

The ModBus XML Schema Definition Specification: Version 0.9b



Jibonananda Sanyal
James Nutaro

29 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

The ModBus XML Schema Definition Specification: Version 0.9b

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. INTRODUCTION	1
2. USE AND DESCRIPTION OF THE XML SCHEMA	2
2.1 XSD ORGANIZATION	2
2.1.1 XML Namespaces.....	3
2.1.2 Device Definition.....	3
2.1.3 Modbus Function Group.....	4
2.1.4 Data Types	4
3. CONCLUSION.....	11
4. LICENCE	12

LIST OF FIGURES

Figure	Page
Figure 1: Seamless interoperability framework.	2
Figure 2: Namespace elements used for validating the XSD.....	3
Figure 3: The highest-level XML element representing a Modbus device.....	3
Figure 4: A group of functions allows Modbus devices to support communications for sensing and control.....	4
Figure 5: The primary descriptors required for generating drive code for a Modbus device.	5
Figure 6: The name of a device can be specified.	6
Figure 7: The description of a device can be specified.	6
Figure 8: A register address must be specified for reading and/or writing values to a given device.....	7
Figure 9: A description of the byte order is necessary for understanding a device register's data.	7
Figure 10: The length of data in a device's register must be specified.	7
Figure 11: The format of a device's register information must be specified.	8
Figure 12: A block label can be used to describe at a higher level the current block's functionality.....	8
Figure 14: Some devices have field multipliers which can dramatically alter the value of a device's register.	8
Figure 14: A unit can be assigned to data communicated from/to a device's register.....	9
Figure 15: Description and actual code for reading from a device's register.	9
Figure 16: Description and actual code for writing to a device's register.	10

ACRONYMS

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

ABSTRACT

This document describes the eXtensible Markup Language (XML) Schema Definition (XSD) for interoperability of Modbus devices. This system uses a common set of XML elements to describe their input and output functions. The goal of the proposed technology is to facilitate the rapid, cost-effective retrofit integration of building automation systems by exposing the functions of sensors, actuators, and other data sources through a uniform software interface.

1. INTRODUCTION

A key challenge in retrofitting small and medium commercial buildings is the integration of legacy sensors, actuators, and other automation devices with new whole-building control solutions. Integrating legacy assets into these modern architectures can be costly if the legacy system comprises more than a handful of devices. Integration cost has two parts:

- a) Discovering what devices and interfaces are available for use
- b) Building the custom software needed to glue existing devices into the integration framework

The technology described in this report are the first major steps toward significantly reducing the costs of retrofitting small and medium commercial buildings with advanced controls by allowing equipment vendors to inexpensively and retroactively provide information required for rapid device driver generation to customers.

Advanced control of HVAC units alone has the potential to reduce whole-building energy consumption by up to 10% (0.5-1.7 of the 17 quads of energy consumed by US commercial buildings). Potential savings by better control of lighting and computing resources are less pronounced but nonetheless significant. By making retrofits cost-effective for building owners, the proposed technology will accelerate the transformation of these potential energy savings into actual energy savings.

The goal of the proposed technology is to facilitate the rapid, cost-effective retrofit integration of building automation systems by exposing the functions of sensors, actuators, and other data sources through a uniform software interface. The market is all commercial buildings and the audience includes buildings retrofit solution providers, OEM sensors and equipment manufacturers.

This document describes a standardized template for describing a device interface that can be used by vendors and application-level programmers to input the device description. A common schema allows for adaptation of existing solutions for device discovery to the selected integration description template. For proprietary protocols, it extends a path towards interoperability with commercial offerings that support device discovery.

2. USE AND DESCRIPTION OF THE XML SCHEMA

It is anticipated that the XML schema described here will be used by device vendors to state the functionality of their devices. This framework demonstrates automatic device driver generation from a device description expressed in the MDL XML format. The specification described here will facilitate the development of various methods and tools for the automatic generation of device drivers from information that will eventually be retrieved by a device discovery service and/or device description template. Where information is partial, the skeleton of a device driver can be generated to provide partial functionality for the device as well as reduce the labor required to produce a complete device driver.

Figure 1 illustrates the envisioned interoperability and automatic framework that allows device-discovery and a common device interface that allows multiple devices to connect and communicate seamlessly.

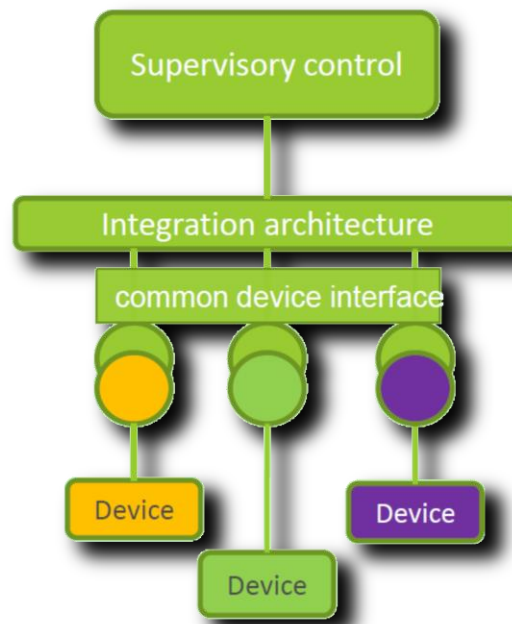


Figure 1: Seamless interoperability framework.

2.1 XSD ORGANIZATION

At a high level, the XML Schema Definition (XSD) has a single root element representing the Modbus device. It consists of the device name, description, and a set of functions organized as an `xs:group`. The `xs:group` consists of XML elements that represent read, write, or both read-write functionality of the registers.

Modbus device vendors supply register tables that most often list device addresses and their corresponding functionality. The XML schema is designed based instead on the device's supported functionality. As such, the XSD proposes a functional listing of the registers of the device instead of the more conventional table of registers and their functionality. Expressing a list of functions of the Modbus device allows for a higher level of abstraction that can be used by various programs and tools to generate device drivers which encapsulate the necessary, register-level communication aspects. The register

address and other information to read the value from a device or to write to a device are all encapsulated in the `xs:element` representing the corresponding function.

The XSD also abstracts the data types for each of the constituent elements of functions. This allows for flexibility in enforcing proper XML syntax. It also allows for greater flexibility in managing changes as this specification evolves to different market requirements.

The following sections describe the different parts of the XML schema in more detail. Note that this has only been tested on a TEMCP ModBus device.

2.1.1 XML Namespaces

The XSD starts with stating the XML version and the supported encoding. At the moment, only US-ASCII characters are supported. The `targetNamespace` and an XML namespace of `xmlns:mdl` is defined for the rest of the document.

```
<?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" >
```

Figure 2: Namespace elements used for validating the XSD

2.1.2 Device Definition

The namespace elements are followed by the root element which is the anchor point for the entire schema and corresponds to the Modbus device being described. The element constitutes of a ‘name’ element, a ‘description’ element, and refers to an `xs:group` named ‘`modbus_functions`’ consisting of Modbus functions. Both the ‘name’ and ‘description’ elements have custom data types which are described later in this document and both are required elements. The ‘`modbus_functions`’ group may occur at most once in the body of the device element.

```
<!-- 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>
```

Figure 3: The highest-level XML element representing a Modbus device.

2.1.3 Modbus Function Group

The ‘modbus_function’ group is an xs:group which encapsulates a sequence of Modbus function elements and is referred to from the device element of the XML. Each element in the sequence is an instance of the data type ‘mdl:function_type’. An instance of ‘function_type’ describes a function of the Modbus device.

```
<!-- definition of a 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>
```

Figure 4: A group of functions allows Modbus devices to support communications for sensing and control.

2.1.4 Data Types

All elements use a derived data type. This allows one to enforce explicit rules on acceptable element values. This design also allows flexibility with respect to future schema changes. The following are the various xs:element data types in the schema.

2.1.4.1 Function Type Definition

The ‘function_type’ encapsulates a read, write, or a read-write element of the schema and is a fundamental XML element which encapsulates a function of the Modbus device. An xs:sequence of ‘function_type’ instances are used to define the ‘modbus_functions’ entry for a device. The ‘function_type’ is an xs:complexType and constitutes of a sequence of the following elements:

- i. A required ‘name’ element of ‘md:name_type’.
- ii. A required ‘description’ element of ‘md:description_type’.
- iii. A list of register addresses, of ‘mdl:address_list_type’. This is a required entry and the values should be separated by spaces.
- iv. A length element describing the length of the register and is of ‘mdl:length_enum_type’. This is an optional entry and defaults to ‘Full word’.
- v. A count element of ‘mdl:count_type’ describing the number of ‘length’ elements for the register. This is an optional entry and defaults to 1.
- vi. A format element of ‘mdl:format_enum_type’ describing the type of native data type of the register. It is an optional element and defaults to INT8.
- vii. An optional block label of ‘mdl:block_label_type’. Some vendors group their registers into named categories. This element describes the vendor grouping, if any, and may occur any number of times.

- viii. An optional multiplier element of 'mdl:multiplier_type' which is used by the code generator as a default scaling factor in the absence of a verbose read or write function description. It defaults to a value of 1.0.
- ix. An optional units element of 'mdl:units_type' describing unit of measure, if applicable.
- x. An optional element describing the read functionality used by the code generator to create the device driver.
- xi. An optional element describing the write functionality used by the code generator to create the device driver.

```

<!-- 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 >

```

Figure 5: The primary descriptors required for generating drive code for a Modbus device.

2.1.4.2 Name Type

This data type is an xs:simpleType that allows only string values.

```
<!-- definition of data types -->
<xs:simpleType name="name_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>This is the name of the device or function, and it will
        be translated into the name of a class that reads from and
        writes to the device.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Figure 6: The name of a device can be specified.

2.1.4.3 Description Type

This data type is an xs:simpleType that allows only string values.

```
<xs:simpleType name="description_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>This is the description of the device or function, and it
        will be translated into the comments of a class and/or
        instructions for code that reads from and writes to the
        device.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Figure 7: The description of a device can be specified.

2.1.4.4 Address List Type

This data type is an xs:simpleType that allows only a list of non-negative integers separated by spaces.

```
<xs:simpleType name="address_list_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>A list of addresses for this register set.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:list itemType="xs:nonNegativeInteger"/>
</xs:simpleType>
```


Figure 8: A register address must be specified for reading and/or writing values to a given device.

2.1.4.5 Length Enumeration Type

This data type is an xs:simpleType that allows only string values ‘Lower byte’, ‘Upper byte’ and ‘Full word’.

```
<xs:simpleType name="length_enum_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>The length of each word that must be read to retrieve
      the data.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Lower byte"/>
    <xs:enumeration value="Upper byte"/>
    <xs:enumeration value="Full word"/>
  </xs:restriction>
</xs:simpleType>
```

Figure 9: A description of the byte order is necessary for understanding a device register’s data.

2.1.4.6 Count Type

This data type is an xs:simpleType allows only positive integers.

```
<xs:simpleType name="count_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>The number of 16 bit words that must be read to retrieve
      all of the data.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>
```

Figure 10: The length of data in a device’s register must be specified.

2.1.4.7 Format Enumeration type

This format enumeration data type is an xs:simpleType that allows only string values ‘INT8’, ‘UINT8’, ‘INT15’, ‘UINT16’, ‘INT32’, and ‘UINT32’.

```

<xs:simpleType name="format_enum_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>The name of Modbus value representation format that will be
      translated to a C language primitive type that will hold the data
      value in an application program.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="INT8"/>
    <xs:enumeration value="UINT8"/>
    <xs:enumeration value="INT16"/>
    <xs:enumeration value="UINT16"/>
    <xs:enumeration value="INT32"/>
    <xs:enumeration value="UINT32"/>
  </xs:restriction>
</xs:simpleType>

```

Figure 11: The format of a device's register information must be specified.

2.1.4.8 Block Label Type

This is an xs:simpleType allows only a string value.

```

<xs:simpleType name="block_label_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>The name of the Modbus block/label.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>

```

Figure 12: A block label can be used to describe at a higher level the current block's functionality.

2.1.4.9 Multiplier Type

This is an xs:simpleType that allows only a floating point number..

```

<xs:simpleType name="multiplier_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>Value of scaling multiplier.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>

```

Figure 13: Some devices have field multipliers which can dramatically alter the value of a device's register.

2.1.4.10 Units Type

This is an xs:simpleType that allows only string values.

```
<xs:simpleType name="units_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>The name of the output units.</p>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Figure 14: A unit can be assigned to data communicated from/to a device's register.

2.1.4.11 Read Function Type

This is an xs:simpleType that allows only a string value. Ideally, it should embody the C code to be used in the driver generation.

```
<xs:simpleType name="read_function_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>This is a fragment of C code that converts
the register values into a data item that is
useful to the application program. This fragment
must be complete except for the definition
of the register variables and the return
argument variable. The register variables are
of type uint16_t and are named r1, r2, ....
The return argument variable has the name arg.
For example, the following divides register
number 1 by 10 and returns the result as a float.</p>
      <br><br>
      <verbatim>arg = (float)r1/10.0f;</verbatim>
    </br></br>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Figure 15: Description and actual code for reading from a device's register.

2.1.4.12 Write Function Type

This is an xs:simpleType comprising of a string value. Ideally, it should embody the C code to be used in the driver generation.

```

<xs:simpleType name="write_function_type">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <p>This is a fragment of C code that converts
      an application supplied argument to register values
      that can be written to the modbus device.
      This fragment must be complete except for the definition
      of the register variables and the return
      argument variable. The register variables are
      of type uint16_t and are named r1, r2, ....
      The return argument variable has the name arg.
      For example, the following multiplies the application
      argument by 10 and the copies it to register
      number 1.</p>
      <br><br>
      <verbatim>r1 = (uint16_t)(arg/10.0f);</verbatim>
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>

```

Figure 16: Description and actual code for writing to a device's register.

3. CONCLUSION

The document describes the XML schema for enabling new device discovery protocols that can overlay legacy technologies and be incorporated into new automation devices. This will help resolve current communication protocol standards used within buildings which support a device discovery procedure but are not uniformly implemented by vendors. The XML schema offers a standardized technique for device enumeration.

The XML schema described here is accompanied by additional software, utility tools, and documentation that walks through the process of device driver creation. The additional utility tools provide parsers that support conversion of existing register data tables to XML files from which device driver software is generated. This is provided to make the use of the XML schema easier for vendors easier to convert to and adopt.

4. 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.