IBPSA
BUILDING SIMULATION CONFERENCE
2015
HYDERABAD
DECEMBER 7 - 9, 2015

Building Simulation Conference - 2015

The Modbus Definition Language: A First Step Towards Device Interoperability

Jibonananda Sanyal, Joshua New, James Nutaro, David Fugate, and Teja Kuruganti

Oak Ridge National Laboratory

Outline

- Device interoperability and buildings
- Seamless interoperability architecture
- Modbus Definition Language
- Utility tools

Sensors and Buildings

- Monitoring of building performance
- Software-in-the-loop simulation
- Challenges:
  - Legacy sensors, actuators, and automation devices
  - Emerging new whole-building control
  - Poor integration
  - Custom software

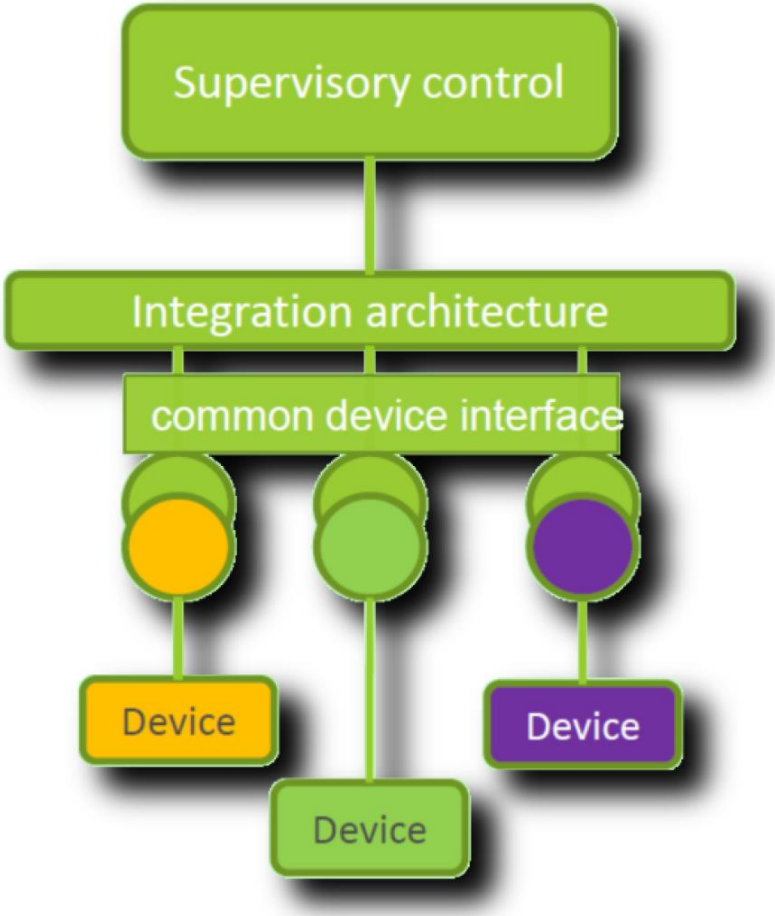
The potential opportunity

- Advanced control of HVACs can save 10% energy in the US
- Simulation informed building management expands this even more
- Immense reduction in device driver costs
- Removes expensive interfacing

Goals of this technology

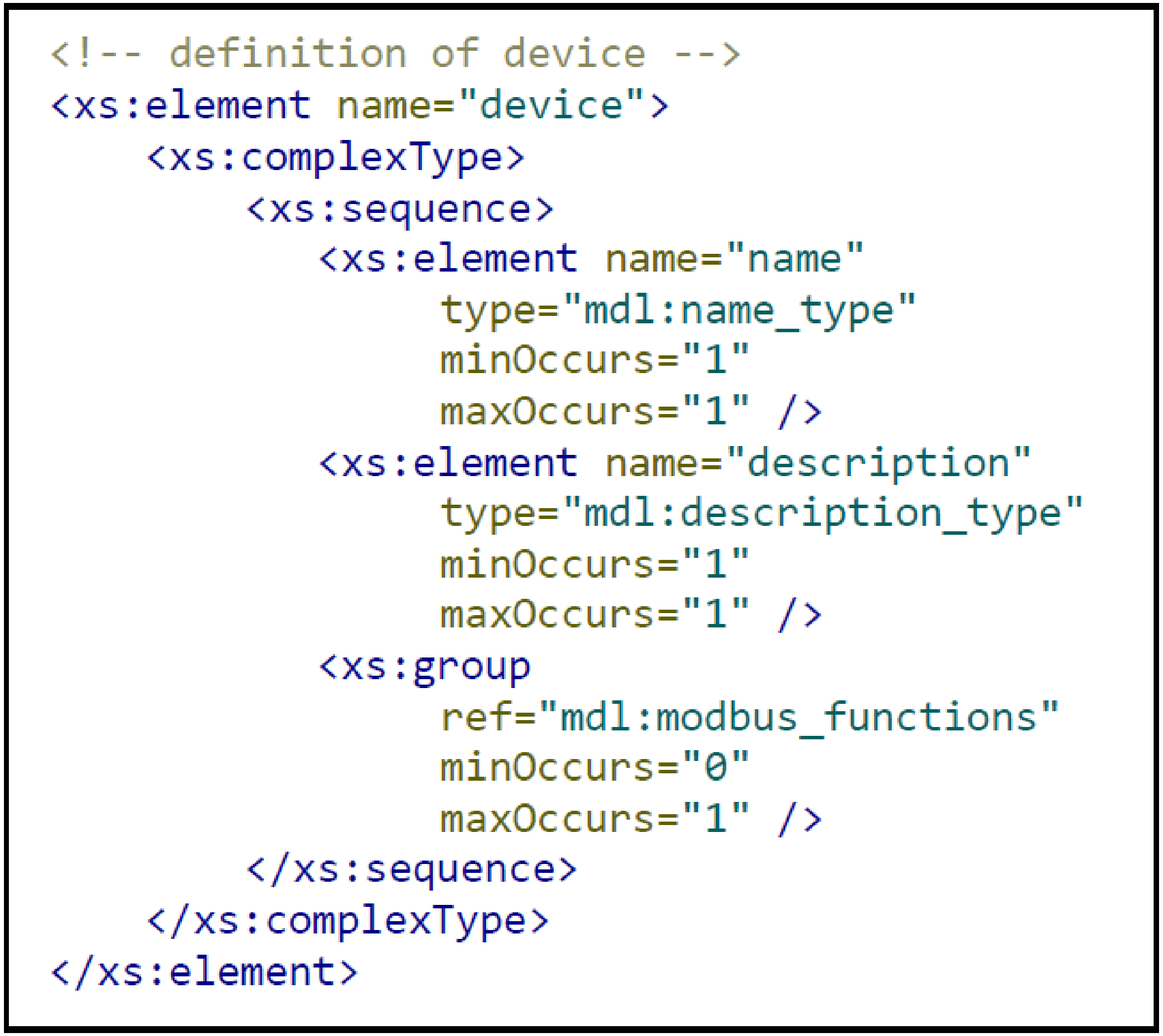
- Automated generation of device driver software for Modbus devices
- Seamless interoperability
- Reduces human effort
- Modbus Definition Language (MDL)
  - New XML based standard
  - XSD for validation

Seamless Interoperability



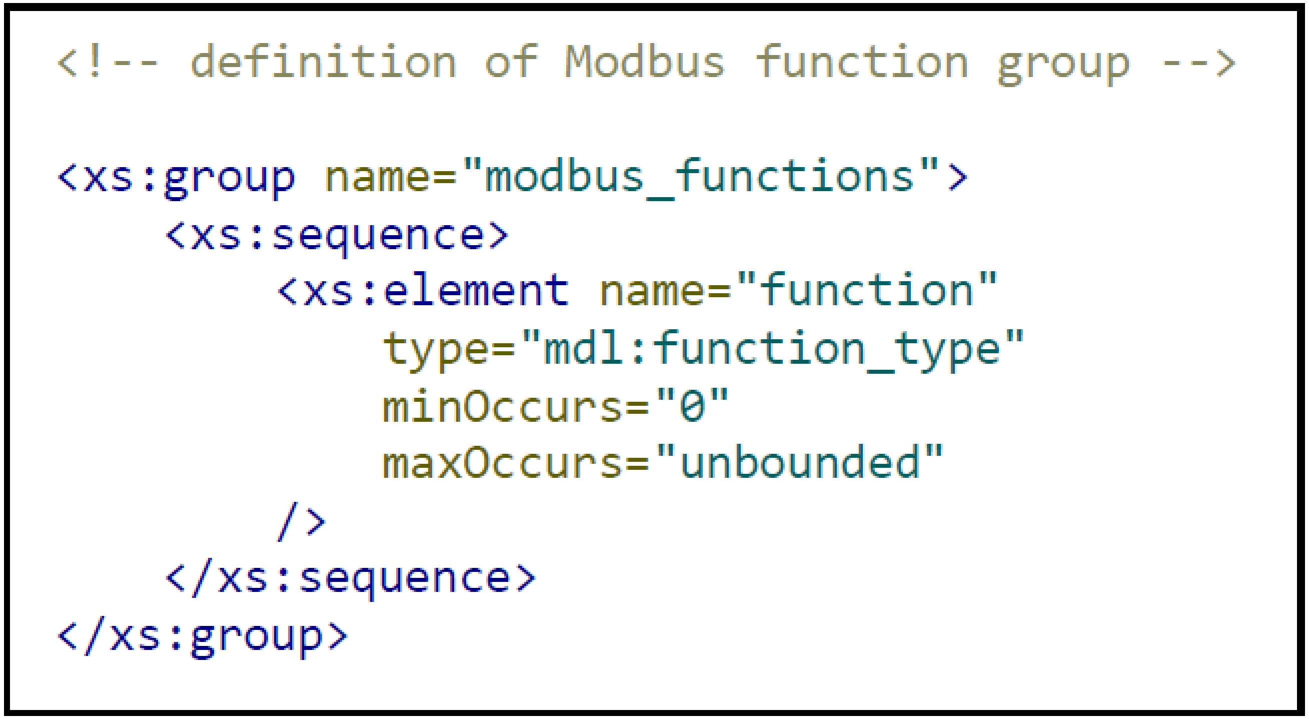
The Modbus Definition Language

Device Definition



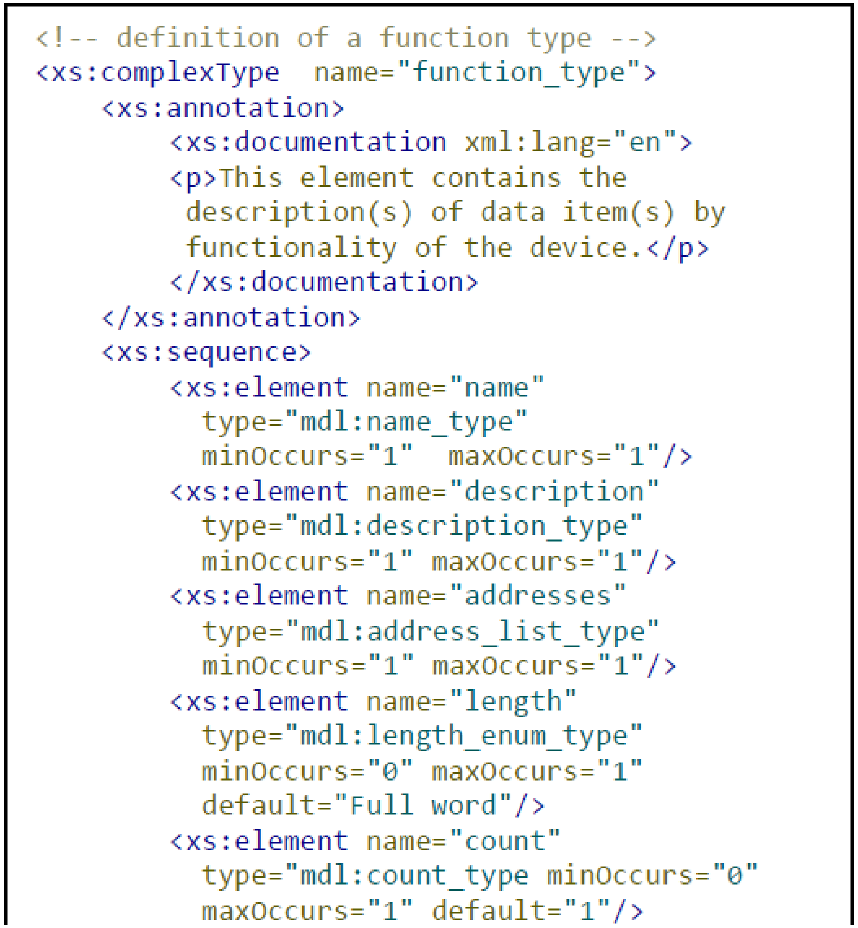
```
<!-- definition of device -->
<xs:element name="device">
    <xs:complexType>
        <xs:sequence>
           <xs:element name="name"
                type="mdl:name_type"
                minOccurs="1"
                maxOccurs="1" />
           <xs:element name="description"
                type="mdl:description_type"
                minOccurs="1"
                maxOccurs="1" />
           <xs:group
                ref="mdl:modbus_functions"
                minOccurs="0"
                maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Function Groups

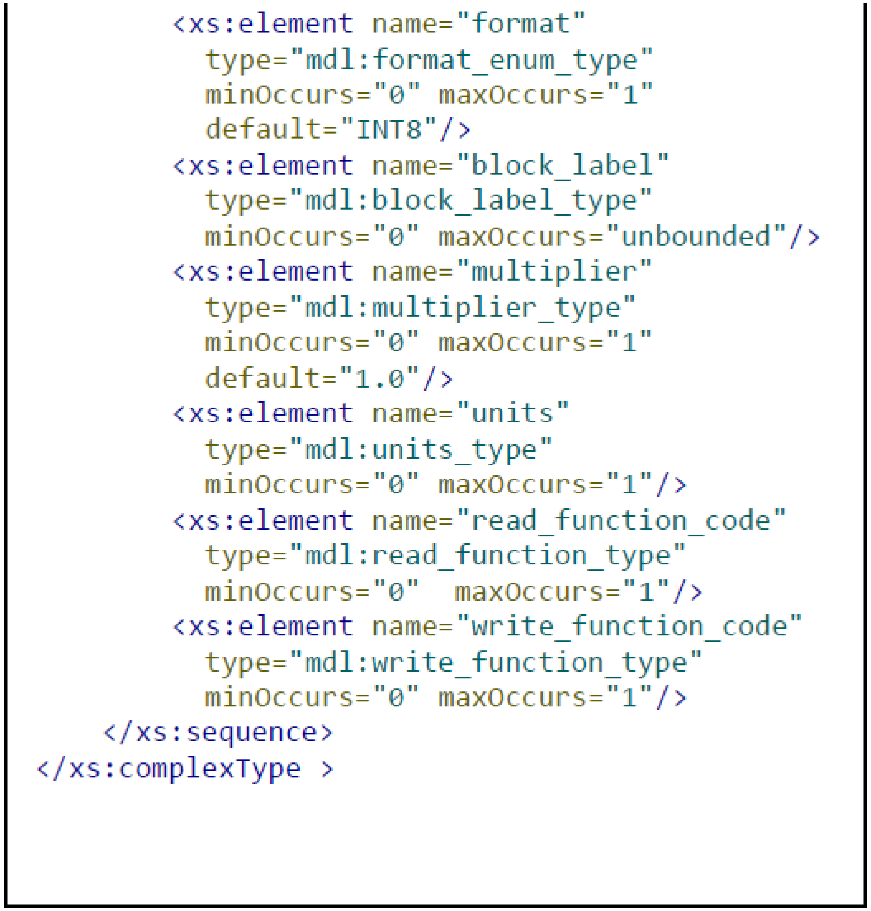


```
<!-- definition of Modbus function group -->

<xs:group name="modbus_functions">
    <xs:sequence>
        <xs:element name="function"
            type="mdl:function_type"
            minOccurs="0"
            maxOccurs="unbounded"
        />
    </xs:sequence>
</xs:group>
```

Functions

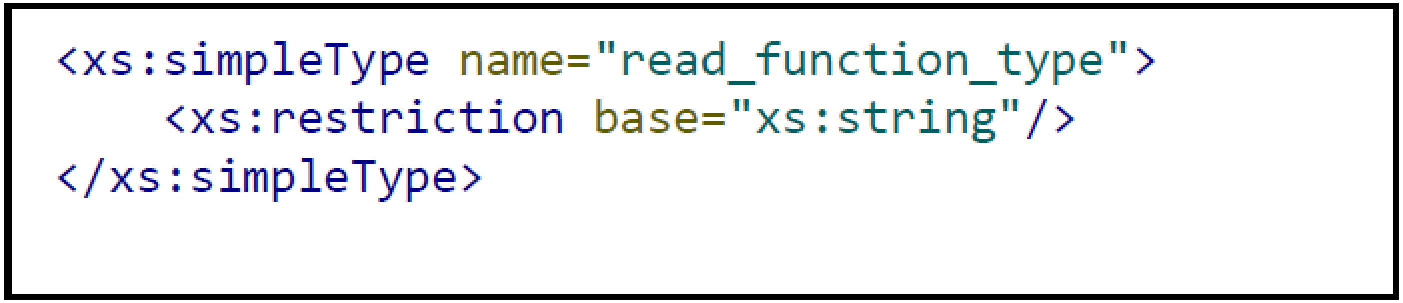


```
<!-- definition of a function type -->
<xs:complexType  name="function_type">
    <xs:annotation>
        <xs:documentation xml:lang="en">
        <p>This element contains the
         description(s) of data item(s) by
         functionality of the device.</p>
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="name"
          type="mdl:name_type"
          minOccurs="1"  maxOccurs="1"/>
        <xs:element name="description"
          type="mdl:description_type"
          minOccurs="1" maxOccurs="1"/>
        <xs:element name="addresses"
          type="mdl:address_list_type"
          minOccurs="1" maxOccurs="1"/>
        <xs:element name="length"
          type="mdl:length_enum_type"
          minOccurs="0" maxOccurs="1"
          default="Full word"/>
        <xs:element name="count"
          type="mdl:count_type minOccurs="0"
          maxOccurs="1" default="1"/>
```

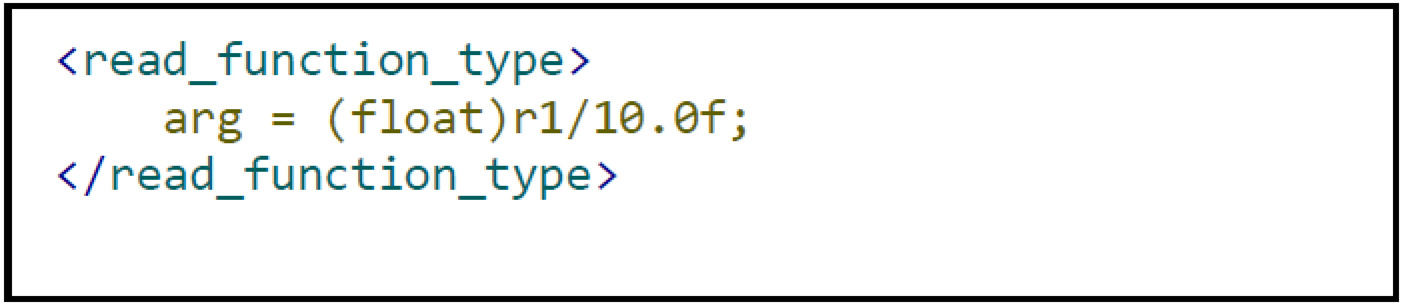


```
        <xs:element name="format"
          type="mdl:format_enum_type"
          minOccurs="0" maxOccurs="1"
          default="INT8"/>
        <xs:element name="block_label"
          type="mdl:block_label_type"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="multiplier"
          type="mdl:multiplier_type"
          minOccurs="0" maxOccurs="1"
          default="1.0"/>
        <xs:element name="units"
          type="mdl:units_type"
          minOccurs="0" maxOccurs="1"/>
        <xs:element name="read_function_code"
          type="mdl:read_function_type"
          minOccurs="0"  maxOccurs="1"/>
        <xs:element name="write_function_code"
          type="mdl:write_function_type"
          minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType >
```

Read and Write



```
<xs:simpleType name="read_function_type">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
```



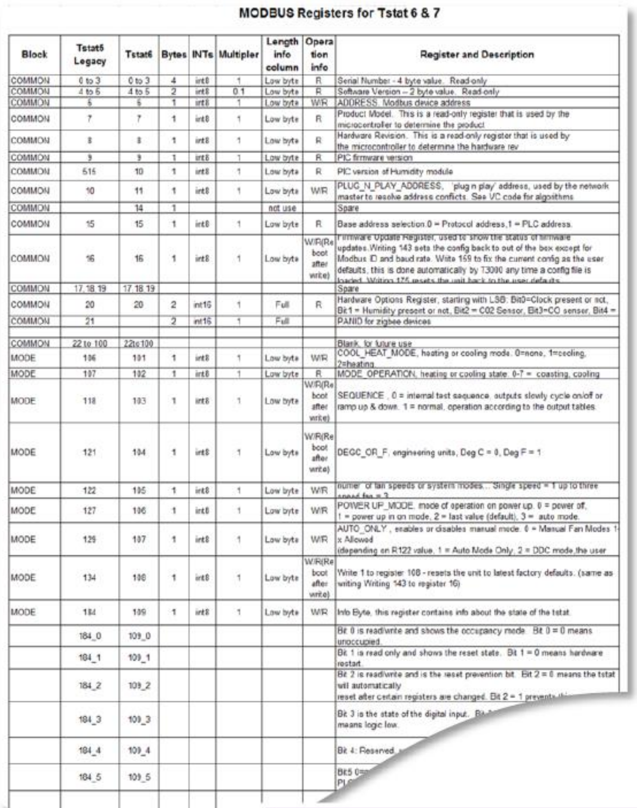
```
<read_function_type>
    arg = (float)r1/10.0f;
</read_function_type>
```

Simple Example

```
<?xml version="1.0" ?>
<device xmlns="http://www.ornl.gov/ModbusXMLSchema">
           <name>TEMPCO Modbus device</name>
           <description>This is a description of the TEMPCO modbus device</description>
           <function>
                      <name>temperature</name>
                      <description>Spare</description>
                      <addresses>14</addresses>
                      <count>1</count>
                      <block_label>COMMON</block_label>
           </function>
           <function>
                      <name>fan_relay_on</name>
                      <description>Relay1 manual output value</description>
                      <addresses>255</addresses>
                      <length>Lower byte</length>
                      <count>1</count>
                      <format>INT8</format>
                      <block_label>OUTPUT</block_label>
                      <multiplier>1</multiplier>
                      <read_function_code> arg = (float)r1/10.0f;  </read_function_code>
           </function>
           …
           …
           …

</device>
```

Supporting Utilities

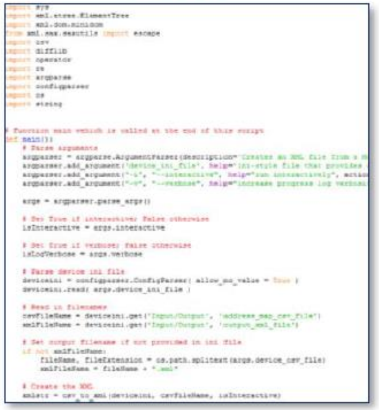
Register table



MODBUS Registers for Tstat 6 & 7

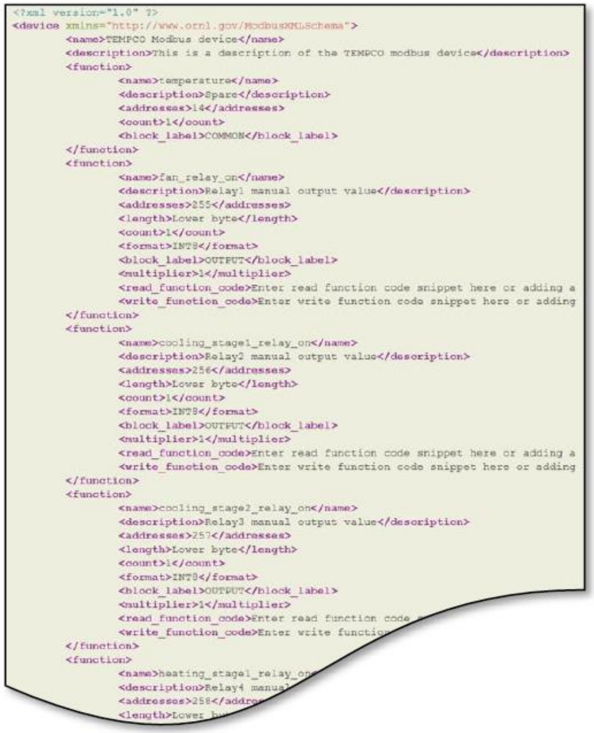
Block	Tstat5 Legacy	Tstat6	Bytes	INTs	Multipler	Length info column	Operation info	Register and Description
COMMON	0 to 3	0 to 3	4	int8	1	Low byte	R	Serial Number - 4 byte value. Read-only
COMMON	4 to 5	4 to 5	2	int8	0.1	Low byte	R	Software Version – 2 byte value. Read-only
COMMON	6	6	1	int8	1	Low byte	W/R	ADDRESS. Modbus device address
COMMON	7	7	1	int8	1	Low byte	R	Product Model. This is a read-only register that is used by the microcontroller to determine the product
COMMON	8	8	1	int8	1	Low byte	R	Hardware Revision. This is a read-only register that is used by the microcontroller to determine the hardware rev
COMMON	9	9	1	int8	1	Low byte	R	PIC firmware version
COMMON	515	10	1	int8	1	Low byte	R	PIC version of Humidity module
COMMON	10	11	1	int8	1	Low byte	W/R	PLUG_N_PLAY_ADDRESS, 'plug n play' address, used by the network master to resolve address conficts. See VC code for algorithms
COMMON		14	1			not use		Spare
COMMON	15	15	1	int8	1	Low byte	R	Base address selection.0 = Protocol address,1 = PLC address.
COMMON	16	16	1	int8	1	Low byte	W/R(Reboot after write)	Firmware Update Register, used to show the status of firmware updates.Writing 143 sets the config back to out of the box except for Modbus ID and baud rate. Write 159 to fix the current config as the user defaults, this is done automatically by T3000 any time a config file is loaded. Writing 175 resets the unit back to the user defaults.
COMMON	17,18,19	17,18,19						Spare
COMMON	20	20	2	int16	1	Full	R	Hardware Options Register, starting with LSB: Bit0=Clock present or not, Bit1 = Humidity present or not, Bit2 = C02 Sensor, Bit3=CO senser, Bit4 =
COMMON	21		2	int16	1	Full		PANID for zigbee devices
COMMON	22 to 100	22to100						Blank, for future use
MODE	106	101	1	int8	1	Low byte	W/R	COOL_HEAT_MODE, heating or cooling mode. 0=none, 1=cooling, 2=heating
MODE	107	102	1	int8	1	Low byte	R	MODE_OPERATION, heating or cooling state. 0-7 = coasting, cooling
MODE	118	103	1	int8	1	Low byte	W/R(Reboot after write)	SEQUENCE , 0 = internal test sequence, outputs slowly cycle on/off or ramp up & down. 1 = normal, operation according to the output tables.
MODE	121	104	1	int8	1	Low byte	W/R(Reboot after write)	DEGC_OR_F, engineering units, Deg C = 0, Deg F = 1
MODE	122	105	1	int8	1	Low byte	W/R	numer of fan speeds or system modes... Single speed = 1 up to three speed fan = 3
MODE	127	106	1	int8	1	Low byte	W/R	POWER UP_MODE. mode of operation on power up. 0 = power off, 1 = power up in on mode, 2 = last value (default), 3 = auto mode.
MODE	129	107	1	int8	1	Low byte	W/R	AUTO_ONLY , enables or disables manual mode. 0 = Manual Fan Modes 1-x Allowed (depending on R122 value. 1 = Auto Mode Only, 2 = DDC mode,the user
MODE	134	108	1	int8	1	Low byte	W/R(Reboot after write)	Write 1 to register 108 - resets the unit to latest factory defaults. (same as writing Writing 143 to register 16)
MODE	184	109	1	int8	1	Low byte	W/R	Info Byte, this register contains info about the state of the tstat.
	184_0	109_0						Bit 0 is read/write and shows the occupancy mode. Bit 0 = 0 means unoccupied.
	184_1	109_1						Bit 1 is read only and shows the reset state. Bit 1 = 0 means hardware restart.
	184_2	109_2						Bit 2 is read/write and is the reset prevention bit. Bit 2 = 0 means the tstat will automatically reset after certain registers are changed. Bit 2 = 1 prevents
	184_3	109_3						Bit 3 is the state of the digital input. Bi means logic low.
	184_4	109_4						Bit 4: Reserved.
	184_5	109_5						Bit5 0= PL
	[illegible]	109_7						

Parser

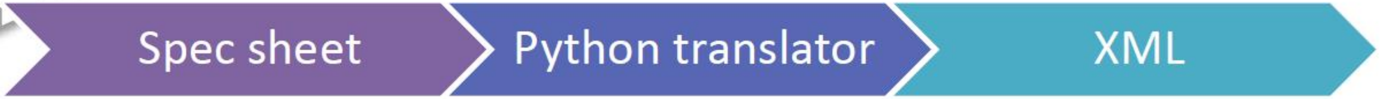


```
import sys
import xml.etree.ElementTree
import xml.dom.minidom
from xml.sax.saxutils import escape
import csv
import difflib
import operator
import re
import argparse
import configparser
import os
import string


# function main which is called at the end of this script
def main():
    # Parse arguments
    argparser = argparse.ArgumentParser(description='Creates an XML file from a M
    argparser.add_argument('device_ini_file', help='ini-style file that provides 
    argparser.add_argument("-i", "--interactive", help="run interactively", actio
    argparser.add_argument("-v", "--verbose", help="increase progress log verbosi

    args = argparser.parse_args()

    # Set true if interactive; false otherwise
    isInteractive = args.interactive

    # Set true if verbose; false otherwise
    isLogVerbose = args.verbose

    # Parse device ini file
    deviceini = configparser.ConfigParser( allow_no_value = True )
    deviceini.read( args.device_ini_file )

    # Read in filenames
    csvFileName = deviceini.get('Input/Output', 'address_map_csv_file')
    xmlFileName = deviceini.get('Input/Output', 'output_xml_file')

    # Set output filename if not provided in ini file
    if not xmlFileName:
        fileName, fileExtension = os.path.splitext(args.device_csv_file)
        xmlFileName = fileName + ".xml"

    # Create the XML
    xmlstr = csv_to_xml(deviceini, csvFileName, isInteractive)
```

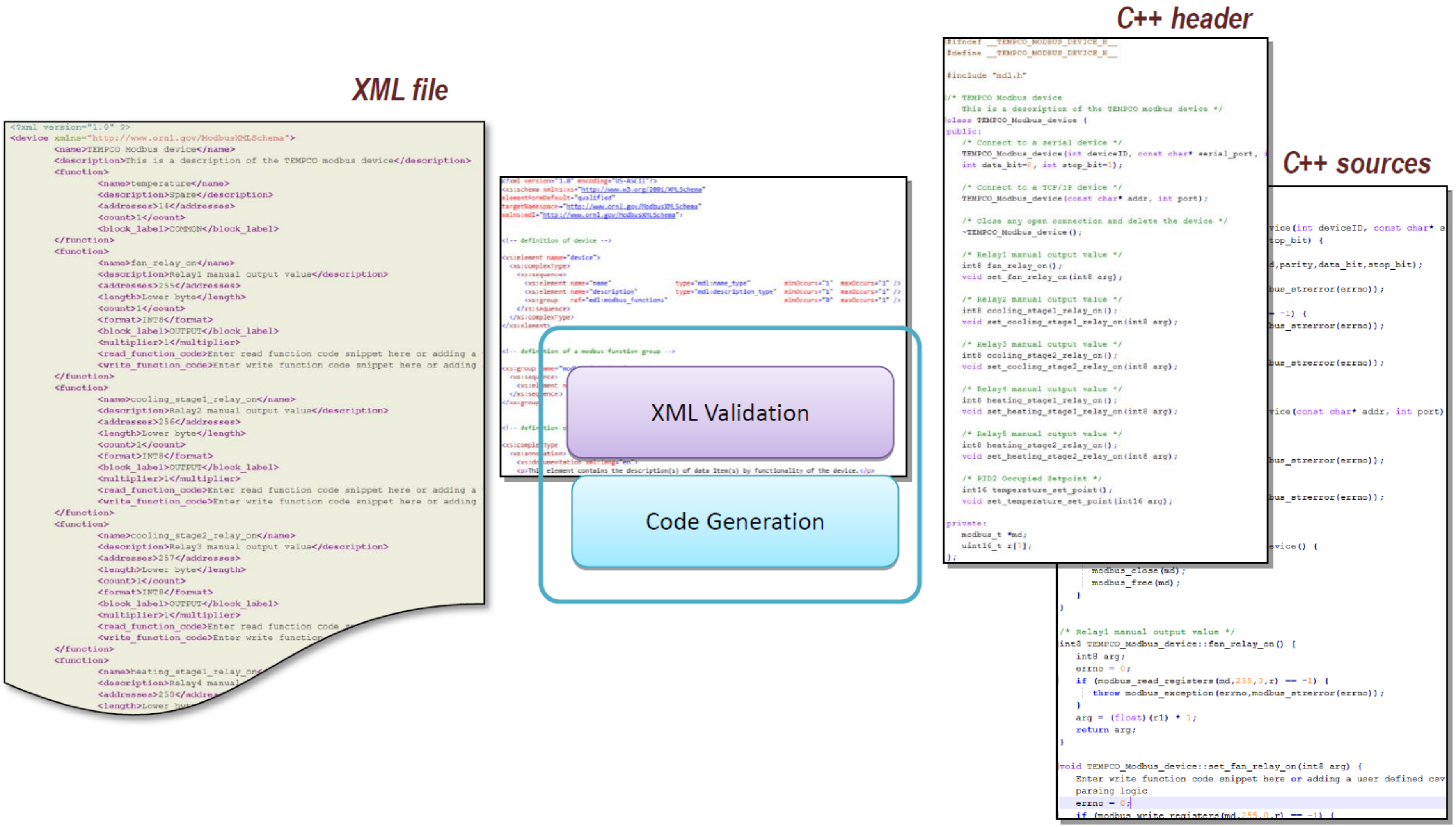
XML file



```
<?xml version="1.0" ?>
<device xmlns="http://www.ornl.gov/ModbusXMLSchema">
        <name>TEMPCO Modbus device</name>
        <description>This is a description of the TEMPCO modbus device</description>
        <function>
                <name>temperature</name>
                <description>Spare</description>
                <addresses>14</addresses>
                <count>1</count>
                <block_label>COMMON</block_label>
        </function>
        <function>
                <name>fan_relay_on</name>
                <description>Relay1 manual output value</description>
                <addresses>255</addresses>
                <length>Lower byte</length>
                <count>1</count>
                <format>INT8</format>
                <block_label>OUTPUT</block_label>
                <multiplier>1</multiplier>
                <read_function_code>Enter read function code snippet here or adding a
                <write_function_code>Enter write function code snippet here or adding
        </function>
        <function>
                <name>cooling_stage1_relay_on</name>
                <description>Relay2 manual output value</description>
                <addresses>256</addresses>
                <length>Lower byte</length>
                <count>1</count>
                <format>INT8</format>
                <block_label>OUTPUT</block_label>
                <multiplier>1</multiplier>
                <read_function_code>Enter read function code snippet here or adding a
                <write_function_code>Enter write function code snippet here or adding
        </function>
        <function>
                <name>cooling_stage2_relay_on</name>
                <description>Relay3 manual output value</description>
                <addresses>257</addresses>
                <length>Lower byte</length>
                <count>1</count>
                <format>INT8</format>
                <block_label>OUTPUT</block_label>
                <multiplier>1</multiplier>
                <read_function_code>Enter read function code
                <write_function_code>Enter write functio
        </function>
        <function>
                <name>heating_stage1_relay_on
                <description>Relay4 manua
                <addresses>258</addr
                <length>Lower by
```



MDL generation from manufacturer’s spec sheet



```xml
<?xml version="1.0" ?>
<device xmlns="http://www.ornl.gov/ModbusXMLSchema">
        <name>TEMPCO Modbus device</name>
        <description>This is a description of the TEMPCO modbus device</description>
        <function>
                <name>temperature</name>
                <description>Spare</description>
                <addresses>14</addresses>
                <count>1</count>
                <block_label>COMMON</block_label>
        </function>
        <function>
                <name>fan_relay_on</name>
                <description>Relay1 manual output value</description>
                <addresses>255</addresses>
                <length>Lower byte</length>
                <count>1</count>
                <format>INT8</format>
                <block_label>OUTPUT</block_label>
                <multiplier>1</multiplier>
                <read_function_code>Enter read function code snippet here or adding a
                <write_function_code>Enter write function code snippet here or adding
        </function>
        <function>
                <name>cooling_stage1_relay_on</name>
                <description>Relay2 manual output value</description>
                <addresses>256</addresses>
                <length>Lower byte</length>
                <count>1</count>
                <format>INT8</format>
                <block_label>OUTPUT</block_label>
                <multiplier>1</multiplier>
                <read_function_code>Enter read function code snippet here or adding a
                <write_function_code>Enter write function code snippet here or adding
        </function>
        <function>
                <name>cooling_stage2_relay_on</name>
                <description>Relay3 manual output value</description>
                <addresses>257</addresses>
                <length>Lower byte</length>
                <count>1</count>
                <format>INT8</format>
                <block_label>OUTPUT</block_label>
                <multiplier>1</multiplier>
                <read_function_code>Enter read function code
                <write_function_code>Enter write function
        </function>
        <function>
                <name>heating_stage1_relay_on
                <description>Relay4 manual
                <addresses>258</addre
                <length>Lower by
```

```xml
<?xml version="1.0" encoding="US-ASCII"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="http://www.ornl.gov/ModbusXMLSchema"
xmlns:mdl="http://www.ornl.gov/ModbusXMLSchema">


<!-- definition of device -->

<xs:element name="device">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name"               type="mdl:name_type"          minOccurs="1"  maxOccurs="1" />
      <xs:element name="description"        type="mdl:description_type"   minOccurs="1"  maxOccurs="1" />
      <xs:group   ref="mdl:modbus_functions"                              minOccurs="0"  maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>


<!-- defin tion of a modbus function group -->

<xs:group  ame="mod
  <xs:seq nce>
    <xs:el ment n
  </xs:seq ence>
</xs:group


<!-- defi tion o

<xs:comple Type
  <xs:anno ation>
    <xs:do umentation xml:lang="en">
    <p>Thi  element contains the description(s) of data item(s) by functionality of the device.</p>
```

```cpp
#ifndef __TEMPCO_MODBUS_DEVICE_H__
#define __TEMPCO_MODBUS_DEVICE_H__

#include "mdl.h"

/* TEMPCO Modbus device
   This is a description of the TEMPCO modbus device */
class TEMPCO_Modbus_device {
public:
   /* Connect to a serial device */
   TEMPCO_Modbus_device(int deviceID, const char* serial_port,
   int data_bit=8, int stop_bit=1);

   /* Connect to a TCP/IP device */
   TEMPCO_Modbus_device(const char* addr, int port);

   /* Close any open connection and delete the device */
   ~TEMPCO_Modbus_device();

   /* Relay1 manual output value */
   int8 fan_relay_on();
   void set_fan_relay_on(int8 arg);

   /* Relay2 manual output value */
   int8 cooling_stage1_relay_on();
   void set_cooling_stage1_relay_on(int8 arg);

   /* Relay3 manual output value */
   int8 cooling_stage2_relay_on();
   void set_cooling_stage2_relay_on(int8 arg);

   /* Relay4 manual output value */
   int8 heating_stage1_relay_on();
   void set_heating_stage1_relay_on(int8 arg);

   /* Relay5 manual output value */
   int8 heating_stage2_relay_on();
   void set_heating_stage2_relay_on(int8 arg);

   /* PID2 Occupied Setpoint */
   int16 temperature_set_point();
   void set_temperature_set_point(int16 arg);

private:
   modbus_t *md;
   uint16_t r[7];
};
```

```cpp
vice(int deviceID, const char* s
top_bit) {

d,parity,data_bit,stop_bit);

bus_strerror(errno));

= -1) {
bus_strerror(errno));

bus_strerror(errno));


vice(const char* addr, int port)


bus_strerror(errno));

bus_strerror(errno));


evice() {
      modbus_close(md);
      modbus_free(md);
   }
}

/* Relay1 manual output value */
int8 TEMPCO_Modbus_device::fan_relay_on() {
   int8 arg;
   errno = 0;
   if (modbus_read_registers(md,255,0,r) == -1) {
      throw modbus_exception(errno,modbus_strerror(errno));
   }
   arg = (float)(r1) * 1;
   return arg;
}

void TEMPCO_Modbus_device::set_fan_relay_on(int8 arg) {
   Enter write function code snippet here or adding a user defined csv
   parsing logic
   errno = 0;
   if (modbus_write_registers(md,255,0,r) == -1) {
```

Device driver generation

Thank You