ZooCADA™ - Mod

Program Customisation Module

Version R01

Reference Manual

Copyright Notice

ZooCADA-Mod Version R01 Reference Manual

Revision History

Original Version 1.0 13-02-2025

Adena Scientific Ltd

P O Box 756
Waikato Mail Centre

5 Pukeko Place
Te Kowhai, New Zealand

Sales and Technical Support

E-mail: service@adena.co.nz
Telephone: + 64 7 829-7063

Software License Agreement

Please read this before using your copy of the software.

This document is a legal agreement between you and Adena Scientific Limited. Any use of the software that accompanies this license indicates your acceptance of the conditions contained in this license.

> If you do not agree to these conditions you must destroy all digital copies of the software and documentation in your possession. If a physical software package was supplied containing the software, printed documentation and/or any other components, you must return it intact to the place of purchase.

Interpretation

The following meanings apply to terms used in this Software License Agreement;

- Device: The computer equipment used operate this software, irrespective of its form, and includes desktop PCs, laptops, tablets, dataloggers, and any other device that is capable of running the software.
- Installed: The software is considered to be installed when it has been loaded into memory or other storage, either permanent or temporary, from where the software can be operated.
- Software: The computer program(s) supplied to you accompanied by this Software License Agreement.
- Modification: Means the addition or deletion of program instructions, reallocation of memory variables, and/or reallocation of datalogger input locations. Changing of the calibration coefficients of measurement instructions in datalogger programs is not considered to be a program modification.

Proprietary Rights

The software and all accompanying documentation are proprietary products of Adena Scientific Limited and are protected under the copyright laws of New Zealand and by international treaties. Ownership of the software and all copies, modified or not, and any merged portions thereof remains at all times with Adena Scientific Limited.

Grant of License

The software and accompanying documentation are licensed to you by Adena Scientific Limited. This means that you have purchased the right to use the software and accompanying documentation only in accordance with the conditions detailed in this Software License Agreement. This license is effective from the date you purchased it until such time as it is terminated.

Single Device License

This license permits you to install the software on one device only. You must purchase a separate license for each device on which the software is to be installed. The software may be transferred from one device to another provided that the software is permanently removed from one device before it is installed onto another.

Annual License Fee

Your license to use the software is conditional upon your payment of any applicable ongoing annual license fee. This fee entitles you to receive, during the period for which the license fee has been paid, any applicable software updates to the software that are made available from time to time by Adena Scientific Limited, and unlimited product support by telephone or email. Site specific software customisation and installation are not included.

The annual license fee is billed in advance, monthly for NZ customers or annually for International customers, commencing one month after the date of your original purchase of the software.

Annual Maintenance Plan Fee

Your license to use the software is conditional upon your payment of any applicable ongoing annual maintenance plan fee. The annual maintenance plan fee includes the annual license fee and the entitlements detailed under "Annual License Fee" above and provides additional benefits of annual on site calibration and functional tests.

Software License Agreement

The annual on site, inspection and testing regime is designed to verify that the control system is operating according to our documented specifications and to identify any maintenance work that may be necessary. The Installation of software updates issued by Adena Scientific Limited is included in the plan.

Site specific software customisation, equipment repair or replacement, and the upgrade or maintenance of software that may be part of an overall system but which is not manufactured by Adena Scientific Limited is not included.

The annual maintenance plan fee is billed monthly in advance, commencing one month after the date of your original purchase of the software.

Backup and Working Copies

You may make one copy of the original software and its electronic format documentation expressly for backup or archival purposes provided that the contents of the original media are copied onto the backup media completely and in an unaltered form complete with the copyright notice.

Modifications to the Program

If the software supplied to you includes the source code, you may make modifications to the software. Before you make any modifications to the software please read the "Limited Warranty" document supplied with the software. Adena Scientific Limited accepts no liability whatsoever for any modifications that you make to the software, nor for any consequences whatsoever that may arise from the use of such modified software. If you modify the software you must enter the date, your name, and the details of the modification(s) that you made, into a "Modifications" comment section at the top of the source code to provide appropriate internal documentation and you must clearly mark all copies of the modified software with the words "Modified Version".

Non Permitted Uses

You may not, except as expressly permitted in this agreement;

1. copy, modify, or transfer the software, in whole or in part, electronically or otherwise, or;
2. translate, disassemble, decompile or otherwise reverse engineer the software, or;
3. rent, lease, sub-license or assign the software to any other person or organisation, or;
4. use the software for any unlawful or substantially unethical purpose.

Transfer of License

This license may be permanently transferred, by you, to another person or organisation, provided that you transfer the original software package, and all copies of the software and accompanying documentation, to that person or organisation. Any such transfer must include your most recent update version and all previous versions and terminates your license to use the software.

Termination of License

You may terminate your license to use the software at any time by either;

1. transferring it to another person or organisation as described above, or;
2. destroying the software and accompanying documentation together with all copies.

Adena Scientific Limited reserves the right to terminate your license to use the software, at our sole discretion and without prejudice to any other rights or remedies we may have in law, in the event that the annual license fee remains unpaid for more than three calendar months, or in any event where we become aware that you have breached the conditions of this Software License Agreement.

Jurisdiction

This agreement will be governed and construed in accordance with the laws of New Zealand.

Limited Warranty

To the original purchaser only;

Adena Scientific Limited warrants for 12 months from the date the software is delivered to you that the software will perform substantially the functions described in the documentation supplied with the software and that the media on which the software is supplied will be free from defects in materials and workmanship.

Adena Scientific Limited undertakes to correct, within a reasonable period of time, any reported software error or documentation error, or replace the distribution media the software was supplied on if it proves to be defective in material or workmanship on an exchange basis without charge. For the purpose of this warranty "software error" means failure of the software to perform substantially the functions described in the documentation supplied with the software.

Adena Scientific Limited does not warrant that operation of the software will be uninterrupted or error free, or that all software errors will be corrected. Adena Scientific Limited accepts no responsibility for any problems caused by changes in computer operating systems or the characteristics of computer hardware that are made after the delivery of the software nor for any problems in the interaction of the software with other software not produced by Adena Scientific Limited.

If Adena Scientific Limited is unable to replace defective distribution media or provide corrected software or corrected documentation within a reasonable period of time, Adena Scientific Limited will, at its sole discretion, either replace the software with a functionally equivalent program at no charge to you or refund the license fee of the software. Adena Scientific Limited will have no responsibility to replace the software or refund the license fee where a failure is caused by accident, abuse, misapplication, or where the software has been modified by you.

These are your sole and exclusive remedies for any breach of this warranty. All other liability of Adena Scientific Limited arising directly or indirectly in connection with this warranty, the software, its use, misuse or otherwise including (but without limitation) any loss of profit, business, revenue, goodwill or anticipated savings is hereby excluded. This exclusion of liability applies to liability in contract and/or in tort, including negligence.

This warranty applies to the exclusion of all warranties expressed or implied including implied warranties of merchantability and fitness for a particular purpose. No oral or written information or advice given by Adena Scientific Limited, its employees, distributors, dealers or agents shall increase the scope of the above warranty or create any new warranties.

The Consumer Guarantees Act 1993 shall not apply to the software when it is acquired for business purposes.

This warranty will be governed and construed in accordance with the laws of New Zealand.

NOTICE

This software License and Warranty applies ONLY to ZooCADA-Mod customisations written for end user customers by Adena Scientific Ltd.

ZooCADA system users familiar with Campbell Scientific datalogger programming may choose to write their own customisations using the ZooCADA-Mod module. User created customisations shall be used ONLY on ZooCADA stations licensed to that user, and MUST NOT be distributed to any third parties by any means whatsoever.

Adena Scientific Ltd shall NOT be responsible in any way whatsoever for the malfunction of any ZooCADA station(s) that result from user created modifications.

This Page Intentionally Left Blank

Contents

Contents

System Overview

The ZooCADA system provides a range of stations that are purpose designed to cater to a wide variety of typical applications at zoological facilities. Some customers may require additional sensors, data processing and/or datalogging to meet their specific needs and in many cases these requirements can be accommodated using spare I/O capacity on a datalogger that is running one of the ZooCADA system programs.

Adding software code into ZooCADA station programs to support customisation requests is extremely unwise as it creates disparate station programs that cannot be maintained within standardised ZooCADA quality control and periodic maintenance systems. The ZooCADA-Mod software module provides a means by which stations can be customised in a carefully structured manner without changing the station program itself.

Each ZooCADA station program has been developed with specific API points that link to the ZooCADA-Mod customisation file sections. Most software code that can be written into a datalogger program can be written into the ZooCADA-Mod customisation file and will operate in exactly the same way as it would if it had been written into the ZooCADA station program itself. Keeping the customisation code in its own file allows it to persist, in the same way the station constants and setpoints do, anytime the station program is updated.

The code written into the ZooCADA-Mod customisation is the same CR Basic code that is used for all the datalogger programming and is subject to exactly the same syntax rules. Likewise the same limitations apply in so far as the program code and the customisation code collectively must execute within the datalogger scan interval, which on all ZooCADA stations is two seconds, and the datalogger's available program memory and data storage memory.

Examples of possible station customisations are:

- Using the enclosure soil water content measurement to switch a spare control port on and off to operate a valve that prevents an irrigation system over watering the enclosure soil.
- Adding a tipping bucket rain gauge to a spare pulse input to measure and record the hourly and daily rainfall.
- Adding a solar pyranometer to an SDI-12 bus port to measure and record the hourly and daily solar radiation.

An empty ZooCADA-Mod customisation module is supplied with each ZooCADA station program and is disabled by default in the Station Constants file. The customisation file can be enabled and used when required and does not need to be loaded into the station unless it is enabled.

Program Features

Sensor Measurements

- SDI-12 sensors are recommended as there is ample address space available on all ZooCADA stations.
- Pulse output sensors such as wind speed and rainfall supported on CR1000X stations.
- Analog output sensors up to 4 on ZooLog, 1 on ZooLife, 1 on ZooLab, none on other stations.

Control Port I/O

- Configurable as Outputs or Inputs, 1 on ZooLog, up to 2 on ZooLife, up to 2 on ZooLab stations.
  See Campbell Scientific datalogger specifications for logic high/low voltage specifications.

Switched 12V DC Outputs

- Suitable for relay control, 1 on ZooLog, up to 2 on ZooLife, up to 2 on ZooLab stations.
  See Campbell Scientific datalogger specifications for output current limits.

Calculations and Data Processing

- Calculate derivative values from existing or new measurements.
- Apply conditional logic to measurements to switch datalogger outputs on/off.
  Amount of additional processing that can be added depends on available program memory.

Data Logging

- Add data tables to record data from additional sensors.
- Add data tables to record data from calculated derivative values.
  Amount of additional data that can be logged depends on available data storage memory.

General

- Define constants.
- Define public variables.
- Define private variables.
- Define and call data tables.
- Insert code into the program startup initialisation.
- Insert code into the beginning of the program scan immediately after timers.
- Insert code into the end of the program scan immediately before data logging.
- Insert code into the end of the program immediately after data logging.
- Insert code into slow scan Aux flag controlled section for SDI-12 sensor measurements.
- Insert code into slow scan uncontrolled section so customisation code can control it.

ZooCADA is a modular control and data acquisition system. Each station can operate standalone or as an integral part of a fully networked, zoo-wide, system with various stations performing different tasks. Using our modular approach, up to 4000 stations, distributed over any geographic area, can be networked provided that network connectivity (typically the site's IP computer network) is available at each station.

Adena Scientific believes that accuracy and reliability are paramount requirements of any system used in applications that support animal welfare, so we purpose designed our ZooCADA system to meet zoological needs, and built it to run on dataloggers manufactured by Campbell Scientific in the USA and available worldwide.

The Customisation File

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Section                                                                *
'* Station Customisations File For:                                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Declare Constants For Customisations Here.
  'Const ExampleConst = 100
#EndIf

#If Section = "PublicVars" Then
  'Declare Public Variables For Customisations Here.
  'Public ExamplePublicVar = 100
#EndIf

#If Section = "PrivateVars" Then
  'Declare Private Variables For Customisations Here.
  'Dim ExamplePrivateVar = 100
#EndIf

#If Section = "DataTables" Then
  'Declare Data Tables For Customisations Here.
  'DataTable (....)
#EndIf

#If Section = "InitCode" Then
  'Declare Initialisation Code For Customisations Here.
  'ExamplePrivateVar = 200
#EndIf

#If Section = "MainCodeStart" Then
  'Declare Main Program Code For Customisations Here.
  'Inserted at start of main program scan code.
  'BattV = Round(BattV,2)
#EndIf

#If Section = "MainCodeEnd" Then
  'Declare Main Program Code For Customisations Here.
  'Inserted at end of main program scan code.
  'BattV = Round(BattV,2)
#EndIf

#If Section = "CallTables" Then
  'Declare Call Data Tables For Customisations Here.
  'Insert reset code for counters after CallTables for logging.
  'CallTable(....)
#EndIf

#If Section = "SlowScanAux" Then
  'Declare Slow Scan Aux Program Code For Customisations Here.
  'Inserted into ReadAuxSensors flag controlled code block.
  'SDI12Recorder (Quantum_PPF,C7,0,"MC!",1.0,0)
#EndIf

#If Section = "SlowScanCode" Then
  'Declare Slow Scan Program Code For Customisations Here.
  'Inserted into slow scan loop always executed code block.
  'If condition Then
  '   Do Something
  'EndIf
#EndIf

'*****************************************************************************************************************
'*                                                END OF INCLUDE FILE                                            *
'*****************************************************************************************************************
```

The customisation file contains all the customisations for a station in the ZooCADA system. All ZooCADA stations support the application of customisations using this file. There is only one customisation file per station and the empty template file is exactly the same, except for its filename extension, for all ZooCADA stations.

The empty customisation file template is shown in the image above.

Each ZooCADA station program is supplied complete with a customisation file template which is set to disabled in the Station Constants file by default. If the customisation file is enabled while the template file is empty, the station program will still run normally.

When the customisation file is disabled in the Station Constants file the customisation file does not need to be loaded into the datalogger but it can be left there for convenience.

When adding custom code to the file it is imperative that the programmer remains aware of the section of the station program their customisation code is being inserted into as each section serves a different purpose. Each section will be discussed separately in the following chapters.

Supported Datalogger Versions

The ZooCADA-Mod customisation file utilises datalogger instructions that Campbell Scientific has recently included in the datalogger operating systems. The ZooCADA-Mod customisation file cannot be used on dataloggers with earlier operating system versions than those listed below. In most cases the datalogger operating systems can be easily upgraded to the latest versions if desired.

CR310

Support is included from OS 11.0 release date 21-02-2024.

CR1000X(e)

Support is included from OS 7.0 release date 16-11-2023.

File Naming

The empty customisation file is named the same for all stations, except for the filename extension, which differs to identify the target datalogger model in the same way the station program files do.

File Name

The customisation file is named STATION_CUSTOMISATIONS and must always be present in the datalogger, even if it's just the empty template, when it is enabled in the Station Constants file. Do not rename this file.

Filename Extension

The customisation filename extension identifies the target datalogger model range as follows:

For CR310 dataloggers the extension is .CR300.

For CR1000X dataloggers the extension is .CR1X.

Using datalogger specific filename extensions enables the use of datalogger program instructions that are specific to a datalogger model range while still maintaining code portability across all stations of that datalogger model. The same file with datalogger specific instructions changed where necessary can then be given the filename extension for a different datalogger model so the same customisation can be used for additional datalogger models.

Enabling The Customisation File

The customisation file is enabled by setting the constant CUSTOM_FILE to True in the STATION_CONSTANTS file and disabled by setting the constant CUSTOM_FILE to False.

When the customisation file is enabled, the CR Basic compiler inserts the customisation code into the datalogger program and compiles it just as if the code had been written directly into the datalogger program.

After loading an updated customisation file and/or an updated station constants file to the datalogger, the updated files have to be compiled into the datalogger program by the datalogger to make them active. The datalogger always recompiles its program file(s) when the datalogger is restarted so to cause a recompile and thus include the updated customisations file, manually stop the datalogger program and then restart it using either the functions in the File Control window of the LoggerNet Connect utility, or the functions provided by File Control page in the datalogger Web Interface. The Web Interface functions are only available to users logged on with administrator privileges.

Editing The Customisation File

The customisation file can be edited in the same way as any other datalogger program using the CR Basic editor in LoggerNet.

We recommend keeping a folder on a computer, use the station name for the folder name, and inside that folder keep a second folder named "Test". The station named folder should hold a copy of the files currently operating in the station. The following example is for the ZooLife program but all other station programs in the ZooCADA system have the same files, just the program name portion of the filename changes.

There should be four files in the station named folder:

STATION_CONSTANTS_ZOOLIFE_R01.CR1X

STATION_SETPOINTS_ZOOLIFE_R01.CR1X

STATION_CUSTOMISATIONS.CR1X

ZOOLIFE_R01.CR1X

If the program has been compiled in the CR Basic editor a table definitions file will also be present:

ZOOLIFE_R01.TDF

Copy all these files to the Test folder. Make the desired changes by editing the file(s) in the Test folder and fully test the program prior to deployment. Once deployed, copy the deployed files into the station named folder.

This way there are always two copies of the files on the computer, the "live" program as currently deployed and the "test" version ready for editing.

To edit the STATION_CUSTOMISATIONS.CR1X file, navigate to the Test folder and open the file in the CR Basic editor. Make the desired changes and save the file.

Open the STATION_CONSTANTS_ZOOLIFE_R01.CR1X file and change the constant from CUSTOM_FILE = False to CUSTOM_FILE = True and save the file.

To compile the program, open the ZOOLIFE_R01.CR1X file in the editor and use the Save and Compile selection from the menus. The compiler will compile the program and display any errors that need to be corrected.

When complied the STATION_CUSTOMISATIONS.CR1X file sections are automatically inserted into the main station program, in this example ZOOLIFE_R01.CR1X.

Once an error-free compile is achieved in the CR Basic editor, the files can be loaded into a datalogger for testing. Using the LoggerNet Connect utility, connect to the datalogger then click the File Control button on the tool bar. Check if any program is running and stop it if necessary.

Next click on the Send button on the toolbar, navigate to the Test folder on the computer and select the files to be loaded into the datalogger. First load the Constants, Setpoints, and Customisations files to the datalogger, then load the program file to the datalogger.

IMPORTANT: All four of the CR1X files, or CR300 files, as appropriate, must be present in the datalogger in order for the datalogger to compile and run the program.

Once all these files are loaded into the datalogger, click on the program file ZOOLIFE_R01.CR1X in this example, then click the Run Options button on the toolbar. Make sure the Run Now and Run on Power Up checkboxes are selected and click the OK button. The datalogger will compile the program into its memory and run it.

Carry out whatever tests are necessary to prove the customisations are working correctly, then the program and the customisations are ready to be deployed.

Demonstration Rainfall Program

In the following chapters a STATION_CUSTOMISATIONS.CR1X file written to add a tipping bucket rain gauge to the ZooLife program is explored section by section to demonstrate the methodology used to create a working customisation.

Each chapter will begin with a listing of the file with the section to be discussed highlighted and subsequent pages will discuss the programming requirements.

Constants Section

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                                 *
'* Station Customisations File For: TEST (Rainfall)                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************************
'*                                                END OF INCLUDE FILE                                            *
'*****************************************************************************************************************
```

The constants section begins with the complier instruction:

```
#If Section = "Constants" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "Constants" in the main program.

All customisation constants must be declared in this section.

Constants

Constants are simply values that are defined at program startup and do not change during program execution.

The keyword Const is used to define a constant.

Once a value is assigned to a constant, which must be done as part of the Const declaration, the programmer can simply type the constant name into the program each time the value is needed, instead of the value itself. The use of constants makes the program easier to follow and easier to modify.

Constants can be type defined as numeric, logical, or string values.

Good practice is to name constants with all uppercase characters so they are clearly differentiated from variables. Separate multiple words in a constant name with the underscore character, and, except for simple decimal values, define the type. For example:

```
Const MY_FLOAT = 21.45                  -  (Default type,  Single-precision floating point number)
Const MY_DOUBLE As Double = 12.345678   -  (Double-precision floating point number)
Const MY_LONG As Long = 123456789       -  (32-bit signed integer.)
Const MY_BOOL As Boolean = True         -  (Boolean, False=0, True = -1)
Const MY_STRING As String = "Hello"     -  (Null-terminated array of characters)
```

The compiler will attempt to determine the type for any constants for which a type is not defined. Note that not all data types types are supported by all datalogger models.

Example

In the example customisation file there is one constant defined as follows:

```
Const BUCKET = 0.2
```

In this example the constant BUCKET is the bucket capacity in millimetres for the tipping bucket rain gauge and it is set to 0.2 millimetres. The value could have been entered directly into the Pulsecount() instruction but a constant was used for clarity when reading the program.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file there is one comment in the constants section as follows:

```
'Tipping bucket rain gauge calibration.
```

Public Variables Section

```
'*****************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                     *
'* Station Customisations File For: TEST (Rainfall)                                                  *
'* Copyright 2015-2024 Adena Scientific Limited                                                      *
'* Datalogger: Campbell Scientific CR1000X                                                           *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                            *
'* Revision Date: 2024-12-07                                                                         *
'*****************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************
'*                                        END OF INCLUDE FILE                                        *
'*****************************************************************************************************
```

The public variables section begins with the complier instruction:

```
#If Section = "PublicVars" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "PublicVars" in the main program.

All customisation public variables must be declared in this section.

Public Variables

Variables are values that change during program execution. Public variables are those that are displayed in the Public data table and can be viewed using the datalogger's Web Interface or the LoggerNet Connect utility.

The keyword Public is used to define public variables.

Public variables can be type defined as numeric, logical, or string values and an initial value for the variable can be set as part of the variable definition. Variable names must start with a letter, an underscore, or a dollar sign and are not case sensitive.

Good practice is to name variables using Pascal case notation. In Pascal case the first letter of each word is in uppercase and all the other letters are in lower case, with all the words joined together with no spaces. With public variables, except for simple decimal values, always define the type. For example:

```
Public MyFloat = 21.45                  -   (Default type,  Single-precision floating point number)
Public MyDouble As Double = 12.345678   -   (Double-precision floating point number)
Public MyLong As Long = 123456          -   (32-bit signed integer.)
Public MY_BOOL As Boolean = True        -   (Boolean, False=0, True = -1)
Public MyString As String = "Hello"     -   (Null-terminated array of characters)
```

Note that not all data types types are supported by all datalogger models.

Example

In the example customisation file there are two public variables defined as follows:

```
Public RainHour = 0
Public RainDay = 0
```

In this example the variable RainHour holds the accumulated total rainfall for the current hour and the variable RainDay holds the accumulated rainfall for the current day. Both values will be displayed in the Public data table. The customisations public variables are always displayed at the bottom of the Public data table.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file there is one comment in the constants section as follows:

```
'Rainfall public variables for display.
```

Public Variable Aliases

In addition to defining new variables, as demonstrated in the example code, the public variables section can be used to alias existing public variables when one or more existing variable names aren't ideal for a particular station deployment. The alias becomes the new name for the variable which the station program will use to display the variable in the Public data table. For example:

```
Alias ExistingVarName = NewVarName
```

Where ExistingVarName is the existing variable name in the station program and NewVarName is the alias that will be displayed.

Note: Public variable aliases should be defined at the bottom of the public variables section immediately before the #EndIf instruction.

In most station programs the use of aliases is unnecessary because the variable names used throughout the ZooCADA system are, for the most part, intentionally generic and self explanatory.

If a variable name for a variable that's logged in one or more data storage tables is aliased, the generic name for the variable will still be used to label the data fields in the logged data. This is intentional behaviour to ensure third party data import templates can be used consistently across the ZooCADA product range irrespective of the deployment site.

In some station programs, such as ZooCADA-Store, the food storage room arrangements may vary from the default of Freezer, Fridge and Pantry. In such cases the user may need to identify the rooms differently, so on these station programs the data storage tables are programmed to allow them to be aliased if desired.

This Page Intentionally Left Blank

Private Variables Section

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                                 *
'* Station Customisations File For: TEST (Rainfall)                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************************
'*                                                END OF INCLUDE FILE                                            *
'*****************************************************************************************************************
```

The private variables section begins with the complier instruction:

```
#If Section = "PrivateVars" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "PrivateVars" in the main program.

All customisation private variables must be declared in this section.

Private Variables

Variables are values that change during program execution. Private variables are not displayed in the Public data table and cannot be viewed using the datalogger's Web Interface or the LoggerNet Connect utility.

The keyword Private is used to define private variables.

Private variables can be type defined as numeric, logical, or string values and an initial value for the variable can be set as part of the variable definition. Variable names must start with a letter, an underscore, or a dollar sign and are not case sensitive.

Good practice is to name variables using Pascal case notation. In Pascal case the first letter of each word is in uppercase and all the other letters are in lower case, with all the words joined together with no spaces. As with private variables, except for simple decimal values, always define the type. For example:

```
Private MyFloat = 21.45                  -  (Default type,  Single-precision floating point number)
Private MyDouble As Double = 12.345678   -  (Double-precision floating point number)
Private MyLong As Long = 123456          -  (32-bit signed integer.)
Private MY_BOOL As Boolean = True        -  (Boolean, False=0, True = -1)
Private MyString As String = "Hello"     -  (Null-terminated array of characters)
```

Note that not all data types types are supported by all datalogger models.

Example

In the example customisation file there is one public variable defined as follows:

```
Private RainTips = 0
```

In this example the variable RainTips holds the number of bucket tips multiplied by the constant BUCKET that is read from the pulse counter each datalogger program scan. ZooCADA stations scan every two seconds so unless it is raining extremely hard this value will be 0.0 most scans and will change to 0.2 if the bucket has tipped since the last scan. If the bucket tipped twice since the last scan the value would be 0.4, etc. This value is added to the total rainfall for the current hour RainHour and the current day RainDay for later logging.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file there is one comment in the constants section as follows:

```
'Rainfall private variables.
```

Data Tables Section

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                                 *
'* Station Customisations File For: TEST (Rainfall)                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************************
'*                                                END OF INCLUDE FILE                                            *
'*****************************************************************************************************************
```

The data tables section begins with the complier instruction:

```
#If Section = "DataTables" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "DataTables" in the main program.

All customisation data tables, functions and subroutines must be defined in this section.

Data Tables

Data tables are used to store the data that is logged by the datalogger. A data table needs to be defined for each desired data set. Normally data tables store the data at regular time intervals such as hourly or daily, but event driven data logging can also be programmed.

Data in a data table is arranged into records (rows of values) and columns (the values in the record)

The DataTable(Name, TrigVar, Size) instruction is the first line of a data table definition. It specifies the Name for the data file, enables or disables the data table with the TrigVar trigger variable, and sets the Size number of records (rows of data) that can be stored in the table. TrigVar can be a program variable which is set True to enable the data table or False to disable the data table. If True is entered as the TrigVar the data table is always enabled. If -1 is entered for the number of rows, the datalogger automatically determines a size that will evenly allocate the amount of available data storage memory to all the tables that have -1 for their size. The dataloggers use a ring memory system as the default data storage so that when a data table is full, the newest data overwrites the oldest data in the table. This behaviour can be changed programmatically if desired.

The DataInterval(TintoInt, Interval, Units, Lapses) instruction is the second line of a data table definition. It defines the interval at which the data is to be logged. The interval is specified as the time into a time interval, such as 0 units into a 60 unit interval, with a range of units available including seconds, minutes, hours, and days. The final parameter is the lapses setting. A lapse is any discontinuity in the records' time intervals. Entering 0 for the Lapses parameter forces every record to include a record number and timestamp, which uses an additional 16 bytes per record. In ZooCADA station programs lapses is always set to 0. If data storage space is not an issue, this option should be used. Please refer to the Campbell Scientific CR Basic editor help system for detailed information on the operation of the lapses setting.

The third and subsequent lines, before the EndTable instruction, define a column in the data table using output processing instructions.

Output processing instructions are in the form of Sample(Reps, Source, DataType) with some having additional parameters. Output processing instructions include Average(), Minimum(), Maximum(), and Totalize(). Each output processing instruction determines the Reps (swath of variables), starting with the variable Source, that will be logged and the DataType. The data type is typically FP2 which is a Campbell Scientific 2-byte floating point data type that is memory efficient and suits a wide range of common measurement values. Sometimes a higher precision is needed, such as for barometric pressure, where ZooCADA uses the IEEE4 data type, a 4-byte value. It is important to match the data types in the data tables to the data types of the variables that are being logged. Each output processing instruction should be entered on its own line in the program code.

The final line in a data table definition is the EndTable() instruction marking the end of the data table definition.

Several data tables may be defined in this section as necessary dependant upon the amount of available data storage memory in the datalogger. Customisations that need to log data must use their own data tables. The customisation code does not enable modification of ZooCADA data tables which should not be modified.

Functions And Subroutines

In addition to data tables, functions and subroutines can be defined in this section. These should be placed after any data tables that are programmed.

Functions and subroutines are programmed in much the same way. Both structures accept arguments (values passed to the function or subroutine) from variables and constants on the program. The most significant difference is that functions can return a value whereas subroutines cannot.

The first line of a function definition is in the form Function Name(Arg1, Arg2, Arg3,...) and similarly for subroutines Sub Name (Arg1, Arg2, Arg3,...). In both cases Name is the name that will be used to call the function or subroutine from the program and Arg1, Arg2, Arg3, ... are variables that are created by, and belong to, the function or subroutine that accept input values from variables in the program. Additional variables for processing purposes can be declared within a function or subroutine and remain private to the function or subroutine.

The second and subsequent lines of a function or subroutine, before the EndFunction or EndSub instruction, are the programming instructions that the function or subroutine will execute.

The full range of CR Basic program instructions, that are supported by the datalogger model in use, can be used for functions and subroutines.

Utilising this flexibility to create customisations for the ZooCADA station program requires considerable care to ensure that customisations do not inadvertently conflict with the programming of the ZooCADA station program as serious program errors can be created.

Please refer to the Campbell Scientific Product Manual (for the datalogger model being programmed), the LoggerNet CR Basic Editor documentation, and the ZooCADA Reference Manual (for the station program) regarding the capabilities and programming requirements of the systems being programmed.

The EndFunction is the final line of a function and EndSub is the final line of a subroutine. These keywords mark the end of the definition.

Example

In this example the first data table is defined as follows:

```
DataTable(RAIN_60M, True, -1)
    DataInterval(0, 60, Min, 0)
    Sample(1, RainHour, FP2)
EndTable
```

In this example only a single data value, the hourly rainfall, is being logged but frequently there will be several values specified. All the output processing instructions are placed between the DataInterval() instruction and the EndTable instruction, with each output instruction entered on its own line.

The first line defines the data file name as RAIN-60M (for 60 minute rainfall), the data table is set to True (always enabled), and the size is set to -1 so the datalogger automatically determines the table size.

The second line defines the DataInterval() which in this case is 0 minutes into a 60 minute interval. The last zero is the lapses parameter which is set to zero so each record is time stamped.

The third line specifies the first value in the record as an instantaneous Sample, with 1 repetition (sample one variable) beginning with the variable RainHour, with a data type of FP2.

In this example the second data table is defined as follows:

```
DataTable(RAIN_24H, True, -1)
    DataInterval(0, 1440, Min, 0)
    Sample(1, RainDay, FP2)
EndTable
```

The format of the second data table is identical to the first but note the different parameters. The filename is set to RAIN_24H (for 24 hour rainfall), the data interval is set to 0 minutes into a 1440 minute interval (daily at midnight) and the variable being sampled is RainDay which is the daily rainfall total accumulator.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file there is a comment on the line above each DataTable definition to indicate what data the data table contains: The first data table has the comment:

```
'Log rainfall hourly data.
```

And the second data table has the comment:

```
'Log rainfall daily data.
```

In addition to data tables, functions and subroutines can be defined in this section. These should be programmed after any data tables are programmed.

Initialisation Code Section

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                                 *
'* Station Customisations File For: TEST (Rainfall)                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************************
'*                                                END OF INCLUDE FILE                                            *
'*****************************************************************************************************************
```

The initialisation code section begins with the complier instruction:

```
#If Section = "InitCode" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "InitCode" in the main program.

All customisation code initialisation routines must be defined in this section.

Initialisation Code

Initialisation code is code that is run once only during program startup to ensure an orderly program startup.

By default numeric variables are initialised to zero when they are defined and strings are initialised as empty strings which may cause program errors unless they are assigned an initial value at startup. Often, variables can simply be given an appropriate value when the variable is declared but sometimes it is necessary to do something more complex which is best achieved with initialisation code.

Most of the CR Basic program instructions, that are supported by the datalogger model in use, can be used in initialisation code, so things like using a loop to initialise an array are possible.

Utilising this flexibility to create customisations for the ZooCADA station program requires considerable care to ensure that customisations do not inadvertently conflict with the programming of the ZooCADA station program as serious program errors can be created.

Example

In this example there is no initialisation code needed so the initialisation section is simply left empty.

Of course the unneeded section could be completely deleted from the customisation file but we recommend leaving it in place, as shown, because this makes it easier to maintain the customisation file if, for example, someone wishes to add another instrument to the customisation and the associated instrument processing does need some initialisation code.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file there is a comment in the empty initialisation code section simply reminding us that it is empty on purpose, as follows:

```
'No Initialisation needed for rainfall program.
```

Main Code Start Section

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                                 *
'* Station Customisations File For: TEST (Rainfall)                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************************
'*                                                END OF INCLUDE FILE                                            *
'*****************************************************************************************************************
```

The main code start section begins with the complier instruction:

```
#If Section = "MainCodeStart" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "MainCodeStart" in the main program.

This section is inserted near the beginning of the datalogger scan immediately after the timers that control the various functions of the main program and the RTC (realtime clock) has been read into the rTime() array. It is the best place in the program to insert timers that set and/or reset flags to control the execution of blocks of program code, measurement instructions for sensors that can be read quickly, and processing instructions for those sensor measurements. General calculations and other processing can also be included at this point.

Program Code

The full range of CR Basic program instructions, that are supported by the datalogger model in use, can be used in this section of code.

When creating program code, it is important to consider the time that the CPU will take to execute the instructions. ZooCADA station programs all have a scan rate of two seconds. Correct execution of the program, requires that the entire program has an execution time that is less than the scan rate. If the execution time exceeds the scan rate it will result in missed scans. Execution times vary depending on the instructions that are executed in any given scan so scans that log data or measure sensors are likely to take longer than most other scans, especially the scan that carries out the midnight logging during which several data tables are written to.

Utilising this flexibility to create customisations for the ZooCADA station program requires considerable care to ensure that customisations do not inadvertently conflict with the programming of the ZooCADA station program as serious program errors can be created.

Please refer to the Campbell Scientific Product Manual (for the datalogger model being programmed), the LoggerNet CR Basic Editor documentation, and the ZooCADA Reference Manual (for the station program) regarding the capabilities and programming requirements of the systems being programmed.

Time Array

The rTime() array can be read by customisation code, if necessary, using the following aliases rather than referencing the array itself so the resultant code is easier to follow. Never write to this array, doing so will cause the timing calculations throughout the main program to fail.

```
rTime(1) - alias = Year
rTime(2) - alias = Month
rTime(3) - alias = DOM
rTime(4) - alias = Hour
rTime(5) - alias = Minute
rTime(6) - alias = Second
rTime(7) - alias = uSecond
rTime(8) - alias = WeekDay
rTime(9) - alias = DayOfYear
```

Example

In this example a tipping bucket rain gauge is measured and processed every scan:

```
PulseCount(RainTips,1,P2,1,0,BUCKET,0)
RainHour += RainTips
RainHour = Round(RainHour,1)
RainDay += RainTips
RainDay = Round(RainDay,1)
```

The first line reads pulse counter P2 and places the tip count into the variable RainTips after multiplying the raw counter value by the value of the constant BUCKET, which is declared in the constants section with a value of 0.2, the rain gauge tipping bucket size in millimetres of rainfall.

The second line of code adds the value of the RainTips variable to the hourly rainfall accumulator RainHour variable.

The third line of code applies the rounding instruction Round() to the RainHour variable to round it to one decimal place. It can be desirable to use rounding instructions on floating point math processing to remove any additional decimal places that sometimes result when floating point math operations are carried out. It is not necessary to do this rounding but it is a nice piece of housekeeping that helps keep logged and/or displayed data tidy.

The fourth line of code adds the value of the RainTips variable to the hourly rainfall accumulator RainDay variable.

The fifth line of code applies the rounding instruction to round the RainDay variable to one decimal place as was done for the hourly rainfall accumulator variable.

The variables RainHour and RainDay are declared in the public variables section so are displayed at the bottom of the Public data table after all the standard ZooCADA program public variables.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file the comment above the measurement instruction informs as to the purpose of the PulseCount() instruction, as follows.

```
'Read pulse count from the rain gauge.
```

Another comment above the rainfall processing lines informs as to their purpose, as follows:

```
'Increment rainfall totals.
```

These brief comments can help anyone who has to maintain the code in the future understand what the author of the code intended. It is not necessary to comment every line of code and brevity is usually best but sometimes a complex code fragment may benefit from an extra line or two of comment at its beginning.

This Page Intentionally Left Blank

Main Code End Section

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                                 *
'* Station Customisations File For: TEST (Rainfall)                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************************
'*                                              END OF INCLUDE FILE                                              *
'*****************************************************************************************************************
```

The main code end section begins with the complier instruction:

```
#If Section = "MainCodeEnd" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "MainCodeEnd" in the main program.

This block of code is inserted near the end of the datalogger scan, immediately before the CallTable() instructions for the ZooCADA program data tables. All of the main program variables are current at this point in the program execution making this is the best place to insert code for customisation calculations or processing that needs to be completed prior to the output data tables being called.

Program Code

The full range of CR Basic program instructions, that are supported by the datalogger model in use, can be used in this section of code. Any of the variables displayed in the Public data table can be read and used to calculate derivative values, which can then be logged. Code to set/reset a control port or switch on/off a 12V DC output in response to a variable can also be added to this section.

When creating program code, it is important to consider the time that the CPU will take to execute the instructions. ZooCADA station programs all have a scan rate of two seconds. Correct execution of the program, requires that the entire program has an execution time that is less than the scan rate. If the execution time exceeds the scan rate it will result in missed scans. Execution times vary depending on the instructions that are executed in any given scan so scans that log data or measure sensors are likely to take longer than most other scans, especially the scan that carries out the midnight logging during which several data tables are written to.

Utilising this flexibility to create customisations for the ZooCADA station program requires considerable care to ensure that customisations do not inadvertently conflict with the programming of the ZooCADA station program as serious program errors can be created.

Please refer to the Campbell Scientific Product Manual (for the datalogger model being programmed), the LoggerNet CR Basic Editor documentation, and the ZooCADA Reference Manual (for the station program) regarding the capabilities and programming requirements of the systems being programmed.

Example

In this example there is no code needed in this section so the section is simply left empty.

Of course the unneeded section could be completely deleted from the customisation file but we recommend leaving it in place, as shown, because this makes it easier to maintain the customisation file if, for example, someone wishes to add some code in the future.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file there is a comment in the empty initialisation code section simply reminding us that it is empty on purpose, as follows:

```
'Nothing needed here for rainfall program.
```

Call Tables Section

```
'*****************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                     *
'* Station Customisations File For: TEST (Rainfall)                                                  *
'* Copyright 2015-2024 Adena Scientific Limited                                                      *
'* Datalogger: Campbell Scientific CR1000X                                                           *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                            *
'* Revision Date: 2024-12-07                                                                         *
'*****************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************
'*                                        END OF INCLUDE FILE                                        *
'*****************************************************************************************************
```

The call tables section begins with the complier instruction:

```
#If Section = "CallTables" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "CallTables" in the main program.

This block of code is inserted at the end of the datalogger scan immediately after the CallTable() instructions for the ZooCADA program data tables. Customisation data tables must be called in this section and any code to reset counter or accumulator variables after data logging should also be placed in this section.

Program Code

Any CallTable() instructions should be the first lines in this section, with any other code, such as counter and accumulator resets, placed after the CallTable() instructions.

The full range of CR Basic program instructions, that are supported by the datalogger model in use, can be used in this section of code.

When creating program code, it is important to consider the time that the CPU will take to execute the instructions. ZooCADA station programs all have a scan rate of two seconds. Correct execution of the program, requires that the entire program has an execution time that is less than the scan rate. If the execution time exceeds the scan rate it will result in missed scans. Execution times vary depending on the instructions that are executed in any given scan so scans that log data or measure sensors are likely to take longer than most other scans, especially the scan that carries out the midnight logging during which several data tables are written to.

Utilising this flexibility to create customisations for the ZooCADA station program requires considerable care to ensure that customisations do not inadvertently conflict with the programming of the ZooCADA station program as serious program errors can be created.

Please refer to the Campbell Scientific Product Manual (for the datalogger model being programmed), the LoggerNet CR Basic Editor documentation, and the ZooCADA Reference Manual (for the station program) regarding the capabilities and programming requirements of the systems being programmed.

Example

In this example the two rainfall data tables, defined in the data tables section, are called and then the accumulators are reset based on the hourly and daily timing.

```
CallTable(RAIN_60M)
CallTable(RAIN_24H)

If IfTime(0,60,Min) Then
   RainHour = 0
EndIf
If IfTime(0,1440,Min) Then
   RainDay = 0
EndIf
```

The first line of code calls the RAIN_60M data table.

The second line of code calls the RAIN_24H data table.

The third line of code is a conditional statement with an IfTime() instruction that is set to 0 minutes into a 60 minute interval, which causes it to return True every hour on the hour (hourly). Its placement in the program ensures the rainfall values have been logged before this block of code is executed.

The fourth line of code is within the hourly conditional block so it is only executed in a scan when the conditional statement is true, at which time it resets the hourly rainfall accumulator to zero.

The fifth line of code is the EndIf keyword which marks the end of the hourly conditional code block.

The sixth, seventh, and eighth lines of code are for a second conditional code block that operates in exactly the same way as the first except that the IfTime() instruction is set to 0 minutes into a 1440 minute interval, which causes it to return True once every day at midnight, and the accumulator being reset is the RainDay variable.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file the first comment simply identifies the start of the CallTables() instructions:

```
'Call rainfall data tables.
```

The next comment informs as to the purpose of the first conditional code block:

```
'Reset hourly rainfall accumulator after logging.
```

The last comment in this section informs as to the purpose of the second conditional code block:

```
'Reset daily rainfall accumulator after logging.
```

These brief comments can help anyone who has to maintain the code in the future understand what the author of the code intended. It is not necessary to comment every line of code and brevity is usually best but sometimes a complex code fragment may benefit from an extra line or two of comment at its beginning.

This Page Intentionally Left Blank

Slow Scan Aux Section

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                                 *
'* Station Customisations File For: TEST (Rainfall)                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************************
'*                                              END OF INCLUDE FILE                                              *
'*****************************************************************************************************************
```

The slow scan aux section begins with the complier instruction:

```
#If Section = "SlowScanAux" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "SlowScanAux" in the main program.

This block of code is inserted into the portion of the slow scan loop that is controlled by the five minute timer. This is the auxiliary sensor code block. Customisation code that reads SDI-12 digital sensors, or other devices that take longer time periods to read and processes should be placed in this section.

Program Code

The full range of CR Basic program instructions, that are supported by the datalogger model in use, can be used in this section of code.

This slow scan loop runs asynchronously of the main program scan which prevents the main program scan execution being delayed by the time required for operations such as communications to digital sensors. Any measurement values obtained in this section should be passed back to the main program scan for logging.

When creating program code, it is important to consider the time that the CPU will take to execute the instructions. ZooCADA station programs all have a scan rate of two seconds. Correct execution of the program, requires that the entire program has an execution time that is less than the scan rate. If the execution time exceeds the scan rate it will result in missed scans. Execution times vary depending on the instructions that are executed in any given scan so scans that log data or measure sensors are likely to take longer than most other scans, especially the scan that carries out the midnight logging during which several data tables are written to.

Utilising this flexibility to create customisations for the ZooCADA station program requires considerable care to ensure that customisations do not inadvertently conflict with the programming of the ZooCADA station program as serious program errors can be created.

Please refer to the Campbell Scientific Product Manual (for the datalogger model being programmed), the LoggerNet CR Basic Editor documentation, and the ZooCADA Reference Manual (for the station program) regarding the capabilities and programming requirements of the systems being programmed.

Example

In this example there is no code needed in this section so the section is simply left empty.

Of course the unneeded section could be completely deleted from the customisation file but we recommend leaving it in place, as shown, because this makes it easier to maintain the customisation file if, for example, someone wishes to add some code in the future.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file there is a comment in the empty initialisation code section simply reminding us that it is empty on purpose, as follows:

```
'Nothing needed here for rainfall program.
```

Slow Scan Code Section

```
'*****************************************************************************************************************
'* ZooCADA Station Program Customisations Module                                                                 *
'* Station Customisations File For: TEST (Rainfall)                                                              *
'* Copyright 2015-2024 Adena Scientific Limited                                                                  *
'* Datalogger: Campbell Scientific CR1000X                                                                       *
'* File name: STATION_CUSTOMISATIONS.CR1X                                                                        *
'* Revision Date: 2024-12-07                                                                                     *
'*****************************************************************************************************************

#If Section = "Constants" Then
  'Tipping bucket rain gauge calibration.
  Const BUCKET = 0.2
#EndIf

#If Section = "PublicVars" Then
  'Rainfall public variables for display.
  Public RainHour = 0
  Public RainDay = 0
#EndIf

#If Section = "PrivateVars" Then
  'Rainfall private variables.
  Dim RainTips = 0
#EndIf

#If Section = "DataTables" Then
  'Log rainfall hourly data.
  DataTable(RAIN_60M,True,-1)
    DataInterval(0,60,Min,0)
    Sample(1,RainHour,FP2)
  EndTable
  'Log rainfall daily data.
  DataTable(RAIN_24M,True,-1)
    DataInterval(0,1440,Min,0)
    Sample(1,RainDay,FP2)
  EndTable
#EndIf

#If Section = "InitCode" Then
  'No initialisation needed for rainfall program.
#EndIf

#If Section = "MainCodeStart" Then
  'Read pulse count from rain gauge.
  PulseCount(RainTips,1,P2,1,0,BUCKET,0)
  'Increment rainfall totals.
  RainHour += RainTips
  RainHour = Round(RainHour,1)
  RainDay += RainTips
  RainDay = Round(RainDay,1)
#EndIf

#If Section = "MainCodeEnd" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "CallTables" Then
  'Call rainfall data tables.
  CallTable(RAIN_60M)
  CallTable(RAIN_24M)
  'Reset hourly rainfall accumulator after logging.
  If IfTime(0,60,Min) Then
    RainHour = 0
  EndIf
  'Reset daily rainfall accumulator after logging.
  If IfTime(0,1440,Min) Then
    RainDay = 0
  EndIf
#EndIf

#If Section = "SlowScanAux" Then
  'Nothing needed here for rainfall program.
#EndIf

#If Section = "SlowScanCode" Then
  'Nothing needed here for rainfall program.
#EndIf

'*****************************************************************************************************************
'*                                              END OF INCLUDE FILE                                              *
'*****************************************************************************************************************
```

The slow scan code section begins with the complier instruction:

```
#If Section = "SlowScanCode" Then
```

The section ends with the instruction:

```
#EndIf
```

The #If and #EndIf instructions are compiler directives which do not form part of the executable program code that runs in the datalogger, but rather they tell the compiler how the program code should be managed during the compiling process. This pair of instructions tells the compiler to include the program code that is contained within these two instructions at the insertion point named "SlowScanCode" in the main program.

This block of code is inserted into the slow scan loop that runs continuously. Code inserted into this section must be controlled by some form or timer or control flag in the main program scan. Customisation code that for SDI-12 digital sensors and/or other code that needs to run at a custom controlled interval should be placed in this section.

Program Code

The full range of CR Basic program instructions, that are supported by the datalogger model in use, can be used in this section of code.

This slow scan loop runs asynchronously of the main program scan which prevents the main program scan execution being delayed by the time required for operations such as communications to digital sensors. Any measurement values obtained in this section should be passed back to the main program scan for logging.

When creating program code, it is important to consider the time that the CPU will take to execute the instructions. ZooCADA station programs all have a scan rate of two seconds. Correct execution of the program, requires that the entire program has an execution time that is less than the scan rate. If the execution time exceeds the scan rate it will result in missed scans. Execution times vary depending on the instructions that are executed in any given scan so scans that log data or measure sensors are likely to take longer than most other scans, especially the scan that carries out the midnight logging during which several data tables are written to.

Utilising this flexibility to create customisations for the ZooCADA station program requires considerable care to ensure that customisations do not inadvertently conflict with the programming of the ZooCADA station program as serious program errors can be created.

Please refer to the Campbell Scientific Product Manual (for the datalogger model being programmed), the LoggerNet CR Basic Editor documentation, and the ZooCADA Reference Manual (for the station program) regarding the capabilities and programming requirements of the systems being programmed.

Example

In this example there is no code needed in this section so the section is simply left empty.

Of course the unneeded section could be completely deleted from the customisation file but we recommend leaving it in place, as shown, because this makes it easier to maintain the customisation file if, for example, someone wishes to add some code in the future.

It is a good idea to add comments in the code to help make the code easy to follow. Comments can be entered on a line of their own, or at the end of a line of code. The comment delimiter is the single quote (') symbol which must be placed at the beginning of the comment text.

In the example customisation file there is a comment in the empty initialisation code section simply reminding us that it is empty on purpose, as follows:

```
'Nothing needed here for rainfall program.
```

Installation Notes

Use these pages to record any notes specific to the site installation or operation.

Templates for Record Keeping

The following pages are intended to assist users with record keeping.

Good records are essential for audit, research and maintenance purposes. We recommend that users keep an ongoing file of all system settings in a manner that is appropriate to their business operations.

Each time settings are changed, a new record of the settings should be created and added to the file. Previous records should be retained when new records are added. This will provide an auditable trail of all setting changes that can assist research with site specific records of what works well for the animal species on display at that site.

These records also assist with maintenance. In the event that system settings have to be reinstated after maintenance work, the correct set of documented settings can be used to complete the reinstatement.

Please photocopy the following pages as needed and use them to create your own record keeping system.

This Page Intentionally Left Blank

Station Change Log

Station Name: ..... Site ID: .....

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Station Change Log

Station Name: ..... Site ID: .....

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	

Date of Change:	Change(s) made:
Changed By:	