

Table of contents

Sensors overview	2
Dataloggers introduction	17
Datalogger support applications	22
Datalogger programming introduction	28

Sensors overview

In the Met Office we use a variety of sensors to measure meteorological parameters such as temperature, wind speed, pressure etc. Whilst these sensors can vary greatly in design and the technology used to measure these parameters, they can be divided into two categories based upon how they present their data output to the end user; **analogue** or **digital**. The type of output determines how we can connect to and record data from the sensor.

Analogue sensors

This type of sensor outputs a voltage, current, resistance or frequency pulse that varies proportionally and in a predictable manner with parameter being measured. This output has to accurately measured and then converted to an actual reading (e.g. temperature, humidity, wind speed) by e.g. a datalogger. Example of output ranges include 0-20mV and 0-1v for a voltage output. The sensor may have a built-in amplifier to lift the output to a more easily measurable range such as 0-5v.

Examples of analogue sensors

- HMP110 humidity sensor outputs 0v at 0% humidity and 1v at 100% humidity. This sensor can also be configured to output a digital reading. Two analogue outputs are provided, one for humidity and the other for temperature from the inbuilt Pt1000 electrical resistance sensing element.



HMP110 Humidity and temperature probe

- CMP10 solar radiation pyranometer. A thermopile sensing element generates a 0-20 millivolt output from incoming short wave solar radiation. No analogue signal amplification occurs. This device is also available in a digital version.



CMP10 Solar radiation sensor

- Pt100 temperature sensor. This consists of a film of platinum embedded in a protective sheath, whose electrical resistance varies in a known manner with temperature. The "Pt" refers to platinum and the "100" refers to its resistance at 0°C in ohms. As temperature increases, the resistance increases in a precisely defined, linear manner.



TempVue10

Digital sensors

These sensors transmit an actual reading of the parameter being measured e.g. 21.3 often in the form of an ASCII formatted text string e.g.

T= 21.3 C H= 79 % P= 1012.5 hPa

but the data could also be in a binary format message or protocol e.g. ModBus, UMB. The electrical interface to the sensor could be RS232 or RS485. Internally the sensor will

make an analogue measurement but this is processed onboard the sensor so that an actual reading is presented to the user.

Examples of digital sensors

- PTB330 pressure transmitter - RS232 interface with ASCII string of the format:

```
P1 1007.14 ***.* * 1007.2 1007.1 1007.1 000E8
```

This instrument normally has 3 pressure sensing elements so that a mean value can be provided in addition to each individual pressure reading. Excessive differences between the readings can be used to flag a suspect sensor. The unit can also be specified with a built-in display screen as well as optional analogue and RS485 outputs.



PTB330

- Thies ultrasonic wind sensor - RS485 full duplex (4-wire) with ASCII string output of wind direction and speed.



Thies ultrasonic

- Lufft SHM31 snow depth sensor - RS485 half duplex, snow depth readings are requested from the instrument using the UMB binary or ASCII protocols. Additionally, many other parameters such as laser temperature, measurement distance etc. can be requested from the instrument.




Lufft SHM31

Analogue vs. Digital

All the sensors we've mentioned so far can be considered analogue as far as the measurement of a meteorological parameter is concerned. This should not be surprising since we are usually measuring analogue changes in the atmosphere. What we refer to as a digital sensor is merely an analogue sensor with extra built-in electronics and processing to make the analogue measurement and transmit an actual reading. In fact, digital sensors are more properly called 'instruments' since they consist of more than just an analogue sensing element.

Such instruments are easier to connect to another digital device such as a PC since most computers don't have the electrical interfaces required to accurately measure small voltages, current or resistance, but they do normally need some configuration before use. Analogue sensors tend to be simpler and don't require configuration and setup, however you need to connect them to a specialist device such as a datalogger that has the electrical interfaces that can measure analogue inputs accurately and stably. The conversion to a usable, digital value is then done by the dataloggers internal processing, usually under the control of a datalogger program.

These days, many sensors can provide both digital and analogue outputs. The specifications page of the sensor manufacturers documentation will contain information regarding the outputs available from the sensor:

Radiometer Connection		
Anschluss • Raccordement • Conexión		
Wire Kabel Fil Cable	Function Funktion Fonction Función	Connect with Anschluss an Relier à Conectar con
3 Green Grün • Vert • Verde	Analogue out	V+/4-20 mA(+)
6 Brown Braun • Brun • Marrón	Analogue ground	V-/4-20 mA(-)
4 Yellow Gelb • Jaune • Amarillo	Modbus® RS-485	B/B'/+
5 Grey Grau • Gris • Gris	Modbus® RS-485	A/A'/-
7 White Weiss • Blanc • Blanco	Power 5 to 30 VDC (12 V recommended)	
8 Black Schwarz • Noir • Negro	Power ground	
1 Red Rot • Rouge • Rojo	None	Not connected
2 Blue Blau • Bleu • Azul	Modbus® common / Ground	
 Shield Abschirmung Protection Malla	Housing Gehäuse Boîte Cubierta	Ground * Erde Terre Tierra
* Connect to ground if radiometer not grounded Mit Erde verbinden, wenn das Radiometer nicht geerdet ist Reliez à la terre si le radiomètre n'est pas connecté Conectar a tierra si el radiómetro no lo está		

Analogue and digital output sensor example

Analogue Sensor interfacing

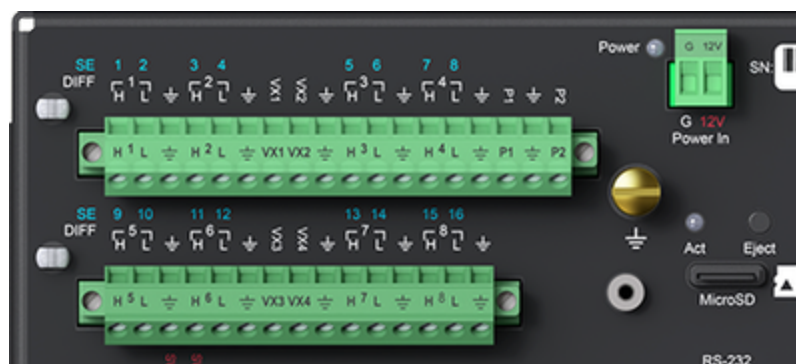
The analogue output (voltage, current or resistance) from an analogue sensor can be connected to a datalogger with a built-in analogue to digital converter (ADC), modern dataloggers use a high resolution 24bit ADC. Under the control of the logger program the analogue output can be converted to a digital value representing the parameter we are trying to measure. You can read more about the process of analogue to digital conversion

here (<https://www.electronics-tutorials.ws/combination/analogue-to-digital-converter.html>).

In addition to the ADC, the datalogger usually has two types of terminal connection for analogue inputs:

- **Single ended (SE)** - one wire of the sensor is connected to an SE input terminal and the other wire to an earth terminal. This way we measure our analogue signal with reference to earth. For higher signal output voltage levels e.g. 0-1 volts, this works fine but for lower output voltages e.g. (10 millivolts) an incorrect reading could occur due to electrical noise on the inputs which could be attributed to the sensor itself.
- **Differential (H/L)** - For low output signals mentioned above it is better to use a differential input. This way we measure the voltage difference between two terminals, normally labelled high (H) and low (L). Now we are measuring the output voltage difference between the two terminals rather than taking a measurement referenced to ground, this method helps to negate any electrical noise on the inputs since such noise will be affecting both inputs simultaneously. The disadvantage of using differential inputs is that we require two input channels rather than the one used by a single ended input.

Here's an image showing the single ended and differential channels on a CR1000X datalogger. Note how the different channels are numbered and marked on the logger - we have 16 single ended and 8 differential inputs:



CR1000X Analogue inputs

Digital sensor interfacing

There is nothing to measure from a digital sensor, the output will be binary data sent over a standard electrical interface such as RS232/RS485. This data has to be decoded into readable text which will contain the measured data from the sensor. A datalogger such as the CR1000X has a number of 'digital' input ports that can be configured for RS232 or RS485 connections. These types of sensors can also be connected to laptop/PC using a USB to RS232/485 converter, which is useful since the best way to set up these sensors (e.g. message output type, units, interval between data transmissions etc.) is using 'terminal' software or a manufacturers software application. This is also the easiest way to check that a digital sensor is working. It is recommended to use a converter based on an "FTDI" chipset as these have been proven to be most reliable. An FTDI driver is available on the Met Office "Company Portal" to allow installation on Met Office laptops if required.

Digital input ports on dataloggers such as the CR1000X are normally labelled as "C" ports, there may also be a separate dedicated input as shown below (marked RS232).



CR1000X digital inputs

USB to RS232/485 converters

It is recommended to use a converter based on an "FTDI" chipset as these have been proven to be most reliable. An FTDI driver is available on the Met Office "Company Portal" to allow installation on Met Office laptops if required. An example of this type of converter is shown below. This one can be purchased from here.

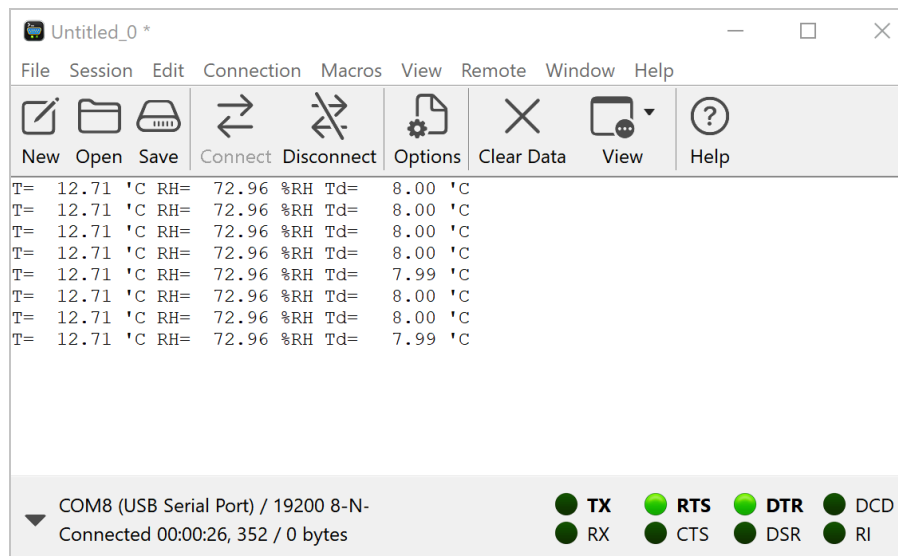
(<https://thepihut.com/products/ft232rnl-usb-to-rs232-485-422-ttl-interface-converter>)



RS232 / 485/ 422 USB converter

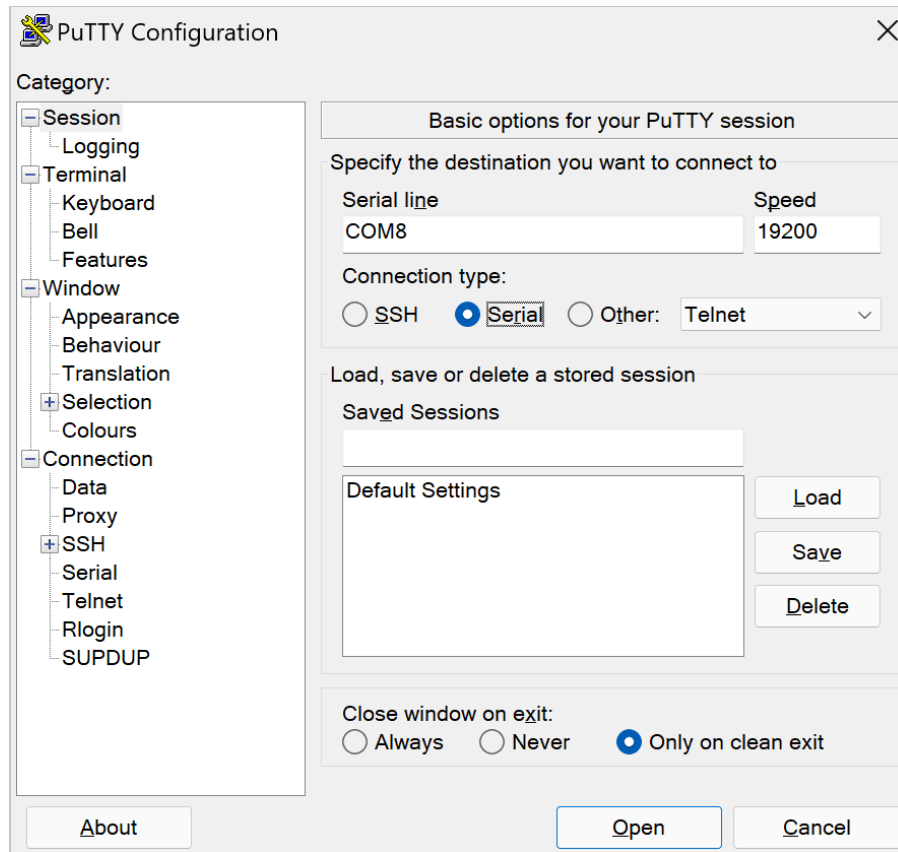
Terminal software and serial settings

A terminal program can be used to receive and display the data from your USB/RS232/485 converter and setup the correct communications parameters. There are a number of free terminal programs available. My personal favourite is 'CoolTerm' which can be obtained from here. (<https://freeware.the-meiers.org/>) This can be installed on a 'dirty' (non Met Office) laptop.

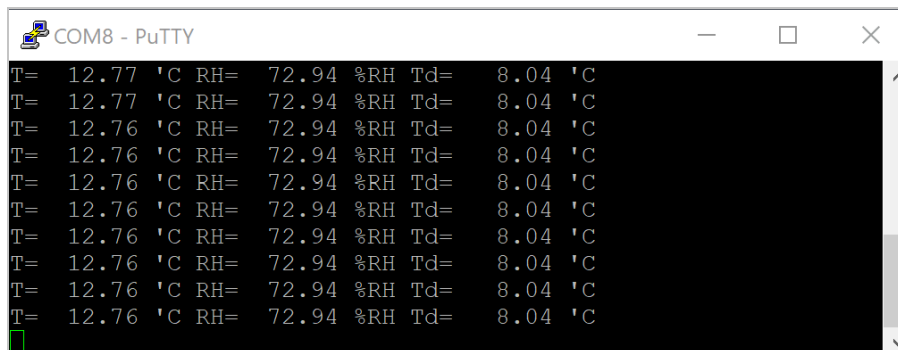


CoolTerm Application

In the 'Company Portal' on a Met Office laptop there is a program called "PuTTY" that can be installed. This is pretty basic as far as serial connections are concerned, but it can do the job.



Putty_settings

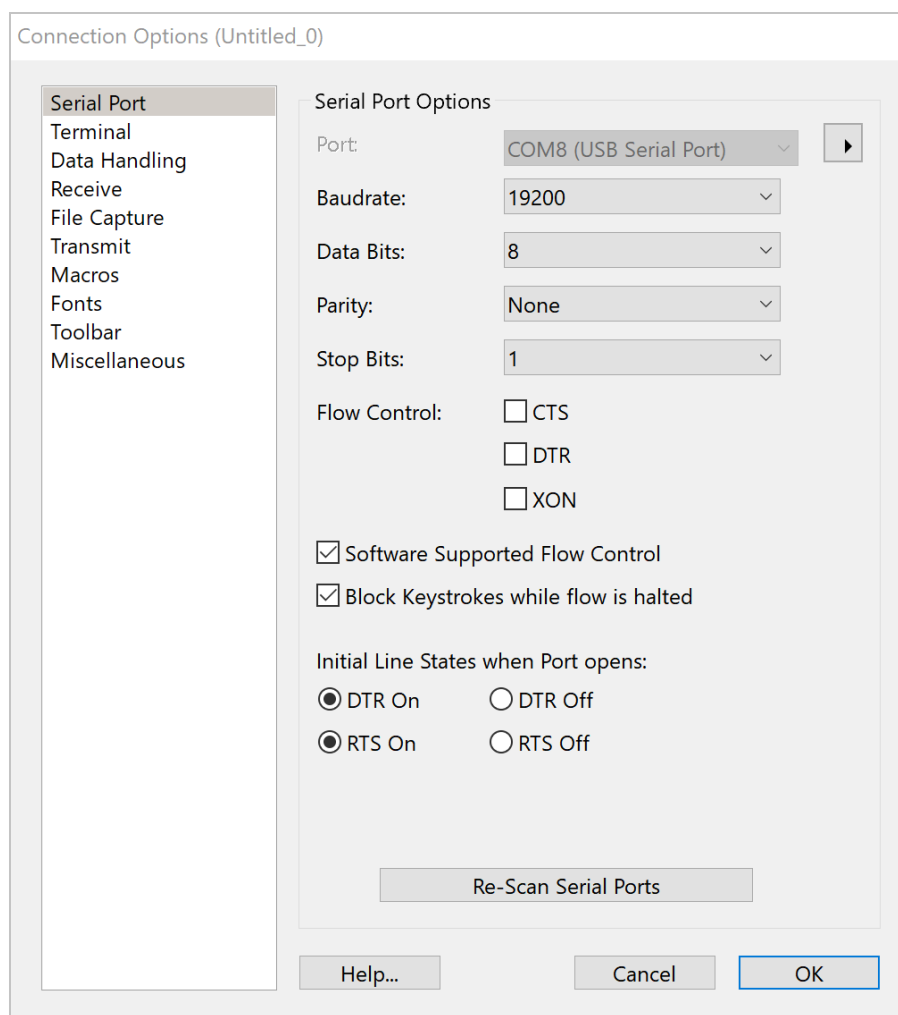


Putty data

When setting up a connection to a serial sensor you will have to select the correct communication parameters for the sensor you are connecting to. This usually means setting the **baud rate** and **parity** (if applicable). The baud rate is the number of bits per second that are used to transfer data over the serial connection. The parity is the method used to check the integrity of the data being transferred. There are two types of parity: **even** and **odd**. Even parity means that the number of 1s in the data stream must be even. Odd parity means that the number of 1s in the data stream must be odd. These

parameters may often be written as e.g. **9600-8-N-1** which would signify 9600 bits per second, 8 data bits, no parity, 1 stop bit. Note these parameters are common between RS232/485/422, all of these employ serial communications the difference being the type of electrical interface used.

Whilst there are other parameters that can be set in these terminal applications you can usually leave them set at the default values. The sensor manufacturers documentation will tell you the default communications settings for the sensor you are using. The default settings can normally be changed by sending specific commands to the sensor from the terminal program. Note that most commands will require you to send a carriage return and line feed `<CR><LF>` or at least a `<CR>` to the terminal program to indicate that the command has been sent. You must of course select the correct serial port for the connection. On Windows this is normally `COM1` or `COM2` etc. On Mac OS X this is normally `/dev/tty.usbserial` and on Linux this is normally `/dev/ttyUSB0` etc.



CoolTerm settings

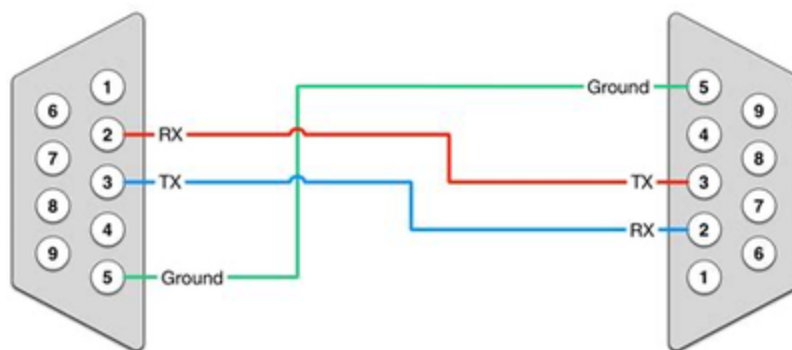
You can read more about serial communications here.

(https://en.wikipedia.org/wiki/Serial_communication)

RS232 wiring

Simple RS232 wiring can be achieved using three wires: RX (receive), TX (transmit) and GND (ground). The RX and TX wires are used to transfer data from the sensor to the computer and vice versa. The GND wire is used to ground the sensor. The RX and TX wires from the sensor are normally connected "crossed over" to the RX and TX wires on the USB to RS232 converter. A 9 pin D-type connector was often used for serial connections and the RS232 specifications calls for the RX to be connected to pin 2, TX to be connected to pin 3 and GND to be connected to pin 5 in such a connector.

A simple 3 wire RS232 example is shown below (diagram also shows 9 pin D-type connector pin assignments):



rs232_wiring.png

Note that RS232 should not be used for long cable runs due to noise degradation of the signal - less than 10 metres is normally fine especially at lower baud rates.

RS485 / RS422 wiring

RS485 can be implemented using either two wires or four wires. The two wire implementation is the most common and is known as "half-duplex" whereas "four wires" provide "full duplex" communications whereby both the transmitter and receiver can send and receive data simultaneously over two pairs pair of transmit and receive wires. Most sensors use a two wire, semi-duplex implementation. RS422 is not very common these days, but this always uses four wires and is very similar to RS485 four wire.

The RS485 interface allows for either a single connection or a "bus" or "network" type wiring whereby multiple sensors can be connected to a single RS485 bus. In such a case

each sensor is known as a "slave" and will have a unique address. The computer or logger controlling the bus is known as a "master" and with such an arrangement it is possible for a master to send and retrieve data to different slaves on the bus using the appropriate address. RS485 buses are not typically used for Met Office setups where each RS485 sensor has its own wiring to the datalogger, in this case every sensor can have the same address e.g. "0" since they are all effectively on a separate bus.

A table showing how to wire RS485 sensors is given below. There are a few different labelling conventions in use for RS485 with regards to how the receive and transmit wires are named, which can be confusing. The table tries to cover most of these conventions. Also remember that different protocols are often used over RS485 e.g. **ModBus** and **UMB**.

RS485 4-wire:

Sensor end	Processor end
Rx+	TA / TxD (A) / TD+
Rx-	TB / TxD (B) / TD-
Tx+	RA / RxD (+) / RD+
Tx-	RB / RxD (-) / RD-

RS485 2-wire:

Sensor end	Processor end
Tx+ (A)	TxD+ / (A)
Tx- (B)	TxD- / (B)

RS485 is a good choice for long cable runs, its differential signalling means that it is not as susceptible to noise degradation as RS232. **Termination resistors** are 120 Ohm resistors that are sometimes installed at either end of the RS485 bus. These resistors are used to prevent reflections occurring on the wires interfering with communications,

however for short cable runs and lower baud rates, these are rarely necessary and not normally used in Met Office sensor wiring.

You can read more about RS485 here. (<https://en.wikipedia.org/wiki/RS-485>)

Quick Quiz

Glossary

A glossary of various terms used on this webpage:

Sensor

A sensor is a device that detects and responds to changes in physical conditions like heat, light, sound, pressure, magnetism or motion and converts them into a measurable signal. Sensors measure physical quantities and convert them into signals which can be read by observers or by instruments.

Interface

An electrical interface is a shared boundary that allows the transfer of electrical signals between two electrical devices or systems. It establishes the standards for how voltage levels, frequencies, and signal durations are transmitted across the connection.

ASCII

ASCII (American Standard Code for Information Interchange) is a character encoding standard that represents English letters, digits, and other basic symbols through a binary code. It assigns a unique numeric value to each character and symbol, allowing computers and other devices to communicate and process text data in a standard format. ASCII is commonly used for text files and communication protocols.

RS232

RS-232 (Recommended Standard 232) is a widely used communication standard for serial transmission of data between devices (e.g. a digital sensor and computer). It defines the electrical and mechanical characteristics of the interface, including signal levels, timing, and connector pin-outs. Some key aspects of RS-232 include:

- Uses +/-12V voltage levels for logic 0 and 1 signals instead of TTL standard 0-5V.
- Asynchronous serial communication where data is transmitted one bit at a time over a single line.
- Common baud rates include 110, 300, 1200, 2400, 9600, 19200, 38400, 57600 and 115200 bits per second.

RS485

RS-485 is a standard defining the electrical characteristics of drivers and receivers for use in serial communications systems. It allows for multipoint connections and can communicate over long distances. RS-485 uses differential signaling to transmit data signals over a pair of wires. This gives it several advantages over single-ended signaling schemes like RS-232, including better noise immunity and ability to interface multiple devices using a bus topology. RS485 allows for half-duplex communication, where data can be transmitted and received on the same pair of wires

- Common baud rates include 110, 300, 1200, 2400, 9600, 19200, 38400, 57600 and 115200 bits per second.

Full duplex, half duplex

When using an RS485 electrical interface, full duplex mode allows simultaneous transmission and reception of data, enabling bidirectional communication between devices (using 4 wires). In contrast, half duplex mode only allows one-way communication at a time, with devices taking turns transmitting and receiving data (uses 2 wires).

ModBus

Modbus is a communication protocol used in industrial automation to establish communication between electronic devices. It enables the exchange of data between a master device (like a computer or controller) and multiple slave devices (such as sensors, actuators, or other controllers) over serial lines or TCP/IP networks. Modbus is widely adopted due to its simplicity, versatility, and interoperability across various manufacturers' devices. Modbus defines a protocol data unit (PDU) that is independent of the lower layer protocols in the protocol stack and allows for the mapping of the Modbus protocol on specific buses or networks.

UMB

The Lufft UMB (Universal Measurement Bus) protocol is a communication protocol used by Lufft environmental sensors to transmit measurement data. It defines a standard format for sensors to send measurements and status information in a binary or ASCII format. The UMB protocol uses a colon ':' as a block separator and addresses each sensor with its unique UMB address. It allows for flexible connection of multiple sensors to a single datalogger or converter unit.

Datalogger

A data logger is an electronic device that records data over time from one or more external sources such as sensors. Data loggers measure and record parameters such as temperature, humidity, voltage, pressure and more. They are commonly used for monitoring environmental conditions, equipment performance and other industrial/scientific applications. Data loggers are small, portable devices that store recorded data internally on memory cards or remotely transfer the data to a computer or cloud for viewing and analysis.

Dataloggers introduction

A datalogger is an electronic device that collects, processes and stores data from analogue and digital sensors. The logger can also be used to control relays and switches. Unlike a microprocessor such as a Raspberry Pi type device, a logger has multiple input and output ports built-in to facilitate the connection of a large number of different types of sensors. Dataloggers are similar to microcontrollers in that have an operating system and can run a user written program but the hardware and operating system is optimised for collecting and storing data real time from multiple inputs.

Due to their limited market, specialist nature, rugged construction, high quality electronics, thorough testing and calibration, dataloggers cost considerably more than generic processing units. At the time of writing a CR1000X datalogger costs approximately £2000.

To use a datalogger such as this, the user must set up the logger using the manufacturers provided configuration utility before connecting a sensor to the correct input ports. A logger program must then be written to tell the logger:

- What sensors are connected and in what way.
- What data to collect from the sensors.
- How to store the data (the logger uses its internal memory to store as much data as it can).
- How to send the data to a remote server (loggers can also store data to a memory card).
- How often you wish to get data from the sensor.
- Apply any processing to the raw sensor data (e.g. calculating a mean or max value over a time period).

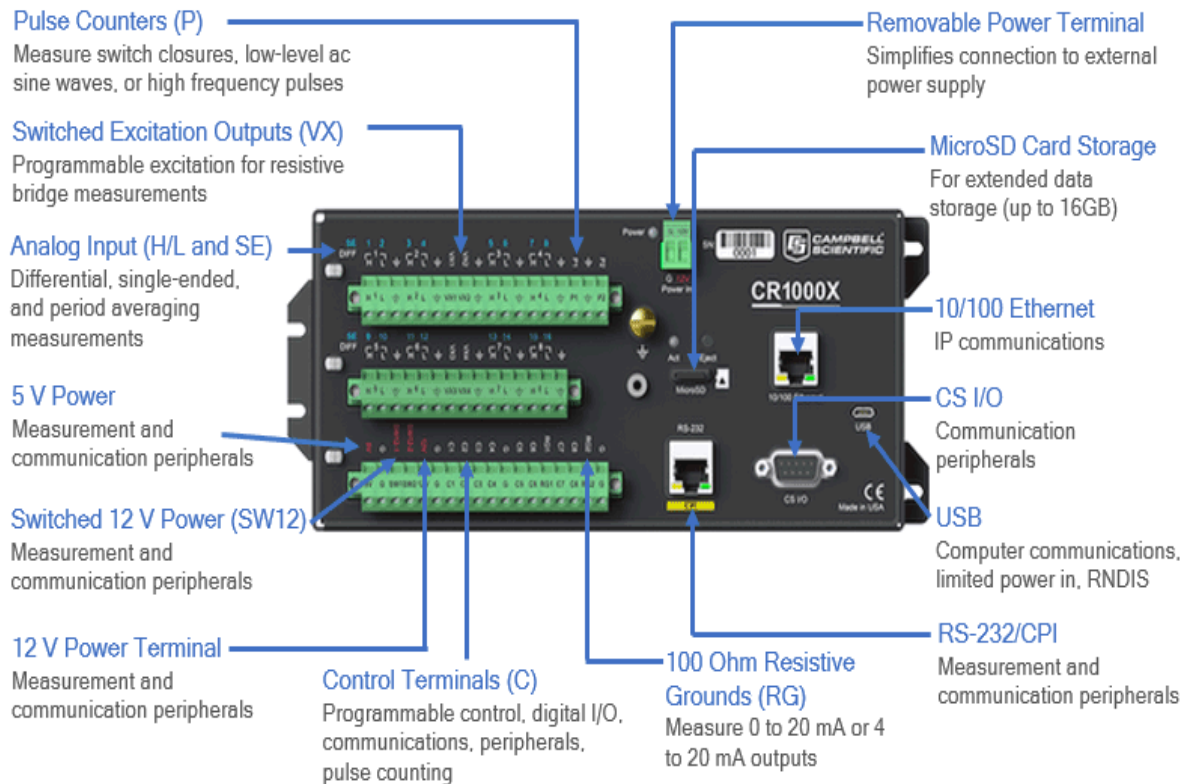
Campbell Scientific CR1000X

The datalogger most commonly in use within the Met Office at this time is the Campbell Scientific CR1000X (<https://www.campbellsci.eu/cr1000x>). This provides multiple interfaces for different types of analogue and digital sensors, a powerful processor with

plenty of memory (for a datalogger!), very fast scan rate (for rapid sampling for sensor data) and very low power requirements (allowing solar panel/battery operation).

In addition, this logger has a USB port for connection to a computer and supports modern network protocols for sending out data such as MQTT. Its digital ports can be configured for RS232 and RS485 and its sufficient memory to allow handling of long data strings. You can access the full, online CR1000X manual [here](https://help.campbellsci.com/CR1000X/Content/Home.htm).

(<https://help.campbellsci.com/CR1000X/Content/Home.htm>)



CR1000X layout

This logger runs a proprietary real time operating system, it's tolerant of sudden power removal and each logger is tested to operate over a wide (industrial) temperature range. The device uses a 24bit analogue to digital converter (ADC) and has enough memory to handle long data strings, large programs and complex processing.

To use the datalogger a logger program must be written in "BASIC" like language called "CRBasic". This program is loaded onto the logger via USB or over a network link using tools provided by Campbell Scientific. You read more about CRBasic and access all the programming instructions from the online manual [here](https://help.campbellsci.com/crbasic/cr1000x/).

(<https://help.campbellsci.com/crbasic/cr1000x/>)

It is not possible to connect a screen or keyboard etc. to the logger, monitoring and other tasks are carried out using Windows based utility applications. Many of the input "ports" of the logger can be configured using "instructions" within a CRBasic logger program. For example the "Control - C" ports can be configured to operate in RS232 or RS485 mode.

Campbell Scientific also make available a large of number of add-on accessories for the CR1000X such as modems, radio links and modules to increase the number of analogue and digital inputs. In addition, it is also possible to purchase battery backup and solar charging units designed to work with this logger. See the glossary below for links to additional information about these subjects.

There are other dataloggers available on the market, but they are not used within the Met Office at present. An alternative type of device has just been released by Vaisala called the "DMU801". This is basically a small, power efficient Linux device with an ARM processor and running an embedded Linux operating system. Plugin cards provide the required number of analogue and digital inputs. You can read more about the DMU801 here. (<https://www.vaisala.com/en/products/data-management-unit-dmu801>)



Vaisala DMU801

Glossary

Datalogger

A datalogger, also known as a data logger or data recorder, is a device or system designed to automatically collect, store, and sometimes transmit data over a period of time. It typically consists of sensors to measure various parameters such as

temperature, humidity, pressure, light intensity, voltage, current, or other environmental or process variables.

CR Basic

CRBasic is a programming language developed by Campbell Scientific for programming their data loggers. Some key things to note about CRBasic: It is a high-level language designed to be easy to use, yet also provide powerful programming capabilities for measurement and control applications.

CRBasic programs are written using a text editor/programmer called the CRBasic Editor, which is designed for creating, editing, compiling and transferring CRBasic programs to Campbell Scientific data loggers.

CRBasic programs have a typical structure that includes a header with metadata, tables to declare variables and constants, and measurement, calculation and control instructions.

Variables can be declared as different data types like Boolean, byte, word and floating point. Arrays and strings can also be used. Functions can be created to modularize and reuse code. Functions allow breaking programs into logical components to simplify programming and troubleshooting.

CRBasic supports conditional structures like If/Then/Else and Select Case as well as loops like Do/Loop and For/Next for control applications. Programs can be compiled in either sequential or pipeline mode to optimize performance for different applications.

Variables and constants can be public to make them accessible to external programs and devices for easy data retrieval and control.

Campbell Scientific Ltd.

Campbell Scientific is a company that specializes in data acquisition and measurement systems for various environmental and research applications. They produce a wide range of data loggers, sensors, and telemetry systems used in fields such as meteorology, hydrology, agriculture, ecology, and geology, among others.

Founded in 1974 by Dr. Gaylon Campbell, the company has become a prominent supplier of instrumentation for environmental monitoring and research. Their products are known for their reliability, accuracy, and ruggedness, making them suitable for use in harsh environmental conditions and remote locations.

Vaisala Oyj.

Vaisala Oyj, commonly referred to as Vaisala, is a Finnish company that specializes in environmental and industrial measurement. Founded in 1936 by Vilho Väisälä, a Finnish scientist and inventor, Vaisala has grown to become a global leader in providing observation and measurement solutions for weather, environmental monitoring, and industrial applications.

The company's product portfolio includes a wide range of instruments and systems for measuring various parameters such as temperature, humidity, pressure, carbon dioxide, liquid concentration, wind speed and direction, and weather phenomena. These instruments are used in applications such as weather forecasting, climate research, aviation, renewable energy, pharmaceuticals, and various industrial processes.

Vaisala is particularly renowned for its weather observation systems, which are widely used by meteorological agencies, airports, and other organizations involved in weather monitoring and forecasting. Their instruments and systems are known for their accuracy, reliability, and durability, making them suitable for use in demanding environments and critical applications.

Datalogger support applications

Campbell Scientific (CSL) produce a number of Windows software applications to support the configuration, programming and management of their dataloggers. Some of these programs have overlapping functionality, most are free to download but some require a license. When starting out it can be difficult to know what application to use, so I hope the following will help. Some tasks can be equally well carried out by different applications, so just use the one you feel most comfortable with or have access to.

Getting help

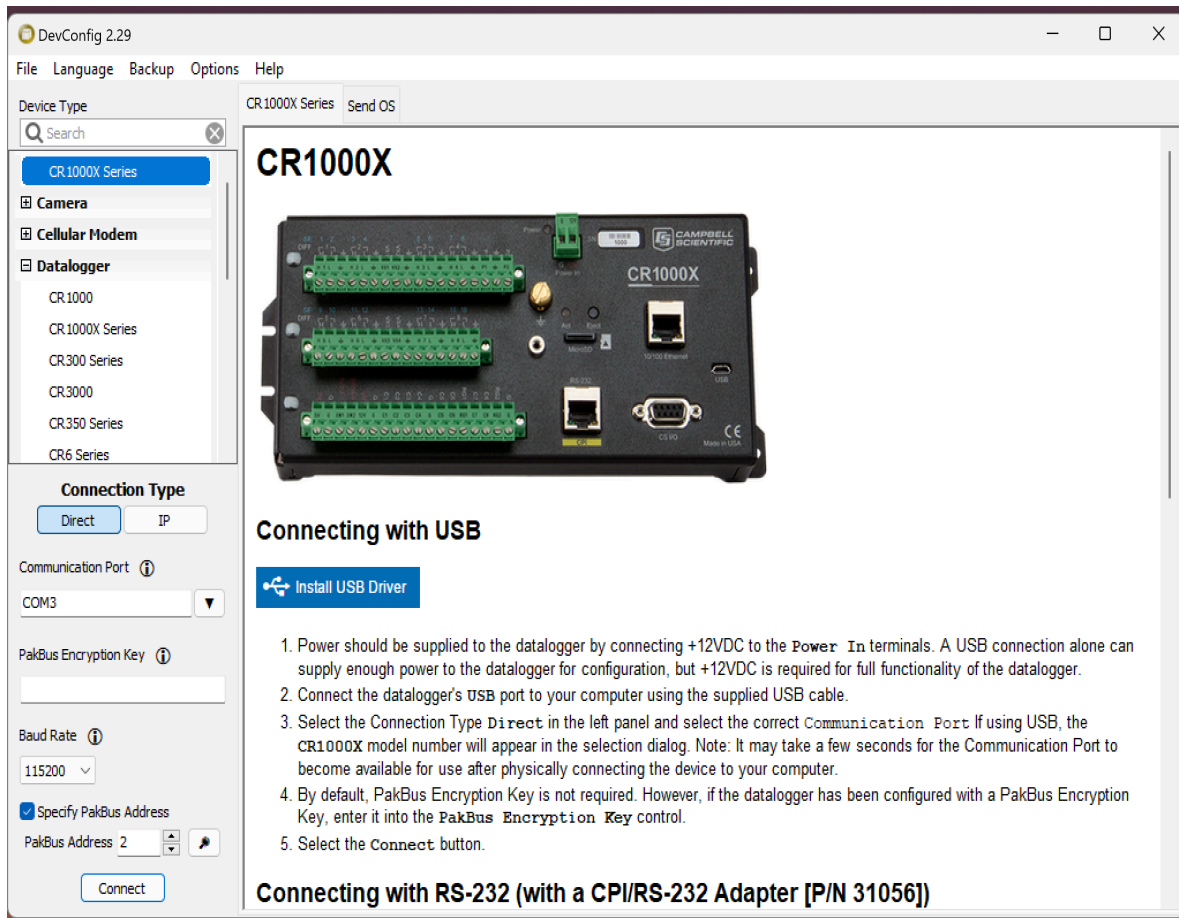
CSL provide excellent online documentation and manuals for these applications, but perhaps the best resource is their large collection of instructional videos available on the CSL website and YouTube. (<https://www.youtube.com/@CampbellScientific>)

These videos are also available on the CSL product pages for each application. They cover basic and more advanced datalogger operations. Don't forget that there are many Engineers with the Met Office (Observations R&D and Field Services in particular) that are very familiar with working with dataloggers so please ask if you're not sure of something - you should at least get pointed in the right direction.

Device configuration utility (DevConfig)

Configuration of Campbell Scientific data loggers and sensors, sending new operating systems, editing settings, sending new programs. Can monitor live data collection. Note that the easiest way to connect to a logger is via the USB port. This application is mostly used (as its name suggests) to initially configure make changes to the configuration of a datalogger.

This application can also be used to configure many of CSL's sensors and datalogger add-on modules. This is probably the most common/popular tool used by Engineers. You can load programs and update the operating system of the logger and check its running state. Read more about it here. (<https://www.campbellsci.com/devconfig>) This software is free to use.

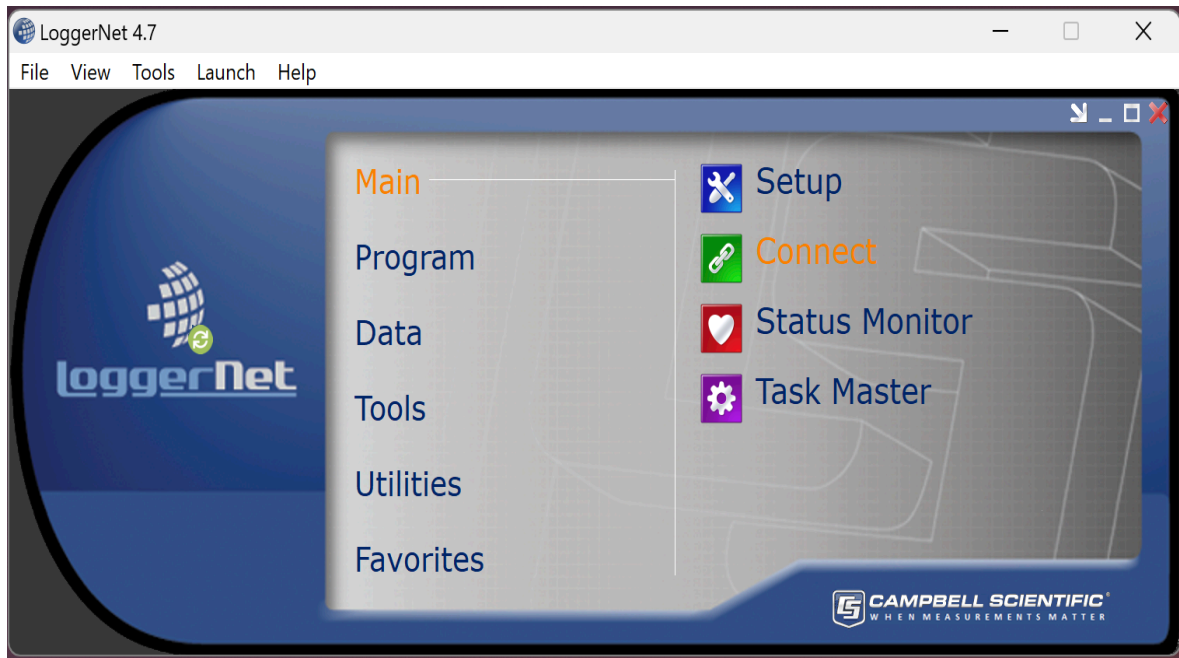


Device configuration utility

Loggernet

CSL's premium software application (requires a license, Met Office does have a large number of licensed copies installed). The only application here that allows for scheduled data collection. Can also perform data logger file control, configuring a network of loggers, data retrieval and monitoring. Loggernet is a bit different in that it has most of the other applications here 'built-in' (DevConfig, ShortCut, CRBasic editor). This makes it very useful but these applications are also now available to download for free as standalone applications. Read more about it here.

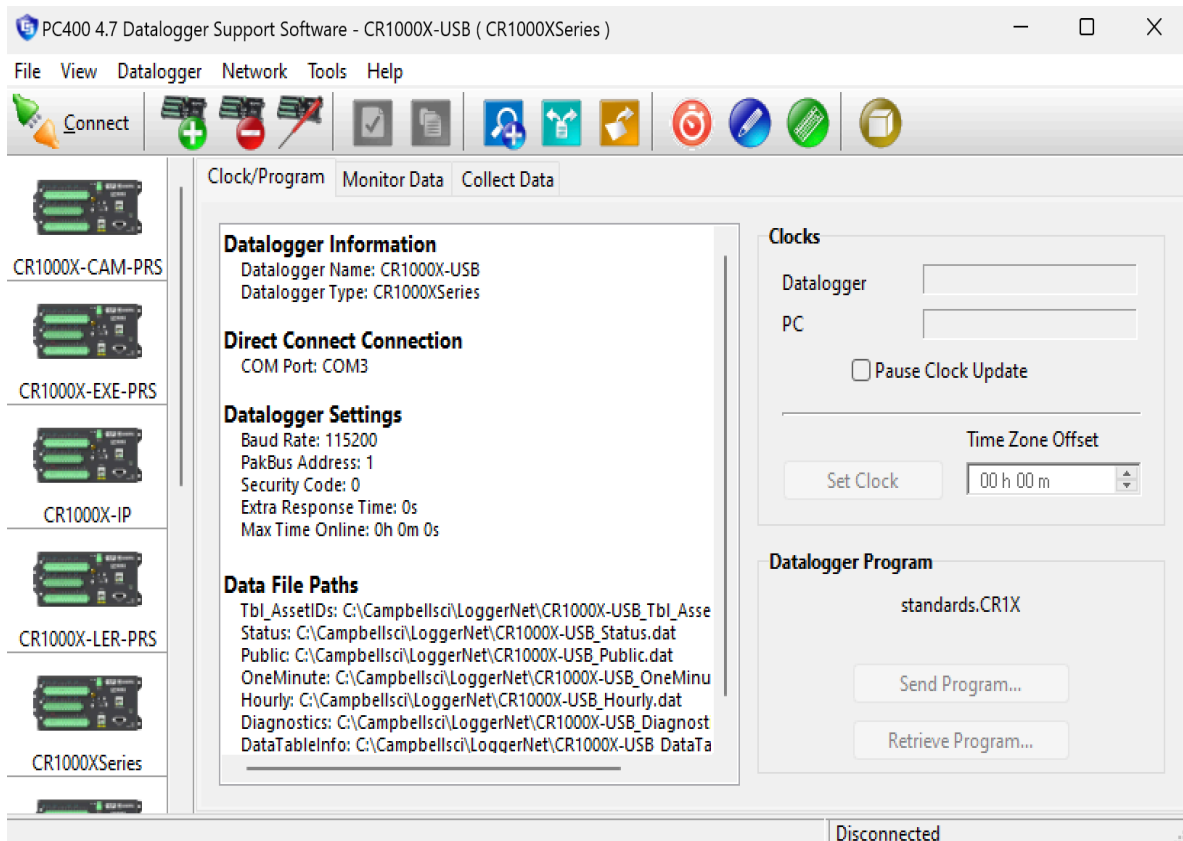
(<https://www.campbellsci.com/loggernet>) This software requires a license to use.



Loggernet

PC400

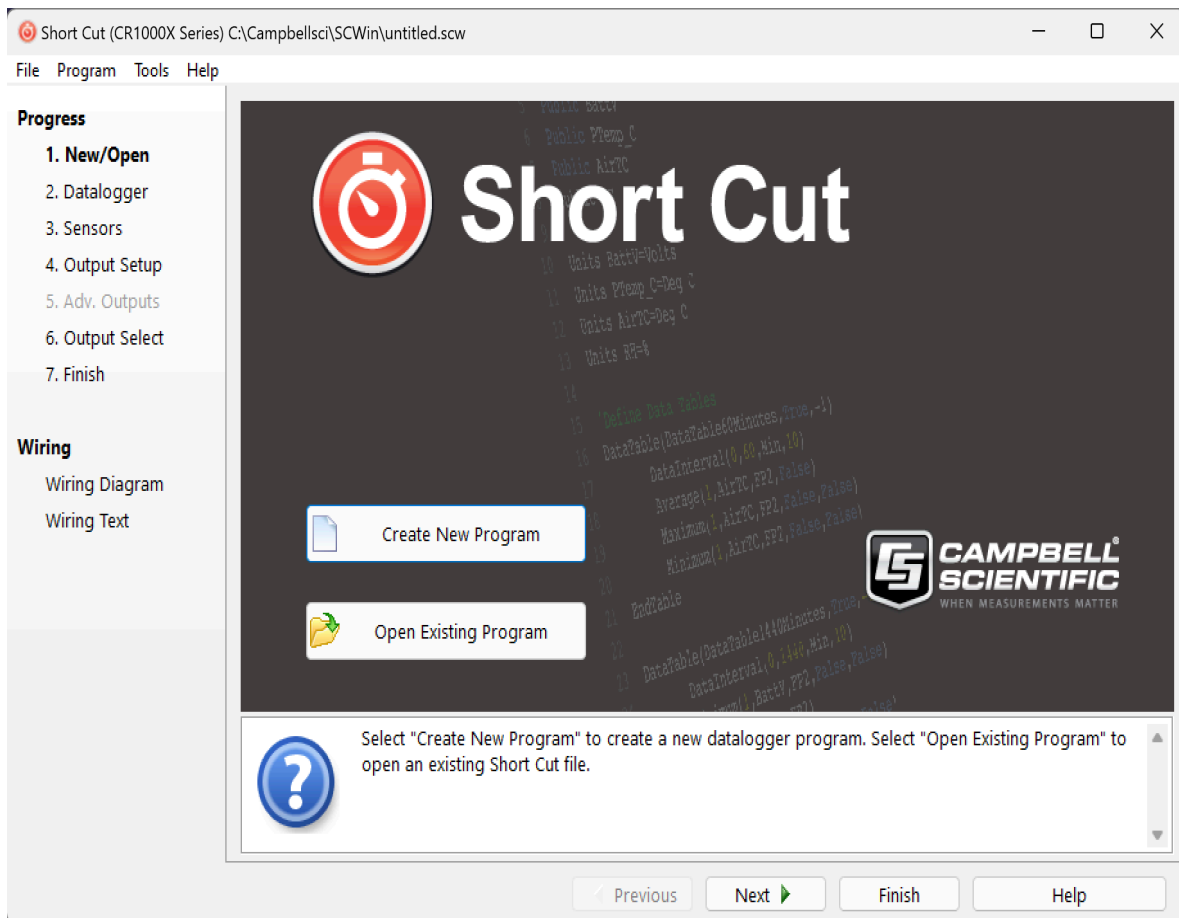
Can do most of what Loggernet can do, very similar but cannot do scheduled data collection. Read more about it here. (<https://www.campbellsci.com/pc400>) This software is free to use.



PC400

ShortCut

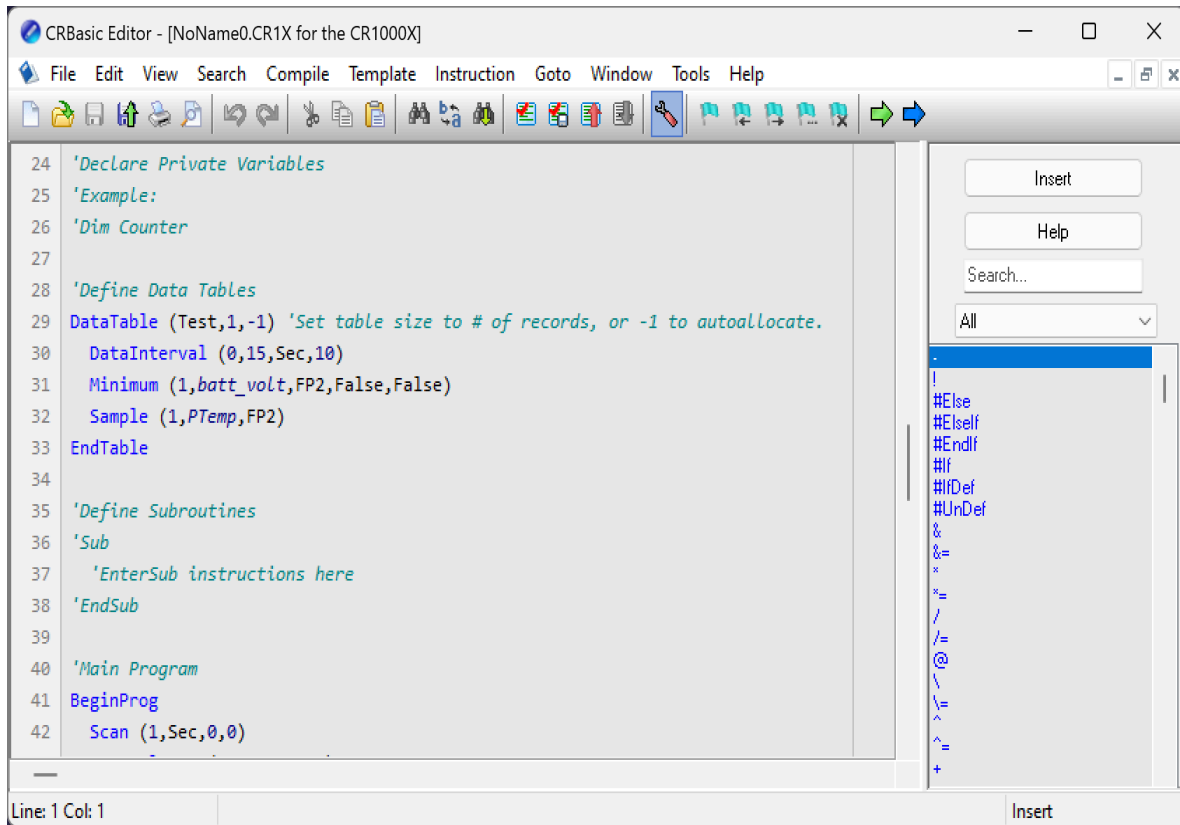
Provides a step-by-step ‘wizard’ that can write a basic data logger program for you provided the sensors you’re using are in the applications sensor list. The end program can be edited later in CRBasic. Handy for creating a program starting point and a good way to start understanding how a CRBasic logger program is written and structured. Read more about it here. (<https://www.campbellsci.com/shortcut>) This software is free to use.



ShortCut

CRBasic editor

A full-featured data logger program editor that can compile and send your program to a connected logger. The program also checks for coding errors before the compilation stage. Read more about it here. (<https://www.campbellsci.com/pc400>) This software is free to use.



CR Basic

Datalogger programming introduction

Overview articles give background information and provide context to a particular subject. Their goal is to explain a concept, not to teach or give instructions.

Campbell scientific documentation and videos

There is a wealth of online documentation and videos available from Campbell Scientific to help your program their dataloggers.

A YouTube channel is available here

(<https://www.youtube.com/@CampbellScientific/videos>).

A web version of the CRBasic documentation is available here

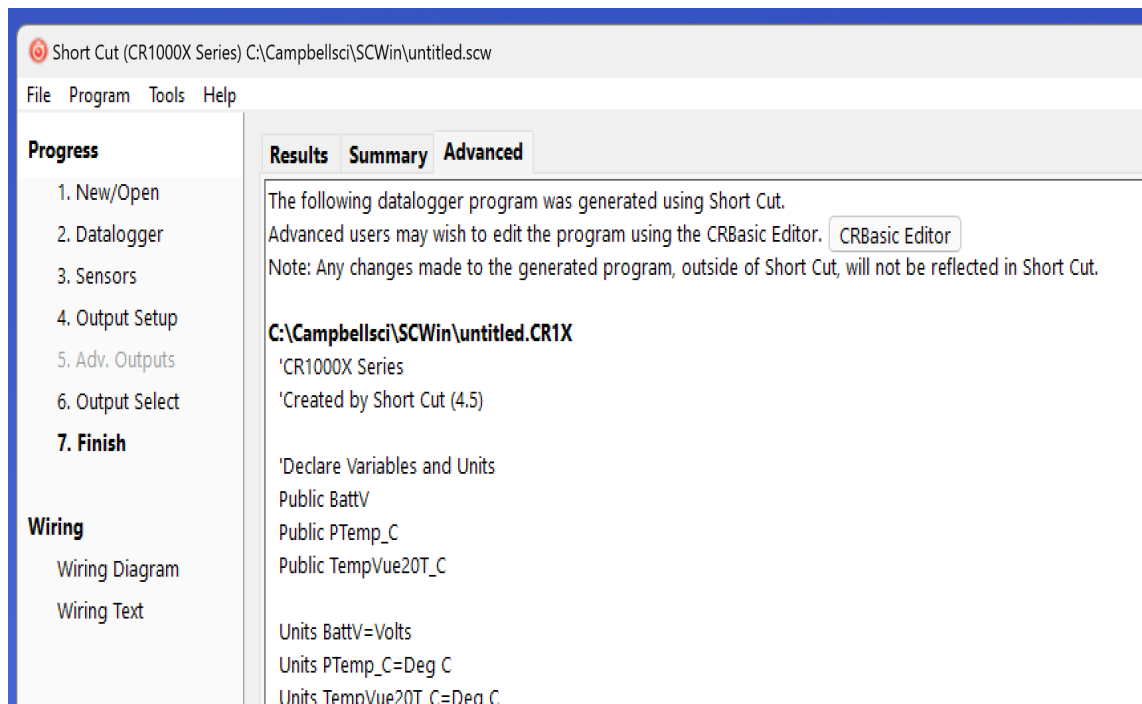
(<https://help.campbellsci.com/crbasic/cr1000x/>). This includes example programs for each of the built-in CRBasic functions and should be your go-to reference when writing a program.

CR1000X getting started tutorial here (<https://www.campbellsci.eu/videos/cr1000x-getting-started-tutorial>)

How to write a program for the CR1000X data logger

There are two ways to write a logger program:

1. Use the 'Short Cut' application wizard to select the sensors you want to use, the measurement values and time intervals to automatically generate a logger program and wiring diagram. This works fine for simple programs where the sensor you want to use is one of those that Short Cut supports in its list of sensors. Using this method can also be useful as a starting point for a program that you will eventually customise by hand as it can produce a file that CRBasic Editor can use (click on the advanced tab when you've finished and saved your program, and you can click on the button to open it up in the CRBasic editor application):



ShortCut advanced mode

2. The other way is to use the CRBasic editor that allows you to write a program from scratch. This will be necessary for a sensor that is not supported by Short Cut and is the tool should ultimately be aiming to use. CRBasic editor had built-in help files for program functions and can check and compile your code before sending it to a connected logger.

Using the 'ShortCut' application to generate a program for you

Open up the Short Cut application, click 'Create new program' and follow the steps through the 'wizard', selecting you logger type, the sensors and parameters you want to measure. The application will also show you how to wire the sensor. You must follow this wiring otherwise the sensor will not work with the code generated by Short Cut. A Short Cut tutorial can be found here (<https://www.campbellsci.eu/videos/programming-with-short-cut-quickstart-part-2>).

How a CR1000X organises and stores data - 'Tables'

This is an important concept to grasp when starting out on your datalogger programming journey. Within CRBasic programs, data storage is typically achieved through the use of

table structures. These tables are essentially organized arrays where data collected from sensors or calculated values are stored.

A data table is essentially a file that resides in data logger memory. The file consists of five or more rows. Each row consists of columns, or fields. The first four rows constitute the file header. Subsequent rows contain data records. Data tables may store individual measurements, individual calculated values, or summary data such as averages, maximums, or minimums.

Typically, files are written to based on time or event. The number of data tables is limited to 250, which includes the Public, Status, DataTableInfo, and ConstTable. You can retrieve data based on a schedule or by manually choosing to collect data using data logger support software.

Defining variables and data tables appropriately in the CRBasic program helps simplify accessing and processing the stored data values later.

TOA5, MyStation, CR6, 1142, CR6.Std.01, CPU:MyTemperature.CR6, 1958, OneMin				
TIMESTAMP	RECORD	BattV_Avg	PTemp_C_Avg	Temp_C_Avg
TS	RN	Volts	Deg C	Deg C
		Avg	Avg	Avg
2019-03-08 14:24:00	0	13.68	21.84	20.71
2019-03-08 14:25:00	1	13.65	21.84	20.63
2019-03-08 14:26:00	2	13.66	21.84	20.63
2019-03-08 14:27:00	3	13.58	21.85	20.62
2019-03-08 14:28:00	4	13.64	21.85	20.52
2019-03-08 14:29:00	5	13.65	21.85	20.64

Datatable example

Constants, variables and alias

A *constant* is value declared at the beginning of a program. It is a value that does not change during the program execution. For example:

```
CONST PI = 3.141592654 'Define constant.
```

A *variable* is a value declared at the beginning of a program and can be changed during the program execution. Variables declared using the **Public** instruction are available for viewing on the datalogger's display and within software such as Loggernet. Variables declared using the **Dim** instruction are not available for viewing. "Scratch" variables and

variables that are not important for the user to monitor are often defined with the **Dim** instruction. For example:

```
'Public Variables
Public count As Long      'create a Long variable
Public Number = 38.3      'create a Float variable and set to
38.3
Public PubString As String 'create a string variable
```

```
'Dim (private) variables
Dim DimArray3D(2, 3, 4) 'create a 3-dimensional array with 24
elements
Dim DimArray1D(9) 'create a 1-dimensional array with 9 elements
Dim DimString As String * 50 'create a string variable - 50
characters
```

An *alias* declaration is used to assign a second name to a Dim or Public variable. This can be useful if you want to use a variable name that is more meaningful to you. Here's an example of the use of alias's:

The example shows how to use the Alias declaration. TCTemp is the original variable array. CoolantT, ManifoldT, and ExhaustT are the aliases that are assigned to these variables:

```
Public TCTemp( 3 ), Ptemp
Alias TCTemp( 1 ) = CoolantT
Alias TCTemp( 2 ) = ManifoldT
Alias TCTemp( 3 ) = ExhaustT
```

Scans and Slow Scans

A **scan** is a series of measurements made at a particular time interval. A **slowscan** is a series of measurements made at a slower time interval. For example, you might wish to read a number of analogue sensors at a 1-second interval that would be your *main* scan and then read a number of digital sensors at a 10-second interval that would be your *slow* scan.

```

BeginProg
Scan( 1, sec, 1, 0 )      'Continuous Scan/Next Scan Loop
    '(measurement instructions, calls to data tables, etc.)
NextScan
EndProg

```

Functions and subroutines

A function is a block of code that can be used to perform a specific task. Functions are useful when you want to perform a specific task multiple times in your program. A subroutine is also a block of code that can be used to perform a specific task. A Subroutine (Sub) is a separate procedure that is called by the main program using a Call statement. A Subroutine can take arguments, perform a series of statements, and change the value of its arguments. One difference between a Sub and a Function is a Function returns a value whereas a subroutine does not. Functions also include the ability to pass in optional parameters.

'Subroutine (**Sub**) to convert temperature in degrees C to degrees F

```

Sub ConvertCtoF (Tmp, TmpF)
    TmpF = Tmp*1.8 +32
EndSub

```

An example of a function - This example creates a function that issues a 2-second toggle to a control port whenever a flag is high and returns the string "BlinkBlink":

```

Public Flag(2), BlinkStatus As String
Function Blink As String
    PortSet (C1,1 )
    Delay (0,2,Sec)
    PortSet (C1,0 )
    Return ("BlinkBlink")
EndFunction

BeginProg
    Scan (1,Sec,3,0)
        If Flag(1) Then

```



```
BlinkStatus=Blink()  
Else  
    BlinkStatus="off"  
EndIf  
NextScan  
EndProg
```

Program comments

Program comments are used to describe the purpose of a program. Comments can be used to explain the purpose of a program, to explain the purpose of a section of code, or to explain the purpose of a variable. In the CRBasic programming language comments preceded by a single quote ('). Comments can be placed anywhere in a program and are ignored by the compiler.

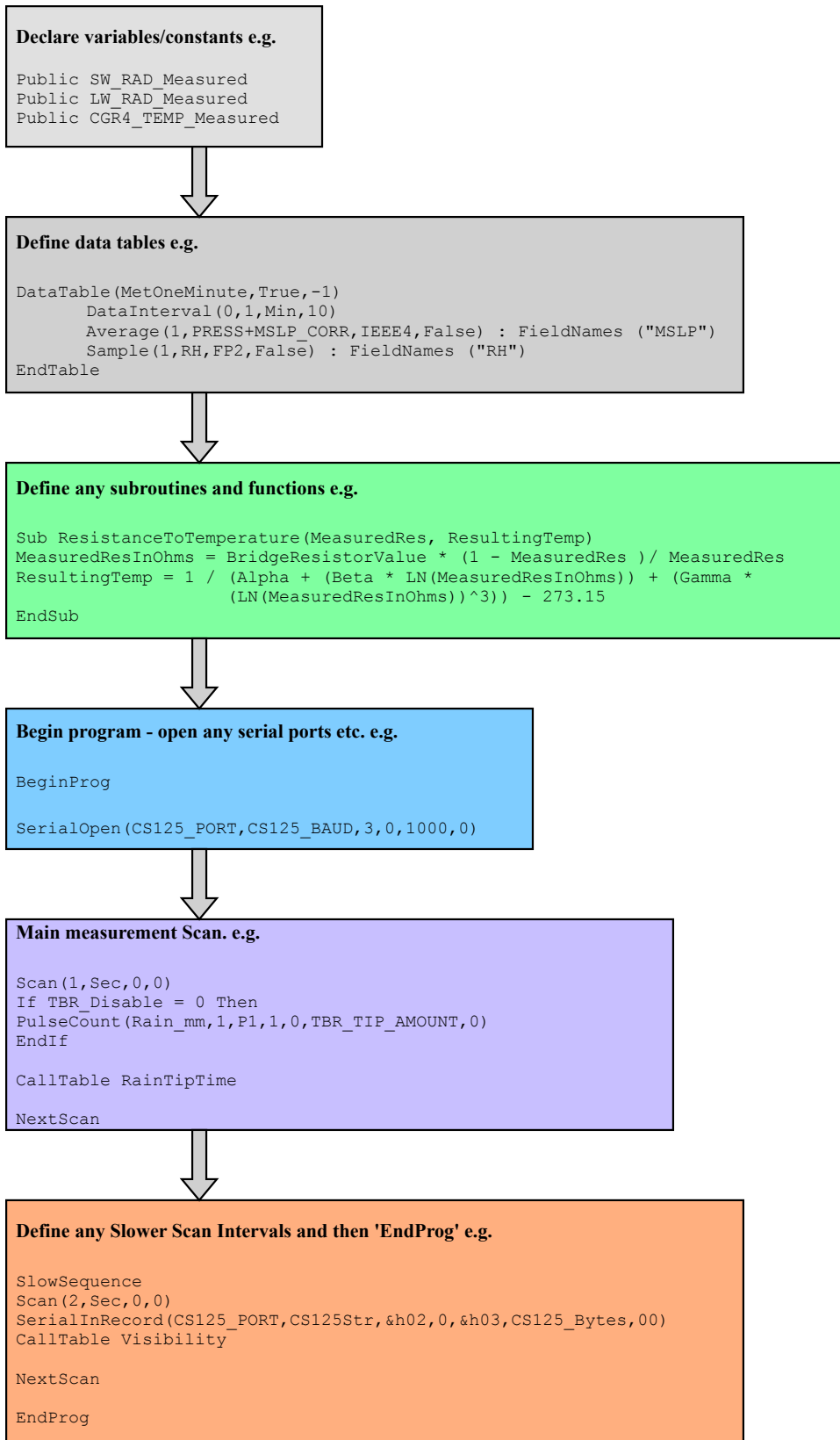
The basic structure of a CRBasic logger program

All CRBasic logger programs follow the same structure no matter how complicated or simple they are:

1. Define variables, constants, aliases (if used) and units (if used).
2. Define the tables that will store your data using the 'DataTable' function (this will include how often data is stored and whether it is maximum/minimum/sample etc. over the storage interval period).
3. Write any functions and subroutines that you want to use within your program.
4. The 'BeginProg' statement that marks the start of the program.
5. Configuration and setup of your program instructions e.g. opening up serial ports.
6. A 'Scan' function that is your primary scan loop e.g. scanning certain sensors every second.
7. 'CallTable' statements to update the data tables you've defined.
8. Optionally 'SlowSequence' functions that are used to scan sensors at a slower rate than the main scan loop.

9. The 'EndProg' statement to signify the end of the program (though the program will continue with its Scan sequences, so this really refers to the end of the program scans setup).

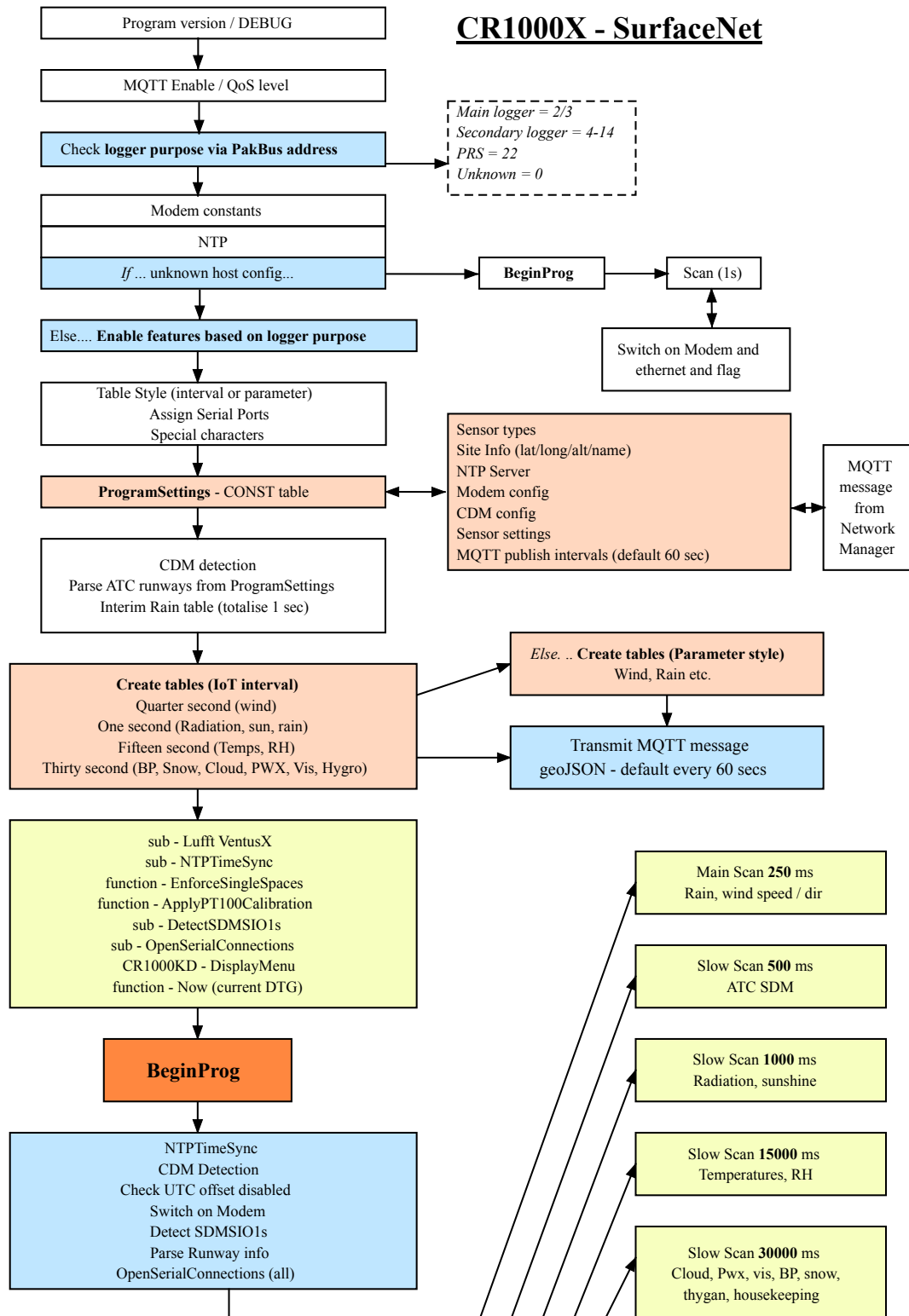
This structure is shown in the diagram below:



CRBasic logger program structure

SurfaceNet CRBasic logger program structure

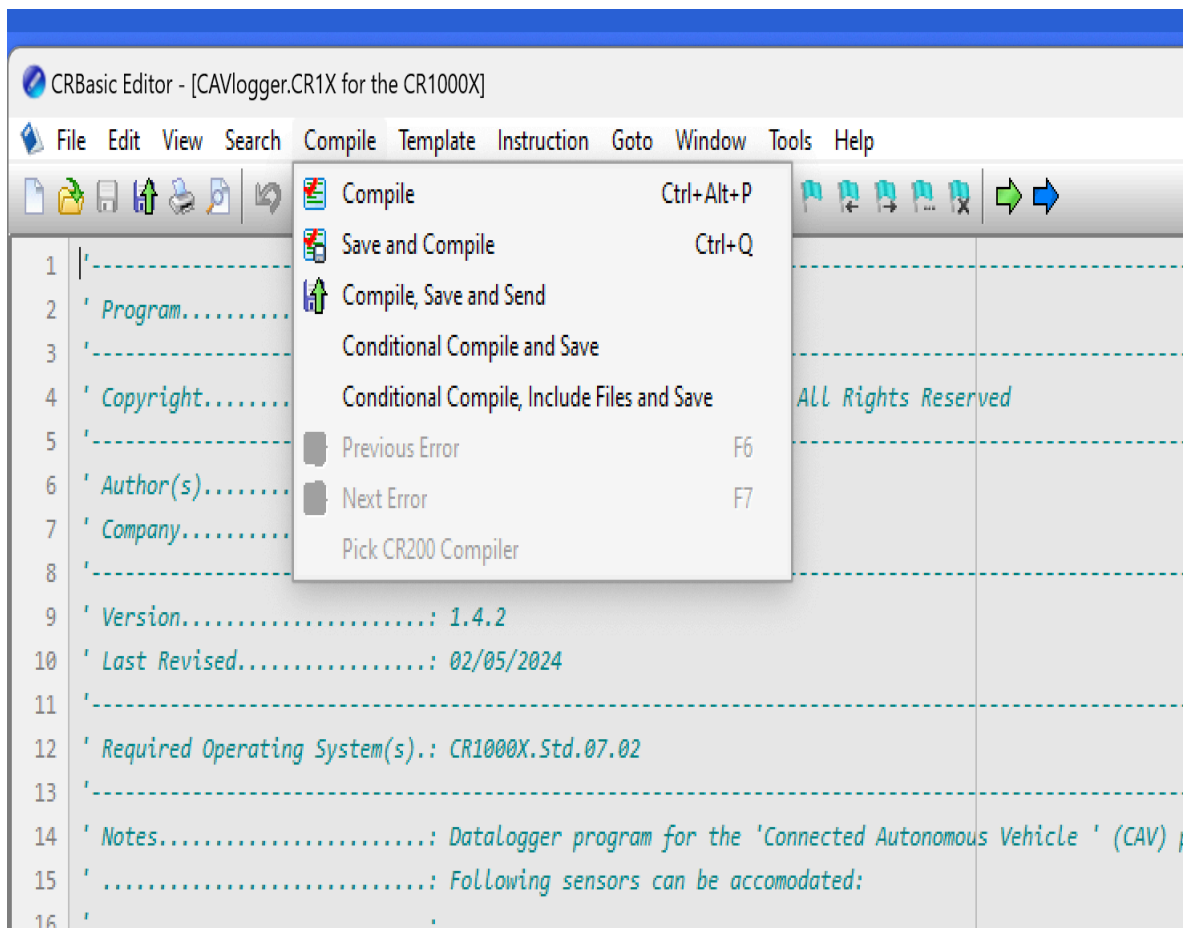
For interest, here is the structure of the SurfaceNet logger program. This a very large and complex data logger program, but you can see that still follows the basic structure of all CRBasic logger programs:



SurfaceNet CRBasic program structure

Compiling your program and sending it to the logger

Once you've written your program, you need to compile it. Compiling a program is the process of converting the program text file into instructions that the datalogger can understand. If you use the 'Short Cut' application to generate your program then it will compile your program before it sends it to a datalogger. The CRBasic Editor application can check your code and compile it before sending it to one of the dataloggers you have set up in LoggerNet or PC400. CRBasic Editor gives you control over whether you want save, compile and send or just compile your program. See the screenshot below:



CRBasic compile menu

Debugging your program

When you compile your program in CRBasic it will check your program for errors and give you a list of errors at the bottom of the CRBasic application window. These could be a typo in your program, an incorrect number of parameters in a function or subroutine. The error description should provide enough details and a program line number to enable to

correct the error. In addition to errors that prevent your program from compiling, 'warnings' will also be shown in the same area. A warning could be that you declared a variable that is not used in your program. Warnings will not a program from being compiled, they are provided for information. See an example of the CRBasic compile information below (DataTable has been spelled incorrectly resulting in several error messages):

```
121 Alias N(6)=Fault_Pressure 'Pressure sensor warning
122 Alias N(7)=Fault_Compass 'Compass fault
123 Alias N(8)=Fault_Undefined 'Undefined fault
124 Alias N(9)=Fault_Checksum 'Checksum error
125
126 Alias DISDROMStrArr(4)=DISDROM_Clock_date
127 Alias DISDROMStrArr(5)=DISDROM_Clock_time
128 Alias DISDROMStrArr(8)=METAR_5m
```

[Version]C:\Campbellsci\Lib\Compilers\CR1000XComp.exe VERSION:CR1000X.Std.07.02 DATE:03/11/2024
CAVlogger.CR1X -- Compile Failed!

line 199: Could not find RadiationOneMinute.
line 199: Invalid, or out of place expression: DataTbl (RadiationOneMinute,True,-1)
.
line 199: Expr illegally occurs before BeginProg.
Error(s) detected in the program. Double-click an error above to navigate to it.

Line: 106 Col: 35 \\Mac\Home\Library\CloudStorage\OneDrive-Personal\Met\CAV\cav-logger\CAVlo

CRBasic error messages

Quick Quiz

Glossary

Programming terminology:

Variables

In computer programming, variables are essentially containers for storing data values. They serve as placeholders for values that may change during the execution of a program. Variables have a name, a data type, and a value. Here's a breakdown of these components:

- **Name:** Also known as an identifier, it is a meaningful name given to the variable by the programmer. It is used to uniquely identify the variable within the program.
- **Data Type:** Defines the type of data that the variable can hold, such as integers, floating-point numbers, characters, strings, etc. The data type determines the operations that can be performed on the variable and the amount of memory it occupies.
- **Value:** The actual data stored in the variable. This value can be changed or modified during the execution of the program.

Constants

In computer programming, a constant is a value that remains unchanged throughout the execution of a program. Constants are typically used to represent fixed values that are known at compile time and are not meant to be modified during runtime. They are helpful for making code more readable, maintainable, and less error-prone by providing meaningful names for values that are used repeatedly throughout the codebase. Constants can be of various data types, including integers, floating-point numbers, characters, strings, and Boolean values. In many programming languages, constants are declared using specific syntax or keywords, such as "const" or "final".

Functions

In computer programming, a function is a block of reusable code that performs a specific task. Functions allow programmers to break down a program into smaller, manageable pieces, making the code more organized, readable, and easier to maintain. Functions typically accept inputs, process them, and then return a result.

They help in modularizing code, promoting code re-usability, and improving overall program structure.

Tables

A data table is a key component in CRBasic programming for Campbell Scientific dataloggers. It is used to store the measured data from sensors and other inputs. The data table must be called by the CRBasic program for the data processing and storage to occur.

Scans

In the context of CRBasic programming for Campbell Scientific data loggers, a scan refers to the fundamental unit of program execution. Specifically, the Scan instruction is used to establish the program scan rate, which determines how often the program will execute.

Compiler

A compiler is a computer program that translates source code written in a high-level programming language into machine code or lower-level code that can be directly executed by a computer's CPU (Central Processing Unit). The process involves several stages, including scanning analysis, syntax analysis, semantic analysis, optimization, and code generation.

Debugging

Debugging, in the context of computer programming, refers to the process of identifying, analyzing, and fixing errors, defects, or bugs within a software application or system. These bugs can cause the program to behave unexpectedly, produce incorrect results, or even crash.

Comments

In computer programming, comments are non-executable portions of code used to annotate or explain the functionality, purpose, or rationale behind specific sections of code. Comments are primarily intended for human readers rather than the computer itself and are typically ignored by the compiler or interpreter during the execution of the program.