MODBUS XML Driver Generation Guide



Jibonananda Sanyal

James Nutao

Date: 1 September, 2014

|  |
| --- |
| **DOCUMENT AVAILABILITY**  Reports produced after January 1, 1996, are generally available free via US Department of Energy (DOE) SciTech Connect.  ***Website*** <http://www.osti.gov/scitech/>  Reports produced before January 1, 1996, may be purchased by members of the public from the following source:  National Technical Information Service  5285 Port Royal Road  Springfield, VA 22161  ***Telephone*** 703-605-6000 (1-800-553-6847)  ***TDD*** 703-487-4639  ***Fax*** 703-605-6900  ***E-mail*** info@ntis.gov  ***Website*** <http://www.ntis.gov/help/ordermethods.aspx>  Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange representatives, and International Nuclear Information System representatives from the following source:  Office of Scientific and Technical Information  PO Box 62  Oak Ridge, TN 37831  ***Telephone*** 865-576-8401  ***Fax*** 865-576-5728  ***E-mail*** reports@osti.gov  ***Website*** <http://www.osti.gov/contact.html> |

|  |
| --- |
| This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. |

Energy and Transportation Science Division

MODBUS XML Driver Generation Guide

Jibonananda Sanyal

James Nutaro

Date Published: 1 September, 2014

Prepared by

OAK RIDGE NATIONAL LABORATORY

Oak Ridge, Tennessee 37831-6283

managed by

UT-BATTELLE, LLC

for the

US DEPARTMENT OF ENERGY

under contract DE-AC05-00OR22725

CONTENTS

Page

[LIST OF FIGURES v](#_Toc397593129)

[ACRONYMS vii](#_Toc397593130)

[ABSTRACT 1](#_Toc397593131)

[1. OVERVIEW 1](#_Toc397593132)

[2. CSV PARSER 2](#_Toc397593133)

[2.1 Installation 2](#_Toc397593134)

[2.2 Dependencies 2](#_Toc397593135)

[2.3 GETTING STARTED 2](#_Toc397593136)

[2.4 DEVICE INI FILE 3](#_Toc397593137)

[2.4.1 Meta information 3](#_Toc397593138)

[2.4.2 I/O functionality 3](#_Toc397593139)

[2.4.3 Column Indices 3](#_Toc397593140)

[2.4.4 Additional user supplied fields 4](#_Toc397593141)

[2.4.5 Function names 4](#_Toc397593142)

[3. DEVICE DRIVER GENERATOR 7](#_Toc397593143)

[3.1 INSTALLATION and DEPENDENCIES 7](#_Toc397593144)

[3.2 DEVICE DRIVER GENERATION 7](#_Toc397593145)

[4. LIMITATIONS 9](#_Toc397593146)

[5. LICENCE 3](#_Toc397593147)

LIST OF FIGURES

Figure Page

1 Usage of the CSV parser script. 2

2 The meta block 3

3 I/O related functionality 3

4 Custom column indices 4

5 User supplied custom columns 4

6 Listing of all Modbus device functions for the device 5

7 Logical representation of the MDL XML file creation 6

8 Logical representation of the device driver creation 8

ACRONYMS

CRC Cyclic Redundancy Check

CSV Comma Separated Value

IEEE Institute of Electrical and Electronics Engineers

MDL Modbus Definition Language

RTU Remote Terminal

XSD XML Schema Definition

XML Extended Markup Language

ABSTRACT

This document describes utility software to facilitate the generation of automatic generation of Modbus device drivers. The first part of this document describes a Python based parser for creating XML encoded files from vendor’s device register tables. The second part of the document describes the generation of device drivers using the XML document.

# OVERVIEW

The Modbus Definition Language (MDL) specification defines a standard for uniform representation of Modbus device register maps allowing the automated generation of device driver software for Modbus devices. The MDL is defined in an Extended Markup Language (XML) Schema Definition (XSD) and instances of MDL describing Modbus devices are XML files adhering to the MDL schema.

The XML files can be manually created by populating the required fields with the appropriate register/data values; however, this can be very cumbersome and tedious for device vendors. To streamline the generation of these XML files, the MDL is supplied with additional utility software that enables the automated transformation of existing device register map tables to XML files from which device driver code can be easily generated. It is expected that the utility software should make the adoption of the new specification easier and less cumbersome for device vendors.

The conversion from device register maps to driver code may be done in the following steps:

Step 1: Conversion of register maps to MDL XML files. The register maps, if expressed in a tabular manner as they most commonly are, in a comma separated value file, can be read into a Python program which converts to the XML file adhering to the MDL.

Step 2: Generation of driver code. The MDL XML file is read by device driver generator code to create C++ source code for the driver.

A Modbus Definition Language (MDL) specification is defined in an accompanying document. The rest of the document describes the main components of the two steps: the CSV parser from step 1 and the driver code generator from step 2. The software is organized in three directories as described below:

ModbusXMLSchema

.\docs

.\schema

.\csv-parser

.\driver-generator

# CSV PARSER

The parser reads in the device register map expressed in a tabular structure in a CSV file and outputs an MDL XML file. The Python program is supplied a command line argument specifying an INI style file. The INI file supplies all the necessary information that the parser requires for the conversion. The following sections describe how to get the software, its dependencies, and describe the INI file structure.

## Installation

A checkout of the source code includes the necessary files for using the software. This is a Python script and requires a working installation of Python 3. The Python modules used are listed in the next section and users may have to install these modules if their Python installation does not have them.

## Dependencies

The following are the Python modules used in the software. Depending on the distribution of your Python package, these modules may already be installed or may be obtained using ‘easy\_install’ or ‘pip’. The modules used are:

1. sys
2. xml
3. csv
4. difflib
5. operator
6. re
7. argparse
8. configparser
9. os
10. string

## GETTING STARTED

The script runs on a command line as follows:

usage: csvxml.py [-h] [-i] [-v] device\_ini\_file

Creates an XML file from a Modbus device address map.

positional arguments:

device\_ini\_file ini-style file that provides device meta information

optional arguments:

-h, --help show this help message and exit

-i, --interactive run interactively

Figure 1: Usage of the CSV parser script.

The script requires a device INI file which specifies the various parameters it needs for generating XML. Among many things, the INI file supplies a list of functional keywords that the software uses to search in the register table for matches. For exact matches, it uses the corresponding address, length, count, etc. values to create an XML entry for the Modbus function (for details on function definitions, please see the Modbus XML specification document). For partial matches, it calculates a similarity ratio using the Python difflib SequencMatcher algorithm. The script may be run interactively or non-interactively. In the interactive mode, the script provides a sorted listing of the closest matches and allows the user to input the best candidate. In the non-interactive mode, the script simply picks what it has algorithmically determined to be the closest match.

An MDL XML file for the device is generated as output.

## DEVICE INI FILE

This section describes the INI style file required by the parser script.

### Meta information

This block describes the Modbus device name and a short description of the device.

# An INI style file describing the inputs to the parser program. All major

# modbus device descriptions go here. Please use this as a template to provide

# the guiding values to the parser program

# Device meta information. All fields in this section are required.

[Modbus Device]

device\_name = TEMPCO Modbus device

device\_description = This is a description of the TEMPCO modbus device

Figure 2: The meta block.

### I/O functionality

This block provides input related to files for I/O. The ‘address\_map\_csv\_file’ element is a required field and describes the csv filename of the Modbus device register table. The ’lines\_to\_skip’ field allows the skipping of some lines from the beginning of this file to skip over comments, headers, etc. The optional ‘output\_xml\_file’ field contains the desired output filename of the generated XML file. If this field is not supplied, the script derives an XML filename from the ‘address\_map\_csv\_file’ field.

# All I/O related functionality. All fields in this section are required.

[Input/Output]

address\_map\_csv\_file = test2.csv

lines\_to\_skip = 1

output\_xml\_file = map.xml

Figure 3: I/O related functionality.

### Column Indices

Different vendors many have different styles of register tables. Some may have a block label to the very left while others may have the register addresses to the very left. While these formats may be varied, the information provided typically is schematically similar. To enable flexibility and low adoption overhead, the column indices of the different fields in the register table can be explicitly specified. This block allows users to input the corresponding column indices

To be adaptable, the software accepts indices of the columns corresponding to the different fields, as illustrated below.

# Column indices (zero-based) for all fields of the address map csv file. If

# a column/field is missing, please leave it blank. Some fields are not

# optional; please consult the schema.

[Column/Field Indices]

description = 9

addresses = 3

length = 7

count = 4

format = 5

block\_label = 0

multiplier = 6

units =

read\_function\_code =

write\_function\_code =

Figure 4: Custom column indices.

### Additional user supplied fields

This section lets the user add additional columns present in their CSV file and extend the functionality of the software. Some vendors may have columns that we may have overlooked in this design. This section allows them to be added these which become available in the script. Vendors’ programmers can then use the column information for the necessary change in logic.

# User supplied column names and indices (zero-based) for additional logic that

# may be required for translating the regoister map to xml. Here,

# 'operation\_info' is used to differentiate between R, W, and R/W addresses in

# the csv2xml.py code. See the section of the code that uses this field.

[Additional Column/Field Indices]

operation\_info = 8

Figure 5: User supplied custom columns.

### Function names

This section lists the functions of the Modbus device that are exposed to the device driver interface. A list of function names and corresponding lists of synonyms must be provided. The synonyms are used for full and partial matches in the device driver map. The resulting MDL XML file will contain function blocks corresponding to the functions listed in this section. Further, when the device driver is generated from the XML, only the functions present in the XML file will have corresponding device driver code.

# Listing of all function names and comma separated list of searchable

# synonyms.

[Function Name Search Synonyms]

temperature = temperature, Temperature

fan\_relay\_on = Relay1 manual output

cooling\_stage1\_relay\_on = Relay2 manual output

cooling\_stage2\_relay\_on = Relay3 manual output

heating\_stage1\_relay\_on = Relay4 manual output

heating\_stage2\_relay\_on = Relay5 manual output

temperature\_set\_point = Occupied setpoint

Figure 6: Listing of all Modbus device functions for the device.

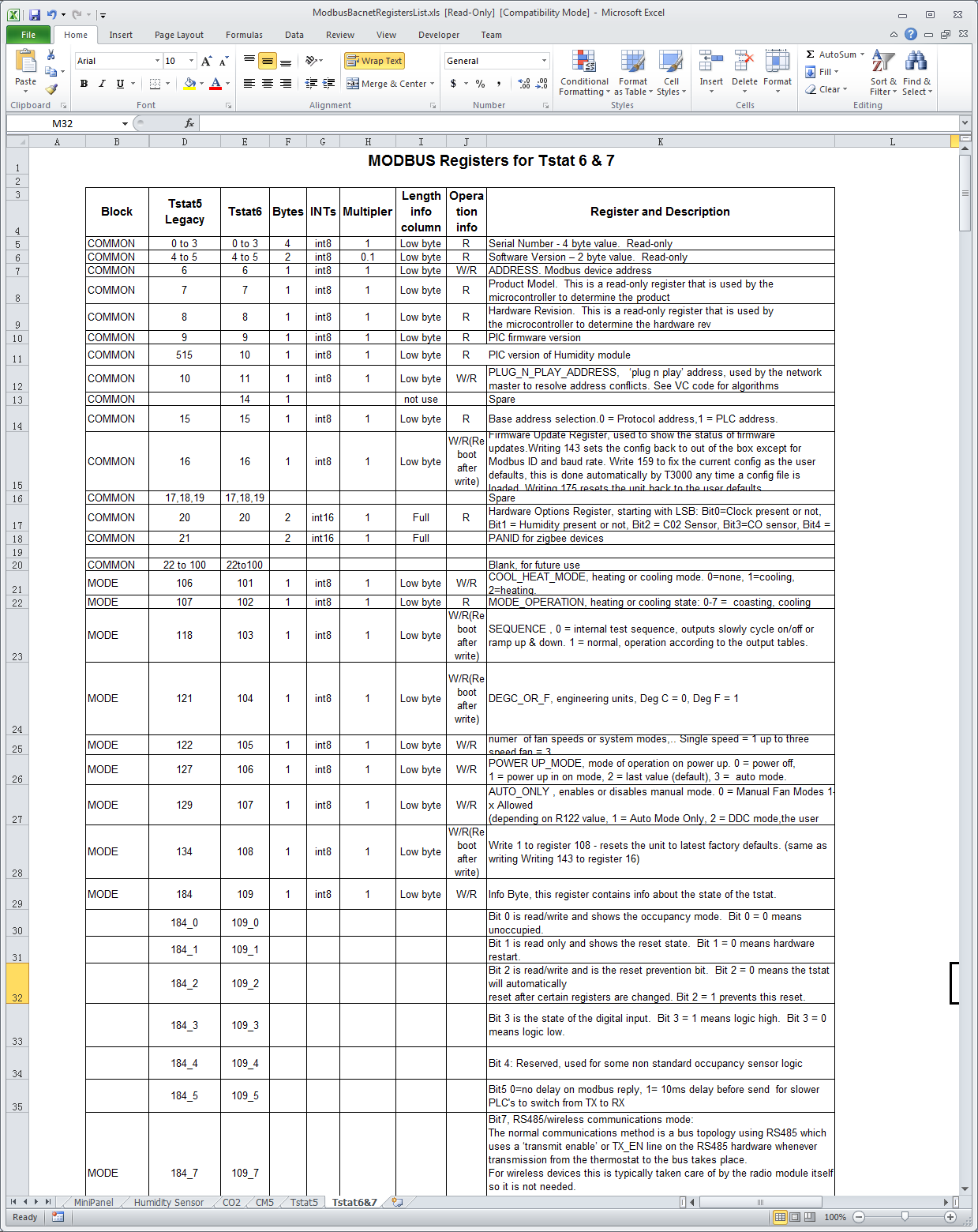
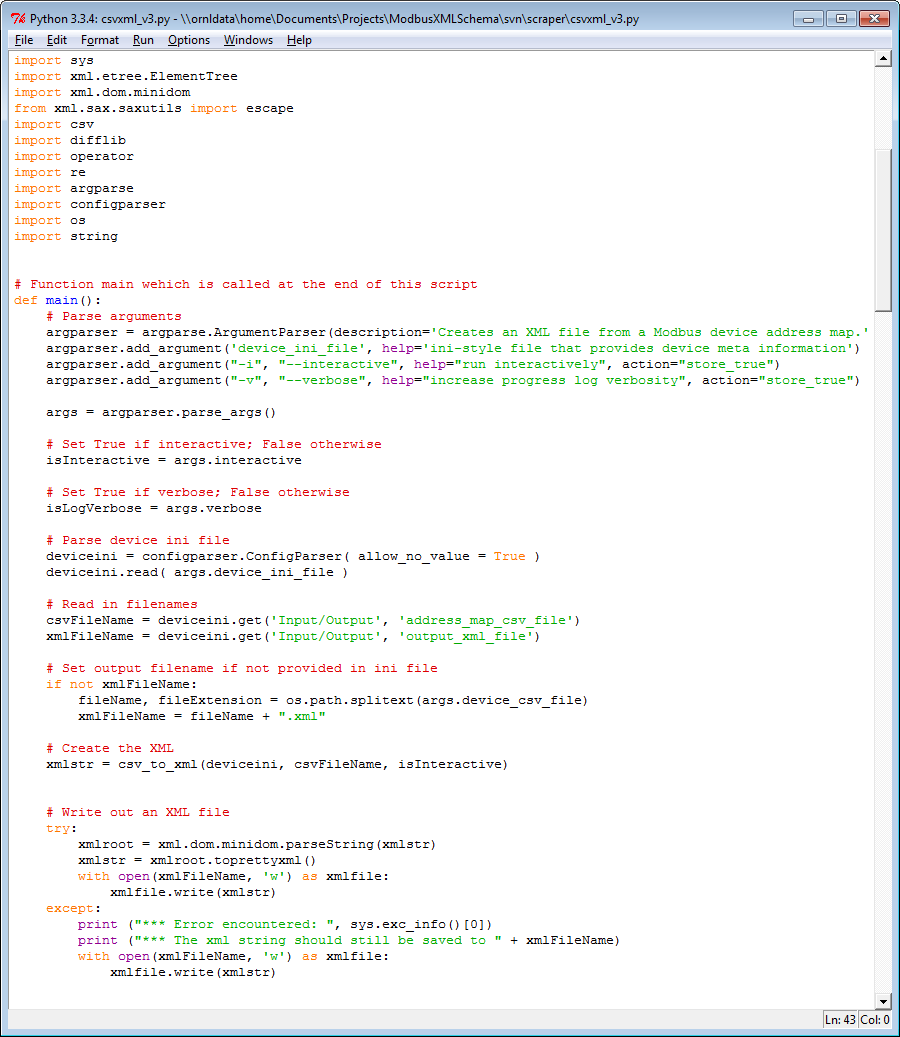


Figure 7: Logical representation of the MDL XML file creation

***Register table***

***XML file***

***Parser***

# DEVICE DRIVER GENERATOR

This section describes the dependencies and method to compile the C++ code for generating device drivers from the XML file generated by the parser.

## INSTALLATION and DEPENDENCIES

A checkout of the source code includes the necessary files for compiling and using the software. This is C/C++ code with several external dependencies. A simple Makefile is supplied which should make the compilation straightforward.

The following are the library dependencies:

1. libmodbus
2. libxml++ 2.6
3. stl

An executable named ‘mdl2code’ is created after compiling the source code.

## DEVICE DRIVER GENERATION

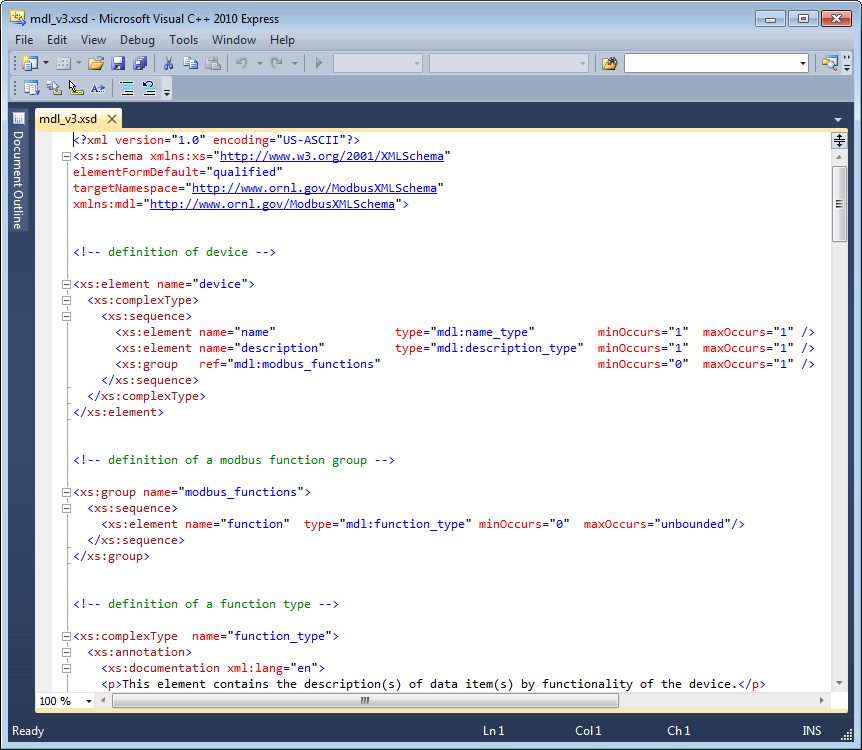
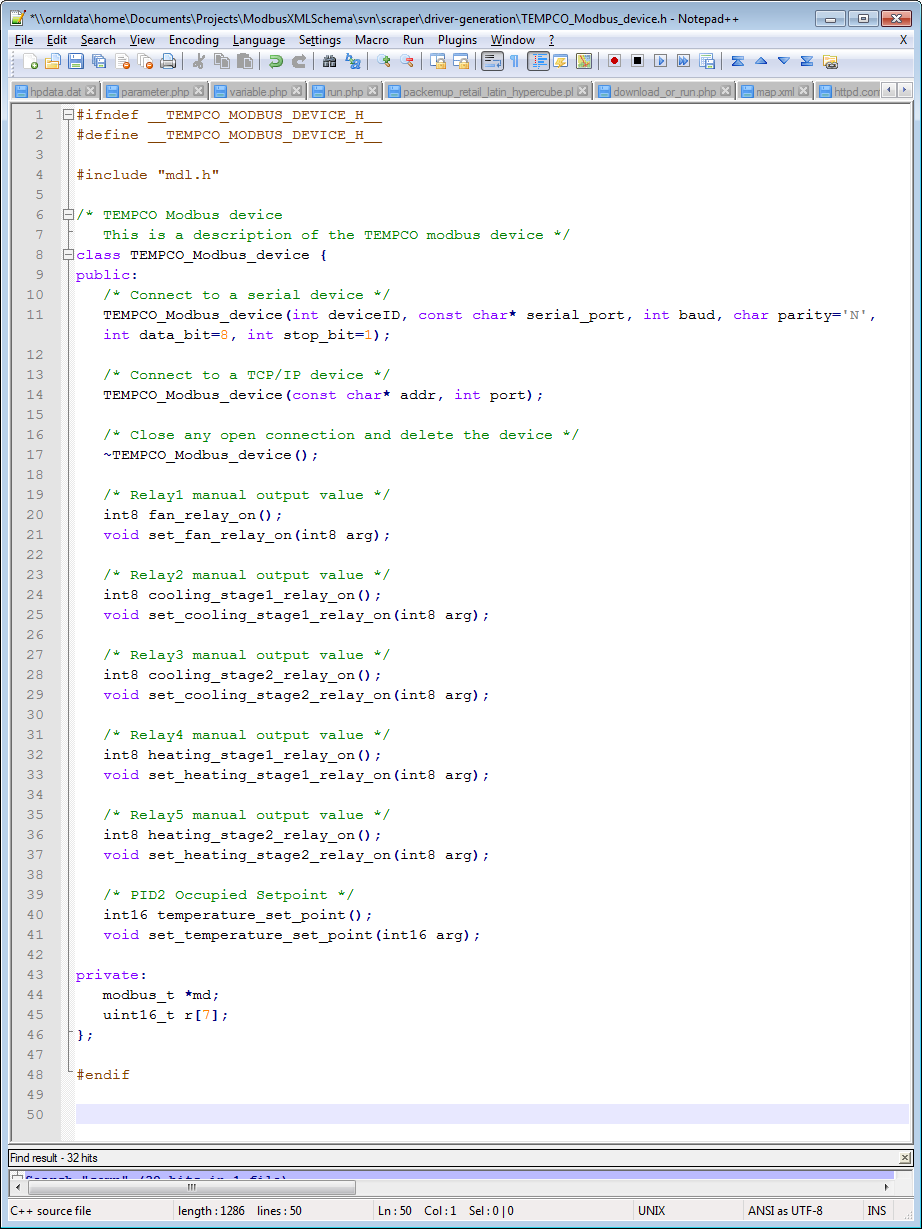
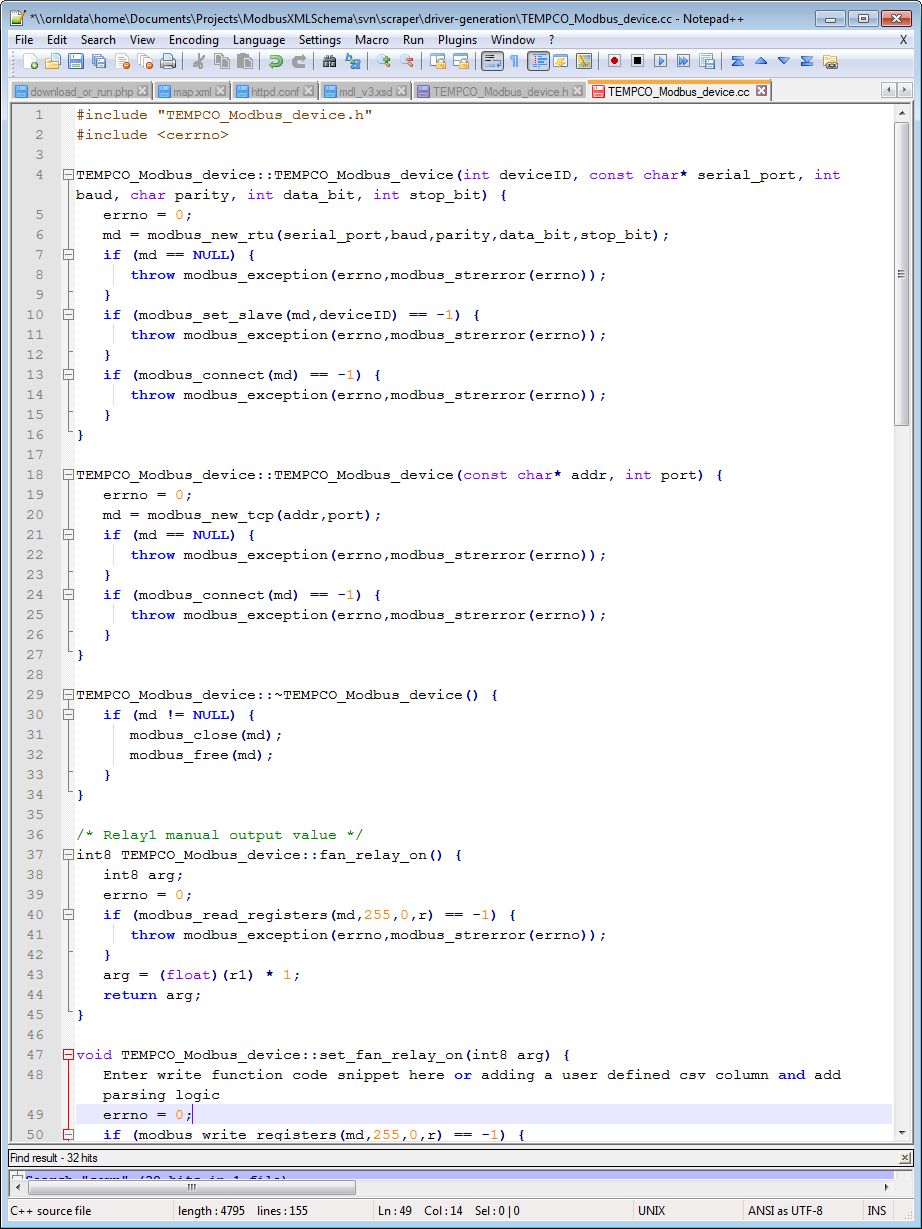
The executable ‘mdl2code’ accets one argument which is the MDL XML file name that has been generated using the CSV parser (or is has been hand created). The executable creates <device\_name> >.h and <device\_name>.cc files which are the corresponding device driver files for the supplied MDL XML. Spaces, if any, in the device name are replaced with underscores to create the filenames of the generated code.



Figure 8: Logical representation of the device driver creation.

***C++ header***

***C++ sources***



XML Validation

Code Generation

***XML file***

# LIMITATIONS

Known limitations of the software include:

CSV parser

1. Developed and tested on Python 3 and untested on Python 2.7.
2. Developed and tested on Linux Ubuntu platform only.
3. Support US-ASCII csv files ony.

Device-driver generator

1. Developed, compiled and tested on Linux Ubuntu only.

# LICENCE

Copyright (c) 2014 Oak Ridge National Laboratory

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sellcopies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.