved.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_VERSION_INFO U2C_GetDllVersion();
```

The *U2C_GetDllVersion* function retrieves the version of the "I2CBrdg.dll" dynamic link library.

Parameters:

None.

Return value:

U2C_VERSION_INFO structure containing "I2CBrdg.dll" dynamic link library version number.

I2C high level and configuration routines

```
U2C_RESULT U2C_SetI2cFreq(
      HANDLE hDevice,
      BYTE Frequency
);
```

The *U2C_SetI2cFreq* function configures I2C bus speed.

Parameters:

hDevice

Handle to the I2CBridge device.

Frequency

The frequency of I2C bus, where:
0 corresponds to I2C bus fast mode.
1 corresponds to I2C bus standard mode.
1+n corresponds to clock period of I2C bus equal to 10+2*n µS.

For convenience following constants were introduced:

constant	frequency
U2C_I2C_FREQ_FAST	I2C bus fast mode
U2C_I2C_FREQ_STD	I2C bus standard mode
U2C_I2C_FREQ_83KHZ	83 kHz
U2C_I2C_FREQ_71KHZ	71 kHz
U2C_I2C_FREQ_62KHZ	62 kHz
U2C_I2C_FREQ_50KHZ	50 kHz
U2C_I2C_FREQ_25KHZ	25 kHz
U2C_I2C_FREQ_10KHZ	10 kHz
U2C_I2C_FREQ_5KHZ	5 kHz
U2C_I2C_FREQ_2KHZ	2 kHz

Return value:

U2C_SUCCESS

The frequency value was successfully set.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_GetI2cFreq(
      HANDLE hDevice,
      BYTE *pFrequency
);
```

The *U2C_GetI2cFreq* function obtains I2C bus speed.

Parameters:

hDevice

Handle to the I2CBridge device.

pFrequency

A pointer to byte to be filled with current I2C bus frequency, where:
0 corresponds to I2C bus fast mode.
1 corresponds to I2C bus standard mode.
1+n corresponds to clock period of I2C bus equal to 10+2*n µS.

For convenience following constants were introduced:

constant	frequency
U2C_I2C_FREQ_FAST	I2C bus fast mode
U2C_I2C_FREQ_STD	I2C bus standard mode
U2C_I2C_FREQ_83KHZ	83 kHz
U2C_I2C_FREQ_71KHZ	71 kHz
U2C_I2C_FREQ_62KHZ	62 kHz
U2C_I2C_FREQ_50KHZ	50 kHz
U2C_I2C_FREQ_25KHZ	25 kHz
U2C_I2C_FREQ_10KHZ	10 kHz
U2C_I2C_FREQ_5KHZ	5 kHz
U2C_I2C_FREQ_2KHZ	2 kHz

Return value:

U2C_SUCCESS

The frequency value was successfully retrieved.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_Read(
      HANDLE hDevice,
      PU2C_TRANSACTION pTransaction
);
```

The *U2C_Read* function reads up to 256 bytes from the I2C slave device.

Parameters:

hDevice

Handle to the I2CBridge device.

pTransaction

Pointer to the U2C_TRANSACTION structure to be used during the read transaction.

Before calling the function this structure have to be partially filled:

- *nSlaveDeviceAddress* – must contain the slave I2C device address;

- *nMemoryAddressLength* – must contain the internal address length (in bytes from 0 up to 4). If *nMemoryAddressLength* is equal to 0, no address will be sent to device and repeated I2C start condition won't be generated.
- *MemoryAddress* – must contain the internal I2C slave device address.
- *nBufferLength* – must contain the number of bytes to be read from the I2C slave device.

After successful completion of the read operation *Buffer* member of the structure will be filled with data read from slave I2C device.

Return value:

U2C_SUCCESS
: The data was successfully read.

U2C_HARDWARE_NOT_FOUND
: I2CBridge device referenced by *hDevice* handle was not found.

U2C_SLAVE_OPENNING_FOR_WRITE_FAILED

U2C_SLAVE_OPENNING_FOR_READ_FAILED

U2C_SENDING_MEMORY_ADDRESS_FAILED

```
U2C_RESULT U2C_Write(
      HANDLE hDevice,
      PU2C_TRANSACTION pTransaction
);
```

The *U2C_Write* function writes up to 256 bytes into the I2C slave device.

Parameters:

hDevice
: Handle to the I2CBridge device.

pTransaction
: Pointer to the U2C_TRANSACTION structure to be used during the write transaction.

Before calling the function this structure have to be filled:

- *nSlaveDeviceAddress* – must contain the slave I2C device address;
- *nMemoryAddressLength* – must contain the internal address length (in bytes from 0 up to 4). If *nMemoryAddressLength* is equal to 0, no address will be sent to device.
- *MemoryAddress* – must contain the internal I2C slave device address.
- *nBufferLength* – must contain the number of bytes to be written into the I2C slave device.
- *nBuffer* – must contain the data to be written.

Return value:

U2C_SUCCESS

The data was successfully written.

U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

U2C_SLAVE_OPENNING_FOR_WRITE_FAILED

U2C_SENDING_MEMORY_ADDRESS_FAILED

U2C_SENDING_DATA_FAILED

```
U2C_RESULT U2C_ScanDevices(
      HANDLE hDevice,
      PU2C_SLAVE_ADDR_LIST pList
);
```

The *U2C_ScanDevices* function scans slave device addresses currently occupied by I2C slave devices connected to the I2C bus.

Parameters:

hDevice
Handle to the I2CBridge device.

pTransaction
Pointer to the U2C_SLAVE_ADDR_LIST structure to be filled with slave device addresses. *nDeviceNumber* member will contain the number of the valid addresses in *List* array.

Return value:

U2C_SUCCESS
Operation was successfully completed and *pList* is filled with valid data.

U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

I2C low level routines

```
U2C_RESULT U2C_Start(
      HANDLE hDevice
);
```

The *U2C_Start* function generates the start condition on the I2C bus.

Parameters:

hDevice
Handle to the I2CBridge device.

Return value:

U2C_SUCCESS
Start condition was successfully generated.

U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_RepeatedStart(
      HANDLE hDevice
);
```

The *U2C_RepeatedStart* function generates the repeated start condition on the I2C bus.

Parameters:
hDevice
Handle to the I2CBridge device.
Return value:
U2C_SUCCESS
Repeated start condition was successfully generated.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_Stop(
      HANDLE hDevice
);
```

The *U2C_Stop* function generates the stop condition on the I2C bus. It can be also used for the generation of the repeated stop condition.

Parameters:
hDevice
Handle to the I2CBridge device.
Return value:
U2C_SUCCESS
Stop condition was successfully generated.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_PutByte(
      HANDLE hDevice,
      BYTE Data
);
```

The *U2C_PutByte* function transmits a single byte to the I2C bus. It assumes that the bus is available and the Start Condition has been generated first. This function doesn't check acknowledge from the I2C slave device, so you must call *U2C_GetAck* to check acknowledge or to use *U2C_PutByteWithAck* instead of *U2C_PutByte* function. This

function can be called several times to implement custom I2C like protocol. The function does not finish the I2C bus transaction after transmission, so at the end of I2C transaction *U2C_Stop* function has to be called.

Parameters:

hDevice

Handle to the I2CBridge device.

Data

Byte to be transmitted to the I2C bus.

Return value:

U2C_SUCCESS

Byte was successfully transmitted to the I2C bus.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_GetByte(
      HANDLE hDevice,
      BYTE* pData
);
```

The *U2C_GetByte* function reads a single byte from the I2C bus. It assumes that the bus is available, the Start Condition has been previously generated and the slave device has been properly addressed. This function doesn't generate acknowledge, so you must call the *U2C_PutAck* function or use *U2C_GetByteWithAck* instead of *U2C_GetByte* function. This function can be called several times to implement custom I2C like protocol. The function does not finish the I2C bus transaction after transmission, so at the end of I2C transaction *U2C_Stop* function has to be called.

Parameters:

hDevice

Handle to the I2CBridge device.

pData

A pointer to byte to be filled with data read from the I2C bus.

Return value:

U2C_SUCCESS

Byte was successfully read from the I2C bus.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_PutByteWithAck(
      HANDLE hDevice,
      BYTE Data
);
```

The *U2C_PutByteWithAck* function transmits a single byte to the I2C bus and checks for acknowledge from I2C slave device. It assumes that the bus is available and the Start Condition has been generated first. This function can be called several times to implement custom I2C like protocol. The function does not finish the I2C bus transaction after transmission, so at the end of I2C transaction *U2C_Stop* function has to be called.

Parameters:

hDevice

Handle to the I2CBridge device.

Data

Byte to be transmitted to the I2C bus.

Return value:

U2C_SUCCESS

Byte was successfully transmitted to the I2C bus and I2C slave device provided acknowledge.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

U2C_NO_ACK

I2C slave device did not acknowledge the transmitted byte.

```
U2C_RESULT U2C_GetByteWithAck(
      HANDLE hDevice,
      BYTE* pData,
      BOOL bAck
);
```

The *U2C_GetByteWithAck* function reads a single byte from the I2C bus and then generates acknowledge or not-acknowledge condition according to the value passed in *bAck* parameter. It assumes that the bus is available, the Start Condition has been previously generated and the slave device has been properly addressed. This function can be called several times to implement custom I2C like protocol. The function does not finish the I2C bus transaction after transmission, so at the end of I2C transaction *U2C_Stop* function has to be called.

Parameters:

hDevice

Handle to the I2CBridge device.

pData

A pointer to byte to be filled with data read from the I2C bus.

bAck

This parameter determines if acknowledge should be generated after the byte is transmitted. If *bAck* is TRUE – acknowledge will be generated, if *bAck* is FALSE – non-acknowledge will be generated.

Return value:

U2C_SUCCESS

Byte was successfully read from I2C bus.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_PutAck(
      HANDLE hDevice,
      BOOL bAck
);
```

The *U2C_PutAck* function generates acknowledge or not-acknowledge condition according to the value passed in *bAck* parameter. This function does not finish the I2C bus transaction after transmission, so at the end of I2C transaction *U2C_Stop* function has to be called.

Parameters:

hDevice

Handle to the I2CBridge device.

bAck

This parameter determines whether acknowledge or non-acknowledge should be generated. If *bAck* is TRUE – acknowledge will be generated, if *bAck* is FALSE – non-acknowledge will be generated.

Return value:

U2C_SUCCESS

Acknowledge / non-acknowledge condition was successfully generated.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_GetAck(
      HANDLE hDevice
);
```

The *U2C_GetAck* function checks for acknowledge from I2C slave device. This function does not finish the I2C bus transaction after transmission, so at the end of I2C transaction *U2C_Stop* function has to be called.

Parameters:

hDevice

Handle to the I2CBridge device.

Return value:

U2C_SUCCESS

I2C slave device provided acknowledge.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

U2C_NO_ACK

I2C slave device did not provide acknowledge.

I2C wire level routines

```
U2C_RESULT U2C_ReadScl(
      HANDLE hDevice,
      U2C_LINE_STATE* pState
);
```

The *U2C_ReadScl* function checks the current state of the SCL line of the I2C bus.

Parameters:

hDevice

Handle to the I2CBridge device.

pState

Pointer to the location to be filled with the SCL line state:
LS_RELEASED – if line is released (high)
LS_DROPPED_BY_I2C_BRIDGE – if I2CBridge device has pulled down the line.
LS_DROPPED_BY_SLAVE – if I2C slave device has pulled down the line.

Return value:

U2C_SUCCESS

The line state was successfully read.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_ReadSda(
      HANDLE hDevice,
      U2C_LINE_STATE* pState
);
```

The *U2C_ReadSda* function checks the current state of the SDA line of the I2C bus.

Parameters:

hDevice

Handle to the I2CBridge device.

pState

Pointer to the location to be filled with the SDA line state:
LS_RELEASED – if line is released (high)

LS_DROPPED_BY_I2C_BRIDGE – if I2CBridge device has pulled down the line.
LS_DROPPED_BY_SLAVE – if I2C slave device has pulled down the line.

Return value:

U2C_SUCCESS
The line state was successfully read.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_ReleaseScl(
      HANDLE hDevice,
);
```

The *U2C_ReleaseScl* function releases the SCL line of the I2C bus. If the line is not pulled down by I2C slave device, it will get high.

Parameters:

hDevice
Handle to the I2CBridge device.

Return value:

U2C_SUCCESS
The line was successfully released.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_ReleaseSda(
      HANDLE hDevice,
);
```

The *U2C_ReleaseSda* function releases the SDA line of the I2C bus. If the line is not pulled down by I2C slave device, it will get high.

Parameters:

hDevice
Handle to the I2CBridge device.

Return value:

U2C_SUCCESS
The line was successfully released.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_DropScl(
      HANDLE hDevice,
);
```

The *U2C_DropScl* function pulls down the SCL line of the I2C bus.

Parameters:

hDevice

Handle to the I2CBridge device.

Return value:

U2C_SUCCESS

The line was successfully dropped.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_DropSda(
      HANDLE hDevice,
);
```

The *U2C_DropSda* function pulls down the SCL line of the I2C bus.

Parameters:

hDevice

Handle to the I2CBridge device.

Return value:

U2C_SUCCESS

The line was successfully dropped.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

GPIO routines

```
U2C_RESULT U2C_SetIoDirection (
      HANDLE hDevice,
      ULONG Value,
      ULONG Mask
);
```

The *U2C_SetIoDirection* function configures input/output direction of the GPIO port pins.

Parameters:

hDevice

Handle to the I2CBridge device.

Value

An unsigned long value specifying the direction of the GPIO.
Value is treated as unsigned long 0xXXCCBBAA, where CC, BB and AA correspond to the C, B and A port pins:

AA bits 7..0 correspond to Port A pins 7..0
BB bits 7..0 correspond to Port B pins 7..0
CC bits 7..0 correspond to Port C pins 7..0
XX bits 7..0 reserved

Bit set to 1 indicates configuration of the corresponding pin as output.
Bit set to 0 indicates configuration of the corresponding pin as input.

Mask

An unsigned long value specifying the data mask to use when modifying the GPIO pins direction. The mask value allows modification of the desired pins only, living rest of the pins unchanged. The bit mapping for *Mask* parameter is exactly the same as for *Value* parameter. Only direction of the pins with mask bit set to 1 will be changed.

Return value:

U2C_SUCCESS

The GPIO pins direction was successfully modified.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_GetIoDirection (
      HANDLE hDevice,
      ULONG *pValue,
);
```

The *U2C_GetIoDirection* function obtains current input/output direction of the GPIO port pins.

Parameters:

hDevice

Handle to the I2CBridge device.

**pValue*

A pointer to unsigned long to be filled with the direction of the GPIO pins. **pValue* is treated as unsigned long 0xXXCCBBAA, where CC, BB and AA correspond to the C, B and A port pins:
AA bits 7..0 correspond to Port A pins 7..0
BB bits 7..0 correspond to Port B pins 7..0
CC bits 7..0 correspond to Port C pins 7..0
XX bits 7..0 reserved

Bit set to 1 indicates configuration of the corresponding pin as output.
Bit set to 0 indicates configuration of the corresponding pin as input.

Return value:

U2C_SUCCESS

The GPIO pins direction was successfully read.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_IoWrite (
      HANDLE hDevice,
      ULONG Value,
      ULONG Mask
);
```

The *U2C_IoWrite* sets the output value of the GPIO port pins. Pins have to be configured as output using *U2C_SetIoDirection* function first.

Parameters:

hDevice

Handle to the I2CBridge device.

Value

An unsigned long value specifying the value to be set to the GPIO pins. *Value* is treated as unsigned long 0xXXCCBBAA, where CC, BB and AA correspond to the C, B and A port pins:
AA bits 7..0 correspond to Port A pins 7..0
BB bits 7..0 correspond to Port B pins 7..0
CC bits 7..0 correspond to Port C pins 7..0
XX bits 7..0 reserved

Mask

An unsigned long value specifying the data mask to use when modifying the GPIO pins output value. The mask value allows modification of the desired pins only, living rest of the pins unchanged. The bit mapping for *Mask* parameter is exactly the same as for *Value* parameter. Only value of the pins with mask bit set to 1 will be changed.

Return value:

U2C_SUCCESS

The GPIO pins output value was successfully modified.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_IoRead (
      HANDLE hDevice,
      ULONG *pValue,
);
```

The *U2C_IoRead* function obtains the value of the GPIO port pins.

Parameters:

hDevice

Handle to the I2CBridge device.

**pValue*

A pointer to unsigned long to be filled with the value of the GPIO pins.

**pValue* is treated as unsigned long 0xXXCCBBAA, where CC, BB and AA correspond to the C, B and A port pins:
AA bits 7..0 correspond to Port A pins 7..0
BB bits 7..0 correspond to Port B pins 7..0
CC bits 7..0 correspond to Port C pins 7..0
XX bits 7..0 reserved

Return value:

U2C_SUCCESS
The GPIO pins state was successfully read.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_SetSingleIoDirection (
      HANDLE hDevice,
      ULONG IoNumber,
      BOOL bOutput
);
```

The *U2C_SetSingleIoDirection* function configures input/output direction of the specified GPIO port pin.

Parameters:

hDevice
Handle to the I2CBridge device.
IoNumber
The number of the GPIO pin to change direction.
Numbers 0..7 correspond to Port A pins 0..7
Numbers 8..15 correspond to Port B pins 0..7
Number 16..23 correspond to Port C pins 0..7
bOutput
The direction of the pin.
bOutput = TRUE configures the pin for output.
bOutput = FALSE configures the pin for input.

Return value:

U2C_SUCCESS
The GPIO pin direction was successfully modified.
U2C_HARDWARE_NOT_FOUND
I2CBridge device referenced by *hDevice* handle was not found.
U2C_BAD_PARAMETER
IoNumber is out of range.

```
U2C_RESULT U2C_GetSingleIoDirection (
      HANDLE hDevice,
      ULONG IoNumber,
      BOOL *pbOutput
);
```

The *U2C_GetSingleIoDirection* function obtains input/output direction of the specified GPIO port pin.

Parameters:

hDevice

Handle to the I2CBridge device.

IoNumber

The number of the GPIO pin to obtain direction.
Numbers 0..7 correspond to Port A pins 0..7
Numbers 8..15 correspond to Port B pins 0..7
Number 16..23 correspond to Port C pins 0..7

pbOutput

A pointer to the boolean to be filled with the direction of the pin.
*pbOutput = TRUE indicates that the pin is configured for output.
*pbOutput = FALSE indicates that the pin is configured for input.

Return value:

U2C_SUCCESS

The GPIO pin direction was successfully read.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

U2C_BAD_PARAMETER

IoNumber is out of range.

```
U2C_RESULT U2C_SingleIoWrite (
      HANDLE hDevice,
      ULONG IoNumber,
      BOOL Value
);
```

The *U2C_SingleIoWrite* function sets the output value of the specified GPIO port pin. Pin must be configured as output using *U2C_SetIoDirection* or *U2C_SetSingleIoDirection* functions first.

Parameters:

hDevice

Handle to the I2CBridge device.

IoNumber

The number of the GPIO pin to set output value to.

Numbers 0..7 correspond to Port A pins 0..7
Numbers 8..15 correspond to Port B pins 0..7
Number 16..23 correspond to Port C pins 0..7

Value

The GPIO pin new output value.

Return value:

U2C_SUCCESS

The GPIO pin output value was successfully modified.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

U2C_BAD_PARAMETER

IoNumber is out of range.

```
U2C_RESULT U2C_SingleIoRead (
      HANDLE hDevice,
      ULONG IoNumber,
      BOOL *pValue
);
```

The *U2C_SingleIoRead* function obtains the value of the specified GPIO port pin.

Parameters:

hDevice

Handle to the I2CBridge device.

IoNumber

The number of the GPIO pin to obtain value from.
Numbers 0..7 correspond to Port A pins 0..7
Numbers 8..15 correspond to Port B pins 0..7
Number 16..23 correspond to Port C pins 0..7

**pValue*

A pointer to boolean to be filled with the GPIO pin state.

Return value:

U2C_SUCCESS

The GPIO pin state was successfully read.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

U2C_BAD_PARAMETER

IoNumber is out of range.

SPI configuration routines

```
U2C_RESULT U2C_SpiSetConfig(
      HANDLE hDevice,
      BYTE CPOL,
      BYTE CPHA
);
```

The *U2C_SpiSetConfig* function configures SPI bus clock polarity and phase.

Parameters:

hDevice

Handle to the I2CBridge device.

CPOL

Clock polarity value determines the CLK line idle state, where:
0 corresponds to “idle low”
1 corresponds to “idle high”

CPHA

Clock phase value determines the clock edge when the data is valid on the bus, where:
0 corresponds to valid data available on leading edge
1 corresponds to valid data available on trailing edge

Return value:

U2C_SUCCESS

The SPI bus was successfully configured.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_SpiGetConfig(
      HANDLE hDevice,
      BYTE *pCPOL,
      BYTE *pCPHA
);
```

The *U2C_SpiGetConfig* function obtains SPI bus configuration (clock polarity and phase).

Parameters:

hDevice

Handle to the I2CBridge device.

CPOL

A pointer to the byte to be filled with current SPI bus clock polarity setting.
Clock polarity determines the CLK line idle state, where:

0 corresponds to “idle low”
1 corresponds to “idle high”

CPHA

A pointer to byte to be filled with current SPI bus clock phase setting. Clock phase value determines the clock edge when the data is valid on the bus, where:
0 corresponds to valid data available on leading edge
1 corresponds to valid data available on trailing edge

Return value:

U2C_SUCCESS

The SPI bus configuration was successfully obtained.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_SpiSetFreq(
      HANDLE hDevice,
      BYTE Frequency
);
```

The *U2C_SpiSetFreq* function configures SPI bus speed.

Parameters:

hDevice

Handle to the I2CBridge device.

Frequency

The frequency of SPI bus, where:
0 corresponds to SPI bus frequency of 200 kHz.
1 corresponds to SPI bus frequency of 100 kHz.
1+n corresponds to the SPI bus clock period equal to 10+2*n µS.

For convenience following constants were introduced:

constant	frequency
U2C_SPI_FREQ_200KHZ	200 kHz
U2C_SPI_FREQ_100KHZ	100 kHz
U2C_SPI_FREQ_83KHZ	83 kHz
U2C_SPI_FREQ_71KHZ	71 kHz
U2C_SPI_FREQ_62KHZ	62 kHz
U2C_SPI_FREQ_50KHZ	50 kHz
U2C_SPI_FREQ_25KHZ	25 kHz
U2C_SPI_FREQ_10KHZ	10 kHz
U2C_SPI_FREQ_5KHZ	5 kHz
U2C_SPI_FREQ_2KHZ	2 kHz

Return value:

U2C_SUCCESS

The frequency value was successfully set.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_SpiGetFreq(
      HANDLE hDevice,
      BYTE *pFrequency
);
```

The *U2C_SpiGetFreq* function obtains SPI bus speed.

Parameters:

hDevice

Handle to the I2CBridge device.

pFrequency

A pointer to byte to be filled with the current SPI bus frequency, where:
0 corresponds to SPI bus frequency of 200 kHz.
1 corresponds to SPI bus frequency of 100 kHz.
1+n corresponds to the SPI bus clock period equal to 10+2*n µS.

For convenience following constants were introduced:

constant	frequency
U2C_SPI_FREQ_200KHZ	200 kHz
U2C_SPI_FREQ_100KHZ	100 kHz
U2C_SPI_FREQ_83KHZ	83 kHz
U2C_SPI_FREQ_71KHZ	71 kHz
U2C_SPI_FREQ_62KHZ	62 kHz
U2C_SPI_FREQ_50KHZ	50 kHz
U2C_SPI_FREQ_25KHZ	25 kHz
U2C_SPI_FREQ_10KHZ	10 kHz
U2C_SPI_FREQ_5KHZ	5 kHz
U2C_SPI_FREQ_2KHZ	2 kHz

Return value:

U2C_SUCCESS

The frequency value was successfully retrieved.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

SPI data transfer routines

```
U2C_RESULT U2C_SpiReadWrite(
      HANDLE hDevice,
      BYTE *pOutBuffer
      BYTE *pInBuffer
      unsigned short Length
);
```

The *U2C_SpiReadWrite* function shifts out (writes) and in (reads) a stream of bytes to/from the SPI slave device.

Parameters:

hDevice

Handle to the I2CBridge device.

pOutBuffer

Pointer to the buffer containing the data to be shifted out to the slave SPI device.

pInBuffer

Pointer to the buffer that receives the data shifted in from the slave SPI device.

Length

Number of bytes to be transferred via SPI bus.

Return value:

U2C_SUCCESS

The data was successfully transmitted via SPI bus.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_SpiWrite(
      HANDLE hDevice,
      BYTE *pOutBuffer
      unsigned short Length
);
```

The *U2C_SpiWrite* function shifts out (writes) a stream of bytes to the SPI slave device.

Parameters:

hDevice

Handle to the I2CBridge device.

pOutBuffer

Pointer to the buffer containing the data to be shifted out to the slave SPI device.

Length

Number of bytes to be shifted out to the slave SPI device.

Return value:

U2C_SUCCESS

The data was successfully written.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.

```
U2C_RESULT U2C_SpiRead(
      HANDLE hDevice,
      BYTE *pInBuffer
      unsigned short Length
);
```

The *U2C_SpiRead* function shifts in (reads) a stream of bytes from the SPI slave device.

Parameters:

hDevice

Handle to the I2CBridge device.

pInBuffer

Pointer to the buffer that receives the data shifted in from the SPI slave device.

Length

Number of bytes to be shifted in.

Return value:

U2C_SUCCESS

The data was successfully read.

U2C_HARDWARE_NOT_FOUND

I2CBridge device referenced by *hDevice* handle was not found.