



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

Software description

ADDIDRIVER

Temperature

5th edition 04/2004

1	INTRODUCTION	1
2	SOFTWARE FUNCTIONS.....	2
1)	b_ADDIDATA_GetNumberOfTemperatureChannels (..).....	2
2)	b_ADDIDATA_GetNumberOfTemperatureModules (..).....	3
3)	b_ADDIDATA_GetNumberOfTemperatureChannelsForTheModule (..)	4
4)	b_ADDIDATA_GetTemperatureChannelInformation (..)	5
5)	b_ADDIDATA_InitTemperatureChannel (..).....	14
6)	b_ADDIDATA_TestTemperatureChannelShortCircuit (..).....	16
7)	b_ADDIDATA_TestTemperatureChannelConnection (..).....	17
8)	b_ADDIDATA_ReadTemperatureChannel (..)	18
9)	b_ADDIDATA_ReadMoreTemperatureChannels (..).....	20
10)	b_ADDIDATA_ConvertDigitalToRealTemperatureValue (..)	22
11)	b_ADDIDATA_ConvertDigitalToRealTemperatureValueWithCorrection Parameters (..).....	23
12)	b_ADDIDATA_ConvertMoreDigitalToRealTemperatureValues (..)	24
13)	b_ADDIDATA_InitTemperatureChannelSCAN (..)	25
14)	b_ADDIDATA_StartTemperatureChannelSCAN (..)	28
15)	b_ADDIDATA_GetTemperatureChannelSCANStatus (..)	29
16)	b_ADDIDATA_ConvertDigitalToRealTemperatureValueSCAN (..).....	30
17)	b_ADDIDATA_StopTemperatureChannelSCAN (..)	31
18)	b_ADDIDATA_CloseTemperatureChannelSCAN (..).....	32
19)	b_ADDIDATA_InitTemperatureWarning (..)	33
20)	b_ADDIDATA_EnableDisableTemperatureWarningChannel (..)	34
21)	b_ADDIDATA_StartAllTemperatureWarnings (..)	35
22)	b_ADDIDATA_StopAllTemperatureWarnings (..).....	36
23)	b_ADDIDATA_ReleaseTemperatureWarning (..)	37
24)	b_ADDIDATA_ReleaseTemperatureChannel (..).....	38
25)	b_ADDIDATA_ConvertDegreeToFahrenheit (..)	39
26)	b_ADDIDATA_ConvertDegreeToKelvin (..)	40
27)	b_ADDIDATA_TestTemperatureAsynchronousFIFOFull (..)	41

Tables

Table 1-1: Type Declaration for Windows 98/NT/2000/XP	1
Table 2-1: Gain/Polarity for a max. voltage of 5V	15
Table 2-2: Selection of the converting time	19
Table 2-3: Time selection of a SCAN	26
Table 2-4: SCAN time mode	26
Table 2-5: SCAN mode	26
Table 2-6: Trigger mode	27

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
VOID	void	void	pointer	any
BYTE	unsigned char	unsigned char	byte	integer
INT	int	int	integer	integer
WORD	unsigned short int	unsigned short int	long	long
DWORD	long	long	longint	long
PBYTE	unsigned char *	unsigned char *	var byte	integer
PINT	int *	int *	var integer	integer
PWORD	unsigned short int *	unsigned short int *	var long	long
PCHAR	char *	char *	var string	string
PDWORD	long *	long *	var longint	long
DOUBLE	double	double	double	double

2 SOFTWARE FUNCTIONS

1) b_ADDIDATA_GetNumberOfTemperatureChannels (..)

Syntax:

<Return value> = b_ADDIDATA_GetNumberOfTemperatureChannels
(DWORD dw_DriverHandle,
WORD pw_ChannelNbr)

Parameters:

- Input:

DWORD	<i>dw</i>	<i>DriverHandle</i>	Driver handle
-------	-----------	---------------------	---------------

- Output:

WORD <i>pw_ChannelNbr</i>	Returns the number of temperature channels
----------------------------------	--

Task:

Returns the number of temperature channels.

Calling convention:

ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
WORD	w_ChannelNbr;

```
b_ReturnValue = b_ADDIDATA_GetNumberOfTemperatureChannels
                (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_GetNumberOfTemperatureModules (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetNumberOfTemperatureModules  
                  (DWORD    dw_DriverHandle,  
                   PWORD    pw_ModuleNbr)
```

Parameters:

- Input:

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

- Output:

PWORD <i>pw_ModuleNbr</i>	Returns the number of temperature modules
---------------------------	---

Task:

Returns the number of temperature modules.

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
WORD	w_ModuleNbr;

```
b_ReturnValue = b_ADDIDATA_GetNumberOfTemperatureModules
                                     (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_GetNumberOfTemperatureChannelsForTheModule (..)

Syntax:

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

Parameters:

- Input:

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

- Output:

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

Task:

Returns the number of temperature channels for the module.

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
WORD	w_ChannelNbr;

```
b_ReturnValue = b_ADDIDATA_GetNumberOfTemperatureChannelsForTheModule(dw_DriverHandle,  
0,  
&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.

4) b_ADDIDATA_GetTemperatureChannelInformation (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetTemperatureChannelInformation
                (DWORD    dw_DriverHandle,
                 WORD     w_Channel,
                 pstr_GetAnalogMesureInformation ps_ChannelInformation,
                 DWORD    dw_StructSize)
```

Parameters:

- Input:

DWORD dw_DriverHandle Driver handle
 DWORD dw_StructSize Size of the structure.

- Output:

pstr_GetAnalogMesureInformation 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_CanGerenatedWarning;
                                         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      Module number
  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_RTD
		ADDIDATA_THERMOCOUPLE
		ADDIDATA_TEMPERATURE_ON_BOARD
BYTE	<i>b_TypePrecision;</i>	Precision of the input
		ADDIDATA_THERMOCOUPLE_TYPE_B
		ADDIDATA_THERMOCOUPLE_TYPE_E
		ADDIDATA_THERMOCOUPLE_TYPE_J
		ADDIDATA_THERMOCOUPLE_TYPE_K
		ADDIDATA_THERMOCOUPLE_TYPE_N
		ADDIDATA_THERMOCOUPLE_TYPE_R
		ADDIDATA_THERMOCOUPLE_TYPE_S
		ADDIDATA_THERMOCOUPLE_TYPE_T
		or
		ADDIDATA_RTD_TYPE_PT
		ADDIDATA_RTD_TYPE_Ni
WORD	<i>w_InputTypeValue;</i>	If a Pt sensor is connected type of the sensor
		100 → Pt100

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, μ s, ms, s, ...) 1: Frequency unit (MHz, kHz, Hz, mHz, ...)
BYTE	<i>b_AvailableConversionUnit;</i>	For time unit: D0: 0: ns not available 1: ns available D1: 0: μ s not available 1: μ s available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available For frequency unit: 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>	For time unit: 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 For frequency unit: 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</i> [255];	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 starst 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, μ 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>	For time unit: D0: 0: ns not available 1: ns available D1: 0: μ s not available 1: μ s available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available

For frequency unit:
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 *w_SCANDelayTimeResolution*; Delay time resolution
8: 8-bit resolution
16: 16-bit resolution, ...

WORD *w_SCANMinDelayTime*; Minimum delay time.
7000: 7000(ns)
10000: 10000(ns), ...

WORD *w_SCANDelayTimeStep*; Conversion delay time steps
20: 20 steps
50: 50 steps, ...

BYTE *b_SCANUnipolarBipolarConfigurable*;
0: Unipolar/bipolar hardware configurable
1: Unipolar/bipolar configurable

BYTE *b_SCANUnipolarAvailable*; 0: Cannot configure in unipolar
1: Can configure in unipolar

BYTE *b_SCANBipolarAvailable*; 0: Cannot configure in bipolar
1: Can configure in bipolar

BYTE *b_SCANDDCCouplingAvailable*;
0: Cannot configure the direct coupling (DC)
1: Can configure direct coupling (DC)

BYTE *b_SCANACCouplingAvailable*;
0: Cannot configure alternative coupling (AC)
1: Can configure alternative coupling (AC)

BYTE *b_SCANBufferAvailable*; 0: No hardware buffer available
1: Hardware buffer available

BYTE *b_SCANNbOfGain*; Returns the number of gain values available

BYTE *b_Reserved7*;

WORD *w_Reserved8*;

DOUBLE *d_SCANGainAvailable*[255]; Defines the available gain values

BYTE *b_SCANOffsetRangeAvailable*;
0: Offset not allowed
1: Offset allowed

BYTE *b_Reserved9*;

WORD *w_SCANOffsetRangeResolution*;
Offset resolution
8 : 8-bit resolution
16: 16-bit resolution, ...

BYTE *b_SCANOffsetRangeDenominator*; Offset denominator step value

BYTE *b_SCANOffsetRangeNumerator*; Offset numerator step value

WORD *w_Reserved10*;

}str_GetAnalogMeasureInformation,*pstr_GetAnalogMeasureInformation;

Task:

Returns information about the temperature channel.

Calling convention:ANSI C:

```
BYTE      b_ReturnValue;  
DWORD     dw_DriverHandle;  
str_GetAnalogMesureInformation s_ChannelInformation;
```

```
    b_ReturnValue = b_ADDIDATA_GetTemperatureChannelInformation  
                    (dw_DriverHandle,  
                     0,  
                     &s_ChannelInformation,  
                     sizeof (str_GetAnalogMesureInformation));
```

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_InitTemperatureChannel (..)

Syntax:

```
<Return value> = b_ADDIDATA_InitTemperatureChannel
                    (DWORD dw_DriverHandle,
                     WORD   w_Channel,
                     WORD   w_Gain,
                     BYTE   b_Polarity,
                     WORD   w_OffsetRange)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Channel</i>	Number of the temperature channel to be initialised
WORD <i>w_Gain</i>	Gain of the channel. Refer to table 2-1
BYTE <i>b_Polarity</i>	Polarity of the selected channel ADDIDATA_UNIPOLAR for unipolar and ADDIDATA_BIPOLAR for bipolar Refer to table 2-1
WORD <i>w_OffsetRange</i>	Function not used. Please set it to "0".

- Output:

No output signal has occurred.

Task:

Initialises the selected temperature channel.

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

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

Calling convention:ANSI C:

BYTE b_ReturnValue;
 DWORD dw_DriverHandle;

```
b_ReturnValue = b_ADDIDATA_InitTemperatureChannel
                                     (dw_DriverHandle,
                                     0,
                                     1,
                                     ADDIDATA_UNIPOLAR,
                                     0);
```

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_Read1TemperatureChannel (..)

Syntax:

```
<Return value> = b_ADDIDATA_Read1TemperatureChannel
(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 temperature 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>	<p>ADDIDATA_DISABLE: No interrupt is generated after the conversion and the variable <i>pdw_ChannelValue</i> returns the digital value.</p> <p>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"</p>

- Output:

PDWORD <i>pdw_ChannelValue</i>	<p>Returns the digital value of the selected channel</p> <p>*<i>pdw_ChannelValue</i>[0] = digital value of the channel</p> <p>*<i>pdw_ChannelValue</i>[1] = digital value of the CJC (if available)</p> <p>*<i>pdw_ChannelValue</i>[2] = digital value of the calibration offset (if available)</p> <p>*<i>pdw_ChannelValue</i>[3] = digital value of the calibration gain (if available)</p> <p>*<i>pdw_ChannelValue</i>[4] = digital value of the calibration offset for the CJC (if available)</p> <p>*<i>pdw_ChannelValue</i>[5] = digital value of the calibration gain for the CJC (if available)</p>
--------------------------------	---

Task:

Returns the digital value (*pdw_ChannelValue*) of the selected channel (*w_Channel*). To obtain the real temperature value, you must call up the

"b_ADDIDATA_ConvertDigitalToRealTemperatureValue (...)" or

"b_ADDIDATA_ConvertDigitalToRealTemperatureValueWithCorrectionParameters (...)" 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 / MHz	10	10 ns / MHz
		20	20 ns / MHz
		300	300 ns / MHz
		1000	1000 ns / MHz
		22222	22222 ns / MHz
1	μs / KHz	10	10 μs / KHz
		20	20 μs / KHz
		300	300 μs / KHz
		1000	1000 μs / KHz
		22222	22222 μs / KHz
2	ms / Hz	10	10 ms / Hz
		20	20 ms / Hz
		300	300 ms / Hz
		1000	1000 ms / Hz
		22222	22222 ms / Hz
3	s / mHz	10	10 s / mHz
		20	20 s / mHz
		300	300 s / mHz
		1000	1000 s / mHz
		22222	22222 s / mHz

Calling convention:ANSI C:

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_ChannelValue[6];

```

```

b_ReturnValue = b_ADDIDATA_ReadTemperatureChannel
                (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. (Table 2-1).

9) b_ADDIDATA_ReadMoreTemperatureChannels (..)

Syntax:

```
<Return value> = b_ADDIDATA_ReadMoreTemperatureChannels
                (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 CJC (if available) * <i>pdw_ChannelValue</i> [2] = digital value of the calibration offset (if available) * <i>pdw_ChannelValue</i> [3] = digital value of the calibration gain (if available) * <i>pdw_ChannelValue</i> [4] = digital value of the calibration offset for the CJC (if available) * <i>pdw_ChannelValue</i> [5] = digital value of the calibration gain for the CJC (if available) * <i>pdw_ChannelValue</i> [6] = digital value of the next channel * <i>pdw_ChannelValue</i> [7] = digital value of the CJC (if available) ...
-------------------------------------	--

Task:

Returns the digital value (*pdw_ChannelValue*) of all selected channels (*w_FirstChannel*, *w_LastChannel*). To get the real temperature value, you must call up the "**b_ADDIDATA_ConvertMoreDigitalToRealTemperatureValues (...)**" function. *dw_ConvertingTime* and *b_ConvertingTimeUnit* determine the converting time.

Calling convention:

ANSI C:

```
BYTE      b_ReturnValue;  
DWORD     dw_DriverHandle;  
DWORD     dw_ChannelArrayValue [72];
```

```
b_ReturnValue = b_ADDIDATA_ReadMoreTemperatureChannels  
                                     (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_ConvertDigitalToRealTemperatureValue (..)

Syntax:

```
<Return value> = b_ADDIDATA_ConvertDigitalToRealTemperatureValue
                    (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 temperature value

- Output:

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

Task:

Converts the digital temperature value (*dw_DigitalValue*) into a real temperature value (*pd_RealValue*) for the selected channel (*w_Channel*)

Calling convention:

ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_DigitalValue;
DOUBLE    d_RealValue;
```

```
b_ReturnValue = b_ADDIDATA_ConvertDigitalToRealTemperatureValue
                    (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_ConvertDigitalToRealTemperatureValueWithCorrectionParameters (..)

Syntax:

<Return value> =

```
b_ADDIDATA_ConvertDigitalToRealTemperatureValueWithCorrectionParameters
                                (DWORD    dw_DriverHandle,
                                 WORD      w_Channel,
                                 PDWORD    pdw_DigitalValue,
                                 PDOUBLE   pd_RealValue)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Channel</i>	Number of the channel to be converted
PDWORD <i>pdw_DigitalValue</i>	Digital temperature value (composed of the digital value of the channel, of the CJC digital value, of the calibration offset and of the calibration gain for the channel and the CJC if these parameters are available)

- Output:

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

Task:

Converts the digital temperature value (*pdw_DigitalValue*) into a real temperature value (*pd_RealValue*) for the selected channel (*w_Channel*).

Calling convention:

ANSI C :

```
BYTE    b_ReturnValue;
DWORD    dw_DriverHandle;
DWORD    dw_DigitalValue[6];
DOUBLE   d_RealValue;
```

b_ReturnValue =

```
b_ADDIDATA_ConvertDigitalToRealTemperatureValueWithCorrectionParameters
                                (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.

12) b_ADDIDATA_ConvertMoreDigitalToRealTemperatureValues (..)

Syntax:

<Return value> = b_ADDIDATA_ConvertMoreDigitalToRealTemperatureValues
 (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 temperature value (composed of the digital value of the channel, of the CJC digital value and of the calibration offset and calibration gain for the channel and the CJC if these parameters are available)

- Output:

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

Task:

Converts the digital temperature value (*pdw_DigitalValue*) into a real temperature value (*pd_RealValue*) for the selected channels.

Calling convention:

ANSI C:

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
DWORD   dw_DigitalValue[24];
DOUBLE  d_RealValue[4];
  
```

```

b_ReturnValue = b_ADDIDATA_ConvertMoreDigitalToRealTemperatureValues
(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.

13) b_ADDIDATA_InitTemperatureChannelSCAN (..)

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_InitTemperatureChannelSCAN
                    (DWORD    dw_DriverHandle,
                     pstr_InitTemperatureChannelSCAN ps_InitParameters,
                     DWORD    dw_StructSize,
                     PDWORD   pdw_SCANHandle)
```

Parameters:**- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
pstr_InitTemperatureChannelSCAN <i>ps_InitParameters</i>	Pointer of the structure which contains the initialisation parameters
DWORD <i>dw_StructSize</i>	Size of the structure.

The structure contains the following information:

WORD <i>w_FirstChannel</i>	First channel to be converted.
WORD <i>w_LastChannel</i>	Last channel to be converted
DWORD <i>dw_ConvertingTime</i>	Converting time. Refer to Table 2-2
BYTE <i>b_ConvertingTimeUnit</i>	Converting time unit. Refer to Table 2-2
BYTE <i>b_SCANTimeMode</i>	Define the mode of the delay. Refer to Table 2-4
DWORD <i>dw_SCANTime</i>	Total time of the SCAN. Refer to Table 2-3 and Table 2-4
BYTE <i>b_SCANTimeUnit</i>	Unit used for the global time. Refer to Table 2-3 and Table 2-4
BYTE <i>b_SCANMode</i>	SCAN mode. Refer to Table 2-5
BYTE <i>b_ExternTriggerMode</i>	External trigger action. Refer to Table 2-6
BYTE <i>b_ExternGateMode</i>	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 <i>dw_SCANCounter</i>	Number of SCAN. Refer to Table 2-5

- Output:

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

Task:

Initialises the temperature acquisition SCAN .

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 / MHz	10	10 ns / MHz
		20	20 ns / MHz
		300	300 ns / MHz
		1000	1000 ns / MHz
		22222	22222 ns / MHz
1	μs / KHz	10	10 μs / KHz
		20	20 μs / KHz
		300	300 μs / KHz
		1000	1000 μs / KHz
		22222	22222 μs / KHz
2	ms / Hz	10	10 ms / Hz
		20	20 ms / Hz
		300	300 ms / Hz
		1000	1000 ms / Hz
		22222	22222 ms / Hz
3	s / mHz	10	10 s / mHz
		20	20 s / mHz
		300	300 s / mHz
		1000	1000 s / mHz
		22222	22222 s / 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

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

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     dw_SCANHandle;
str_InitTemperatureChannelSCAN s_InitParameters;

```

```

b_ReturnValue = b_ADDIDATA_InitTemperatureChannelSCAN
                (dw_DriverHandle,
                 &s_InitParameters,
                 sizeof (str_InitTemperatureChannelSCAN),
                 &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_StartTemperatureChannelSCAN (..)**Syntax:**

<Return value> = b_ADDIDATA_StartTemperatureChannelSCAN
 (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_InitTemperatureChannelSCAN" 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_StartTemperatureChannelSCAN
                                   (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.

15) b_ADDIDATA_GetTemperatureChannelSCANStatus (..)**Syntax:**

<Return value> = b_ADDIDATA_GetTemperatureChannelSCANStatus
 (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_InitTemperatureChannelSCAN"

- 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_GetTemperatureChannelSCANStatus
                                                         (dw_DriverHandle,
                                                         dw_SCANHandle,
                                                         &b_SCANStatus);
```

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_ConvertDigitalToRealTemperatureValueSCAN (..)**Syntax:**

```
<Return value> = b_ADDIDATA_ConvertDigitalToRealTemperatureValueSCAN
                    (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_InitTemperatureChannelSCAN"
PDWORD <i>pdw_DigitalValueArray</i>	Digital value array of the temperature channel

- Output:

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

Task:

Converts the digital temperature value array into a real temperature 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_ConvertDigitalToRealTemperatureValueSCAN
                    (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.

17) b_ADDIDATA_StopTemperatureChannelSCAN (..)

Syntax:

<Return value> = b_ADDIDATA_StopTemperatureChannelSCAN
(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_InitTemperatureChannelSCAN"

- 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_StopTemperatureChannelSCAN(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_CloseTemperatureChannelSCAN (..)

Syntax:

<Return value> = b_ADDIDATA_CloseTemperatureChannelSCAN
(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_InitTemperatureChannelSCAN"

- 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_CloseTemperatureChannelSCAN(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.

19) b_ADDIDATA_InitTemperatureWarning (..)**Syntax:**

```
<Return value> = b_ADDIDATA_InitTemperatureWarning
                                (DWORD    dw_DriverHandle,
                                WORD      w_Channel,
                                WORD      w_Gain,
                                BYTE      b_Polarity,
                                WORD      w_OffsetRange,
                                DOUBLE     d_WarningValue)
```

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Channel</i>	Number of the temperature channel to be initialised
WORD	<i>w_Gain</i>	Gain. Refer to table 2-1
BYTE	<i>b_Polarity</i>	Polarity ADDIDATA_UNIPOLAR for unipolar and ADDIDATA_BIPOLAR for bipolar Refer to table 2-1
WORD	<i>w_OffsetRange</i>	Offset range. Not used: Set to 0
DOUBLE	<i>d_WarningValue</i>	Limit temperature value. If the limit value is reached or even exceeded, an interrupt is generated. Refer to the chapter "Interrupt"

Output:

No output signal has occurred.

Task:

Initialises the warning logic for the selected temperature channel (*w_Channel*). If the limit value is reached or exceeded (*d_WarningValue*), an interrupt is generated.

Calling convention:ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_InitTemperatureWarning
                                (dw_DriverHandle,
                                0,
                                1,
                                ADDIDATA_UNIPOLAR,
                                0,
                                52.0);
```

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_EnableDisableTemperatureWarningChannel (..)

Syntax:

<Return value> = b_ADDIDATA_EnableDisableTemperatureWarningChannel
(DWORD dw_DriverHandle,
WORD w_Channel,
BYTE b_WarningFlag)

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Channel</i>	Selected temperature channel
BYTE	<i>b_WarningFlag</i>	ADDIDATA_DISABLE: Disables the warning logic for the selected channel. ADDIDATA_ENABLE: Enables the warning logic for the selected channel.

- Output:

No output signal has occurred.

Task:

Enables or disables the warning logic for the selected temperature channel (w_Channel).

Calling convention:

ANSI C :

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;

```

```
b_ReturnValue = b_ADDIDATA_EnableDisableTemperatureWarningChannel
(dw_DriverHandle,
0,
ADDIDATA_ENABLE);
```

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_StartAllTemperatureWarnings (..)

Syntax:

<Return value> = b_ADDIDATA_StartAllTemperatureWarnings
(DWORD dw_DriverHandle)

Parameters:

- Input:

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

- Output:

No output signal has occurred.

Task:

Starts the temperature warning logic.

Calling convention:

ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_StartAllTemperatureWarnings(dw_DriverHandle);
```

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_StopAllTemperatureWarnings (..)

Syntax:

<Return value> = b_ADDIDATA_StopAllTemperatureWarnings
(DWORD dw_DriverHandle)

Parameters:

- Input:

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

- Output:

No output signal has occurred.

Task:

Stops the temperature warning logic.

Calling convention:

ANSI C:

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;

```

```
b_ReturnValue = b_ADDIDATA_StopAllTemperatureWarnings(dw_DriverHandle);
```

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_ReleaseTemperatureWarning (..)

Syntax:

<Return value> = b_ADDIDATA_ReleaseTemperatureWarning
(DWORD dw_DriverHandle,
WORD w_Channel)

Parameters:**- Input:**

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

- Output:

No output signal has occurred.

Task:

Releases the temperature warning logic for the selected channel (*w_Channel*) to allow a new warning initialisation.

Calling convention:ANSI C:

BYTE b_ReturnValue;
DWORD dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_ReleaseTemperatureWarning
(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.

24) b_ADDIDATA_ReleaseTemperatureChannel (..)

Syntax:

<Return value> = b_ADDIDATA_ReleaseTemperatureChannel
(DWORD dw_DriverHandle,
WORD w_Channel)

Parameters:

- Input:

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

- Output:

No output signal has occurred.

Task:

Releases the temperature 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_ReleaseTemperatureChannel(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.

25) b_ADDIDATA_ConvertDegreeToFahrenheit (..)**Syntax:**

```
<Return value> = b_ADDIDATA_ConvertDegreeToFahrenheit  
                  (DOUBLE      d_DegreeValue,  
                  PDOUBLE     pd_FahrenheitValue)
```

Parameters:**- Input:**

DOUBLE *d_DegreeValue* Degree value to be converted

- Output:

PDOUBLE *pd_FahrenheitValue* Value in Fahrenheit.

Task:

Converts the degree value in a Fahrenheit value.

Calling convention:ANSI C :

```
BYTE      b_ReturnValue;  
DOUBLE    d_FValue;
```

```
b_ReturnValue = b_ADDIDATA_ConvertDegreeToFahrenheit  
                (25.55,  
                &d_FValue);
```

Return Value:

1: No error

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

"i_ADDIDATA_GetLastError", to find the error source.

26) b_ADDIDATA_ConvertDegreeToKelvin (..)**Syntax:**

<Return value> = b_ADDIDATA_ConvertDegreeToKelvin
(DOUBLE d_DegreeValue,
PDOUBLE pd_KelvinValue)

Parameters:**- Input:**

DOUBLE *d_DegreeValue* Degree value to be converted

- Output:

PDOUBLE *pd_KelvinValue* Value in Kelvin.

Task:

Converts the degree value in a Kelvin value.

Calling convention:ANSI C:

BYTE b_ReturnValue;

DOUBLE d_KValue;

b_ReturnValue = b_ADDIDATA_ConvertDegreeToKelvin
(25.55,
&d_KValue);

Return Value:

1: No error

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

"i_ADDIDATA_GetLastError", to find the error source.

27) b_ADDIDATA_TestTemperatureAsynchronousFIFOFull (..)**Syntax:**

```
<Return value> = b_ADDIDATA_TestTemperatureAsynchronousFIFOFull
                    (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_TestTemperatureAsynchronousFIFOFull
                    (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.