



Technical support:  
+49 (0)7223 / 9493-0

## **Software description**

# **ADDIDRIVER**

## **Pressure**

4<sup>th</sup> edition 03/2005



<b>1</b>	<b>INTRODUCTION.....</b>	<b>1</b>
<b>2</b>	<b>SOFTWARE FUNCTIONS.....</b>	<b>2</b>
1)	b_ADDIDATA_GetNumberOfPressureChannels (..).....	2
2)	b_ADDIDATA_GetNumberOfPressureModules (..) .....	3
3)	b_ADDIDATA_GetNumberOfPressureChannelsForTheModule (..) .....	4
4)	b_ADDIDATA_GetPressureChannelInformation (..) .....	5
5)	b_ADDIDATA_GetPressureChannelReferenceVoltage (..).....	13
6)	b_ADDIDATA_GetPressureChannelGainFactor (..) .....	14
7)	b_ADDIDATA_InitPressureChannel (..).....	15
8)	b_ADDIDATA_Read1PressureChannel (..).....	17
9)	b_ADDIDATA_ReadMorePressureChannels (..).....	19
10)	b_ADDIDATA_ConvertDigitalToRealPressureValue (..) .....	21
11)	b_ADDIDATA_ConvertMoreDigitalToRealPressureValues (..) .....	22
12)	b_ADDIDATA_InitPressureSCAN (..) .....	23
13)	b_ADDIDATA_StartPressureSCAN (..) .....	26
14)	b_ADDIDATA_GetPressureSCANStatus (..) .....	27
15)	b_ADDIDATA_ConvertDigitalToRealPressureValueSCAN (..) .....	28
16)	b_ADDIDATA_StopPressureSCAN (..).....	29
17)	b_ADDIDATA_ClosePressureSCAN (..).....	30
18)	b_ADDIDATA_ReleasePressureChannel (..).....	31
19)	b_ADDIDATA_ConvertBarToPa (..).....	32
20)	b_ADDIDATA_ConvertBarToPsi (..) .....	33
21)	b_ADDIDATA_TestPressureAsynchronousFIFOFull .....	34
22)	b_ADDIDATA_GetPressureChannelReferenceVoltage (..).....	35
23)	b_ADDIDATA_GetPressureChannelGainFactor (..) .....	36

## Tables

Table 1-1: Type Declaration for Windows 98/NT/2000/XP .....	1
Table 2-1: Gain for a max. voltage of 5V .....	16
Table 2-2: Selection of the converting time.....	18
Table 2-3: Time selection of a SCAN.....	24
Table 2-4: SCAN time mode.....	24
Table 2-5: SCAN mode.....	24
Table 2-6: Trigger mode .....	25

# 1 INTRODUCTION

**i**

## IMPORTANT!

Note the following conventions in the text:

Function: "b\_ADDIDATA\_GetNumberOfAnalogInputs"  
Variable *dw\_DriverHandle*

**Table 1-1: Type Declaration for Windows 98/NT/2000/XP**

	Borland C	Microsoft C	Borland Pascal	Microsoft Visual Basic Windows
<b>VOID</b>	void	void	pointer	any
<b>BYTE</b>	unsigned char	unsigned char	byte	integer
<b>INT</b>	int	int	integer	integer
<b>WORD</b>	unsigned short int	unsigned short int	long	long
<b>DWORD</b>	long	long	longint	long
<b>PBYTE</b>	unsigned char *	unsigned char *	var byte	integer
<b>PINT</b>	int *	int *	var integer	integer
<b>PWORD</b>	unsigned short int *	unsigned short int *	var long	long
<b>PCHAR</b>	char *	char *	var string	string
<b>PDWORD</b>	long *	long *	var longint	long
<b>DOUBLE</b>	double	double	double	double

## 2 SOFTWARE FUNCTIONS

### 1) b\_ADDIDATA\_GetNumberOfPressureChannels (..)

### Syntax:

<Return value> = b\_ADDIDATA\_GetNumberOfPressureChannels  
(DWORD dw\_DriverHandle,  
WORD pw\_ChannelNbr)

### Parameters:

**- Input:**

DWORD	<i>dw_DriverHandle</i>	Driver handle
-------	------------------------	---------------

**- Output:**

PWORD	<i>pw_ChannelNbr</i>	Returns the number of pressure inputs
-------	----------------------	---------------------------------------

### Task:

Returns the number of pressure inputs.

### Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
WORD	w_ChannelNbr;

```
b_ReturnValue = b_ADDIDATA_GetNumberOfPressureChannels
                (dw_DriverHandle,
                 &w_ChannelNbr);
```

### Return Value:

1: No error

0: Error by calling up the function.

Use the function "i\_ADDIDATA\_GetLastError", to find the error source.

**2) b\_ADDIDATA\_GetNumberOfPressureModules (..)****Syntax:**

<Return value> = b\_ADDIDATA\_GetNumberOfPressureModules  
(DWORD dw\_DriverHandle,  
PWORD pw\_ModuleNbr)

**Parameters:****- Input:**

DWORD *dw\_DriverHandle*      Driver handle

**- Output:**

PWORD *pw\_ModuleNbr*      Returns the number of pressure modules

**Task:**

Returns the number of pressure modules.

**Calling convention:**ANSI C :

BYTE      b\_ReturnValue;  
DWORD     dw\_DriverHandle;  
WORD      w\_ModuleNbr;

b\_ReturnValue = b\_ADDIDATA\_GetNumberOfPressureModules  
(dw\_DriverHandle,  
&w\_ModuleNbr);

**Return Value:**

1: No error

0: Error by calling up the function. Use the function

"i\_ADDIDATA\_GetLastError", to find the error source.

### 3) b\_ADDIDATA\_GetNumberOfPressureChannelsForTheModule (..)

### Syntax:

```
<Return value> = b_ADDIDATA_GetNumberOfPressureChannelsForTheModule
                                (DWORD    dw_DriverHandle,
                                WORD      w_Module,
                                PWORD     pw_ChannelNbr)
```

### Parameters:

**- Input:**

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Pressure module

**- Output:**

<b>WORD</b> <i>pw_ChannelNbr</i>	Returns the number of pressure channels for the given module.
----------------------------------	---

### Task:

Returns the number of pressure channels for the module w\_Module.

### Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
WORD	w_ChannelNbr;

[illegible]

### Return Value:

1: No error  
0: Error by calling up the function. Use the function  
"i\_ADDIDATA\_GetLastError", to find the error source.



**4) b\_ADDIDATA\_GetPressureChannelInformation (..)****Syntax:**

```
<Return value> = b_ADDIDATA_GetPressureChannelInformation
                (DWORD    dw_DriverHandle,
                 WORD     w_Channel,
                 pstr_GetAnalogMeasureInformation ps_ChannelInformation,
                 DWORD    dw_StructSize)
```

**Parameters:****- Input:**

DWORD *dw\_DriverHandle*            Driver handle  
 DWORD *dw\_StructSize*            Size of the structure entered in the pointer.

**- Output:**

pstr\_GetAnalogMeasureInformation *ps\_ChannelInformation*  
 Channel information.

**structure returned :**

```
typedef struct
```

```
{
  BYTE    b_InputsResolution;            Returns the input resolution
                                             8: 8-bit resolution
                                             16: 16-bit resolution, ...
  BYTE    b_CanUsedInterrupt;            0: No interrupt can be generated
                                             1: Interrupt can be generated
  BYTE    b_UnipolarBipolarConfigurable
                                             0: unipolar/bipolar hardware configurable
                                             1: Unipolar/bipolar configurable
  BYTE    b_UnipolarAvailable;           0: Cannot configure in unipolar
                                             1: Can configure in unipolar
  BYTE    b_BipolarAvailable;            0: Cannot configure in bipolar
                                             1: Can configure in bipolar
  BYTE    b_SingleDifferenceSelected;    0: single mode selected
                                             1: Differential mode selected
  BYTE    b_DCCouplingAvailable;        0: Cannot configure the direct coupling (DC)
                                             1: Can configure direct coupling (DC)
  BYTE    b_ACCouplingAvailable;        0: Cannot configure alternative coupling (AC)
                                             1: Can configure alternative coupling (AC)
  BYTE    b_BufferAvailable;            0: No hardware buffer available
                                             1: Hardware buffer available
  BYTE    b_CanGeneratedWarning;        0: Alarm not available
                                             1: Alarm available
  BYTE    b_CurrentSourceSetBySoft;    0: The current source must not be
                                             activated by software
                                             1: The current source must be
                                             activated by software.
  BYTE    b_NbrOfGain;                  Returns the number of gain values available
  DWORD    dw_ModuleNumber              Returns the module number of the channel
  DOUBLE d_GainAvailable[255];        Defines the available gain value
  BYTE    b_OffsetRangeAvailable;       0: Offset not allowed
                                             1: Offset allowed
}
```

BYTE	<i>b_Reserved2;</i>	
WORD	<i>w_OffsetRangeResolution;</i>	Offset resolution
		8: 8-bit resolution
		16: 16-bit resolution, ...
BYTE	<i>b_OffsetRangeDenominator;</i>	Offset denominator step value
BYTE	<i>b_OffsetRangeNumerator;</i>	Offset numerator step value
BYTE	<i>b_OpenInputDetection;</i>	0: Open input detection not available
		1: Open input detection available
BYTE	<i>b_ShortCircuitDetection;</i>	0: Short-circuit detection not available
		1: Short-circuit detection available
DOUBLE	<i>d_Umax;</i>	Returns the maximum input voltage value in V or A
DOUBLE	<i>d_URef;</i>	Returns the reference input voltage value in V or A
BYTE	<i>b_InputType;</i>	Selected user input type ADDIDATA_PRESSURE_CHANNEL
BYTE	<i>b_TypePrecision;</i>	Precision of the input Not used; Returns "0"
WORD	<i>w_InputTypeValue;</i>	Not used; Returns "0"

#### Module Information

BYTE	<i>b_AutoCalibration;</i>	0: Auto-calibration not available
		1: Auto-calibration available
BYTE	<i>b_CJCAvailable;</i>	0: Without CJC (cold junction compensation)
		1: With CJC
BYTE	<i>b_ConversionMustSetting;</i>	0: The conversion time for a single acquisition is determined by hardware (jumper)
		1: The conversion time for a single acquisition is determined by software
BYTE	<i>b_ConversionCalcType;</i>	0: Binary type (XX000,XX001,XX010,XX011,XX100)
		1: Multiple type (60, 120, 240, 480, 960, ...)
BYTE	<i>b_ConversionUnitType;</i>	0: Time unit (ns, $\mu$ s, ms, s, ...)
		1: Frequency unit (MHz, kHz, Hz, mHz, ...)
BYTE	<i>b_AvailableConversionUnit;</i>	<b>For time unit:</b>
		D0: 0: ns not available
		1: ns available
		D1: 0: $\mu$ s not available
		1: $\mu$ s available
		D2: 0: ms not available
		1: ms available
		D3: 0: s not available
		1: s available
		<b>For frequency unit:</b>
		D0: 0: MHz not available
		1: MHz available
		D1: 0: kHz not available
		1: kHz available
		D2: 0: Hz not available
		1: Hz available
		D3: 0: mHz not available
		1: mHz available

WORD	<i>w_ConversionResolution</i> ;	8 : 8-bit resolution 16: 16-bit resolution, ...
WORD	<i>w_MinConversionTime</i> ;	Minimum conversion time. 7000: 7000(ns) 10000: 10000(ns), ...
WORD	<i>w_ConversionStep</i> ;	Conversion time steps 20: 20 steps 50: 50 steps, ...
BYTE	<i>b_SEQArrayAvailable</i> ;	0: Sequence array not available 1: Sequence array available
BYTE	<i>b_SEQConfigurable</i> ;	0: Sequence fixed 1: Sequence configurable
BYTE	<i>b_SEQHardwareTriggerAvailable</i> ;	0: Hardware trigger not available 1: Hardware trigger available
BYTE	<i>b_SEQHardwareTriggerHighAvailable</i> ;	0: Hardware high trigger level not available 1: Hardware high trigger level available
BYTE	<i>b_SEQHardwareTriggerLowAvailable</i> ;	0: Low level not available for hardware trigger 1: Low level available for hardware trigger
BYTE	<i>b_SEQHardwareTriggerAvailableMode</i> ;	001: External trigger start a 1 DMA cycle 010: Each trigger starts a sequence 100: Each trigger starts a DMA cycle
BYTE	<i>b_SEQHardwareGateAvailable</i> ;	0: Hardware gate not available 1: Hardware gate available
BYTE	<i>b_SEQHardwareGateHighAvailable</i> ;	0: High level not available for hardware gate 1: High level available for hardware gate
BYTE	<i>b_SEQHardwareGateLowAvailable</i> ;	0: Low level not available for hardware gate 1: Low level available for hardware gate
BYTE	<i>b_SEQClrIndexAvailable</i> ;	0: It is not possible to restart the acquisition with the next selected channel in the sequence 1: It's possible to restart the acquisition with the next selected channel in the sequence
WORD	<i>w_SEQAcquisitionMode</i> ;	00: The acquisition can operate in single mode 01: The acquisition can operate in continuous mode 11: The acquisition can operate in single and continuous mode
BYTE	<i>b_DMAAvailable</i> ;	0: Board cannot use the DMA 1: Board can use the DMA
BYTE	<i>b_DualDMAChannel</i> ;	0: Only 1 DMA channel can be used 1: Board can use 2 DMA channels
BYTE	<i>b_SEQCounterAvailable</i> ;	0: DMA conversion counter not available 1: DMA conversion counter available
BYTE	<i>b_SEQCounterMode</i> ;	01: Counter can count the number of sequences 10: Counter can count the number of DMA cycles

		11: Counter can count the number of DMA cycles or the number of sequences
BYTE	<i>b_SEQCommonGain;</i>	Defines if the gain can differ for each input 0: Gain is the same for each input 1: Gain can differ for each input.
BYTE	<i>b_SEQCommonPolarity;</i>	Defines if the polarity can differ for each input 0: Polarity is the same for each input 1: Polarity can differ for each input.
BYTE	<i>b_SEQCommonOffsetRange;</i>	Defines if the offset can differ for each input 0: Offset is the same for each input 1: Offset can differ for each input.
BYTE	<i>b_SEQCommonCoupling;</i>	
WORD	<i>w_SEQCounterResolution;</i>	Counter resolution 8 : 8-bit resolution 16: 16-bit resolution, ...
BYTE	<i>b_SEQDelayTimeConfigurable;</i>	0: Delay after each sequence not available 1: Delay after each sequence available
BYTE	<i>b_SEQDelayMode;</i>	
BYTE	<i>b_SEQDelayCalcType;</i>	0: Binary type (XX000,XX001,XX010,XX011,XX100) 1: Multiple type (60,120,240,480,960, ...)
BYTE	<i>b_SEQDelayTimeUnitType;</i>	0: Time unit (ns,µs,ms,s, ...) 1: Frequency unit (MHz,kHz,Hz,mHz, ...)
BYTE	<i>b_SEQDelayValueType;</i>	0: Value to write in the register is in s or in Hz 1: The step multiplier is to be written in the register
BYTE	<i>b_SEQDelayTimeUnit;</i>	<b>For time unit:</b> D0: 0: ns not available 1: ns available D1: 0: µs not available 1: µs available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available <b>For frequency unit:</b> D0: 0: MHz not available 1: MHz available D1: 0: kHz not available 1: kHz available D2: 0: Hz not available 1: Hz available D3: 0: mHz not available 1: mHz available
WORD	<i>w_SEQDelayTimeResolution;</i>	Delay time resolution 8: 8-bit resolution 16: 16-bit resolution, ...
WORD	<i>w_SEQMinDelayTime;</i>	Minimum delay time. 7000: 7000(ns)

		10000: 10000(ns), ...
WORD	<i>w_SEQDelayTimeStep</i> ;	Conversion delay time steps
		20: 20 steps
		50: 50 steps, ...
BYTE	<i>b_SEQUnipolarBipolarConfigurable</i> ;	0: Unipolar/Bipolar hardware configurable
		1: Unipolar/Bipolar configurable
BYTE	<i>b_SEQUnipolarAvailable</i> ;	0: Cannot configure in unipolar
		1: Can configure in unipolar
BYTE	<i>b_SEQBipolarAvailable</i> ;	0: Cannot configure in bipolar
		1: Can configure in bipolar
BYTE	<i>b_SEQDCCouplingAvailable</i> ;	0: Cannot configure the direct coupling (DC)
		1: Can configure direct coupling (DC)
BYTE	<i>b_SEQACCouplingAvailable</i> ;	0: Cannot configure alternative coupling (AC)
		1: Can configure alternative coupling (AC)
BYTE	<i>b_SEQBufferAvailable</i> ;	0: No hardware buffer available
		1: Hardware buffer available
BYTE	<i>b_SEQNbrOfGain</i> ;	Returns the number of gain values available
BYTE	<i>b_Reserved3</i> ;	
WORD	<i>w_Reserved4</i> ;	
DWORD	<i>dw_Reserved5</i> ;	
DOUBLE	<i>d_SEQGainAvailable[255]</i> ;	Defines the available gain value
BYTE	<i>b_SEQOffsetRangeAvailable</i> ;	0: Offset not allowed
		1: Offset allowed
BYTE	<i>b_Reserved6</i> ;	
WORD	<i>w_SEQOffsetRangeResolution</i> ;	Offset resolution
		8 : 8-bit resolution
		16: 16-bit resolution, ...
BYTE	<i>b_SEQOffsetRangeDenominator</i> ;	Offset denominator step value
BYTE	<i>b_SEQOffsetRangeNumerator</i> ;	Offset numerator step value
BYTE	<i>b_SCANAvailable</i> ;	0: SCAN not available
		1: SCAN available
BYTE	<i>b_SCANConfigurable</i> ;	0: SCAN fixed
		1: SCAN configurable (The first and the last channel can be given)
BYTE	<i>b_SCANHardwareTriggerAvailable</i> ;	0: Hardware trigger not available
		1: Hardware trigger available
BYTE	<i>b_SCANHardwareTriggerHighAvailable</i> ;	0: Hardware high trigger level not available
		1: Hardware high trigger level available
BYTE	<i>b_SCANHardwareTriggerLowAvailable</i> ;	0: Hardware low trigger level not available
		1: Hardware low trigger level available
BYTE	<i>b_SCANHardwareTriggerAvailableMode</i> ;	0001: External trigger starts each acquisition from a SCAN
		0010: Each trigger start each SCAN
		1000: the first Trigger starts the SCAN cycle

BYTE	<i>b_SCANHardwareGateAvailable;</i>	0: Hardware gate not available 1: Hardware gate available
BYTE	<i>b_SCANHardwareGateHighAvailable;</i>	0: Hardware high gate level not available 1: Hardware high gate level available
BYTE	<i>b_SCANHardwareGateLowAvailable;</i>	0: Low level not available for hardware gate 1: Low level available for hardware gate
BYTE	<i>b_SCANClrIndexAvailable;</i>	0: It is not possible to restart the acquisition with the next selected channel of SCAN 1: It is possible to restart the acquisition with the next selected channel of SCAN
WORD	<i>w_SCANAcquisitionMode;</i>	00: The acquisition can operate in single mode 01: The acquisition can operate in continuous mode 11: The acquisition can operate in single and continuous mode
BYTE	<i>b_SCANCounterAvailable;</i>	0: SCAN conversion counter not available 1: SCAN conversion counter available
BYTE	<i>b_SCANCounterMode;</i>	01: Counter can count the number of SCANS
BYTE	<i>b_SCANCommonGain;</i>	Defines if the gain can differ for each input 0: Gain is the same for each input 1: Gain can differ for each input.
BYTE	<i>b_SCANCommonPolarity;</i>	Defines if the polarity can differ for each input 0: Polarity is the same for each input 1: Polarity can be differ for each input.
BYTE	<i>b_SCANCommonOffsetRange;</i>	Defines if the offset can differ for each input 0: Offset is the same for each input 1: Offset can differ for each input.
BYTE	<i>b_SCANCommonCoupling;</i>	
WORD	<i>w_SCANCounterResolution;</i>	Counter resolution 8: 8-bit resolution 16: 16-bit resolution, ...
BYTE	<i>b_SCANDelayTimeConfigurable;</i>	0: Delay after each sequence not available 1: Delay after each sequence available
BYTE	<i>b_SCANDelayMode;</i>	
BYTE	<i>b_SCANDelayCalcType;</i>	0: Binary type (XX000,XX001,XX010,XX011,XX100) 1: Multiple type (60,120,240,480,960, ...)
BYTE	<i>b_SCANDelayTimeUnitType;</i>	0: Time unit (ns, $\mu$ s,ms,s, ...) 1: Frequency unit (MHz,kHz,Hz,mHz, ...)
BYTE	<i>b_SCANDelayValueType;</i>	0: Value to write in the register is the value in s or in Hz 1: The step multiplier is to be written in the register

BYTE	<i>b_SCANDelayTimeUnit;</i>	<b>For time unit:</b> D0: 0: ns not available 1: ns available D1: 0: $\mu$ s not available 1: $\mu$ s available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available <b>For frequency unit:</b> D0: 0: MHz not available 1: MHz available D1: 0: kHz not available 1: kHz available D2: 0: Hz not available 1: Hz available D3: 0: mHz not available 1: mHz available
WORD	<i>w_SCANDelayTimeResolution;</i>	Delay time resolution 8: 8-bit resolution 16: 16-bit resolution, ...
WORD	<i>w_SCANMinDelayTime;</i>	Minimum delay time. 7000: 7000(ns) 10000: 10000(ns), ...
WORD	<i>w_SCANDelayTimeStep;</i>	Conversion delay time steps 20: 20 steps 50: 50 steps, ...
BYTE	<i>b_SCANUnipolarBipolarConfigurable;</i>	0: Unipolar/bipolar hardware configurable 1: Unipolar/bipolar configurable
BYTE	<i>b_SCANUnipolarAvailable;</i>	0: Cannot configure in unipolar 1: Can configure in unipolar
BYTE	<i>b_SCANBipolarAvailable;</i>	0: Cannot configure in bipolar 1: Can configure in bipolar
BYTE	<i>b_SCANDCCouplingAvailable;</i>	0: Cannot configure the direct coupling (DC) 1: Can configure direct coupling (DC)
BYTE	<i>b_SCANACCouplingAvailable;</i>	0: Cannot configure alternative coupling (AC) 1: Can configure alternative coupling (AC)
BYTE	<i>b_SCANBufferAvailable;</i>	0: No hardware buffer available 1: Hardware buffer available
BYTE	<i>b_SCANNbOfGain;</i>	Returns the number of gain values available
BYTE	<i>b_Reserved7;</i>	
WORD	<i>w_Reserved8;</i>	
DOUBLE	<i>d_SCANGainAvailable[255];</i>	Defines the available gain values
BYTE	<i>b_SCANOffsetRangeAvailable;</i>	0: Offset not allowed 1: Offset allowed
BYTE	<i>b_Reserved9;</i>	

WORD	<i>w_SCANOffsetRangeResolution</i> ; Offset resolution 8 : 8-bit resolution 16: 16-bit resolution, ...
BYTE	<i>b_SCANOffsetRangeDenominator</i> ; Offset denominator step value
BYTE	<i>b_SCANOffsetRangeNumerator</i> ; Offset numerator step value
WORD	<i>w_Reserved10</i> ; }str_GetAnalogMeasureInformation,*pstr_GetAnalogMeasureInformation;

### Task:

Returns information about the pressure input channels.

### Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
str_GetAnalogMeasureInformation s_ChannelInformation;

b_ReturnValue = b_ADDIDATA_GetPressureChannelInformation
                (dw_DriverHandle,
                 0,
                 &s_ChannelInformation,
                 sizeof (str_GetAnalogMeasureInformation));

```

### Return Value:

1: No error  
0: Error by calling up the function. Use the function  
"i\_ADDIDATA\_GetLastError", to find the error source.



**5) b\_ADDIDATA\_GetPressureChannelReferenceVoltage (..)****Syntax:**

```
<Return value> = b_ADDIDATA_GetPressureChannelReferenceVoltage
                                (DWORD      dw_DriverHandle,
                                 WORD        w_ChannelNbr,
                                 DOUBLE      *pd_ReferenceVoltage)
```

**Parameters:****- Input:**

DWORD dw_DriverHandle	Driver handle
DWORD w_ChannelNbr	Number of the pressure input channel for which the reference voltage is requested.

**- Output:**

DOUBLE *pd_ReferenceVoltage	Returns the reference voltage (V).
-----------------------------	------------------------------------

**Task:**

Returns the reference voltage (pd\_ReferenceVoltage) of the selected channel (w\_ChannelNbr).

**Calling convention:**ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DOUBLE    d_ReferenceVoltage;
```

```
b_ReturnValue = b_ADDIDATA_GetPressureChannelReferenceVoltage
                                (dw_DriverHandle,
                                 0,
                                 &d_ReferenceVoltage);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function i\_ADDIDATA\_GetLastError" to find the error source.

**6) b\_ADDIDATA\_GetPressureChannelGainFactor (..)****Syntax:**

```
<Return value> = b_ADDIDATA_GetPressureChannelGainFactor
                                (DWORD      dw_DriverHandle,
                                WORD        w_ChannelNbr,
                                DOUBLE      *pd_GainFactor)
```

**Parameters:****- Input:**

DWORD dw_DriverHandle	Driver handle
DWORD w_ChannelNbr	Number of the pressure input channel for which the gain correction factor is requested

**- Output:**

DOUBLE* pd_GainFactor	Returns the gain correction factor
-----------------------	------------------------------------

**Task:**

Returns the gain correction factor (pd\_GainFactor) allocated to the selected channel (w\_ChannelNbr).

**Calling convention:**ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DOUBLE    d_GainFactor;
```

```
b_ReturnValue = b_ADDIDATA_GetPressureChannelGainFactor
                (dw_DriverHandle,
                 0,
                 &d_GainFactor);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function i\_ADDIDATA\_GetLastError", to find the error source.

**7) b\_ADDIDATA\_InitPressureChannel (..)**

<Return value> = b\_ADDIDATA\_InitPressureChannel  
 (DWORD dw\_DriverHandle,  
 WORD w\_Channel,  
 pstr\_InitPressureChannel ps\_InitParameters,  
 DWORD dw\_StructSize)

**Parameters:****- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Channel</i>	Number of the pressure channel to be initialised
pstr_InitPressureChannel ps_InitParameters	
DWORD dw_StructSize	Size of the structure entered in the pointer. The structure must contain the following information

typedef struct

{	
DOUBLE <i>d_Gain</i>	Gain of the channel. Refer to Table 2-1
DOUBLE <i>d_OffsetVoltage</i>	Sensor offset voltage (mV). Refer to the sensor documentation
DOUBLE <i>d_SensorSensibility</i>	Sensor sensibility (mV/V/bar). Refer to the sensor documentation
} str_InitPressureChannel	

**- Output:**

No output signal has occurred.

**Task:**

Initialises the selected pressure channel.

**Table 2-1: Gain for a max. voltage of 5V**

Gain selection	Min. pressure input voltage	Max. pressure input voltage
1	0 V	5V
2	0 V	2.5V
4	0 V	1.25 V
5	0 V	1 V
8	0 V	0.625 V
10	0 V	0.5 V
16	0 V	0.3125 V
20	0 V	0.250 V
32	0 V	156.25 mV
50	0 V	100 mV
64	0 V	78.125 mV
100	0 V	50 mV
128	0 V	39,0625 mV
...		

**Calling convention:**ANSI C:

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
str_InitPressureChannel s_InitParameters;

```

```

b_ReturnValue = b_ADDIDATA_ InitPressureChannel
                (dw_DriverHandle,
                 0,
                 &s_InitParameters,
                 sizeof (str_InitPressureChannel));

```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function

"i\_ADDIDATA\_GetLastError", to find the error source.

**8) b\_ADDIDATA\_Read1PressureChannel (..)****Syntax:**

```
<Return value> = b_ADDIDATA_Read1PressureChannel
                (DWORD    dw_DriverHandle,
                 WORD     w_Channel,
                 DWORD    dw_ConvertingTime,
                 BYTE     b_ConvertingTimeUnit,
                 BYTE     b_InterruptFlag,
                 PDWORD   pdw_ChannelValue)
```

**Parameters:****- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>w_Channel</i>	Number of the pressure channel to be read
DWORD <i>dw_ConvertingTime</i>	Converting time. Refer to table 2-2
BYTE <i>b_ConvertingTimeUnit</i>	Determines the converting time unit. Refer to table 2-2
BYTE <i>b_InterruptFlag</i>	ADDIDATA_DISABLE: No interrupt is generated after the conversion and the variable <i>pdw_ChannelValue</i> returns the digital value. ADDIDATA_ENABLE: An interrupt is generated after the conversion. The digital value of the selected channel is returned through the interrupt function. Refer to the chapter "Interrupt"

**- Output:**

PDWORD <i>pdw_ChannelValue</i>	Returns the digital value of the selected channel * <i>pdw_ChannelValue</i> [0] = digital value of the channel * <i>pdw_ChannelValue</i> [1] = digital value of the calibration offset (if available) * <i>pdw_ChannelValue</i> [2] = digital value of the calibration gain (if available)
--------------------------------	---

**Task:**

Returns the digital value (*pdw\_ChannelValue*) of the selected channel (*w\_Channel*). To obtain the real pressure value, you must call up the

"**b\_ADDIDATA\_ConvertDigitalToRealPressureValue (...)**" function.

*dw\_ConvertingTime* and *b\_ConvertingTimeUnit* determine the converting time.

**Table 2-2: Selection of the converting time**

<i>b_ConvertingTimeUnit</i>	Unit selection	<i>dw_ConvertingTime</i>	Converting time
0	ns or MHz	10	10 ns or MHz
		20	20 ns or MHz
		300	300 ns or MHz
		1000	1000 ns or MHz
		22222	22222 ns or MHz
1	$\mu$ s or KHz	10	10 $\mu$ s or KHz
		20	20 $\mu$ s or KHz
		300	300 $\mu$ s or KHz
		1000	1000 $\mu$ s or KHz
		22222	22222 $\mu$ s or KHz
2	ms or Hz	10	10 ms or Hz
		20	20 ms or Hz
		300	300 ms or Hz
		1000	1000 ms or Hz
		22222	22222 ms or Hz
3	s or mHz	10	10 s or mHz
		20	20 s or mHz
		300	300 s or mHz
		1000	1000 s or mHz
		22222	22222 s or mHz

**Calling convention:**ANSI C:

BYTE        b\_ReturnValue;  
 DWORD      dw\_DriverHandle;  
 DWORD      dw\_ChannelValue[4];

b\_ReturnValue = b\_ADDIDATA\_Read1PressureChannel

(dw\_DriverHandle,  
 0,  
 10,  
 1,  
 ADDIDATA\_DISABLE,  
 dw\_ChannelValue);

**Return Value:**

1: No error  
 0: Error by calling up the function. Use the function  
 "i\_ADDIDATA\_GetLastError", to find the error source.

**i****IMPORTANT!**

The **interrupt mask** for the function is detailed in the "**Interrupt**" function description. (Tables 2-1 and 2-2).

**9) b\_ADDIDATA\_ReadMorePressureChannels (..)****Syntax:**

```
<Return value> = b_ADDIDATA_ReadMorePressureChannels
                (DWORD    dw_DriverHandle,
                 WORD     w_FirstChannel,
                 WORD     w_LastChannel,
                 DWORD    dw_ConvertingTime,
                 BYTE     b_ConvertingTimeUnit,
                 BYTE     b_InterruptFlag,
                 PDWORD   pdw_ChannelArrayValue)
```

**Parameters:****- Input:**

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_FirstChannel</i>	Selection of the first channel to be read
WORD	<i>w_LastChannel</i>	Selection of the last channel to be read
DWORD	<i>dw_ConvertingTime</i>	Converting time. Refer to table 2-2
BYTE	<i>b_ConvertingTimeUnit</i>	Determines the unit of the converting time. Refer to table 2-2
BYTE	<i>b_InterruptFlag</i>	ADDIDATA_DISABLE: No interrupt is generated after the last conversion and the variable <i>pdw_ChannelArrayValue</i> returns the digital value. ADDIDATA_ENABLE: An interrupt is generated after the last conversion. The digital value of all selected channels is returned through the interrupt function. Refer to the chapter "Interrupt"

**- Output:**

PDWORD	<i>pdw_ChannelArrayValue</i>	Returns the digital value of all selected channels
		* <i>pdw_ChannelValue</i> [0] = digital value of first channel
		* <i>pdw_ChannelValue</i> [1] = digital value of the calibration offset (if available)
		* <i>pdw_ChannelValue</i> [2] = digital value of the calibration gain (if available)
		* <i>pdw_ChannelValue</i> [3] = digital value of the next channel
		* <i>pdw_ChannelValue</i> [4] = digital value of the calibration offset (if available)
		* <i>pdw_ChannelValue</i> [5] = digital value of the calibration gain (if available)
		...

**Task:**

Returns the digital value (*pdw\_ChannelValue*) of all selected channels (*w\_FirstChannel*, *w\_LastChannel*). To get the real pressure value, you must call up the "**b\_ADDIDATA\_ConvertMoreDigitalToRealPressureValues (...)**" function. *dw\_ConvertingTime* and *b\_ConvertingTimeUnit* determine the converting time.

**Calling convention:**ANSI C:

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_ChannelArrayValue [48];

```

```

b_ReturnValue = b_ADDIDATA_ReadMorePressureChannels
                                                    (dw_DriverHandle,
                                                    0,
                                                    11,
                                                    10,
                                                    1,
                                                    ADDIDATA_DISABLE,
                                                    dw_ChannelArrayValue);

```

**Return Value:**

1: No error  
 0: Error by calling up the function. Use the function "*i\_ADDIDATA\_GetLastError*", to find the error source.



**10) b\_ADDIDATA\_ConvertDigitalToRealPressureValue (..)****Syntax:**

```
<Return value> = b_ADDIDATA_ConvertDigitalToRealPressureValue
                    (DWORD      dw_DriverHandle,
                     WORD       w_Channel,
                     DWORD      dw_DigitalValue,
                     PDOUBLE    pd_RealValue)
```

**Parameters:****- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Channel</i>	Number of the channel number to be converted
DWORD <i>dw_DigitalValue</i>	Gives the digital pressure value (composed of the digital value of the channel, of the calibration offset and calibration gain if these parameters are available)

**- Output:**

PDOUBLE <i>pd_RealValueArray</i>	Returns the real pressure value
----------------------------------	---------------------------------

**Task:**

Converts the digital pressure value (*dw\_DigitalValue*) into a real pressure value (*pd\_RealValue*) for the selected channel (*w\_Channel*).

**Calling convention:**ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_DigitalValue[4];
DOUBLE    d_RealValue;
```

```
b_ReturnValue = b_ADDIDATA_ConvertDigitalToRealPressureValue
                    (dw_DriverHandle,
                     0,
                     dw_DigitalValue,
                     &d_RealValue);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError", to find the error source.

**11) b\_ADDIDATA\_ConvertMoreDigitalToRealPressureValues (..)****Syntax:**

<Return value> = b\_ADDIDATA\_ConvertMoreDigitalToRealPressureValues  
 (DWORD dw\_DriverHandle,  
 WORD w\_FirstChannel,  
 WORD w\_LastChannel,  
 PDWORD pdw\_DigitalValue,  
 PDOUBLE pd\_RealValue)

**Parameters:****- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_FirstChannel</i>	Selection of the first channel
WORD <i>w_LastChannel</i>	Selection of the last channel
PDWORD <i>pdw_DigitalValue</i>	Digital pressure value (composed of the digital value of the channel, of the calibration offset and calibration gain if these parameters are available)

**- Output:**

PDOUBLE <i>pd_RealValueArray</i>	Returns the real pressure value
----------------------------------	---------------------------------

**Task:**

Converts the digital pressure value (*pdw\_DigitalValue*) into a real pressure value (*pd\_RealValue*) for the selected channels.

**Calling convention:**ANSI C :

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
DWORD   dw_DigitalValue[4];
DOUBLE  d_RealValue;
```

```

b_ReturnValue = b_ADDIDATA_ConvertMoreDigitalToRealPressureValues
(dw_DriverHandle,
0,
3,
dw_DigitalValue,
&d_RealValue);
```

**Return Value:**

1: No error  
 0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError", to find the error source.

12) **b\_ADDIDATA\_InitPressureSCAN (..)****i****IMPORTANT!****This function can only be operated on one module.****Please check that all the channels you selected are on the same module.****Syntax:**

```

<Return value> = b_ADDIDATA_InitPressureSCAN
                    (DWORD    dw_DriverHandle,
                    pstr_InitPressureSCAN ps_InitParameters,
                    DWORD    dw_StructSize
                    PDWORD   pdw_SCANHandle)

```

**Parameters:****- Input:**

DWORD dw_DriverHandle	Driver handle
pstr_InitPressureSCAN ps_InitParameters	Pointer of the structure which contains the initialisation parameters
DWORD dw_StructSize	Size of the structure.

The structure contains the following information:

WORD w_FirstChannel	First channel to be converted.
WORD w_LastChannel	Last channel to be converted
DWORD dw_ConvertingTime	Converting time. Refer to table 2-2
BYTE b_ConvertingTimeUnit	Converting time unit. Refer to table 2-2
BYTE b_SCANTimeMode	Define the mode of the delay. Refer to table 2-4
DWORD dw_SCANTime	Total time of the SCAN. Refer to table 2-3 and 3-4
BYTE b_SCANTimeUnit	Unit used for the global time. Refer to table 2-3 and 3-4
BYTE b_SCANMode	SCAN mode. Refer to table 2-5
BYTE b_ExternTriggerMode	External trigger action. Refer to table 2-6
BYTE b_ExternGateMode	External gate action. ADDIDATA_DISABLE: External gate not used ADDIDATA_LOW_LEVEL: The SCAN is started when the external gate is low ADDIDATA_HIGH_LEVEL: The SCAN is started when the external gate is high
DWORD dw_SCANCounter	Number of SCAN. Refer to table 2-5

**- Output:**

PDWORD pdw_SCANHandle	Handle of the initialised SCAN. This handle is used for all SCAN functions.
-----------------------	--

**Task:**

Initialises the SCAN function for the acquisition of pressure channels.

**Table 2-3: Time selection of a SCAN**

<i>b_SCANTimeUnit</i>	Unit selection	<i>dw_SCANTime</i>	Total time of the SCAN
0	ns or MHz	10	10 ns or MHz
		20	20 ns or MHz
		300	300 ns or MHz
		1000	1000 ns or MHz
		22222	22222 ns or MHz
1	μs or KHz	10	10 μs or KHz
		20	20 μs or KHz
		300	300 μs or KHz
		1000	1000 μs or KHz
		22222	22222 μs or KHz
2	ms or Hz	10	10 ms or Hz
		20	20 ms or Hz
		300	300 ms or Hz
		1000	1000 ms or Hz
		22222	22222 ms or Hz
3	s or mHz	10	10 s or mHz
		20	20 s or mHz
		300	300 s or mHz
		1000	1000 s or mHz
		22222	22222 s or mHz

**Table 2-4: SCAN time mode**

<i>b_SequenceTimeMode</i>	Mode description
ADDIDATA_DELAY_NOT_USED	The delay between two SCANS is not used
ADDIDATA_DELAY_MODE1_USED	The delay between two SCANS is used. The conversion is started at once, and the delay is used after the first SCAN.
ADDIDATA_DELAY_MODE2_USED	The delay between two SCANS is used. The conversion is started after a delay.

**Table 2-5: SCAN mode**

<b>b_SCANMode</b>	<b>dw_SCANCounter</b>	<b>SCAN description</b>
ADDIDATA_SINGLE_SCAN	Not used	Only one SCAN is started after calling up the function "b_ADDIDATA_StartPressure SCAN"
ADDIDATA_DEFINED_SCAN_NUMBER	Determines the number of SCANS	A predefined number of SCANS is started after calling up the function "b_ADDIDATA_StartPressure SCAN"
ADDIDATA_CONTINUOUS_SCAN	Not used	An undefined number of SCANS is started after calling up the function "b_ADDIDATA_StartPressure SCAN"

**Table 2-6: Trigger mode**

ADDIDATA_DISABLE	External trigger not used
ADDIDATA_FIRST_LOW_EDGE_START_ALL_SCAN	The first low edge starts all SCANS
ADDIDATA_FIRST_HIGH_EDGE_START_ALL_SCAN	The first high edge starts all SCANS
ADDIDATA_FIRST_EDGE_START_ALL_SCAN	The first low/high edge starts all SCANS
ADDIDATA_EACH_LOW_EDGE_START_A_SCAN	The first low edge starts a single SCAN
ADDIDATA_EACH_HIGH_EDGE_START_A_SCAN	The first high edge starts a single SCAN
ADDIDATA_EACH_EDGE_START_A_SCAN	Every edge starts a single SCAN
ADDIDATA_EACH_LOW_EDGE_START_A_SINGLE_ACQUISITION	Each low edge starts a single acquisition of the SCAN
ADDIDATA_EACH_HIGH_EDGE_START_A_SINGLE_ACQUISITION	Each high edge starts a single acquisition of the SCAN
ADDIDATA_EACH_EDGE_START_A_SINGLE_ACQUISITION	Each edge starts a single acquisition of the SCAN

**Calling convention:**ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     w_SCAN [6] = {0,1,2,3,4,5};
DWORD     dw_SCANHandle;
str_InitPressureSCAN s_InitParameters;

```

```

b_ReturnValue = b_ADDIDATA_InitPressureSCAN
                (dw_DriverHandle,
                 &s_InitParameters,
                 sizeof (str_InitPressureSCAN),
                 &dw_SCANHandle);

```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function

"i\_ADDIDATA\_GetLastError", to find the error source.

**13) b\_ADDIDATA\_StartPressureSCAN (..)****Syntax:**

```
<Return value> = b_ADDIDATA_StartPressureSCAN
                  (DWORD    dw_DriverHandle,
                   DWORD    dw_SCANHandle)
```

**Parameters:****- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>dw_SCANHandle</i>	SCAN handle. This handle is returned by the "b_ADDIDATA_InitPressureSCAN" function

**- Output:**

No output signal has occurred.

**Task:**

Starts the selected SCAN (*dw\_SCANHandle*).

**Calling convention:**ANSI C:

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
DWORD   dw_SCANHandle;
```

```
b_ReturnValue = b_ADDIDATA_StartPressureSCAN
                                     (dw_DriverHandle,
                                     dw_SCANHandle);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError", to find the error source.

**14) b\_ADDIDATA\_GetPressureSCANStatus (..)****Syntax:**

<Return value> = b\_ADDIDATA\_GetPressureSCANStatus  
                                   (DWORD     dw\_DriverHandle,  
                                   DWORD     dw\_SCANHandle,  
                                   PBYTE     pb\_SCANStatus)

**Parameters:****- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>dw_SCANHandle</i>	SCAN handle. This handle is returned by the function "b_ADDIDATA_InitPressureSCAN"

**- Output:**

PBYTE <i>pb_SCANStatus</i>	Returns the status of the SCAN .
	0: SCAN not started
	1: SCAN started
	2: SCAN completed
	3: SCAN completed and new SCAN started

**Task:**

Returns the status (*pb\_SCANStatus*) of the selected SCAN (*dw\_SCANHandle*).

**Calling convention:**ANSI C :

```
BYTE     b_ReturnValue;
DWORD    dw_DriverHandle;
DWORD    dw_SCANHandle;
BYTE     b_SCANStatus;
```

```
b_ReturnValue = b_ADDIDATA_GetPressureSCANStatus
                                                         (dw_DriverHandle,
                                                         dw_SCANHandle,
                                                         dw_SCANArrayValue);
```

**Return Value:**

1: No error  
 0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError", to find the error source.

**15) b\_ADDIDATA\_ConvertDigitalToRealPressureValueSCAN (..)****Syntax:**

```
<Return value> = b_ADDIDATA_ConvertDigitalToRealPressureValueSCAN
                    (DWORD      dw_DriverHandle,
                     DWORD      dw_SCANHandle,
                     PDWORD     pdw_DigitalValueArray,
                     PDOUBLE    pd_RealValueArray)
```

**Parameters:****- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>dw_SCANHandle</i>	SCAN handle. This handle is returned by the function "b_ADDIDATA_InitPressureSCAN"
PDWORD <i>pdw_DigitalValueArray</i>	Digital value array of the pressure channel

**- Output:**

PDOUBLE <i>pd_RealValueArray</i>	Real pressure value array
----------------------------------	---------------------------

**Task:**

Converts the digital pressure value array into a real pressure value array.

**Calling convention:**ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SCANHandle;
DWORD     dw_DigitalValueArray [600];
DOUBLE    d_RealValueArray [600];
```

```
b_ReturnValue = b_ADDIDATA_ConvertDigitalToRealPressureValueSCAN
                    (dw_DriverHandle,
                     dw_SCANHandle,
                     dw_DigitalValueArray,
                     d_RealValueArray);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError", to find the error source.



**16) b\_ADDIDATA\_StopPressureSCAN (..)****Syntax:**

<Return value> = b\_ADDIDATA\_StopPressureSCAN  
   (DWORD     dw\_DriverHandle,  
   DWORD     dw\_SCANHandle)

**Parameters:****- Input:**

WORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>dw_SCANHandle</i>	SCAN handle. This handle is returned by the function "b_ADDIDATA_InitPressureSCAN"

**- Output:**

No output signal has occurred.

**Task:**

Stops the selected SCAN (*dw\_SCANHandle*).

**Calling convention:**ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SCANHandle;
```

```
b_ReturnValue = b_ADDIDATA_StopPressureSCAN
                                                         (dw_DriverHandle,
                                                         dw_SCANHandle);
```

**Return Value:**

1: No error  
 0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError", to find the error source.

### 17) b\_ADDIDATA\_ClosePressureSCAN (..)

### Syntax:

<Return value> = b\_ADDIDATA\_ClosePressureSCAN  
(DWORD dw\_DriverHandle,  
DWORD dw\_SCANHandle)

### Parameters:

**- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>dw_SCANHandle</i>	SCAN handle. This handle is returned by the function "b_ADDIDATA_InitPressureSCAN"

**- Output:**

No output signal has occurred.

### Task:

Closes the selected SCAN (*dw\_SCANHandle*) and releases the SCAN handle.

### Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SCANHandle;

```

```
b_ReturnValue = b_ADDIDATA_ClosePressureSCAN(dw_DriverHandle, dw_SCANHandle);
```

### Return Value:

1: No error  
0: Error by calling up the function. Use the function  
"i\_ADDIDATA\_GetLastError", to find the error source.

**18) b\_ADDIDATA\_ReleasePressureChannel (..)****Syntax:**

<Return value> = b\_ADDIDATA\_ReleasePressureChannel  
   (DWORD     dw\_DriverHandle,  
   WORD     w\_Channel)

**Parameters:****- Input:**

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Channel</i>	Number of the pressure channel to be released before a new initialisation.

**- Output:**

No output signal has occurred.

**Task:**

Releases the pressure logic for the selected channel (*w\_Channel*) to allow a new initialisation.

**Calling convention:**

ANSI C:

```
BYTE     b_ReturnValue;
DWORD    dw_DriverHandle;
```

```
b_ReturnValue = b_ADDIDATA_ReleasePressureChannel
                                                         (dw_DriverHandle,
                                                         0);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function  
 "i\_ADDIDATA\_GetLastError", to find the error source.

**19) b\_ADDIDATA\_ConvertBarToPa (..)****Syntax:**

<Return value> = b\_ADDIDATA\_ConvertBarToPa  
(DOUBLE d\_BarValue,  
PDOUBLE pd\_PaValue)

**Parameters:****- Input:**

DOUBLE *d\_BarValue* Bar value to be converted

**- Output:**

PDOUBLE *pd\_PaValue* Value in Pascal (Pa).

**Task:**

Converts the Bar value in a Pascal value ( $\text{Pa} = \text{N/m}^2$ ).

**Calling convention:**ANSI C :

BYTE b\_ReturnValue;  
DOUBLE d\_PaValue;

b\_ReturnValue = b\_ADDIDATA\_ConvertBarToPa  
(25,55,  
&d\_PaValue);

**Return Value:**

1: No error

0: Error by calling up the function. Use the function  
"i\_ADDIDATA\_GetLastError", to find the error source.

**20) b\_ADDIDATA\_ConvertBarToPsi (..)****Syntax:**

```
<Return value> = b_ADDIDATA_ConvertBarToPsi  
                  (DOUBLE    d_BarValue,  
                  PDOUBLE   pd_PsiValue)
```

**Parameters:****- Input:**

DOUBLE *d\_BarValue*                      Bar value to be converted

**- Output:**

PDOUBLE *pd\_PsiValue*                      Value in Psi.

**Task:**

Converts the Bar value in a Psi value.

**Calling convention:**ANSI C:

```
BYTE    b_ReturnValue;  
DOUBLE  d_PsiValue;
```

```
b_ReturnValue = b_ADDIDATA_ConvertBarToPsi  
                  (25,55,  
                  &d_PsiValue);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function

"i\_ADDIDATA\_GetLastError", to find the error source.

**21) b\_ADDIDATA\_TestPressureAsynchronousFIFOFull****Syntax:**

```
<Return value> = b_ADDIDATA_TestPressureAsynchronousFIFOFull
                    (DWORD    dw_DriverHandle,
                     PBYTE    pb_Full)
```

**Parameters:****- Input:**

DWORD *dw\_DriverHandle*      Driver handle

**- Output:**

PBYTE *pb\_Full*      0: Asynchronous interrupt FIFO memory not full  
                          1: Asynchronous interrupt FIFO memory is full

**Task:**

Tests if the asynchronous interrupt FIFO memory is full or not.

The asynchronous interrupt FIFO memory is the FIFO memory in which the asynchronous events generated by the asynchronous interrupt are stored.

**Calling convention:**ANSI C:

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
BYTE    b_FIFOFull
```

```
b_ReturnValue = b_ADDIDATA_TestPressureAsynchronousFIFOFull
                    (dw_DriverHandle,
                     &b_FIFOFull);
```

**Return value:**

1: No error

0: Error by calling up the function. Use the function  
 "i\_ADDIDATA\_GetLastError",  
 to find the error source.

**22) b\_ADDIDATA\_GetPressureChannelReferenceVoltage (..)****Syntax:**

```
<Return value> = b_ADDIDATA_GetPressureChannelReferenceVoltage
                (DWORD      dw_DriverHandle,
                 WORD       w_ChannelNbr,
                 DOUBLE     *pd_ReferenceVoltage)
```

**Input:**

DWORD	dw_DriverHandle	Driver handle
DWORD	w_ChannelNbr	Number of the pressure input channel to get the reference voltage

**Output:**

DOUBLE	*pd_ReferenceVoltage	Return the reference voltage (V).
--------	----------------------	-----------------------------------

**Task:**

Return the reference voltage (pd\_ReferenceVoltage) apply the selected channel (w\_ChannelNbr).

**Calling convention:**

ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DOUBLE    d_ReferenceVoltage;
```

```
b_ReturnValue = b_ADDIDATA_GetPressureChannelReferenceVoltage
                (dw_DriverHandle,
                 0,
                 &d_ReferenceVoltage);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError", to find the error source.

**23) b\_ADDIDATA\_GetPressureChannelGainFactor (..)****Syntax:**

<Return value> = b\_ADDIDATA\_GetPressureChannelGainFactor  
 (DWORD dw\_DriverHandle,  
 WORD w\_ChannelNbr,  
 DOUBLE \*pd\_GainFactor)

**Input:**

DWORD	dw_DriverHandle	Driver handle
DWORD	w_ChannelNbr	Number of the pressure input channel to get the gain correction factor

**Output:**

DOUBLE	* pd_GainFactor	Return the gain correction factor.
--------	-----------------	------------------------------------

**Task:**

Return the gain correction factor (pd\_GainFactor) apply the selected channel (w\_ChannelNbr).

**Calling convention:**

ANSI C :

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
DOUBLE  d_GainFactor;
```

```

b_ReturnValue = b_ADDIDATA_GetPressureChannelGainFactor
                (dw_DriverHandle,
                 0,
                 &d_GainFactor);
```

**Return Value:**

1: No error

0: Error by calling up the function. Use the function  
 "i\_ADDIDATA\_GetLastError", to find the error source.