Campbell Scientific, Inc.
BMP5 SDK End User License Agreement (EULA)

NOTICE OF AGREEMENT: Please carefully read this EULA. By installing or using this software, you are agreeing to comply with the terms and conditions herein. The term "developer" herein refers to anyone using this BMP5 Direct SDK.

By accepting this agreement, you acknowledge and agree that Campbell Scientific may from time-to-time, and without notice, make changes to one or more components of the SDK or make changes to one or more components of other software on which the SDK relies. In no instance will Campbell Scientific be responsible for any costs or liabilities incurred by you or other third parties as a result of these changes.

The core operational files included with this BMP5 Direct SDK (hereinafter referred to as “BMP5 Direct Binaries”) include the files: SimplePB.DLL and coralib3d.dll. Developer may distribute or sell their software including the BMP5 Direct Binaries subject to the terms hereafter set forth.

RELATIONSHIP

Campbell Scientific, Inc. hereby grants a license to use BMP5 Direct Binaries in accordance with the license statement above. No ownership in Campbell Scientific, Inc. patents, copyrights, trade secrets, trademarks, or trade names is transferred by this Agreement. Developer may use these BMP5 Direct Binaries to create as many applications as desired and freely distribute them. Campbell Scientific, Inc. expects no royalties or any other compensation. Developer is responsible for supporting applications created using the BMP5 Direct Binaries.

RESPONSIBILITIES OF DEVELOPER

The Developer agrees:

- To provide a competent programmer familiar with Campbell Scientific, Inc. data logger programming to write the applications.
- Not to sell or distribute documentation on use of the BMP5 Direct Binaries.
- Not to sell or distribute the applications that are provided as examples in the BMP5 Direct SDK.
- To develop original works. Developers may copy and paste portions of the code into their own applications, but their applications are expected to be unique creations.
- This Agreement does not give Developer the right to sell or distribute any other Campbell Scientific, Inc. Software (e.g., PC200W, VisualWeather, LoggerNet or any of their components, files, documentation, etc.) as part of Developer's application. Distribution of any other Campbell Scientific, Inc. software requires a separate distribution agreement.
- Not to sell or distribute applications that compete directly with any application developed by Campbell Scientific, Inc. or its affiliates.
- Not to use Campbell Scientific’s name, trademarks, or service marks in connection with any program you develop with the SDK. You may not state or infer in any way that Campbell Scientific endorses any program you develop, unless prior written approval is received from Campbell Scientific.
• To assure that each application developed with BMP5 Direct Binaries clearly states the name of the person or entity that developed the application. This information should appear on the first window the user will see.

**WARRANTY**

There is no written or implied warranty provided with the BMP5 Direct SDK software other than as stated herein. Developer agrees to bear all warranty responsibility of any derivative products distributed by Developer.

**TERMINATION**

Any license violation or breach of Agreement will result in immediate termination of the developer's rights herein and the return of all BMP5 Direct SDK materials to Campbell Scientific, Inc.

**MISCELLANEOUS**

Notices required hereunder shall be in writing and shall be given by certified or registered mail, return receipt requested. Such notice shall be deemed given in the case of certified or registered mail on the date of receipt. This Agreement shall be governed and construed in accordance with the laws of the State of Utah, USA. Any dispute resulting from this Agreement will be settled in arbitration.

This Agreement sets forth the entire understanding of the parties and supersedes all prior agreements, arrangements and communications, whether oral or written pertaining to the subject matter hereof. This Agreement shall not be modified or amended except by the mutual written agreement of the parties. The failure of either party to enforce any of the provisions of this Agreement shall not be construed as a waiver of such provisions or of the right of such party thereafter to enforce each and every provision contained herein. If any term, clause, or provision contained in this Agreement is declared or held invalid by a court of competent jurisdiction, such declaration or holding shall not affect the validity of any other term, clause, or provision herein contained. Neither the rights nor the obligations arising under this Agreement are assignable or transferable.

If within 30 days of receiving the BMP5 Direct SDK product developer does not agree to the terms of license, developer shall return all materials without retaining any copies of the product and shall remove any use of the BMP5 Direct Binaries in any applications developed or distributed by Developer. In the absence of such return, CSI shall consider Developer in agreement with the herein, stated license terms and conditions.

**COPYRIGHT**

This software is protected by United States copyright law and international copyright treaty provisions. This software may not be altered in any way without prior written permission from Campbell Scientific. All copyright notices and labeling must be left intact.
Limited Warranty

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.
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.

1. BMP5 Direct SDK Overview .............................................. 1
   1.1 General Notes on BMP5 Direct SDK Usage.................................1
   1.2 Data Logger Program Table Structure .......................................2
   1.3 Developing Applications Using the .NET Framework ...............2

2. SimplePB.dll Reference ............................................. 2
   2.1 OpenPort() ..........................................................................2
   2.2 ClosePort() ..........................................................................3
   2.3 OpenIPPort() .........................................................................3
   2.4 CloseIPPort() .........................................................................3
   2.5 GetClock() ...........................................................................4
   2.6 SetClock() ...........................................................................4
   2.7 GetValue() ...........................................................................5
   2.8 SetValue() ...........................................................................6
   2.9 GetData() ...............................................................................7
   2.10 GetDataHeader() ...................................................................9
   2.11 GetCommaData() ................................................................10
   2.12 File_Send() .........................................................................11
   2.13 GetAddress() ......................................................................12
   2.14 GetStatus() .........................................................................13
   2.15 GetTableNames() ................................................................14
   2.16 GetDLLVersion() ..................................................................15
   2.17 GetLastResults() ..................................................................15
   2.18 FileControl() .......................................................................16
   2.19 SetSecurity() .......................................................................16
   2.20 GetTableRecordsCount() .........................................................17

Appendix

A. Sample Program Table Structure ................................. A-1
   A.1 CR1000X Data Logger Program Tables .................................. A-2
       A.1.1 CR1000X Data Logger Program .......................................... A-7

CRBasic Example

   A-1. CR1000X Data Logger Program .......................................... A-7
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 “dialing” 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 behavior 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 runtime (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:

```csharp
[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.
2.1 OpenPort()

Syntax

```c
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

```c
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

```c
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

```c
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**

```c
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

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
```c
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**

```c
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

```c
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

```c
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**

```c
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**

```c
BSTR _stdcall GetLastResults ()
```

2.18 FileControl()

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

**Syntax**

```c
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 Inloc 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

<table>
<thead>
<tr>
<th>Record No</th>
<th>Time Stamp</th>
<th>Data Field 1</th>
<th>Data Field 2</th>
<th>Data Field 3-72</th>
<th>Data Field 73</th>
</tr>
</thead>
</table>

Table 2 – User Defined

<table>
<thead>
<tr>
<th>RN 0</th>
<th>Time Stamp</th>
<th>Data Field 1</th>
<th>Data Field 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RN 1</td>
<td>Time Stamp</td>
<td>Data Field 1</td>
<td>Data Field 2</td>
</tr>
<tr>
<td>RN 2</td>
<td>Time Stamp</td>
<td>Data Field 1</td>
<td>Data Field 2</td>
</tr>
</tbody>
</table>

Table 3 – User Defined

<table>
<thead>
<tr>
<th>RN 0</th>
<th>Time Stamp</th>
<th>Data Field 1</th>
<th>Data Field 2</th>
<th>Data Field 3</th>
<th>Data Field 4</th>
<th>Data Field 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>RN 1</td>
<td>Time Stamp</td>
<td>Data Field 1</td>
<td>Data Field 2</td>
<td>Data Field 3</td>
<td>Data Field 4</td>
<td>Data Field 5</td>
</tr>
<tr>
<td>RN 2</td>
<td>Time Stamp</td>
<td>Data Field 1</td>
<td>Data Field 2</td>
<td>Data Field 3</td>
<td>Data Field 4</td>
<td>Data Field 5</td>
</tr>
<tr>
<td>RN 3</td>
<td>Time Stamp</td>
<td>Data Field 1</td>
<td>Data Field 2</td>
<td>Data Field 3</td>
<td>Data Field 4</td>
<td>Data Field 5</td>
</tr>
</tbody>
</table>

Table 4 – DataTableInfo

<table>
<thead>
<tr>
<th>Record No</th>
<th>Time Stamp</th>
<th>Data Field 1</th>
<th>Data Field 2</th>
<th>Data Field 3-17</th>
<th>Data Field 18</th>
</tr>
</thead>
</table>
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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field 11</td>
<td>WatchdogErrors</td>
<td>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.</td>
</tr>
<tr>
<td>Field 12</td>
<td>PanelTemp</td>
<td>Deg C</td>
</tr>
<tr>
<td>Field 13</td>
<td>Battery</td>
<td>Volts</td>
</tr>
<tr>
<td>Field 14</td>
<td>LithiumBattery</td>
<td>Volts</td>
</tr>
<tr>
<td>Field 15</td>
<td>Low12VCount</td>
<td></td>
</tr>
<tr>
<td>Field 16</td>
<td>CompileResults</td>
<td></td>
</tr>
<tr>
<td>Field 17</td>
<td>StartUpCode</td>
<td></td>
</tr>
<tr>
<td>Field 18</td>
<td>ProgErrors</td>
<td></td>
</tr>
<tr>
<td>Field 19</td>
<td>VarOutOfBound</td>
<td></td>
</tr>
<tr>
<td>Field 20</td>
<td>SkippedScan</td>
<td></td>
</tr>
<tr>
<td>Field 21</td>
<td>SkippedSystemScan</td>
<td></td>
</tr>
<tr>
<td>Field 22</td>
<td>ErrorCalib</td>
<td></td>
</tr>
<tr>
<td>Field 23</td>
<td>MemorySize</td>
<td>Bytes</td>
</tr>
<tr>
<td>Field 24</td>
<td>MemoryFree</td>
<td>Bytes</td>
</tr>
<tr>
<td>Field 25</td>
<td>CommsMemFree</td>
<td></td>
</tr>
<tr>
<td>Field 26</td>
<td>FullMemReset</td>
<td></td>
</tr>
<tr>
<td>Field 27</td>
<td>CardStatus</td>
<td></td>
</tr>
</tbody>
</table>
### 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 $\mu$s
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 $\mu$s
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 $\mu$s
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 $\mu$s
Time required to process auto (background) calibration.

### Field 36 MaxSystemProcTime $\mu$s
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)
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>PortConfig(1)</td>
</tr>
<tr>
<td>46</td>
<td>PortConfig(2)</td>
</tr>
<tr>
<td>47</td>
<td>PortConfig(3)</td>
</tr>
<tr>
<td>48</td>
<td>PortConfig(4)</td>
</tr>
<tr>
<td>49</td>
<td>PortConfig(5)</td>
</tr>
<tr>
<td>50</td>
<td>PortConfig(6)</td>
</tr>
<tr>
<td>51</td>
<td>PortConfig(7)</td>
</tr>
<tr>
<td>52</td>
<td>PortConfig(8)</td>
</tr>
<tr>
<td>53</td>
<td>SW12Volts(1)</td>
</tr>
<tr>
<td>54</td>
<td>SW12Volts(2)</td>
</tr>
<tr>
<td>55</td>
<td>PakBusRoutes</td>
</tr>
<tr>
<td>56</td>
<td>Messages</td>
</tr>
<tr>
<td>57</td>
<td>CalVolts(1)</td>
</tr>
<tr>
<td>58</td>
<td>CalVolts(2)</td>
</tr>
<tr>
<td>59</td>
<td>CalVolts(3)</td>
</tr>
<tr>
<td>60</td>
<td>CalRefSlope(1)</td>
</tr>
<tr>
<td>61</td>
<td>CalRefSlope(2)</td>
</tr>
<tr>
<td>62</td>
<td>CalRefSlope(3)</td>
</tr>
<tr>
<td>63</td>
<td>CalRefOffset(1)</td>
</tr>
<tr>
<td>64</td>
<td>CalRefOffset(2)</td>
</tr>
<tr>
<td>65</td>
<td>CalRefOffset(3)</td>
</tr>
<tr>
<td>66</td>
<td>CalGain(1)</td>
</tr>
<tr>
<td>67</td>
<td>CalGain(2)</td>
</tr>
<tr>
<td>68</td>
<td>CalGain(3)</td>
</tr>
<tr>
<td>69</td>
<td>CalOffset(1)</td>
</tr>
<tr>
<td>70</td>
<td>CalOffset(2)</td>
</tr>
<tr>
<td>71</td>
<td>CalOffset(3)</td>
</tr>
<tr>
<td>72</td>
<td>CalCurrent(1)</td>
</tr>
<tr>
<td>73</td>
<td>CalCurrent(2)</td>
</tr>
</tbody>
</table>
### Table Number 2 – Hourly: The Hourly table contains the minimum battery voltage and the minimum and maximum temperature over a 60-minute interval.

<table>
<thead>
<tr>
<th>Field Number</th>
<th>Field Name</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field 1</td>
<td>Battery_Min</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>Field 2</td>
<td>Battery_Time</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Field 2</td>
<td>Temp_Min</td>
<td>Deg C</td>
<td></td>
</tr>
<tr>
<td>Field 3</td>
<td>Temp_Max</td>
<td>Deg C</td>
<td></td>
</tr>
</tbody>
</table>

### Table Number 3 – DataTableInfo

<table>
<thead>
<tr>
<th>Field Number</th>
<th>Field Name</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field 1</td>
<td>DataTableName(1)</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Field 2</td>
<td>SkippedRecord</td>
<td></td>
<td>Reports how many times records have been skipped in a data table. For multiple tables, each table has its own entry in an array.</td>
</tr>
<tr>
<td>Field 3</td>
<td>DataRecordSize(1,1)</td>
<td>Records</td>
<td>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.</td>
</tr>
<tr>
<td>Field 4</td>
<td>DataRecordSize(1,2)</td>
<td>Records</td>
<td></td>
</tr>
<tr>
<td>Field 5</td>
<td>SecsPerRecord</td>
<td>Seconds</td>
<td>Reports the data output interval for a data table. For multiple tables, each table has its own entry in an array.</td>
</tr>
<tr>
<td>Field 6</td>
<td>DataFillDays(1,1)</td>
<td>Days</td>
<td>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.</td>
</tr>
<tr>
<td>Field 7</td>
<td>DataFillDays(1,2)</td>
<td>Days</td>
<td></td>
</tr>
<tr>
<td>Field 8</td>
<td>DataFilled(1,1)</td>
<td>Percentage</td>
<td>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.</td>
</tr>
<tr>
<td>Field 9</td>
<td>DataFilled(1,2)</td>
<td>Percentage</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Field Number</th>
<th>Field Name</th>
<th>Units</th>
<th>Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field 1</td>
<td>Batt_Volt</td>
<td>Volts</td>
<td></td>
</tr>
<tr>
<td>Field 2</td>
<td>Temp</td>
<td>Deg C</td>
<td></td>
</tr>
</tbody>
</table>
## A.1.1 CR1000X Data Logger Program

<table>
<thead>
<tr>
<th>CRBasic Example A-1. CR1000X Data Logger Program</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>'CR1000X Series Data Logger</strong></td>
</tr>
<tr>
<td><strong>'Declare Variables and Units</strong></td>
</tr>
<tr>
<td>Public Batt_Volt, Temp</td>
</tr>
<tr>
<td>Units Batt_Volt=Volts</td>
</tr>
<tr>
<td>Units Temp=Deg C</td>
</tr>
<tr>
<td><strong>'Define Data Tables</strong></td>
</tr>
<tr>
<td>DataTable(Hourly,True,-1)</td>
</tr>
<tr>
<td>DataInterval(0,60,Min,10)</td>
</tr>
<tr>
<td>Minimum(1,Batt_Volt,FP2,False,True)</td>
</tr>
<tr>
<td>FieldNames(&quot;Battery_Min,Battery_Time&quot;)</td>
</tr>
<tr>
<td>Minimum(1,Temp,FP2,False,False)</td>
</tr>
<tr>
<td>Maximum(1,Temp,FP2,False,False)</td>
</tr>
<tr>
<td>EndTable</td>
</tr>
<tr>
<td><strong>'Main Program</strong></td>
</tr>
<tr>
<td>BeginProg</td>
</tr>
<tr>
<td>Scan(10,Sec,3,0)</td>
</tr>
<tr>
<td><strong>'Default Data Logger Battery Voltage measurement Batt_Volt:</strong></td>
</tr>
<tr>
<td>Battery(Batt_Volt)</td>
</tr>
<tr>
<td><strong>'109 Temperature Probe measurement Temp:</strong></td>
</tr>
<tr>
<td>Therm109(Temp,1,1,Vx1,0,60,1.0,0)</td>
</tr>
<tr>
<td><strong>'Call Data Tables and Store Data</strong></td>
</tr>
<tr>
<td>CallTable(Hourly)</td>
</tr>
<tr>
<td>EndScan</td>
</tr>
<tr>
<td>EndProg</td>
</tr>
</tbody>
</table>