Product Manual

Software

BMP5

Direct SDK

Revision: 04/2020

CSL I.D - 538

RELIABLE
SINCE 1974
MONITORING

Limited Guarantee

The following warranties are in effect for ninety (90) days from the date of shipment of the original purchase. These warranties are not extended by the installation of upgrades or patches offered free of charge.

Campbell Scientific warrants that the installation media on which the software is recorded and the documentation provided with it are free from physical defects in materials and workmanship under normal use. The warranty does not cover any installation media that has been damaged, lost, or abused. You are urged to make a backup copy (as set forth above) to protect your investment. Damaged or lost media is the sole responsibility of the licensee and will not be replaced by Campbell Scientific.

Campbell Scientific warrants that the software itself will perform substantially in accordance with the specifications set forth in the instruction manual when properly installed and used in a manner consistent with the published recommendations, including recommended system requirements. Campbell Scientific does not warrant that the software will meet licensee's requirements for use, or that the software or documentation are error free, or that the operation of the software will be uninterrupted.

Campbell Scientific will either replace or correct any software that does not perform substantially according to the specifications set forth in the instruction manual with a corrected copy of the software or corrective code. In the case of significant error in the installation media or documentation, Campbell Scientific will correct errors without charge by providing new media, addenda, or substitute pages. If Campbell Scientific is unable to replace defective media or documentation, or if it is unable to provide corrected software or corrected documentation within a reasonable time, it will either replace the software with a functionally similar program or refund the purchase price paid for the software.

All warranties of merchantability and fitness for a particular purpose are disclaimed and excluded. Campbell Scientific shall not in any case be liable for special, incidental, consequential, indirect, or other similar damages even if Campbell Scientific has been advised of the possibility of such damages. Campbell Scientific is not responsible for any costs incurred as a result of lost profits or revenue, loss of use of the software, loss of data, cost of re-creating lost data, the cost of any substitute program, telecommunication access costs, claims by any party other than licensee, or for other similar costs.

This warranty does not cover any software that has been altered or changed in any way by anyone other than Campbell Scientific. Campbell Scientific is not responsible for problems caused by computer hardware, computer operating systems, or the use of Campbell Scientific's software with non-Campbell Scientific software.

Licensee's sole and exclusive remedy is set forth in this limited warranty. Campbell Scientific's aggregate liability arising from or relating to this agreement or the software or documentation (regardless of the form of action; e.g., contract, tort, computer malpractice, fraud and/or otherwise) is limited to the purchase price paid by the licensee.

Licence for Use

CAMPBELL® SCIENTIFIC
WHEN MEASUREMENTS MATTER

Campbell Scientific Ltd,
80 Hathern Road,
Shepshed, Loughborough, LE12 9GX, UK
Tel: +44 (0) 1509 601141
Fax: +44 (0) 1509 270924
Email: support@campbellsci.co.uk
http://www.campbellsci.co.uk

Table of Contents

PDF viewers: These page numbers refer to the printed version of this document. Use the PDF reader bookmarks tab for links to specific sections.

BMP5 Direct SDK

1. BMP5 Direct SDK Overview

The BMP5 Direct Software Development Kit (SDK) comprises a simple call-level API (SimplePB.dll) wrapper for the included coralib3d.dll communications server. Client applications developed using the SDK will execute calls to the C-type functions exposed by the SimplePB.dll to effect data logger communications via the coralib3d.dll.

The SDK components and example applications are installed by default in *C:\Campbellsci\BMP5DirectSDK*. The SDK does not require registration on the host computer. However, the SimplePB.dll wrapper and the coralib3d.dll communications server must be installed into the same folder as the client application's executable.

NOTE If you have been using version 4.3 or earlier of the BMP5Direct SDK on your machine, you may wish to uninstall, remove, or relocate the files located in the C:\Campbellsci\BMP5DirectSDK\Examples folder before installing this version. This will help avoid confusion about code locations after installation.

This version uses a folder structure in this form:
\Examples\C#
\Examples\MFC-VS2015
\Examples\VB.NET

Older versions use a folder structure like this:
\Examples\C#\SmplPB_CS
\Examples\MFC
\Examples\VBNET

1.1 General Notes on BMP5 Direct SDK Usage

The SDK supports only PakBus® data logger communication via a serial port (COM) link or a TCP/IP socket connection. PakBus packet routing is **not** supported. Only a single, directly connected (leaf node) PakBus data logger is accessible at any one time.

The "dialling" of communication devices such as a dial-up phone modem or an RF500M modem is **not** supported. However, a connection via a transparent bridging device such as an RF450 or an RF401 radio is possible.

A successful call to the OpenPort() or OpenIPPort() function will start the CORALIB3D communications server (hereafter, referred to as "the Server"). The application should stop the Server by calling either the CloseIPPort() or ClosePort() function before exiting.

Both the Server and the SimplePB.dll wrapper write log files to *C:\Campbellsci\SimplePB\Ver#\logs*; where "Ver#" is the version number of the SimplePB.dll. These files can provide useful information about the Server's behaviour when troubleshooting connection issues. Refer to Appendix D of the *LoggerNet Instruction Manual* for information regarding log files.

Once a connection is established, additional functions can be called to accomplish the desired task. For example: send and manage data logger programs, check or set the data logger clock, query the data logger for data table information, get/set table values, and collect table records.

1.2 Data Logger Program Table Structure

The application developer must understand the table structure of the program running in the data logger because table and field names and numbers are used as arguments for many of the functions exposed by the SimplePB.dll. The GetTableNames() function can be used to obtain a list of tables and their associated numbers. Refer to Appendix A, *Sample Program Table Structure* ***(p. A-1)***, for information regarding the table structure of PakBus data loggers.

1.3 Developing Applications Using the .NET Framework

From the perspective of the .NET Framework, the SimplePB.dll is unmanaged code; not unlike the native functions of the Windows® API. Therefore, the platform invoke (P/Invoke) services provided by the common language run-time (CLR) can be used to directly access the SimplePB.dll functions.

Fundamentally, the implementation involves attaching a “DllImport” attribute (requires the System.Runtime.InteropServices namespace) to a static or shared declaration of the external function. The DllImport attribute notifies the CLR of the name of the DLL to load and the exposed function to call. An example of using the OpenPort() function is shown in the following C# code snippet:

```
[DllImportAttribute("SimplePB.dll", EntryPoint = "OpenPort", CallingConvention =
CallingConvention.StdCall)]
public static extern int OpenPort(int comPortNumber, int baudRate);
```

Attention should be paid to the marshalling of parameter data types. Particularly, the “Strings” in the managed code and the “char” arrays in the unmanaged functions. The SimplePB.dll functions expect the “char” arrays to be null-terminated and UTF8 encoded.

The recommended method for accommodating the C-type pointers used by many of the SimplePB.dll functions is to marshal the parameter as a System.IntPtr type. In the case of pointer to a pointer types (char**), pass the IntPtr by reference (ref or ByRef). Optionally, the “unsafe” keyword in C# allows for the direct use of pointer types.

Best practice is to encapsulate or “wrap” the SimplePB.dll function calls into a shared class and expose them to application code via public functions. This approach is implemented in both the C# and VB.NET example applications provided with the SDK.

2. SimplePB.dll Reference

The following C-style functions are exposed by the SimplePB.dll.

2.1 OpenPort()

Opens a COM port (serial port) on the host computer using the specified COM port and baud rate.

Syntax

int _stdcall OpenPort (int com_port_no, int baud)

Parameters

com_port_no: COM port to open.

baud: Baud rate to be used by the COM port.

Return Codes

0 = Successful.
–1 = Port failed to open or is already open.

2.2 ClosePort()

Closes the currently open COM port or IP port connection.

Syntax

int _stdcall ClosePort()

Return Codes

0 = Successful.
–1 = Port failed to close or was not open.

2.3 OpenIPPort()

Opens a TCP socket connection with a network device using the specified IP address and port number. An appropriate device would be a cell modem, serial server, or data logger. IPv4 and IPv6 addresses or fully qualified domain names are supported.

Syntax

int _stdcall OpenIPPort (char const *ip_address, int tcp_port)

Parameters

ip_address: Pointer to the memory location of a char array defining the IP address to be used. Must be a null-terminated array of UTF8 encoded bytes.

tcp_port: Port number that will be used when communicating with the data logger.

Return Codes

0 = Successful.
–1 = IP port failed to open or is already open.

2.4 CloseIPPort()

Closes the currently open IP port (synonymous with ClosePort()).

Syntax

```
int _stdcall CloseIPPort()
```

Return Codes

  0 = Successful.
–1 = IP port failed to close or was not open.

2.5 GetClock()

Queries the data logger for its current date and time.

Syntax

```
int _stdcall GetClock ( int pakbus_address, int device_type, char **return_data,
int *return_data_len )
```

Parameters

pakbus_address: PakBus® address of the data logger.

device_type: Type of data logger:

1	= CR200
2	= CR10XPB, CR23XPB, CR510PB
3	= CR1000
4	= CR3000
5	= CR800 Series
9	= CR6 Series
13	= CR300 Series
14	= CR1000X Series
15	= GRANITE 9
16	= GRANITE 10
17	= GRANITE 6

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the data logger.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

  0 = Successful.
–1 = Communication timed out.
–2 = Port is not open.

Example of data returned by function call

```
14:12:35  04/16/2004
```

2.6 SetClock()

Sets the date and time of the data logger to match the host computer clock.

Syntax

int _stdcall SetClock (int pakbus_address, int device_type, char **return_data, int *return_data_len)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:

1 =	CR200
2 =	CR10XPB, CR23XPB, CR510PB
3 =	CR1000
4 =	CR3000
5 =	CR800 Series
9 =	CR6 Series
13 =	CR300 Series
14 =	CR1000X Series
15 =	GRANITE 9
16 =	GRANITE 10
17 =	GRANITE 6

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the data logger.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0 = Successful.
–1 = Communication timed out.
–2 = Port is not open.

Example of data returned by function call

```
14:22:51  04/16/2004  (Old Time Old Date)
14:22:27  04/16/2004  (New Time New Date)
```

2.7 GetValue()

Queries the data logger for a value or an array of values from the specified table and field.

Syntax

int _stdcall GetValue (int pakbus_address, int device_type, int swath, char const *table_name, char const *field_name, char **return_data, int *return_data_len)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:

1 =	CR200
2 =	CR10XPB, CR23XPB, CR510PB
3 =	CR1000
4 =	CR3000
5 =	CR800 Series
9 =	CR6 Series
13 =	CR300 Series
14 =	CR1000X Series
15 =	GRANITE 9
16 =	GRANITE 10
17 =	GRANITE 6

swath: The number of values to collect starting at the location specified in the field_name parameter. The requested swath must be within the bounds of an indexed array or an error will occur.

table_name: Pointer to the memory location of a char array defining the name of the table in which the value(s) exist. Must be a null-terminated array of UTF8 encoded bytes.

field_name: Pointer to the memory location of a char array defining the field in which the value(s) exist. Field_name may specify an array element (example: “Temp(3)”). Must be a null-terminated array of UTF8 encoded bytes.

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the data logger.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0 = Successful.
–1 = Communication timed out.
–2 = Port is not open.

Example of data returned by function call

```
12.753,111.9,1.239      (Swath of 3 values from fields)
```

2.8 SetValue()

Set the value of the specified field in the specified data logger table.

Syntax

```
int _stdcall SetValue ( int pakbus_address, int device_type, char const
*table_name, char const *field_name, char const *value )
```

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:

1	= CR200
2	= CR10XPB, CR23XPB, CR510PB
3	= CR1000
4	= CR3000
5	= CR800 Series
9	= CR6 Series
13	= CR300 Series
14	= CR1000X Series
15	= GRANITE 9
16	= GRANITE 10
17	= GRANITE 6

table_name: Pointer to the memory location of a char array defining the name of the table in which the field will be set. Must be a null-terminated array of UTF8 encoded bytes.

field_name: Pointer to the memory location of a char array defining the field that will be set with the new value. Must be a null-terminated array of UTF8 encoded bytes.

value: Pointer to the memory location of a char array defining the value used to set the field. Must be a null-terminated array of UTF8 encoded bytes.

Return Codes

0	= Successful.
–1	= Communication timed out.
–2	= Port is not open.

2.9 GetData()

Queries the data logger for records and returns each record formatted as a list of fieldname:value pairs. A return code of ‘1’ indicates that additional records remain to be transferred. The function call should be iterated until the return code is ‘0’.

Syntax

int _stdcall GetData (int pakbus_address, int device_type, int table_no, int record_no, char **return_data, int *return_data_len)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:

1	= CR200
2	= CR10XPB, CR23XPB, CR510PB
3	= CR1000
4	= CR3000
5	= CR800 Series
9	= CR6 Series
13	= CR300 Series
14	= CR1000X Series
15	= GRANITE 9
16	= GRANITE 10
17	= GRANITE 6

table_no: The number for the table from which to collect data.

record_no: The record number where data collection will start. All records following this record number will be included in the collection. Therefore, if the record number is set to 0, all records in the table will be collected. In addition, if the record number specified does not exist in the data logger, all existing records from the oldest to the newest will be returned. However, if the record number is set to a negative number, only the most recent record in the table will be collected. There is not a way to specify and collect a single record from a table using this command unless that record is the most recent record in the table.

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the data logger.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0	= Complete.
1	= Successful but more data to collect.
–1	= Communication timed out.
–2	= Port is not open.
–3	= Invalid table number.

Example of data returned by function call

```
"2004-04-16 14:18:03",1          (Time stamp, Record number)
1,OSversion,v03A                 (Field number, Field name, Field value)
2,OSDate,06-Jan-04
3,ProgName,BATT.CR2
4,ProgSig,54451
5,CalOffset,2.625
6,PakBusAddress,1
7,RfInstalled,424
8,RfNetAddr,0
9,RfAddress,0
10,RfHopSeq,0
11,RfPwrMode,RF1_Sec
12,Rf_ForceOn,0
13,RfSignalLevel,0
14,RfRxPakBusCnt,0
15,VarOutOfBounds,0
16,SkipScan,0
17,TrapCode,0
18,WatchDogCnt,0
19,ResetTables,0
20,BattVoltage,12.3943
```

2.10 GetDataHeader()

Returns the TOA5 file header for the specified table.

Syntax

int _stdcall GetDataHeader (int pakbus_address, int device_type, int table_no, char **return_data, int *return_data_len)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:
1 = CR200
2 = CR10XPB, CR23XPB, CR510PB
3 = CR1000
4 = CR3000
5 = CR800 Series
9 = CR6 Series
13 = CR300 Series
14 = CR1000X Series
15 = GRANITE 9
16 = GRANITE 10
17 = GRANITE 6

table_no: The number of the table for which the header will be generated.

return_data: Pointer to a pointer to the memory location of a char array containing the header returned by the DLL.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0 = Successful.
1 = Successful but more data to collect.
–1 = Communication timed out.
–2 = Port is not open.
–3 = Invalid table number.

Example of data returned by function call

"TIMESTAMP","RECORD", OSVersion, OSDate, OSSignature

2.11 GetCommaData()

Queries the data logger for records and returns each record in a TOA5 comma-separated format. A return code of '1' indicates that additional records remain to be transferred. The function call should be iterated until the return code is '0'.

Syntax

int _stdcall GetData (int pakbus_address, int device_type, int table_no, int record_no, char **return_data, int *return_data_len)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:
1 = CR200
2 = CR10XPB, CR23XPB, CR510PB
3 = CR1000
4 = CR3000
5 = CR800 Series
9 = CR6 Series
13 = CR300 Series
14 = CR1000X Series
15 = GRANITE 9
16 = GRANITE 10
17 = GRANITE 6

table_no: The number for the table from which to collect data.

record_no: The record number where data collection will start. All records following this record number will be included in the collection. Therefore, if the record number is set to 0, all records in the table will be collected. In addition, if the record number specified does not exist in the data logger, all existing records from the oldest to the newest will be returned. However, if the record number is set to a negative number, only the most recent record in the table will be collected. There is not a way to specify and collect a single record from a table using this command unless that record is the most recent record in the table.

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the data logger.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0 = Complete.
1 = Successful but more data to collect.
–1 = Communication timed out.
–2 = Port is not open.
–3 = Invalid table number.

Example of data returned by function call

"2005-09-08 14:13:47",1,"CR1000.Std.05","050624",47178

2.12 File_Send()

Sends the specified program to the data logger. A return code of ‘1’ indicates that a fragment of the file has been successfully transferred, but additional fragments remain. The array pointed to by ‘return_data’ will contain a string indicating the current progress of the file transfer. The function call should be iterated until the return code is ‘0’. Once the operation is complete, ‘return_data’ will point to an array containing the compile results.

Sending a .CR2 file to a CR200 will cause the Server to attempt to invoke the CR200 compiler located at *C:\Campbellsci\Lib\CR200Compilers*. If the compiler is not installed, an error will be returned.

Syntax

int _stdcall File_Send (int pakbus_address, int device_type, char const *file_name, char **return_data, int *return_data_len)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:
1 = CR200
2 = CR10XPB, CR23XPB, CR510PB
3 = CR1000
4 = CR3000
5 = CR800 Series
9 = CR6 Series
13 = CR300 Series
14 = CR1000X Series
15 = GRANITE 9
16 = GRANITE 10
17 = GRANITE 6

file_name: Pointer to the memory location of a char array defining the path and file name of the program file to be sent to the data logger. Must be a null-terminated array of UTF8 encoded bytes.

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the DLL.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0 = Complete.
1 = Successful but more data to transfer.
–1 = Communication timed out.
–2 = Port is not open.
–3 = Cannot open source file.
–4 = File name is too long.
–5 = Data logger timed out.
–6 = File offset does not match.
–7 = Data logger reported an error.
–8 = File control error.
–9 = Cannot get program status.

Example of data returned from a CR1000

```
OS Version: CR1000.Std.05
OS Signature: 19128
Serial Number: 1031
PowerUp Progr: CPU:Program.cr1
Compile Status: Data Logger Program Running
Program Name: CPU:Program.cr1
Program Sig.: 32083
Compile Result: Compiled in SequentialMode.
```

2.13 GetAddress()

Queries the open port for a connected PakBus device; if found, the PakBus address is returned. If multiple PakBus devices are connected, only the first to respond is reported.

Syntax

```
int _stdcall GetAddress ( int device_type, char **return_data, int
*return_data_len )
```

Parameters

device_type: Type of data logger:
1 = CR200
2 = CR10XPB, CR23XPB, CR510PB
3 = CR1000
4 = CR3000
5 = CR800 Series
9 = CR6 Series
13 = CR300 Series
14 = CR1000X Series
15 = GRANITE 9
16 = GRANITE 10
17 = GRANITE 6

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the DLL.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0 = Successful.
–1 = Communication timed out.
–2 = Port is not open.

Example of data returned by function call

PakBusAddress=1;

2.14 GetStatus()

Queries the data logger for a summary of its current status.

Syntax

int _stdcall GetStatus (int pakbus_address, int device_type, char **return_data, int *return_data_len)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:

1 =	CR200
2 =	CR10XPB, CR23XPB, CR510PB
3 =	CR1000
4 =	CR3000
5 =	CR800 Series
9 =	CR6 Series
13 =	CR300 Series
14 =	CR1000X Series
15 =	GRANITE 9
16 =	GRANITE 10
17 =	GRANITE 6

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the data logger.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0 = Successful.
–1 = Communication timed out.
–2 = Port is not open.

Example of data returned from a CR200

```
OS Version: v03A
OS Signature: 43529
Serial Number:
PowerUp Progr:
Compile Status: Data Logger Program Running
Program Name: BATT.CR2
Program Sig.: 54451
Compile Result: Program Running
Batt=12.38V
```

2.15 GetTableNames()

Query the data logger for its table names and numbers.

Syntax

int _stdcall GetTableNames (int pakbus_address, int device_type, char **return_data, int *return_data_len)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:

1	= CR200
2	= CR10XPB, CR23XPB, CR510PB
3	= CR1000
4	= CR3000
5	= CR800 Series
9	= CR6 Series
13	= CR300 Series
14	= CR1000X Series
15	= GRANITE 9
16	= GRANITE 10
17	= GRANITE 6

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the data logger.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0	= Successful.
–1	= Communication timed out.
–2	= Cannot read table definitions from the data logger.

Example of data returned by function call

```
1 Status
2 DataTable1
3 DataTable2
4 Public
```

2.16 GetDLLVersion()

Gets the version of the SimplePB.dll being used.

Syntax

int _stdcall GetDLLVersion (char **return_data, int *return_data_len)

Parameters

return_data: Pointer to a pointer to the memory location of a char array containing the data returned from the data logger.

return_data_len: Pointer to the memory location containing the length of the char array returned from the DLL.

Return Codes

0 = Successful.

Example of data returned by function call

SimplePB.dll Version 2.0 / 2,2,3,0

2.17 GetLastResults()

Retrieves the return_data results from memory for the previous function as a String. This function is useful for developers that don’t want to manage memory pointers. A new BSTR is allocated each time this function is called.

Syntax

BSTR _stdcall GetLastResults ()

2.18 FileControl()

Used to control compilation and execution of the data logger program and do file management.

Syntax

int _stdcall FileControl (int pakbus_address, int device_type, char const *file_name, int command)

Parameters

pakbus_address: PakBus address of the data logger.

device_type: Type of data logger:

1 = CR200
2 = CR10XPB, CR23XPB, CR510PB
3 = CR1000
4 = CR3000
5 = CR800 Series
9 = CR6 Series
13 = CR300 Series
14 = CR1000X Series
15 = GRANITE 9
16 = GRANITE 10
17 = GRANITE 6

file_name: Pointer to the memory location of a char array defining the path and file name of the device or file subject to the specified command.

command: Specifies the action to be executed upon the specified device or file:

1 = Compile and run; marks the program as “run on power up”
2 = Run on power up
3 = Make hidden
4 = Delete file
5 = Format device
6 = Compile and run (preserve data if no table changed)
7 = Stop running program
8 = Stop running program and delete associated files
9 = Make the specified file the operating system
10 = Compile and run but do not change the "run on power up" program
11 = Pause execution of the running program
12 = Resume execution of the running program
13 = Stop the running program, delete associated files, and mark as run now and on power up
14 = Stop the running program, delete associated files, and mark as run now but not on power up

Return Codes

0 = Successful.
–1 = Communication timed out.
–2 = Port is not open.

2.19 SetSecurity()

Sets the security code that will be used to communicate with the data logger.

Syntax

int _stdcall SetSecurity (int security_code)

Parameter

Security_code: Security code to use.

Return Codes

0 = Success.
–1 = Failure.

2.20 GetTableRecordsCount()

Queries the data logger to determine the number of records that are available for collection from the specified table.

Syntax

int _stdcall GetTableRecordsCount (int pakbus_address, int device_type, int table_no, unsigned long *records_count)

Parameters

pakbus_address: The PakBus address of the data logger.

Device_type: Type of data logger:

1 = CR200
2 = CR10XPB, CR23XPB, CR510PB
3 = CR1000
4 = CR3000
5 = CR800 Series
9 = CR6 Series
13 = CR300 Series
14 = CR1000X Series
15 = GRANITE 9
16 = GRANITE 10
17 = GRANITE 6

table_no: Number of the table from which to get the records count.

records_count: Pointer to the memory location where the records count value will be written.

Return Codes

0 = Successful.
1 = Successful but more data to collect.
–1 = Communication timed out.
–2 = Port is not open.
–3 = Invalid table number.

Appendix A. Sample Program Table Structure

The table structure of a PakBus® data logger is given in the following example. This example shows a data logger with two user defined tables plus the **Status** table, **DataTableInfo** table, and **Public** or **Inlocs** table. The second table in the following example contains three records and the third table contains four records. The **Status** table, **DataTableInfo** table, and **Public** or **Inlocs** table will always return the most recent records and will not contain any historical data records.

The first table is the **Status** table, which shows the status of the data logger. The **DataTableInfo** table shows information such as name, skipped records, size, and time to fill for all user-defined tables. The **Public** or **Inlocs** table contains all public variables or input locations. All other tables found in the data logger are created and defined by the user in the data logger program. The tables in a PakBus data logger will always contain a record number and timestamp followed by the data fields.

NOTE The **DataTableInfo** table is only present in newer data loggers and/or with newer operating systems.

Table 1 – Status

Record No	Time Stamp	Data Field 1	Data Field 2	Data Field 3-72	Data Field 73

Table 2 – User Defined

RN 0	Time Stamp	Data Field 1	Data Field 2
RN 1	Time Stamp	Data Field 1	Data Field 2
RN 2	Time Stamp	Data Field 1	Data Field 2

Table 3 – User Defined

RN 0	Time Stamp	Data Field 1	Data Field 2	Data Field 3	Data Field 4	Data Field 5
RN 1	Time Stamp	Data Field 1	Data Field 2	Data Field 3	Data Field 4	Data Field 5
RN 2	Time Stamp	Data Field 1	Data Field 2	Data Field 3	Data Field 4	Data Field 5
RN 3	Time Stamp	Data Field 1	Data Field 2	Data Field 3	Data Field 4	Data Field 5

Table 4 – DataTableInfo

Record No	Time Stamp	Data Field 1	Data Field 2	Data Field 3-17	Data Field 18

Table 5 – Public or Inlocs

Record No	Time Stamp	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6

A.1 CR1000X Data Logger Program Tables

The following tables show the table structure from a program installed in a CR1000X data logger. This program measures and stores the minimum battery voltage and the minimum and maximum temperature over a 60-minute interval. When communicating with a data logger using the BMP5 Direct SDK, knowing the table structure of the running program is necessary for some commands.

NOTE Although each record of a table has an associated timestamp and record number, they are not relevant when determining which field number to use, and therefore not shown.

Table Number 1 – Status

Field Number	Field Name	Units	Notes
Field 1	OSVersion		Version of the operating system. Updated at OS startup.
Field 2	OSDate		Build date of the operating system in the format mmddyyyy. Updated at startup.
Field 3	OSSignature		Signature of the operating system.
Field 4	SerialNumber		Serial number assigned by the factory when the data logger was calibrated. Stored in flash memory. Updated at startup.
Field 5	RevBoard		Electronics board revision in the form xxx.yyy, where xxx = hardware revision number; yyy = clock chip software revision. Stored in flash memory. Updated at startup.
Field 6	StationName		Station name stored in flash memory. This is not the same name as that is entered into your data logger support software. This station name can be sampled into a data table, but it is not the name that appears in data file headers. Updated at startup or when the name is changed.
Field 7	ProgName		Name of current (running) program; updates at startup.
Field 8	StartTime		Time (date and time) the CRBasic program started. Updates at beginning of program compile.
Field 9	RunSignature		Signature of the running binary (compiled) program. Value is independent of comments or non-functional changes. Often changes with operating-system changes. Updates after compiling and before running the program.
Field 10	ProgSignature		Signature of the running CRBasic program including comments. Does not change with operating-system changes. Updates after compiling the program.

Field 11	WatchdogErrors		Number of watchdog errors that have occurred while running this program. Resets automatically when a new program is compiled. Enter 0 to reset. Updated at startup and at occurrence.
Field 12	PanelTemp	Deg C	Current temperature of the data logger's main processing board.
Field13	Battery	Volts	Voltage (Vdc) of the battery powering the system. Updates when viewing the Status table or via program code.
Field 14	LithiumBattery	Volts	Voltage of the internal lithium battery.
Field 15	Low12VCount		Counts the number of times the primary CR1000X supply voltage drops below ≈9.0 VDC.
Field 16	CompileResults		Contains error messages generated at compilation or during runtime. Updated after compile. Also appended to at run time for run time errors such as variable out of bounds.
Field 17	StartUpCode		Indicates how the running program was compiled. Updated at startup. 0 = Normal shutdown -1 = Restart due to a power loss. -2 = Restart due to watchdog reset
Field 18	ProgErrors		Number of compile or runtime errors for the running program. Updated after compile.
Field 19	VarOutOfBound		Number of attempts to write to an array outside of the declared size. The write does not occur. Indicates a CRBasic program error. If an array is used in a loop or expression, the pre-compiler and compiler do not check to see if an array is accessed out-of-bounds (i.e., accessing an array with a variable index such as arr(index) = arr(index–1), where index is a variable). Updated at run time when the error occurs.
Field 20	SkippedScan		Number of skipped program scans that have occurred while running the CRBasic program. Does not include scans intentionally skipped as may occur with the use of ExitScan and Do / Loop instructions. Updated when they occur.
Field 21	SkippedSystemScan		Number of scans skipped in the background calibration.
Field 22	ErrorCalib		Number of erroneous calibration values measured. Erroneous values are discarded.
Field 23	MemorySize	Bytes	Total final storage memory size. Updated at startup.
Field 24	MemoryFree	Bytes	Unallocated final storage memory on the CPU. All free memory may not be available for data tables. As memory is allocated and freed, holes of unallocated memory, which are unusable for final-storage memory, may be created. Updated after compile completes.
Field 25	CommsMemFree		Memory allocations for communications. Numbers outside of parentheses reflect current memory allocation. Numbers inside parentheses reflect the lowest memory size reached.
Field 26	FullMemReset		Enter 98765 to start a full-memory reset.
Field 27	CardStatus		Contains a string with the most recent status information for the removable memory card.

Field 28	MeasureOps		Reports the number of task-sequencer opcodes required to do all measurements. Calculated at compile time. Includes operation codes for calibration (compile time), auto (background) calibration (system), and Slow Sequences. Assumes all measurement instructions run each scan. Updated after compile and before running.
Field 29	MeasureTime	µs	Reports the time needed to make measurements in the current scan. Calculated at compile time. Includes integration and settling time. Assumes all measurement instructions will run each scan. Updated when a main scan begins.
Field 30	ProcessTime	µs	Processing time of the last scan. Time is measured from the end of the EndScan instruction (after the measurement event is set) to the beginning of the EndScan (before the wait for the measurement event begins) for the subsequent scan. Calculated on-the-fly. Updated at the conclusion of scan processing, prior to waiting for the next scan.
Field 31	MaxProcTime	µs	Maximum time required to run through processing for the current scan. Value is reset when the scan exits. Enter 0 to reset. Updated at the conclusion of scan processing, prior to waiting for the next scan.
Field 32	BuffDepth		Shows the current pipeline mode processing buffer depth, which indicates how far the processing task is currently behind the measurement task. Updated at the conclusion of scan processing, prior to waiting for the next scan.
Field 33	MaxBuffDepth		Maximum number of buffers the CR1000X will use to process lagged measurements.
Field 34	LastSystemScan		Reports the time of the of the last auto (background) calibration, which runs in a hidden slow-sequence type scan.
Field 35	SystemProcTime	µs	Time required to process auto (background) calibration.
Field 36	MaxSystemProcTime	µs	Maximum time required to process the auto (background) calibration, which runs in a hidden slow-sequence type scan. Displays 0 until a background calibration runs.
Field 37	PortStatus(1)		States of C terminals configured for control. On/high (True) or off/low (False). Array elements in numeric order of C terminals. Default = False. Updates when state changes.
Field 38	PortStatus(2)		
Field 39	PortStatus(3)		
Field 40	PortStatus(4)		
Field 41	PortStatus(5)		
Field 42	PortStatus(6)		
Field 43	PortStatus(7)		
Field 44	PortStatus(8)		

Field 45	PortConfig(1)		Sets up C terminals in numeric order of terminals. Set up for input, output, SDI-12, COM port. Default = Input. Updates when the port configuration changes.
Field 46	PortConfig(2)		
Field 47	PortConfig(3)		
Field 48	PortConfig(4)		
Field 49	PortConfig(5)		
Field 50	PortConfig(6)		
Field 51	PortConfig(7)		
Field 52	PortConfig(8)		
Field 53	SW12Volts(1)		Status of switched, 12 Vdc terminal. True = on. Updates when the state changes.
Field 54	SW12Volts(2)		
Field 55	PakBusRoutes		Lists routes or router neighbours known to the data logger at the time the setting was read. Each route is represented by four components separated by commas and enclosed in parentheses: (port, via neighbour address, pakbus address, response time in ms). Default = (1, 4089, 4089, 1000). Updates when routes are added or deleted.
Field 56	Messages		Contains a string of manually entered messages.
Field 57	CalVolts(1)		Array of floating-point values reporting a factory calibrated correction factor for the different voltage ranges.
Field 58	CalVolts(2)		
Field 59	CalVolts(3)		
Field 60	CalRefSlope(1)		Displays voltage reference temperature compensation slope.
Field 61	CalRefSlope(2)		
Field 62	CalRefSlope(3)		
Field 63	CalRefOffset(1)		Displays voltage reference temperature compensation offset.
Field 64	CalRefOffset(2)		
Field 65	CalRefOffset(3)		
Field 66	CalGain(1)		Array of floating-point values reporting calibration gain (mV) for each integration / range combination.
Field 67	CalGain(2)		
Field 68	CalGain(3)		
Field 69	CalOffset(1)		Displays the offset calibration factor for the different voltage ranges.
Field 70	CalOffset(2)		
Field 71	CalOffset(3)		
Field 72	CalCurrent(1)		Shows the offset calibration factor for the resistor used in 0-20 and 4-20 mA measurements on RG terminals. Measured once during production calibration.
Field 73	CalCurrent(2)		

Table Number 2 – Hourly: The **Hourly** table contains the minimum battery voltage and the minimum and maximum temperature over a 60-minute interval.

Field Number	Field Name	Units	Notes
Field 1	Battery_Min	Volts	
Field 2	Battery_Time	Time	
Field 2	Temp_Min	Deg C	
Field 3	Temp_Max	Deg C	

Table Number 3 – DataTableInfo

Field Number	Field Name	Units	Notes
Field 1	DataTableName(1)		Reports the name of the data table. Each table has its own entry in an array. Array elements are in the order the data tables are declared in the CRBasic program.
Field 2	SkippedRecord		Reports how many times records have been skipped in a data table. For multiple tables, each table has its own entry in an array.
Field 3	DataRecordSize(1,1)	Records	Reports the number of records allocated to a data table. Each table has its own entry in a two-dimensional array. First dimension is for on-board memory. Second dimension is for card memory.
Field 4	DataRecordSize(1,2)	Records	
Field 5	SecsPerRecord	Seconds	Reports the data output interval for a data table. For multiple tables, each table has its own entry in an array.
Field 6	DataFillDays(1,1)	Days	Reports the time required to fill a data table. Each table has its own entry in a two-dimensional array. First dimension is for on-board memory. Second dimension is for card memory.
Field 7	DataFillDays(1,2)	Days	
Field 8	DataFilled(1,1)	Percentage	Reports the current field level of the table as a percentage of total. Each table has its own entry in a two-dimensional array. First dimension is for on-board memory. Second dimension is for card memory.
Field 9	DataFilled(1,2)	Percentage	

NOTE With multiple tables, these field numbers will change as additional array elements are added for each table.

Table Number 4 – Public: The **Public** table contains only the most recent “real-time” record for the variable described in the data logger program.

Field Number	Field Name	Units	Notes:
Field 1	Batt_Volt	Volts	
Field 2	Temp	Deg C	

A.1.1 CR1000X Data Logger Program

CRBasic Example A-1. CR1000X Data Logger Program

```
'CR1000X Series Data Logger

'Declare Variables and Units
Public Batt_Volt, Temp
Units Batt_Volt=Volts
Units Temp=Deg C

'Define Data Tables
DataTable(Hourly,True,-1)
  DataInterval(0,60,Min,10)
  Minimum(1,Batt_Volt,FP2,False,True)
  FieldNames("Battery_Min,Battery_Time")
  Minimum(1,Temp,FP2,False,False)
  Maximum(1,Temp,FP2,False,False)
EndTable

'Main Program
BeginProg
  Scan(10,Sec,3,0)
    'Default Data Logger Battery Voltage measurement Batt_Volt:
    Battery(Batt_Volt)
    '109 Temperature Probe measurement Temp:
    Therm109(Temp,1,1,Vx1,0,60,1.0,0)
    'Call Data Tables and Store Data
    CallTable(Hourly)
  NextScan
EndProg
```

INFO

CAMPBELL SCIENTIFIC®
WHEN MEASUREMENTS MATTER

Global Sales & Support Network

A worldwide network to help meet your needs

Campbell Scientific regional offices

Australia

Location:	Garbutt, QLD Australia
Phone:	61.7.4401.7700
Email:	info@campbellsci.com.au
Website:	www.campbellsci.com.au

Brazil

Location:	São Paulo, SP Brazil
Phone:	11.3732.3399
Email:	vendas@campbellsci.com.br
Website:	www.campbellsci.com.br

Canada

Location:	Edmonton, AB Canada
Phone:	780.454.2505
Email:	dataloggers@campbellsci.ca
Website:	www.campbellsci.ca

China

Location:	Beijing, P. R. China
Phone:	86.10.6561.0080
Email:	info@campbellsci.com.cn
Website:	www.campbellsci.com.cn

Costa Rica

Location:	San Pedro, Costa Rica
Phone:	506.2280.1564
Email:	info@campbellsci.cc
Website:	www.campbellsci.cc

France

Location:	Vincennes, France
Phone:	0033.0.1.56.45.15.20
Email:	info@campbellsci.fr
Website:	www.campbellsci.fr

Germany

Location:	Bremen, Germany
Phone:	49.0.421.460974.0
Email:	info@campbellsci.de
Website:	www.campbellsci.de

India

Location:	New Delhi, DL India
Phone:	91.11.46500481.482
Email:	info@campbellsci.in
Website:	www.campbellsci.in

South Africa

Location:	Stellenbosch, South Africa
Phone:	27.21.8809960
Email:	sales@campbellsci.co.za
Website:	www.campbellsci.co.za

Spain

Location:	Barcelona, Spain
Phone:	34.93.2323938
Email:	info@campbellsci.es
Website:	www.campbellsci.es

Thailand

Location:	Bangkok, Thailand
Phone:	66.2.719.3399
Email:	info@campbellsci.asia
Website:	www.campbellsci.asia

UK

Location:	Shepshed, Loughborough, UK
Phone:	44.0.1509.601141
Email:	sales@campbellsci.co.uk
Website:	www.campbellsci.co.uk

USA

Location:	Logan, UT USA
Phone:	435.227.9120
Email:	info@campbellsci.com
Website:	www.campbellsci.com