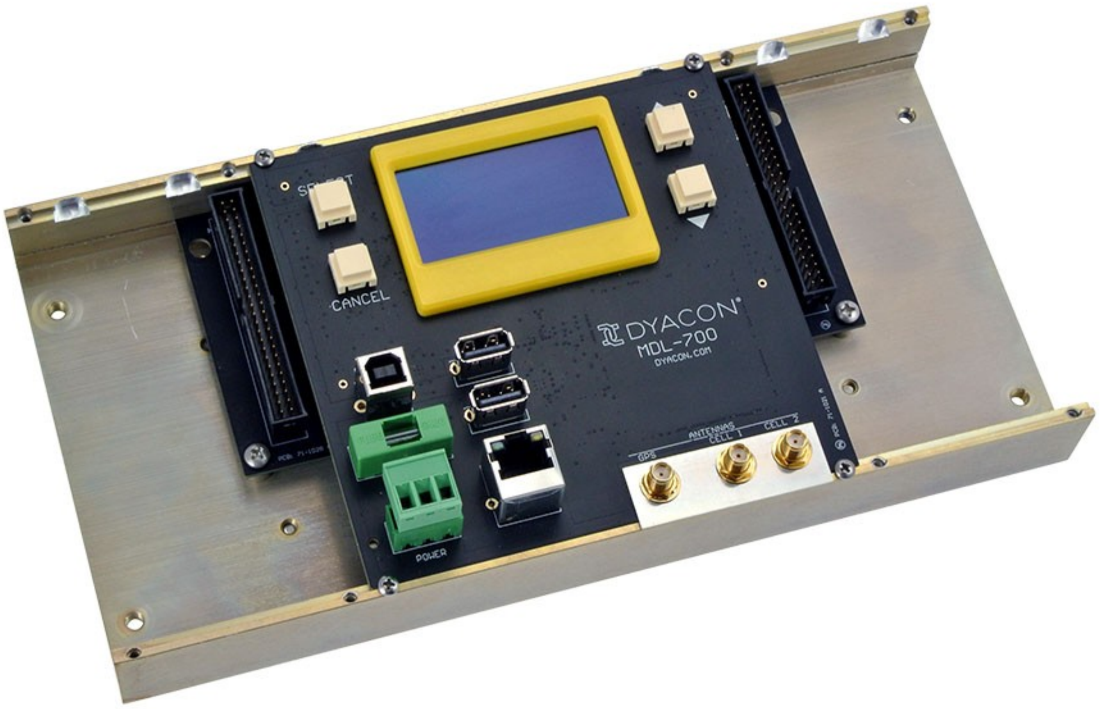


MDL-700 Modular Data Logger Manual

57-6088 Rev X2 – Preliminary Release

This page intentionally left blank.

CONTENTS

1.0 NOTICES.....4

2.0 INTRODUCTION.....6

2.1 Scope.....6
2.2 Technical Support.....7

3.0 QUICK-START GUIDE.....8

4.0 PRODUCT INTRODUCTION.....9

4.1 Product Description.....9
4.2 Computer Features.....9
4.3 Expansion Modules.....11

5.0 FIRST BOOT.....13

5.1 Power.....13
5.2 Terminal Connection.....13
5.3 Login.....13

6.0 LOADING NEW OS.....14

6.1 Required Components.....14
6.2 Procedure.....14

7.0 FILE TRANSFER.....16

7.1 PC to MDL File Transfer.....16
7.2 USB File Transfer.....16
7.3 Transfer Files Using Internet.....18

8.0 PYTHON DEVELOPMENT FOR MDL.....19

8.1 Installing a Python library on the MDL using PIP.....19
8.2 Developing for the MDL.....20

9.0 EXECUTING PYTHON ON MDL.....22

10.0 DYACON DATALOGGER CODE.....23

10.1 DataBear.....23
10.2 pyMDL.....23
10.3 Running Code on Start-up.....29

11.0 MDL SETTINGS AND COMMANDS.....30

11.1 LED Control.....30
11.2 Buttons.....30
11.3 Display.....31
11.4 OS Version.....31
11.5 Format Data Partition.....31
11.6 System Power Nets.....31

12.0 EMSP-1 – SERIAL PORT EXPANSION MODULE.....32

12.1 Features.....32
12.2 Connections.....32
12.3 Serial Port Expansion Module.....34

13.0 OS REVISION HISTORY.....37

14.0 DOCUMENT REVISION HISTORY.....38

1.0 Notices

1.1.1 © Copyright 2020 Dyacon, Inc.

1.1.2 Manufacturer

Dyacon, Inc.
1770 Research Park Way
Suite 168
Logan, UT 84341-1959
USA

1.1.3 Declarations

Dyacon WSD-1™, TPH-1™, LD-1™, GT-1™, and CM-1™ are low-power electronic industrial devices designed and manufactured by Dyacon.

RoHS

All electronic and mechanical components conform to RoHS, Directive 2002/95/EC.

FCC CFR Part 15

This equipment complies with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a commercial installation.

CE

**Preliminary statement ** Dyacon MDL-700 meets CE standards, complying with EN 61326-1 Industrial Immunity and EN55011 (CISPR 11) Class A Emissions.

1.1.4 Warranty Information

Limited Hardware Warranty

Dyacon, Inc. warrants that all Dyacon products and components shall be free from defects in materials and workmanship for a period of two (2) years from the date of shipment when installed according to instruction manuals accompanying said hardware and used for the purpose for which said hardware was designed. In the event a defect in materials or workmanship is discovered and reported to Dyacon within the warranty period, Dyacon will at its option repair the defect or replace the defective product. This warranty does not apply where the product has been operated outside the specifications of the product. Dyacon's obligation hereunder will be limited to repair or replacement of Dyacon equipment. Customers shall have the responsibility to ship the defective equipment to Dyacon at the customer's expense, with all cost of shipment prepaid. Dyacon will ship the repaired or replaced item at Dyacon's expense using the preferred shipment method of Dyacon. On-site warranty repair of equipment is provided at the discretion of Dyacon.

Disclaimer of Warranties

The warranties set forth above are in lieu of all other warranties of Dyacon, whether written, oral, or implied. Dyacon makes no warranties regarding its products (hardware or software), including without limitation warranties as to merchantability, fitness for a particular purpose, any warranty arising from course of performance, course of dealing or usage of trade whether any of the foregoing warranties are either expressed or implied. Dyacon specifically makes no warranties as to the suitability of its products for any particular application. Dyacon shall in no event be liable for performance, or use of any product covered by this agreement whether such claim is based upon warranty contract (express or implied), strict liability, negligence, or otherwise. Any responsibility and/or liability of Dyacon shall, in connection with a warranted product, be limited in maximum amount to the original purchase price of that product.

Removal of Serial Number

Removal of the original serial number label or reprogramming of the electronic serial number voids any warranty on the device. Dyacon will not repair or update products if the serial number label missing or legitimate ownership cannot be verified. Dyacon may not return equipment that is missing serial numbers or where legitimate ownership is in question.

Updates or Modifications

Dyacon shall be under no obligation to update or modify its products except as herein noted to correct defects or errors. Customer agrees that all representation and warranties contained herein shall be immediately null and void in the event of any incorrect installation, modification, alteration, or change in or to any product affected by or on behalf of customer except for a change made by Dyacon or other direct supervision thereof.

2.0 Introduction

2.1 Scope

This manual covers the hardware and operation of the Dyacon MDL-700 (modular data logger).

The MDL-700 is an industrial computer with a custom-built version of the Linux operating system. In its base version, the MDL simply provides an open-programmable platform for end users. Users must create or install software to perform data acquisition and storage.

It is assumed that users have experience interfacing to and/or programming Linux computers. The user is expected to be familiar with basic computer principles, computer architecture, data protocols, and Python or C programming. Consequently, the instructions contained in this manual may be insufficient for novice users. Future versions of the MDL platform may include a web-configuration interface or be supplied as turn-key systems. Additional documents and manuals may address software applications and development, including source code examples.

With the exception of simple examples, sensors and peripherals are not documented in this manual. Please refer to the respective manufactures for sensor operational details.

Dyacon cannot guarantee compatibility with all sensors, networks, peripherals, and third-party software. Users should evaluate the technical requirements of each component and exercise standard software development and testing practices.

2.2 Technical Support

Contact Information

Dyacon, Inc.

1770 Research Park Way
Suite 168
Logan, UT 84341-1959

Phone:	(435) 753-1002
Email:	support@dyacon.com
Internet:	www.Dyacon.com

Normal business hours are from 9:00 am to 5:00 pm. (Mountain Time Zone)

Phone / Email Support

If you need technical support via the phone or email, please have the following information ready:

Product name, model number, and serial number.
Your name and name of the purchaser of the equipment.
Name of company, institution, or agency.
Phone number, email address.
Billing and Shipping address.

A clear description of the question or problem.

3.0 Quick-Start Guide

A separate document covering power and connection requirements is pending.

4.0 Product Introduction

4.1 Product Description

The MDL-700^{TM} is a programmable Linux field computer with I/O and capabilities found in data loggers. It can be programmed using conventional tools such as Python and C. Depending on the required libraries and features, programs may be developed and complied directly on the data logger.

The use of common programming languages allows researchers, automation engineers, and software developers to program the MDL-700 using their existing skills, knowledge, and experience.

Developers may also find a wealth of source code examples, libraries, and support available in the public domain. Additional libraries and tools may be installed using standard Linux package management.

Standard PC I/O is accessible on the front panel of the computer along with a 64 x 128 OLED display.

Like other Dyacon products, sensors and power connections use pluggable terminal blocks. This provides excellent usability and versatility.

The MDL^{TM} uses expansion modules to customize the data logger to the specific application. Digital and analog sensor modules may be added to the right side of the computer board. Power input, solar charge controller, and relay control outputs may be attached to the left side of the computer board.

A cell phone and GPS module may be installed on the computer board. WiFi may be added via SD card or USB.

The tight integration and modularity of MDL-700 ensure a flexible platform for many commercial, industrial, and research applications.

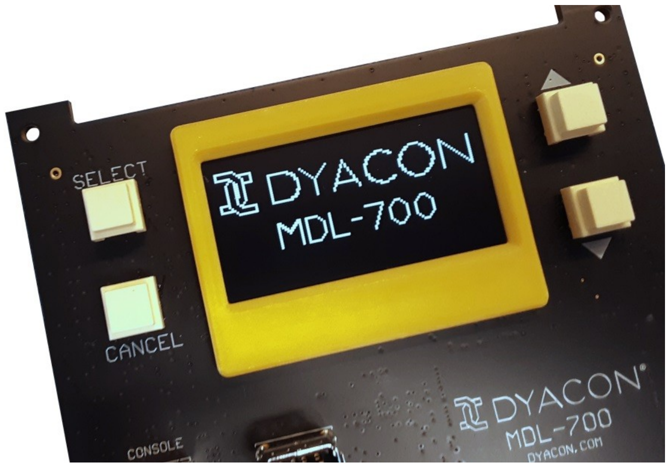
Custom-designed modules may be developed for integrators.

4.2 Computer Features

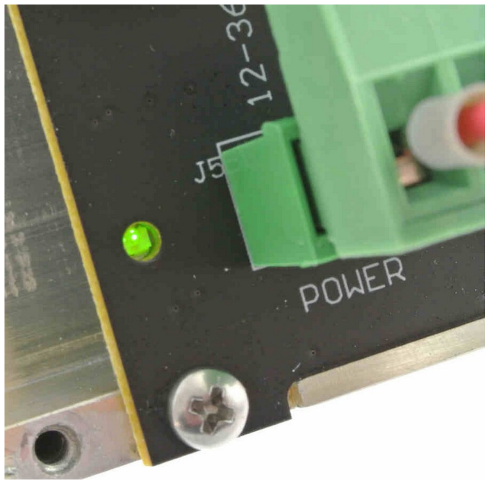
Front Panel

64x128 pixel graphic OLED display

4-button interface



Status LED



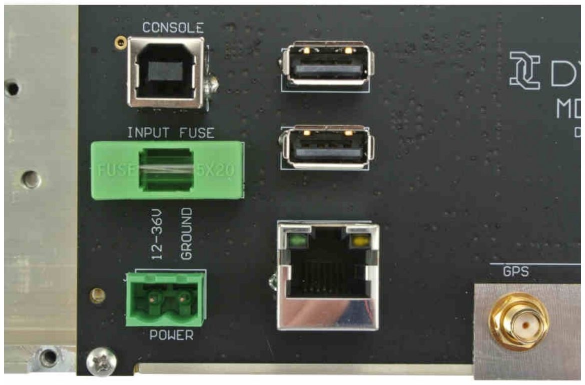
Ethernet port

USB-B device port for terminal connection

2x USB-A host ports for USB flash drive and other peripherals

Power input (12 VDC to 24 VDC)

User-serviceable fuse



SMA antenna connectors for cell phone and GPS receiver.

OS, Memory, and Processor

500 MHz, 32-bit ARM Cortex-A5

Linux 4.14 operating system – Check this, I think it might be 4.19

128 MB DDR2 SDRAM

4 GB eMMC flash – After our discussion I will look at making this larger on the next version.

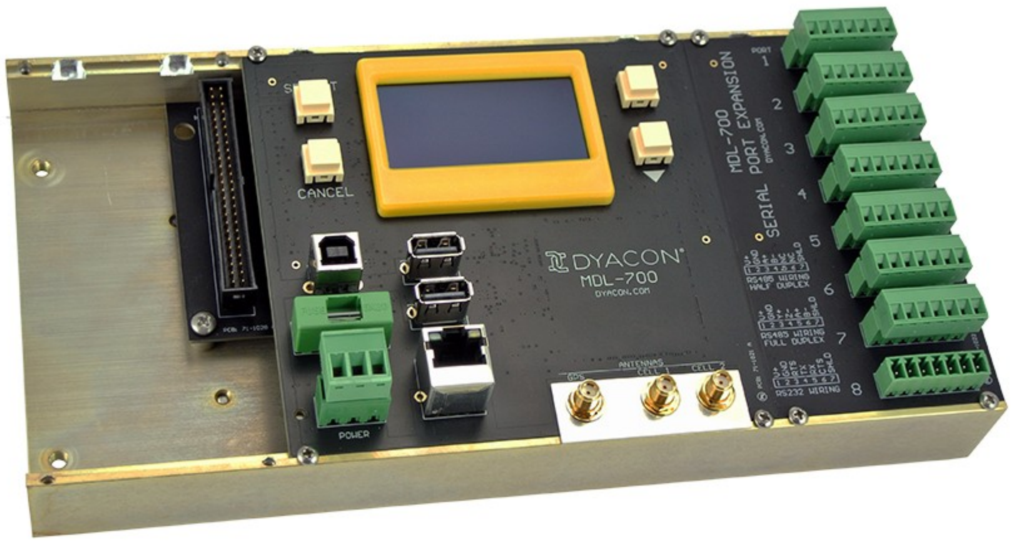
Real-time clock, battery-backed

4.3 Expansion Modules

MDL-700 expansion modules can be added to the left and right sides of the main computer board.

The left side is designed to accommodate power input and control output modules.

The right side modules are sensor inputs.



4.3.1 Power and Control Expansion Modules

The following modules are added to the left side of the computer board. A backplane board is required.

Multiple modules may be added.

Future Release

- Solar charge controller – 12 V to 36 V nominal solar panel input, maximum power point tracking (MPPT), 7 to 100 Ah battery capacity.

MDL has the capability for the following modules. These will be developed as needed.

- AC/DC power input converter.
- Control output – Mechanical relay or solid state relay.

4.3.2 Sensor Input Expansion Modules

Sensor modules are added to the right side of the computer board. A backplane board is required.

Multiple expansion modules may be added.

Integrators may prefer a custom solution in order to provide a mix of signals required for their unique application.

Currently Available

- EMSP-1 - Serial Port Expansion Module

  8x – Software selectable RS-232/422/485, half-duplex or full-duplex

  Ports 7 and 8 are SDI-12 compatible.

Future Release

- Analog Input, 8x – 24-bit ADC, single-ended, 11 VDC and 5 VDC reference
- Mixed Signal – 1x RS-485 (with 6 connectors), 1x SDI-12, 1x Pulse Counter, 4x 24-bit ADC, 1 x Frequency input, 2x Control output

5.0 First Boot

5.1 Power

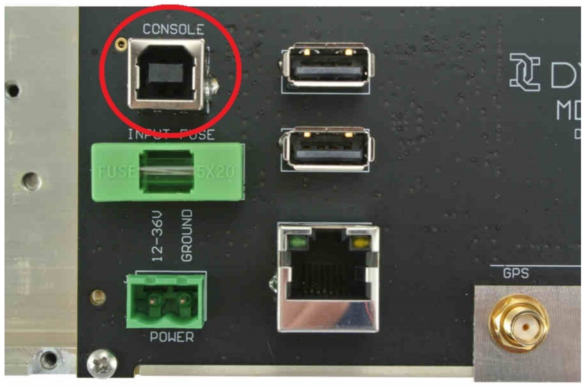
Depending on the configuration ordered, the MDL-700 requires 12 VDC to 24 VDC input (38 VDC max).

The computer itself draws less than 50 mA, depending on the state of the OLED display (more pixels = more power). Connected sensors may draw a higher amount of current.

A 12 VDC bench supply or wall adapter capable of 800 mA or more should be sufficient for most applications.

Using the supplied terminal block, connect 12 VDC to 24V DC VDC power.

5.2 Terminal Connection



A standard Linux terminal and command prompt is accessed through the front pane USB-B connector.

Using a standard terminal program connect with the following settings:

Baud rate: 115200

Data Bits: 8

Parity: None

Stop Bits: 1

Flow Control: None

System messages may be seen during boot and occasionally during normal operation.

In some cases, a serial connection may not be established initially. The USB cable may need to be unplugged and reinserted after both computers are booted and the terminal program is running.

5.3 Login

By default, there is no password for the root user.

Log in using the following.

```
User: root
```

6.0 Loading New OS

As Dyacon develops MDL, additional features may become available. The operating system can be upgraded using the following method.

6.1 Required Components

Loading a new operating system on the MDL requires the following:

- Windows personal computer
- Operating system files (typically a .zip file).
- USB A-B cable to connect the PC to the MDL console port.

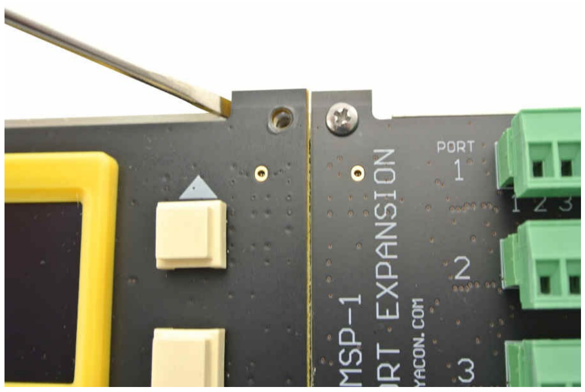
6.2 Procedure

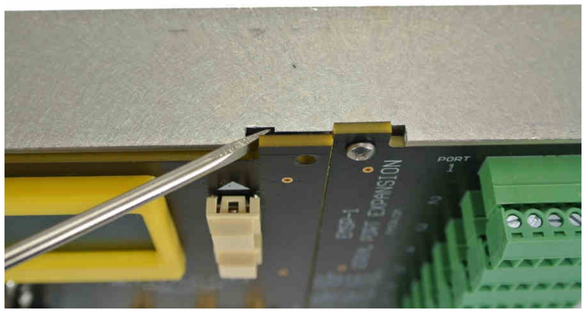
1. Unzip (uncompress) the OS files to a convenient folder on the PC.
2. Disconnect power from MDL.
3. Remove the MDL computer board from the enclosure.

   Remove the four (4) screws holding the computer board.

   Insert a screwdriver under the top tabs and gently pry the board up.

   Use the tabs with your fingers to slowly release the PCB from the backplane connectors.





4. Connect a USB cable from the PC to the USB-B connector on the MDL.
5. Apply power to MDL.
6. Use a terminal program to verify the MDL is operating normally. (See terminal port settings in the First Boot section.)
7. Start the boot loader.

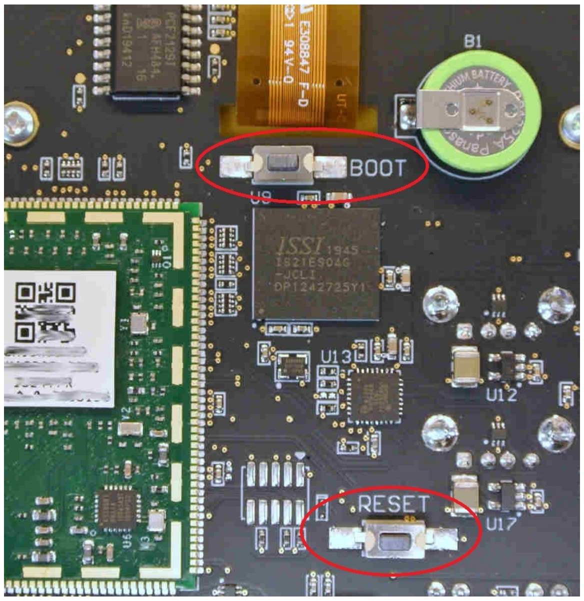
   With the MDL powered and connected to the PC, access the back of the MDL.

   Press and hold 'Boot' switch.

   Momentarily press 'Reset'.

   The terminal program should display "RomBOOT" on the command line.
8. After "RomBOOT" is displayed on the command line, release the Boot switch.

   This sequence may require a few attempts if the boot pins are not well connected.



9. Execute the 'Load emmc.bat' file by double clicking on the file or using a command window.

   Progress messages will show in the command window during the OS loading.
10. After the new OS has loaded, reconnect to the MDL with a terminal program and reset the MDL.
11. Re-install the computer board to the enclosure as soon as convenient.

7.0 File Transfer

The MDL is a Linux computer. As such, there are many different ways to interact with the device. The following are three methods for transferring files, such as programs and scripts, from the work station of the developer to the MDL.

7.1 PC to MDL File Transfer

Transferring files on to and off of MDL can be completed using the serial console port connection.

The following is a description using ZMODEM. The transfer protocol is used and initiated by the Linux commands ‘sz’ and ‘rz’.

TeraTerm is one terminal that supports ZMODEM protocol.

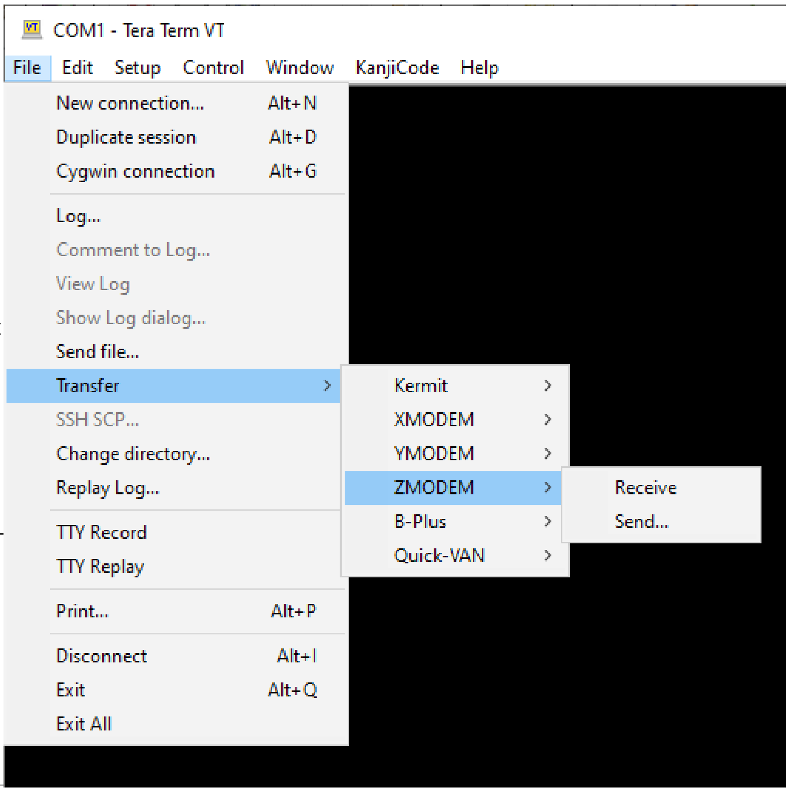
7.1.1 Receiving Files on MDL

1. Connect to the MDL via the console port.
2. Navigate to the destination directory on MDL.
3. On the MDL enter the receive command.

```
:/# rz
rz waiting to receive.**...
```

4. In TeraTerm, select File > Transfer > ZMODEM, and then select the file(s) to be sent.

After transfer the files will be located in the directory where ‘rz’ was run.

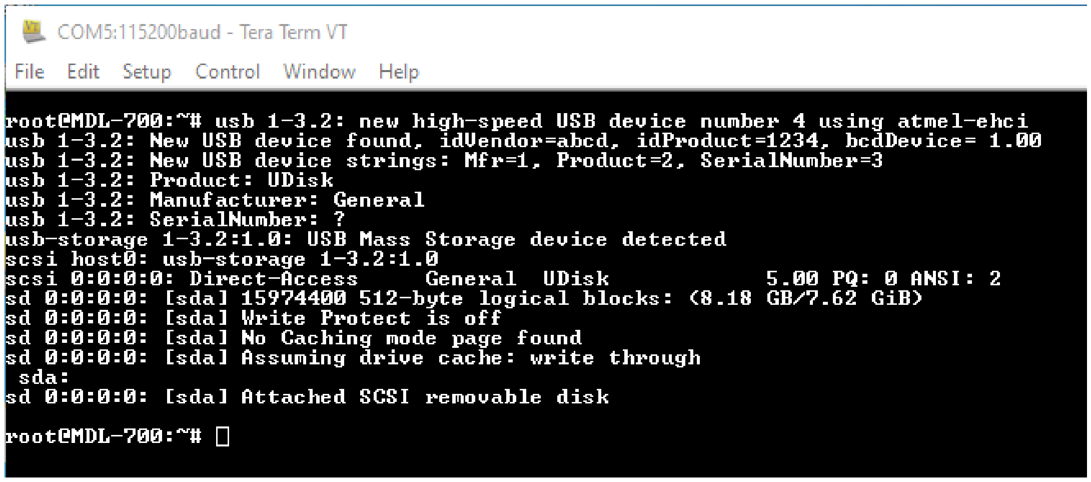


7.1.2 Send Files From

Follow a similar procedure to send a file to the PC, but use ‘sz’ on MDL.

7.2 USB File Transfer

The MDL has two USB ports that can be used to access a USB storage device. When a USB is connected to the port, the OS will automatically detect it and output information on the command line.


COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
root@MDL-700:~# usb 1-3.2: new high-speed USB device number 4 using atmel-ehci
usb 1-3.2: New USB device found, idVendor=abcd, idProduct=1234, bcdDevice= 1.00
usb 1-3.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-3.2: Product: UDisk
usb 1-3.2: Manufacturer: General
usb 1-3.2: SerialNumber: ?
usb-storage 1-3.2:1.0: USB Mass Storage device detected
scsi host0: usb-storage 1-3.2:1.0
scsi 0:0:0:0: Direct-Access     General  UDisk            5.00 PQ: 0 ANSI: 2
sd 0:0:0:0: [sda] 15974400 512-byte logical blocks: (8.18 GB/7.62 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] No Caching mode page found
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda:
sd 0:0:0:0: [sda] Attached SCSI removable disk

root@MDL-700:~# 
```

The drive can be accessed via the device name:

```
/dev/sd*n
```

where:

`'*'` is a, b, c, ... depending on how many devices are connected.

`'n'` is a number of partitions. Typically, flash drives do not contain partitions and will not have a number in this position.

In the picture above, the drive is assigned ‘a’, [sda]. A second drive would typically be assigned [sdb].

You can list the connected drives with the following command.

```
:/# ls /dev/sd*
/dev/sda /dev/sdb
```

Files on the drive are accessed by mounting the drive and then navigating through the drives file system.

7.2.1 Mounting and Transferring a File

1. Plug in the USB to any open USB port.
2. Make a directory, a mount point, for the device. This directory may be at the home, root, or other location of your choice. The following creates a ‘mydrive’ directory at the home

   ```
   :/# mkdir ~/mydrive
   ```
3. Assuming the device is already formatted, mount the device via.

   ```
   :/# mount -t <type> <devicename> <mountpoint>
   ```

where:

`'type'` is the file system on the device, typically vfat. Type is automatically detected and does not need to be explicitly given.

`'devicename'` would be sda or sdb.

`'mountpoint'` is the directory created in the previous step

```
:# mount -t sda mydrive
```

4. Navigate as needed to the file after the device is mounted.

```
:/# cd ~/mydrive
```

5. Copy the file to the MDL

```
:/# cp ~/mydrive/<file> <destination path>
```

Example:

```
:/# cp ~/mydrive/logger1.2 /home/logger/logger_app
```

6. Unmount the drive before removing it to ensure all data is synced.

```
:# umount mydrive
```

7.3 Transfer Files Using Internet

Files can also be transferred to the MDL using internet protocols. When using Python for development, a typical approach would be to store the scripts in a repository like Github or Bitbucket. Repositories can be cloned using Git, which is pre-installed on the MDL. Python package manager, PIP, can also be used for packages stored on PYPI (pypi.org).

8.0 Python Development for MDL

This section give a brief introduction to the Python programming language and how it can be used on the Dyacon MDL-700™.

One of the many advantages of the MDL is that it is an open platform. Whereas other data loggers use proprietary languages and scripts, MDL may be programmed using common programming tools and languages. Python is common in academic and commercial environments. (By the way, “Python” is a reference to the British comedian Monty Python, not the reptile.)

Python is an interpreted language. This means that it is processed through a program that reads the text in the program file, a compiler is not used. Interpreted languages separate the program from the machine on which it is executed. This makes Python relatively fast to learn, easy to use, and it protects the system from programmer errors. Ideally, the same interpreter layer allows Python programs to be run on multiple operating systems and platforms with minimal changes to the code.

MDL is pre-installed with Python 3.7.3 and Numpy 1.16.3. Many other libraries can be installed using PIP, Python’s package manager. However, some Python libraries, including popular libraries like SciPy and Matplotlib, contain C-code that is compiled for a particular operating system and these will not work on the MDL unless built from source. (Which is beyond the scope of this document.)

8.1 Installing a Python library on the MDL using PIP

PIP is an automatic package or software manager used by Python. It will find, download, and install Python packages automatically.

MDL must be connected to Ethernet in order for PIP to function. Cell connection should also work, but may incur data charges.

8.1.1 Install a Python Package on The MDL

Using a terminal connection with MDL, execute the following command.

```
:/# pip install <package>
```

This will download the most recent version of the package from pypi.org and install it.

PIP will also install any necessary dependencies for a particular library. For example, the Modbus library MinimalModbus depends on the PySerial library.

Experienced users can specify a particular version if needed.

8.2 Developing for the MDL

There are a number of different approaches to developing Python code for the MDL. Each programmer will find tools and methods that fit their abilities and needs. The following is one method that we hope will get you started.

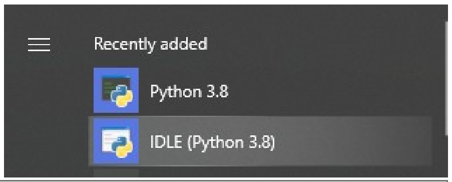
8.2.1 Install Python's IDLE

IDLE is Python's Integrated Development and Learning Environment. IDLE includes a text editor with context highlighting and allows the developer to execute the program from the same window.

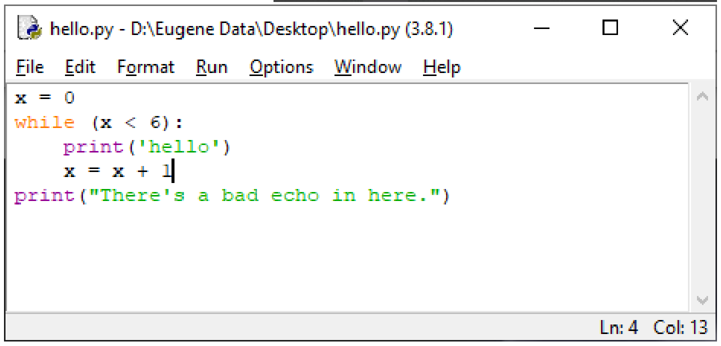
Install Python to your Windows computer from Microsoft Store. Just search for Python. Version 3.8 is the latest, but 3.7 will work too.

The latest version available from Python.org may be newer than the version on MDL. This will not typically be a problem, unless developing with the latest functions.

Once the Windows installer completes, IDLE may be started by selecting it from the Start menu program list.

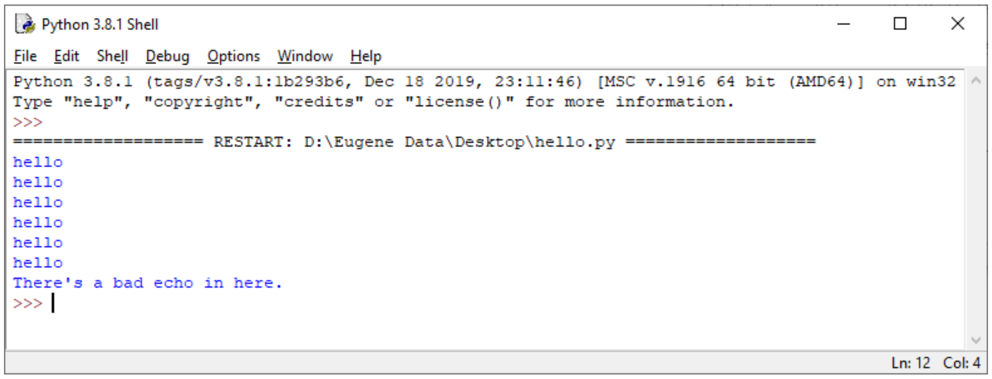


The IDLE is a context-aware Python code editor. This means that reserved words and commands are highlighted.


hello.py - D:\Eugene Data\Desktop\hello.py (3.8.1)
File Edit Format Run Options Window Help
Ln: 4 Col: 13

```
x = 0
while (x < 6):
    print('hello')
    x = x + 1
print("There's a bad echo in here.")
```

Pressing F5 short cut (or select Run) will start the Python shell and execute the program.



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: D:\Eugene Data\Desktop\hello.py ===================
hello
hello
hello
hello
hello
hello
There's a bad echo in here.
>>> 
```

New users of Python may be interested in the following.

Python Tutorial: https::/#docs.python.org/3.8/tutorial/index.html

Python Documentation: https::/#docs.python.org/3.8/index.html

Python is a cross-platform language, so code developed on Windows should run on the MDL (Linux). Some Python functions are specific to a particular platform, so be aware that you may need to change some of your code after you move it to the MDL if it doesn't run.

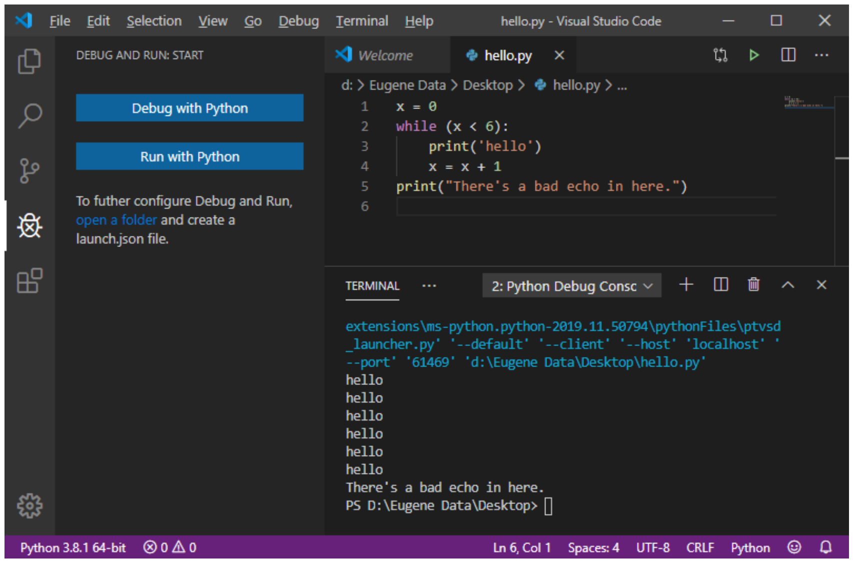
Serial ports will always be different on different platforms, so these will likely need to be changed when moving your code from your development PC to MDL.

Review any error messages output by Python for clues to what is causing a failure.

8.2.2 Code Editors

Python may be developed using a simple text editor or a full interactive development environment like Visual Studio Code (https://code.visualstudio.com/).

A plugin allows code to be executed inside of the editor.


File Edit Selection View Go Debug Terminal Help
hello.py - Visual Studio Code
DEBUG AND RUN: START
Debug with Python
Run with Python
Welcome
hello.py
d: > Eugene Data > Desktop > hello.py > ...
TERMINAL
2: Python Debug Consc
Python 3.8.1 64-bit ⊗ 0 ⚠ 0
Ln 6, Col 1 Spaces: 4 UTF-8 CRLF Python

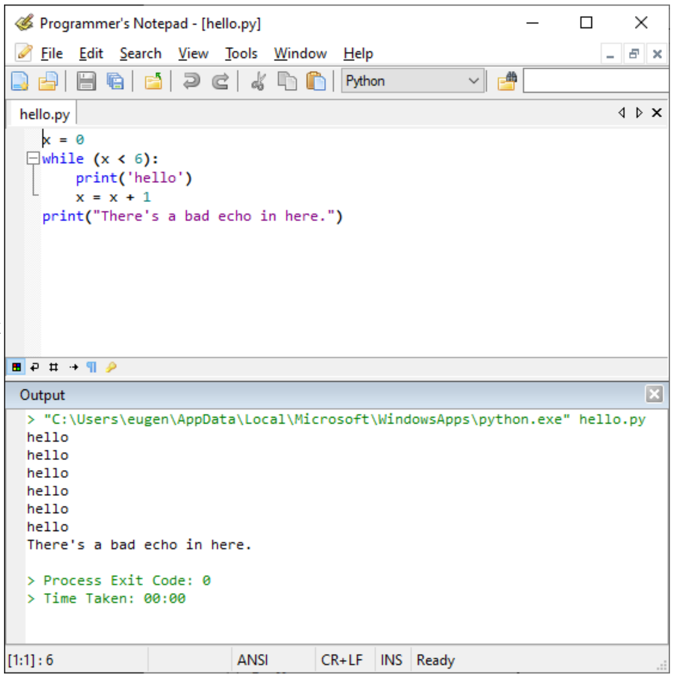
To futher configure Debug and Run, open a folder and create a launch.json file.

```
x = 0
while (x < 6):
    print('hello')
    x = x + 1
print("There's a bad echo in here.")
```

```
extensions\ms-python.python-2019.11.50794\pythonFiles\ptvsd
_launcher.py' '--default' '--client' '--host' 'localhost' '
--port' '61469' 'd:\Eugene Data\Desktop\hello.py'
hello
hello
hello
hello
hello
hello
There's a bad echo in here.
PS D:\Eugene Data\Desktop>
```

Programmer's Notepad (http://www.pnotepad.org/) is a language-aware text editor that can be configured to execute the Python interpreter without leaving the editor.

Notepad++ (https://notepad-plus-plus.org/) is example of an editor that can be configured to run Python IDLE on the active file.


Programmer's Notepad - [hello.py]
File Edit Search View Tools Window Help
Python
hello.py
Output
[1:1] : 6
ANSI
CR+LF
INS
Ready

```
x = 0
while (x < 6):
    print('hello')
    x = x + 1
print("There's a bad echo in here.")
```

```
> "C:\Users\eugen\AppData\Local\Microsoft\WindowsApps\python.exe" hello.py
hello
hello
hello
hello
hello
hello
There's a bad echo in here.

> Process Exit Code: 0
> Time Taken: 00:00
```

8.2.3 Moving Code to MDL

For convenience, Python can be developed on your personal computer.

When your code is ready, move it to the MDL using ZMODEM, a USB drive, or by packaging it on a repository that can be accessed via PIP and/or Git.

See the File Transfer section of this document for more information.

9.0 Executing Python on MDL

Execute Python code on the MDL just as you would on a personal computer.

```
:/# python <script.py>
```

10.0 Dyacon Datalogger Code

While the MDL can be used in a variety of contexts, the primary functionality is that of a data logger. As such, Dyacon is developing versatile data logger software, primarily using Python. This software is open source and can be extended and changed by the user as needed.

MDL datalogger software currently consists of two independent projects. The first project is DataBear (https://github.com/chrisrycx/databear), which implements core data logger functionality and is non-platform specific. The second project, pyMDL, utilizes DataBear while performing MDL specific configuration tasks.

This section is a brief introduction to both projects.. Full code documentation, including source code is available on GitHub.

10.1 DataBear

Platform agnostic Python data logger code, DataBear, is an open-source project that is compatible with the Linux operating system, including project boards such as Raspberry Pi and BeagleBone. Currently implemented features include basic scheduled measurement and data storage, simple processing of data samples, and output of data to a CSV. Datalogger operation is configured via a YAML file. Sensor operation is defined using a standardize interface.

DataBear is pre-installed on the MDL.

Installation

```
:/# pip install databear
```

Source Code - Github

```
https://github.com/chrisrycx/databear
```

10.2 pyMDL

pyMDL is a Python based open-source program that provides an MDL specific hardware adaptation for DataBear. pyMDL simplifies hardware configuration (GPIO settings) and will eventually facilitate use of the MDL display. pyMDL is pre-installed on the MDL.

Installation

pyMDL currently cannot be installed using PIP and a name conflict on PYPI may force a change in the project name at some point. For now, install pyMDL by cloning the repository using Git.

Source

```
https://github.com/dyacon/pyMDL
```

10.2.1 Example

NEW EXAMPLE

This section provides step by step instructions for setting up a new MDL to make serial port measurements using pyMDL and DataBear. This example uses Dyacon TPH-1 and WSD-1 sensors, which are pre-defined in DataBear. In general, a sensor interface will need to be developed. See DataBear documentation for more information.

1. Initial Setup:

   This step will typically be completed by Dyacon, but is included here for clarity.

   a) Create a data logger directory on the MDL to hold scripts, configuration, data, and log files for this example.

   ```
   :/# mkdir mylogger
   ```

   b) Install DataBear and minimalmodbus

   ```
   :/# pip install databear minimalmodbus
   ```

   c) Clone the pyMDL repository to a convenient directory

   ```
   :/# git clone https://github.com/dyacon/pyMDL.git
   ```

   d) Change directory to pyMDL and install the pyMDL package. Using the ‘-e’ option will create an “editable” install, which will be useful if any core functionality of pyMDL is customized.

   ```
   :/pyMDL/# pip install -e .
   ```

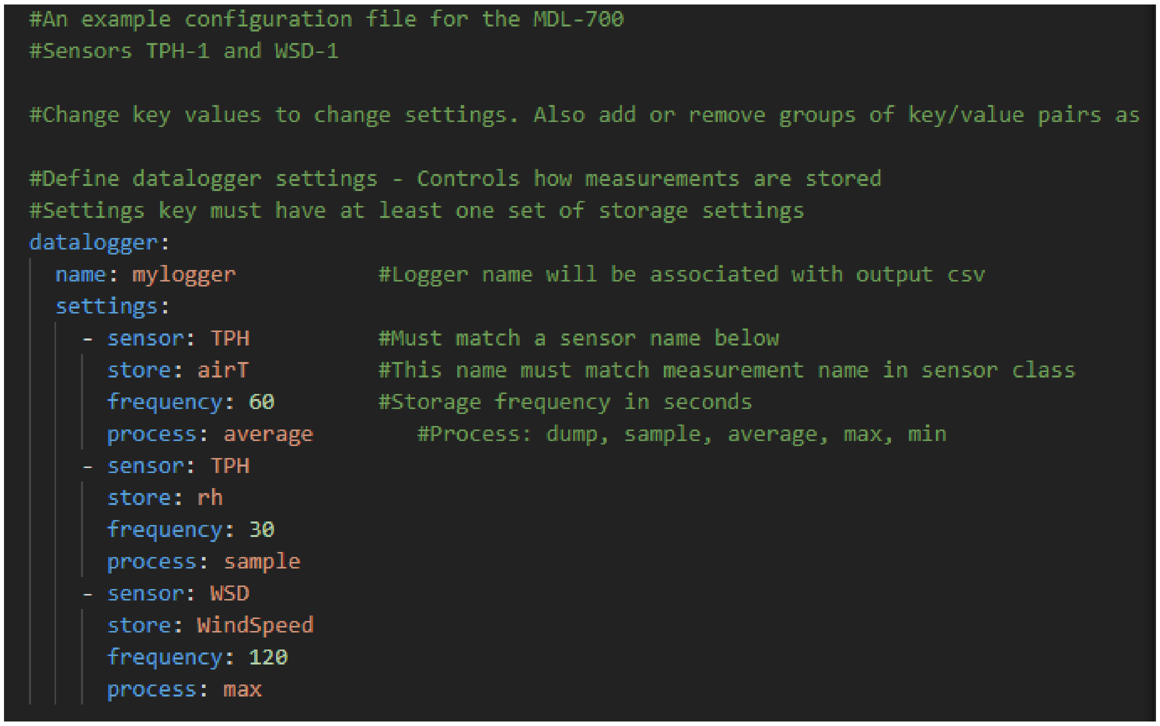
2. Create a configuration YAML file for the desired settings

   The tables below detail how measurements are performed and are arbitrary for this example.

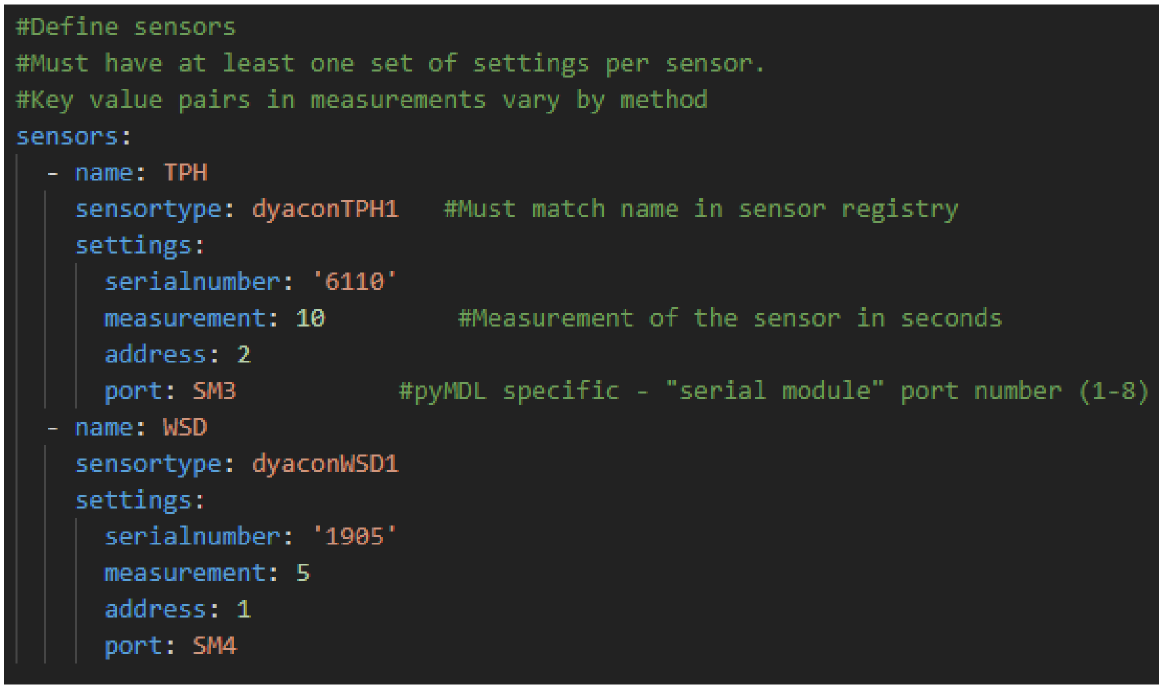
Sensor	Measurement Interval
TPH - 1	10 seconds
WSD - 1	5 seconds

Measurement	Storage Processing	Storage Interval
Air Temperature	Average	1 min
Relative Humidity	Sample	30 seconds
Wind speed	Max	2 min

   a) In the data logger directory, create the following YAML file. Use example files in the pyMDL folder as a template.

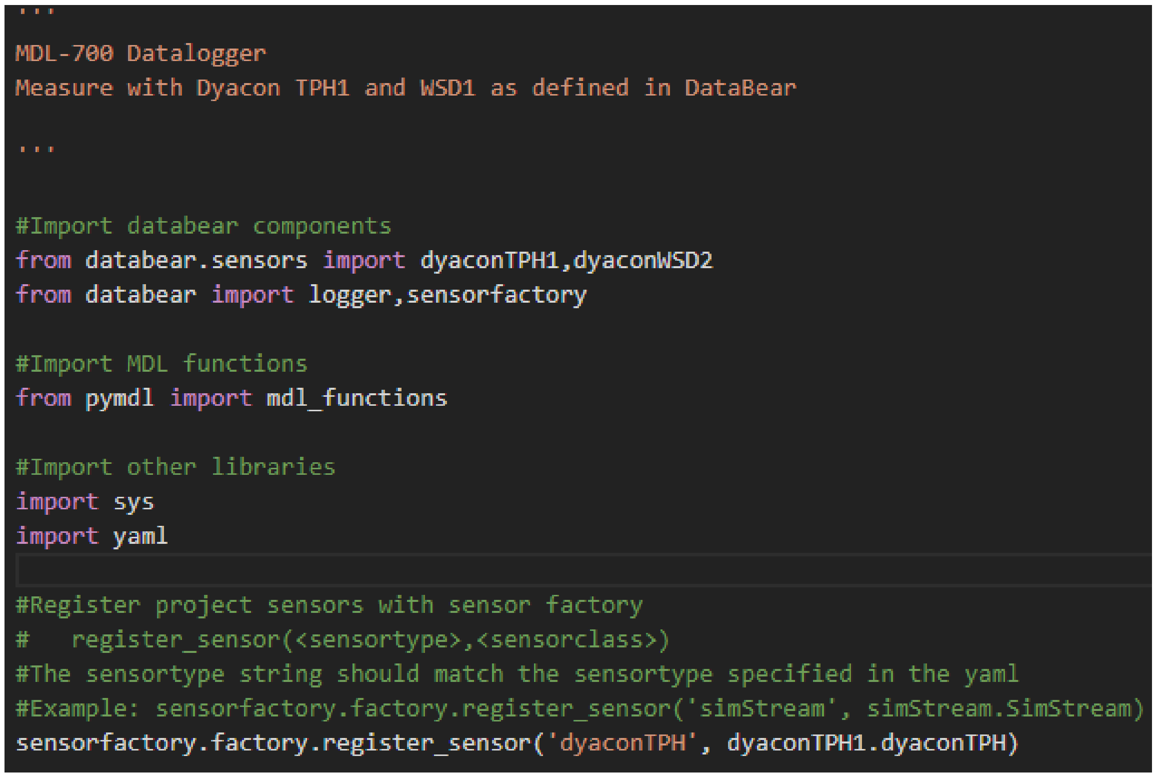


```
#An example configuration file for the MDL-700
#Sensors TPH-1 and WSD-1

#Change key values to change settings. Also add or remove groups of key/value pairs as

#Define datalogger settings - Controls how measurements are stored
#Settings key must have at least one set of storage settings
datalogger:
  name: mylogger           #Logger name will be associated with output csv
  settings:
    - sensor: TPH          #Must match a sensor name below
      store: airT          #This name must match measurement name in sensor class
      frequency: 60        #Storage frequency in seconds
      process: average        #Process: dump, sample, average, max, min
    - sensor: TPH
      store: rh
      frequency: 30
      process: sample
    - sensor: WSD
      store: WindSpeed
      frequency: 120
      process: max
```



```
#Define sensors
#Must have at least one set of settings per sensor.
#Key value pairs in measurements vary by method
sensors:
  - name: TPH
    sensortype: dyaconTPH1   #Must match name in sensor registry
    settings:
      serialnumber: '6110'
      measurement: 10         #Measurement of the sensor in seconds
      address: 2
      port: SM3           #pyMDL specific - "serial module" port number (1-8)
  - name: WSD
    sensortype: dyaconWSD1
    settings:
      serialnumber: '1905'
      measurement: 5
      address: 1
      port: SM4
```

b) In the data logger directory, create a python script to run the datalogger. Use example in pyMDL directory as a template. Most of this script consists of content from the template with only changes to the top sections.



```
'''
MDL-700 Datalogger
Measure with Dyacon TPH1 and WSD1 as defined in DataBear

'''

#Import databear components
from databear.sensors import dyaconTPH1,dyaconWSD2
from databear import logger,sensorfactory

#Import MDL functions
from pymdl import mdl_functions

#Import other libraries
import sys
import yaml

#Register project sensors with sensor factory
#   register_sensor(<sensortype>,<sensorclass>)
#The sensortype string should match the sensortype specified in the yaml
#Example: sensorfactory.factory.register_sensor('simStream', simStream.SimStream)
sensorfactory.factory.register_sensor('dyaconTPH', dyaconTPH1.dyaconTPH)
```

- The only changes that need to be made to the script consist of importing the needed sensor class from DataBear or a custom script. After import, the class must be registered with the "sensor factory".

3. Run the datalogger from the command line. Data will be saved to a CSV in the datalogger directory.

```
:/# python path/to/mylogger.py path/to/config.yaml
```

```
***** Delete prior example?
```

This section provides step by step instructions for setting up a new MDL with pyMDL and making serial port measurements. In this example, two different sensors are utilized: a Dyacon TPH-1 sensor and an RM Young barometric pressure sensor. The TPH sensor uses Modbus communication protocols and the RM Young sensor streams data at a frequency of 1 measurement per second. In order to demonstrate configuration flexibility in pyMDL, each measurement is configured to occur at different, arbitrarily chosen rates (table x). Data storage is also arbitrarily set to every 30 seconds for each sensor, but this could be changed to any value.

Sensor	Measurement	Frequency	Storage Frequency
TPH-1 (Modbus)	Air Temperature	10 sec	30 sec
TPH-1 (Modbus)	Relative Humidity	15 sec	30 sec
RM Young (Streaming)	Barometric Pressure	5 sec	30 sec

Arbitrary measurement configuration for example.

1. Obtain a copy of pyMDL from Dyacon.

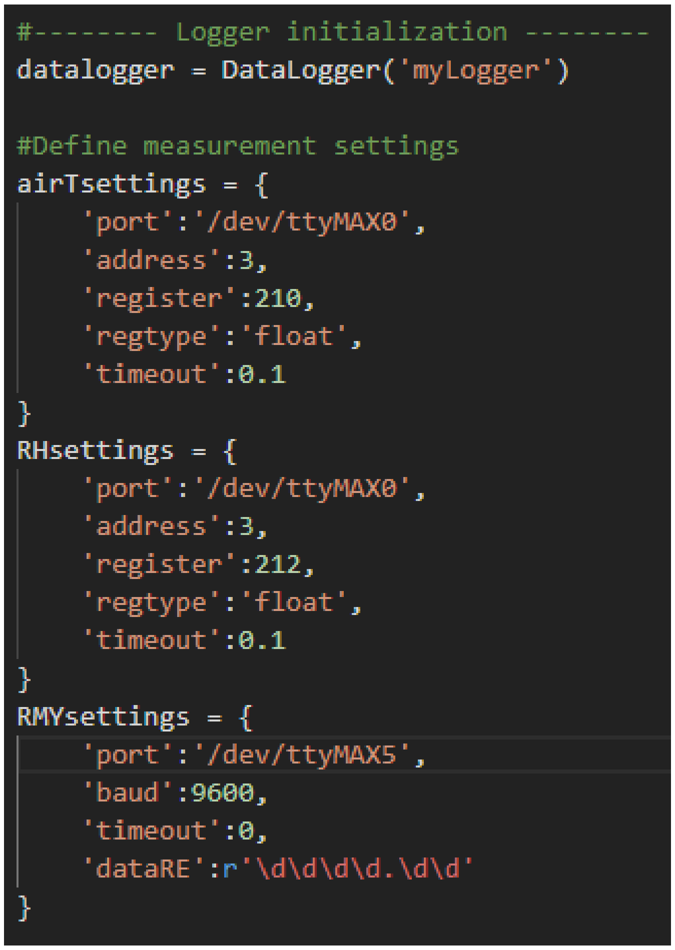
   This can be accomplished using Git or via a download from the pyMDL repository. Contact Dyacon for more information.

2. Configure pyMDL for the desired measurement settings by editing the 'logger.py' script. In this case, the configuration will be based on Table X.

a) Select a name for the data logger and input that name into the section of code where the ‘datalogger’ object is instantiated (see figure below). Data is stored to a CSV with the same name as the data logger.

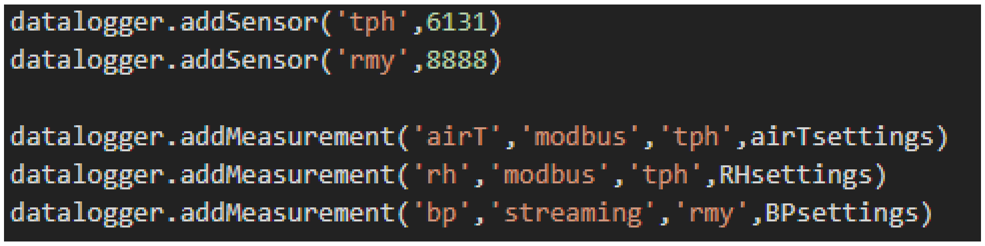
b) Next create a dictionary of settings for each measurement:

c) Table X has three measurements, so three dictionaries are created. The settings (key:value pairs) must conform with the requirements of the measurement methods defined in ‘sensor.py’. The “RHsettings” and “airTsettings” shown below are associated with a Modbus measurement and have components like address and register. “BPsettings” are associated with a streaming serial measurement. Review measurement method code in ‘sensor.py’ to determine what settings are needed for a particular measurement. Note that the ports in these settings must correspond to the serial port on the MDL where the corresponding sensor is connected.



```
#-------- Logger initialization --------
datalogger = DataLogger('myLogger')

#Define measurement settings
airTsettings = {
    'port':'/dev/ttyMAX0',
    'address':3,
    'register':210,
    'regtype':'float',
    'timeout':0.1
}
RHsettings = {
    'port':'/dev/ttyMAX0',
    'address':3,
    'register':212,
    'regtype':'float',
    'timeout':0.1
}
RMYsettings = {
    'port':'/dev/ttyMAX5',
    'baud':9600,
    'timeout':0,
    'dataRE':r'\d\d\d\d.\d\d'
}
```

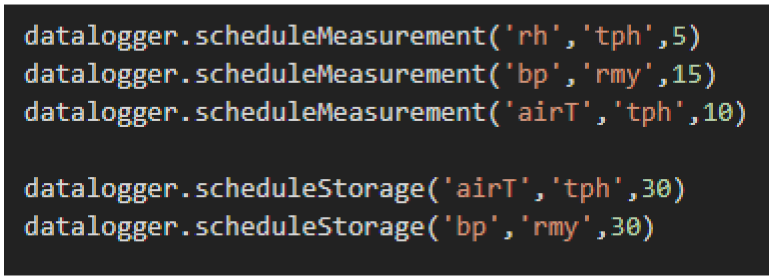
d) Next add sensors and measurements to the datalogger object:



```
datalogger.addSensor('tph',6131)
datalogger.addSensor('rmy',8888)

datalogger.addMeasurement('airT','modbus','tph',airTsettings)
datalogger.addMeasurement('rh','modbus','tph',RHsettings)
datalogger.addMeasurement('bp','streaming','rmy',BPsettings)
```

This code configures the data logger for two different sensors and three different measurements.

e) Configure the logger to measure and store using the “scheduleMeasurement” and “scheduleStorage” methods.



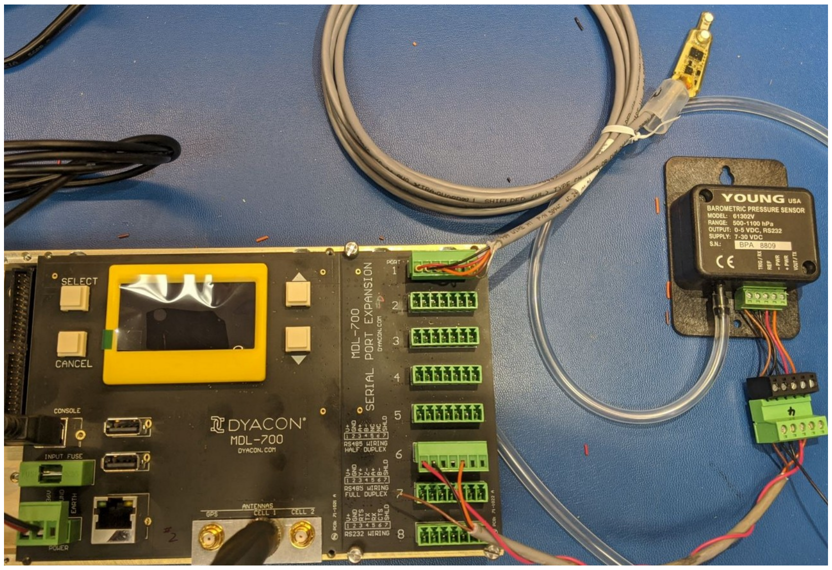
```
datalogger.scheduleMeasurement('rh','tph',5)
datalogger.scheduleMeasurement('bp','rmy',15)
datalogger.scheduleMeasurement('airT','tph',10)

datalogger.scheduleStorage('airT','tph',30)
datalogger.scheduleStorage('bp','rmy',30)
```

Here the logger will measure every 5 to 15 seconds depending on the measurement and store data every 30 seconds. The data stored will simply be the latest sample as other data processing functions like 'average' have not been implemented in V0.

3. Copy the edited logger, sensor, and schedule.py scripts to the MDL using a method described in section 6 of this manual.
4. Ensure any Python dependencies associated with the logger code are present on the MDL. The command "pip list" will list the Python libraries currently installed. Modbus and serial communications require "pySerial" and "minimalmodbus". Both can be installed (using Ethernet or Cell) via:

   ```
   >> pip install minimalmodbus
   ```

   Since minimalmobus depends on pySerial, pip will install both libraries.
5. Plug the sensors into an open serial port.

   Here the TPH-1 is plugged into port 1, following the wiring diagram for RS485 half-duplex. The RM Young sensor is plugged into port 6 following the diagram for RS232 sensors. The ports used in this example are randomly chosen.
6. The final step before running the data logger program is configuration of the serial ports for the type of serial communication used by the sensor. This is done using GPIO commands described in section 10. The GPIO commands for the sensors and ports used here are:

   ```
   >> gpioset gpiochip1 0=1 1=1 2=1 3=1
   ```

   ```
   >> gpioset gpiochip2 4=0 5=0 6=0 7=0
   ```
7. Run the data logger program:

   ```
   >> python logger.py
   ```

   The data logger will begin measurement and storage at the top of the next minute and will output the current measurement to the command line. To stop program execution use Ctrl-C.

10.3 Running Code on Start-up

An application, including Python code, can be configured to run when the MDL is powered up. This eliminates the need to use the console port to access the MDL command line to run the logger. Once configured, the MDL easily be deployed by installing it in a field location and simply turning it on.

1. Edit the file /app/appRun using a text editor like 'vim'.
2. Add the command for running the code to the file under section XXX
   - Picture
   - Example: python /path/to/code.py

This method will not restart the code if it unexpectedly exits. To continuously monitor and restart an application, use the "inittab" approach... **Not sure how much to elaborate.

11.0 MDL Settings and Commands

The Linux operating system is compiled specifically for the MDL. The computer hardware and expansion modules have unique features that support data logging activities.

The following are operating system functions that configure or access the unique features of the computer board and some of the common expansion modules.

11.1 LED Control

LED is controlled by the operating system. There is no application control of the LEDs.

11.2 Buttons

The buttons are mapped to the following.

```
/dev/input/event0
```

This is the standard keyboard input and is accessible using the Python `input( )` function.

Button Label	Event Code	Key Name
Select	28	Key_Enter
Cancel	1	Key_Esc
Up	103	Key_Up
Down	108	Key_Down

Key presses generate both EV_SYN and EV_KEY events.

When connected to a terminal screen, the following command will display the button presses.

```
evtest /dev/input/event0
```

11.3 Display

The MDL includes a 128 x 64 pixel OLED white on black display. The display is controlled using a “frame buffer” which is simply a memory segment where each bit in memory corresponds to a display pixel. Bytes are written to the frame buffer from left to right and row by row. The frame buffer has to total of 1024 bytes (16 bytes per row times 64 rows).

Device file: /dev/fb0

Example:

- Writing \x0f (0000 1111) would result in the first 4 pixels in the display black and the subsequent 4 white.
- `cat /etc/splash > /dev/fb0`
  - This command will show the Dyacon MDL splash screen seen on start up by writing the data in the file to the frame buffer.

Python can be used to control the display by opening and writing to the device file. The Python library Pillow (https://pillow.readthedocs.io/en/stable/) can be used to generate images and convert them to bytes.

11.4 OS Version

OS build version can be displayed with the following:

```
cat /etc/buildinfo
```

11.5 Format Data Partition

When installing an OS upgrade, it may be necessary to format the data partition of the flash memory system.

```
mkfs.ext4 /dev/mmcblk0p8
```

11.6 System Power Nets

MDL-700 has the capability to monitor internal and external power net voltage and current. The values are displayed using a bash script.

Whether during bench testing or programatically in the field, monitoring power usage will help with power budgeting and system monitoring.

From the root director:

```
./displayPower
```

Or,

```
bash displayPower
```

12.0 EMSP-1 – Serial Port Expansion Module

Expansion modules may be purchased with MDL-700 or separately.

EMSP-1 is a common module for reading multiple digital sensors.

12.1 Features

8x Multi-protocol serial ports.

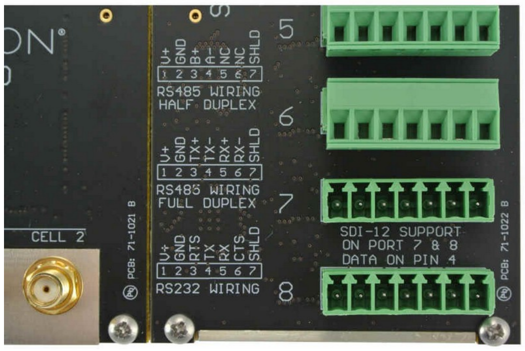
One pluggable connector for each serial port.

Individually software configurable as RS-232, 2-wire RS-485, 4-wire RS-485.

Ports 7 and 8 are SDI-12 signal level compatible (5 V).

12 VDC nominal (11.6 VDC regulated) and ground on each serial port connector.

Total power monitoring, both current and voltage.



12.2 Connections

12.2.1 Ports 1 Through 6

Each port uses an individual UART and multi-protocol transceiver.

All ports are discretely configured, allowing for a mix of sensor types, whether RS-485 2-wire, RS-485 4-wire or RS-232.

Modbus RTU or Modbus ASCII protocols may be implemented at the software layer.

Bias and termination resistors for RS-485 modes may be enabled using software instructions.

Pin-out

Pin	Label	Function	Notes
1	+V	Power 12 VDC (11.6 V regulated)	Shared power net with all serial ports. Total current limited to 100 mA.
2	Gnd	Power ground.	
3	B+ TX+ RTS	RS-485 2-wire: B+ RS-485 4-wire: TX+ RS-232: Ready to send	Software selectable electrical protocol between three modes. Software enabled RS-485 bias and termination resistors.
4	A- TX- TX	RS-485 2-wire: A- RS-485 4-wire: TX- RS-232: Transmit	
5	NC RX+ RX	RS-485 2-wire: No connect RS-485 4-wire: RX+ RS-232: Receive	
6	NC RX- CTS	RS-485 2-wire: No connect RS-485 4-wire: TX+ RS-232: Clear to send	
7	Shield	Earth ground	

12.2.2 Ports 7 and 8

These ports are identical to 1 through 6, differing only in the 5 V logic level used for the transceivers. This makes the ports compatible with SDI-12 signal levels.

Ports utilize standard Linux serial port functions. This may result in some timing irregularities that may require some software adaptation for some sensors. Dyacon Python data logger project pyMDL has been successfully tested on several SDI-12 sensors.

Pin-out

Pin	Label	Function	Notes
1	+V	Power 12 VDC (11.6 V regulated)	Shared power net with all serial ports. Total current limited to 100 mA.
2	Gnd	Power ground.	
3	B+ TX+ RTS	RS-485 2-wire: B+ RS-485 4-wire: TX+ RS-232: Ready to send	Software selectable electrical protocol between three modes. Software enabled RS-485 bias and termination resistors.
4	A- TX- TX Data	RS-485 2-wire: A- RS-485 4-wire: TX- RS-232: Transmit SDI-12: Data	See SDI-12 description above this table.
5	NC RX+ RX	RS-485 2-wire: No connect RS-485 4-wire: RX+ RS-232: Receive	
6	NC RX- CTS	RS-485 2-wire: No connect RS-485 4-wire: TX+ RS-232: Clear to send	
7	Shield	Earth ground	

12.3 Serial Port Expansion Module

The 8-port serial expansion module is one of the common options. Each port can be individually configured for RS-485 or RS-232 communications. Additionally, RS-485 mode can be configured for full or half-duplex (4- or 2-wire), termination resistors enabled, and voltage bias resistors set.

On start up, the default script (/etc/init.d/S01serialmodule) sets all ports to full duplex RS-485.

12.3.1 Port Configuration

Serial ports can be configured using GPIO commands either within the start up script or at the command line.

```
gpioset gpiochip[UART_Address] [Type_Address]=[Type_Mode]
                                [Duplex_Address]=[Duplex_Mode]
                                [Termination_Address]=[Term_Enable]
                                [Bias_Address]=[Bias_Enable]

gpioset gpiochip[1|2] [0|4|8|12]=[1|0]
                      [1|5|9|13]=[1|0]
                      [2|6|10|14]=[1|0]
                      [3|7|11|15]=[1|0]
```

Examples

Configure port 4 as half-duplex RS-485 with termination resistor and bias enabled:

```
:/#gpioset gpiochip1 12=1 13=1 14=1 15=1
```

Connect to a port via '/dev/ttyMAXn' where n is the port number minus 1.

```
    :/#/dev/ttyMAX3
```

Parameter	Description	Values
UART_Address (UA)	Identifies one of two quad UARTS. The first UART serves ports 1-4 and the second UART serves ports 5-8.	0 – Address settings for ports 1-4 1 – Address settings for ports 5-8 Shown as UAn below. (Second serial port board would use UA2 and UA3.)
Type_Address	Identifies the particular port where the electrical protocol type is set.	0 – Port 1 for UA0 or 5 for UA1 4 – Port 2 or 6 8 – Port 3 or 7 12 – Port 4 or 8
Type_Mode	Sets the transceiver mode as either RS-232 or RS-485	0 – RS-232 1 – RS-485
Duplex_Address	Identifies the port for the setting of the duplex mode.	1 – Port 1 or 5 5 – port 2 or 6 9 – Port 3 or 7 13 – Port 4 or 8
Duplex_Mode	In RS485 mode, the transceiver can operate as either half-duplex (2-wire) or full-duplex (4-wire).	0 – Full-duplex RS-485 1 – Half-duplex RS-485
Termination_Address	Identifies the port address for setting the termination resistor when in RS-485 mode.	2 – Port 1 or 5 6 – port 2 or 6 10 – Port 3 or 7 14 – Port 4 or 8
Term_Enable	Enables or disables the termination resistor for RS-485 mode. This should be coordinated with other termination resistors on the data bus.	0 – No termination resistor 1 – 120 ohm termination resistor enabled
Bias_Address	Identifies the port for setting the bias resistors.	3 – Port 1 or 5 7 – port 2 or 6 11 – Port 3 or 7 15 – Port 4 or 8
Bias_Enable	Enables the pull-up and pull-down resistors on the RS-485 signal lines. This prevents errors when no device is driving the bus. This setting should be coordinated with other devices on the data bus.	0 – No bias resistors. 1- Bias resistors enabled

13.0 OS Revision History

Version	Description of Changes	Date

14.0 Document Revision History

Rev	Description	Author	Date
X2	Preliminary release.	E. Bodrero	2020-04-23