ModBus XML Driver Generation Guide

Jibonananda Sanyal
James Nutaro

29 September 2014

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

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 Updated: 29 September, 2014
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
ACRONYMS vii
ABSTRACT 1
1. OVERVIEW 1
2. CSV PARSER 2
2.1 INSTALLATION 2
2.2 DEPENDENCIES 2
2.3 GETTING STARTED 2
2.4 DEVICE INI FILE 3
2.4.1 Meta information 3
2.4.2 I/O functionality 3
2.4.3 Column Indices 4
2.4.4 Additional user supplied fields 4
2.4.5 Function names 5
3. DEVICE DRIVER GENERATOR 7
3.1 INSTALLATION AND DEPENDENCIES 7
3.2 DEVICE DRIVER GENERATION 7
4. LIMITATIONS 9
5. LICENCE 10

LIST OF FIGURES

Figure **Page**

Figure 1: How to use the CSV parser script. 2
Figure 2: Structure of a device's meta information. 3
Figure 3: Flexible handling for converting a device manufacturer's CSV file to a device XML file. 4
Figure 4: Flexible mapping of data columns in a device manufacturer's CSV file to specific information required to create device's XML file. 4
Figure 5: Additional columns can be specified for inclusion for modifying specific parsing behavior that uses additional input. 5
Figure 6: Listing of all Modbus device functions for the device which can be used to communicate with the device via the automatically generated device driver. 5
Figure 7: Logical representation of the CSV parser to MDL XML file 6
Figure 8: Logical representation of the device driver is validated and used to generate driver code for communicating with the device. 8

ACRONYMS

CSV	Comma Separated Value
INI	INItialization file
MDL	Modbus Definition Language
XML	eXtensible Markup Language
XSD	XML Schema Definition

ABSTRACT

This document describes utility software to facilitate automatic generation of Modbus device drivers which allows interoperable control of devices. Two software utilities are discussed: a CSV parser and a device driver generator. The CSV parser is a Python utility which translates a vendor's device register tables into an XML encoded file that describes the device and its registers. The device driver generator is C++ code that generates device drivers using the XML document created by the CSV parser.

1. OVERVIEW

The Modbus Definition Language (MDL) specification defines a uniform representation as a standard for describing Modbus device register maps. This allows automated generation of device driver software for Modbus devices. The MDL is defined by an eXtensible Markup Language (XML) Schema Definition (XSD). A specific instance of an MDL can describe individual Modbus devices using an individual XML file for each device. Please see the MDL specification for more details.

The XML files can be manually created by populating the required fields with the appropriate register/data values; however, this can be very tedious and error-prone 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 an XML files which can be used to easily generate device driver code. It is expected that the utility software should make the adoption of the new specification easier for device vendors.

The conversion from device register maps to driver code involves the following 2 step process:

Step 1: Conversion of register maps to MDL XML files. The register maps, commonly expressed in a tabular manner as a comma separated value (CSV) file, can be read into a Python program which creates an 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.

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 directories as described below:

ModbusXMLSchema
- .\docs
- .\schema
- .\csv-parser
- .\driver-generator

All descriptions in this document use the TEMPCO thermostat as an example of a ModBus device.

2. 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 INItialization (INI) file structure.

2.1 INSTALLATION

A checkout of the source code includes the necessary files for using the software from the following GitHub repository:

https://github.com/ORNL-BTRIC/ModbusXMLSchema

Use of this software 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.

2.2 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:

a. sys
b. xml
c. csv
d. difflib
e. operator
f. re
g. argparse
h. configparser
i. os
j. string

2.3 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: How to use 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. As an example, reading and writing the temperature set point for a thermostat device is fairly common, so the code looks through all instances of 'temperature' in a manufacturer's spreadsheet as a possible data item that should be read or written to by a device driver. For exact matches, the script uses values in fields (such as address, length, count, etc.) to create an XML entry for the Modbus function (for details on function definitions, please see the Modbus XML specification document). However, it is impossible to have a completely consistent nomenclature across all device manufacturers. For partial matches, it calculates a similarity ratio using the Python difflib SequenceMatcher algorithm. The script may be run interactively or non-interactively. In interactive mode, the script provides a sorted listing of the closest matches and allows the user to determine which field in the manufacturer's device spreadsheet corresponds to the field being required by the device's XML data description. In the non-interactive mode, the script simply picks what it has algorithmically determined to be the closest match. While tests may in some cases indicate an incorrect mapping between device specifications (manufacturer's *.csv to modbus' *.xml), the non-interactive mode will likely require manual correction by someone familiar with the current device.

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

2.4 DEVICE INI FILE

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

2.4.1 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: Structure of a device's meta information.

2.4.2 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 program to ignore the header lines for comments, units, and other information typically found in the top rows of a CSV file. 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. The script will overwrite the output file if it exists.

```
# 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: Flexible handling for converting a device manufacturer's CSV file to a device XML file.

2.4.3 Column Indices

Different vendors have different styles of register tables for recording the communication-specific details of their devices. Some may have a block label in the first column while others may have the register address in the first column. While these formats may be varied, the information provided is often schematically similar. To enable flexibility and ease of adoption, 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: Flexible mapping of data columns in a device manufacturer's CSV file to specific information required to create device's XML file.

2.4.4 Additional user supplied fields

This section allows the user to specify additional columns present in the CSV file and extend the functionality of the software. Some vendors may have columns that have overlooked in this schema design. This section allows additional columns of data to be added and can be used by the parsing script. Vendors' programmers can then use the column information for any necessary changes in communication logic.

```
# User supplied column names and indices (zero-based) for additional logic that
# may be required for translating the register 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: Additional columns can be specified for inclusion for modifying specific parsing behavior that uses additional input.

2.4.5 Function names

This section lists the functions of the Modbus device that are exposed to the device driver interface, allowing the device to be used through these functions. A list of function names and corresponding list of synonymous search terms 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 in the driver source code 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 which can be used to communicate with the device via the automatically generated device driver.

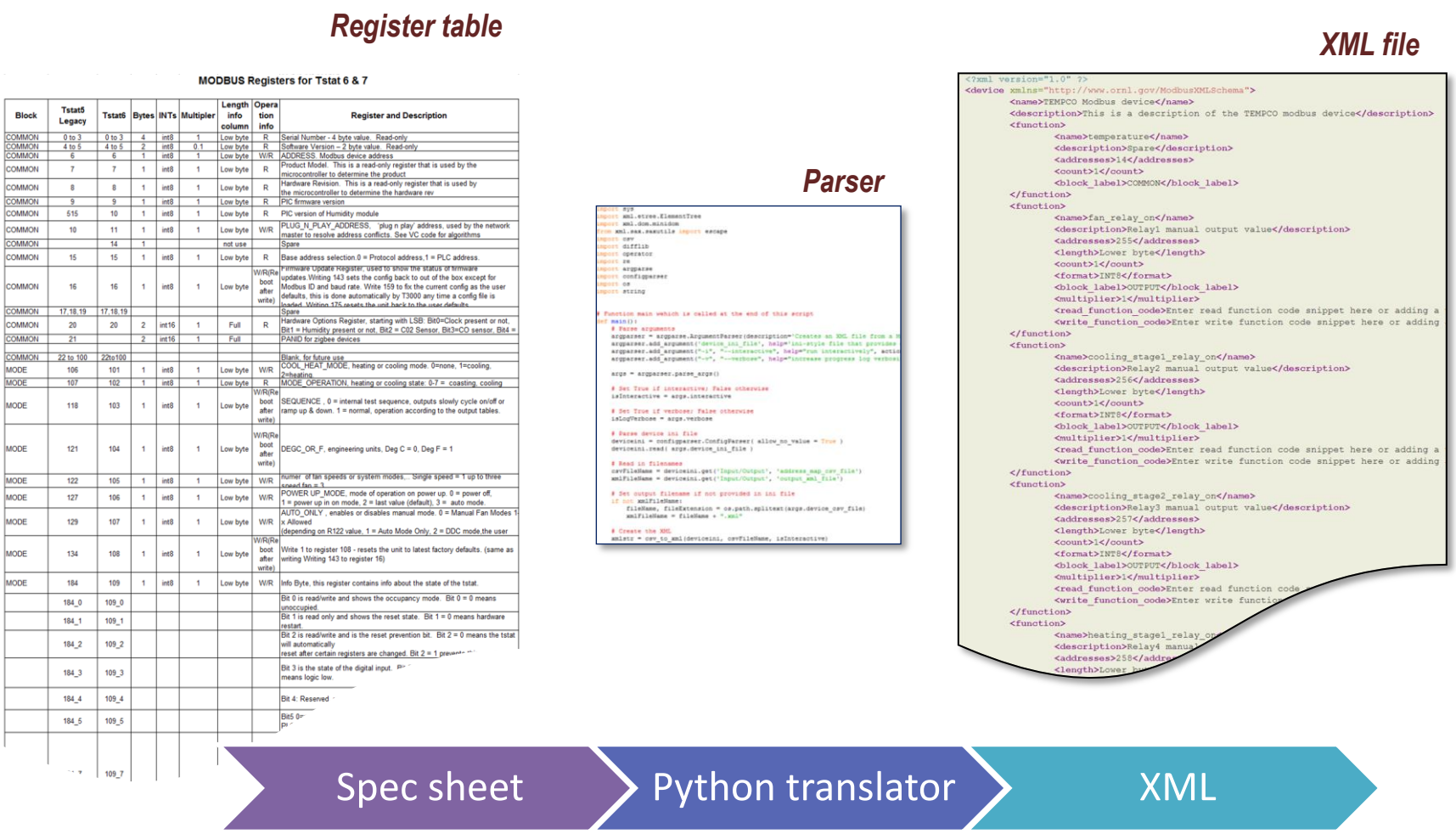

Figure 7: Logical representation of the CSV parser to MDL XML file

3. 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.

3.1 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:

a. libmodbus
b. libxml++ 2.6
c. stl

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

3.2 DEVICE DRIVER GENERATION

The executable ‘mdl2code’ accepts one argument which is the MDL XML filename (which can be created/edited manually or can be generated using the CSV parser). 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

.

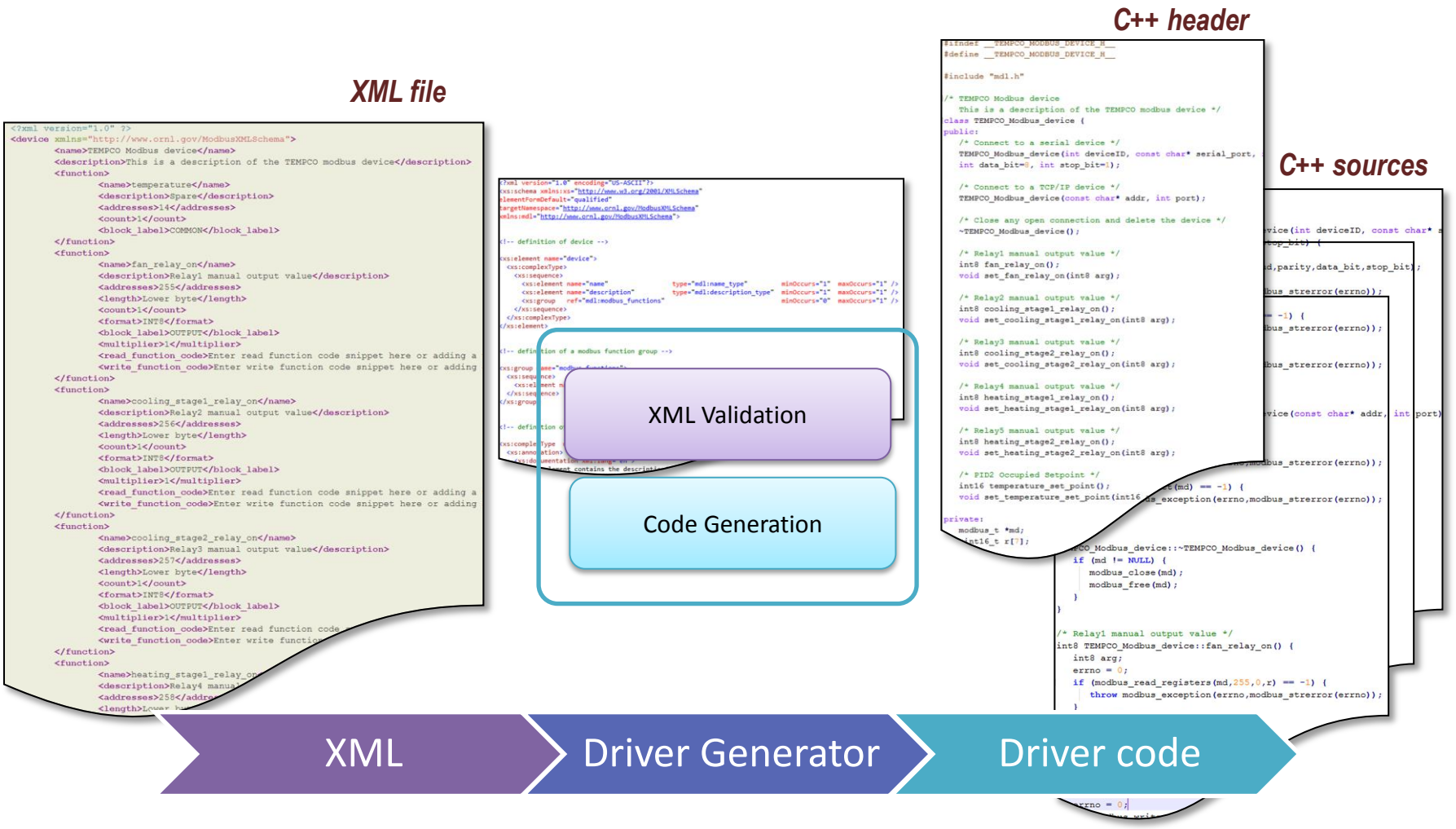

Figure 8: Logical representation of the device driver is validated and used to generate driver code for communicating with the device.

4. LIMITATIONS

Known limitations of the software include:

CSV parser

a. Developed and tested on Python 3 and untested on Python 2.7.
b. Developed and tested on Linux Ubuntu platform only.
c. Support US-ASCII csv files only.

Device-driver generator

a. Developed, compiled and tested on Linux Ubuntu only.
b. Only tested on a TEMCO thermostat.

5. LICENSE

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 sell copies 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.