The MODBUS XML Schema Definition Specification: Version 0.9b

Jibonananda Sanyal
James Nutaro

1 September, 2014

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

Energy and Transportation Science Divsion

The MODBUS XML Schema Definition Specification: Version 0.9b

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

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 the eXtended Markup Language (XML) Schema definition (XSD) for interoperability of Modbus devices using a common and expressive 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 proposed technology will significantly reduce 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 device interface that can be used by vendors and application-level programmers to input the device description. A common schema allows for adapting existing solutions for device discovery to the selected integration description template. For proprietary protocols, it extend 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 will allow for the automatic driver generation from the device description for a given application framework. This will facilitate the development of various methods and tools employing the methods for the automatic generation of device drivers from information retrieved by the 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, reduce the labor required to produce a complete device driver, or both.

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.

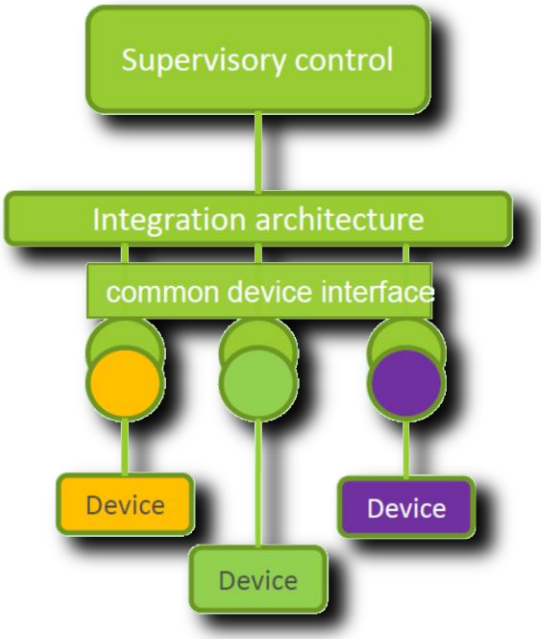

Figure 1: Seamless interoperability framework.

2.1 XSD ORGANIZATION

At a high level, the XML Schema 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 with functionality of device. To this effect, 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 register level communication aspects. The register address and other information to read the value off the device or to write to the device are all encapsulated in the xs:element representing the corresponding function.

The XSD also abstracts out the data types for each of the constituent elements of functions. This allows for flexibility in enforcing compliance in the XML encoding. 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.

2.1.1 XML Namespaces

The XSD starts with stating the XML version and the supported encoding. At the moment, only US-ASCII is supported encoding. 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 in the XSD

2.1.2 Device Definition

The namespace elements are followed by the root element which anchors the entire schema. The root elemet corresponds to the Modbus device and 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: Device definition section

2.1.3 Modbus Function Group

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

2.1.4 Data Types

All elements use a derived data type which allows enforcing explicit rules on acceptable values for the elements. This design also allows for future schema changes. Th e 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 th schema. It is also the fundamental element encapsulating a function of the Modbus device. An xs:sequence of instances of the ‘function_type’ constitutes the ‘modbus_functions’ entry in a device element.

i. The ‘function_type’ is an xs:complexType and constitutes of a sequence of the following elements:

ii. A required ‘name’ element of ‘md:name_type’.

iii. A required ‘description’ element of ‘md:description_type’.

iv. A list of register addresses, of ‘mdl:address_list_type’. This is a required entry and the values should be separated by spaces.

v. 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’.

vi. 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.

vii. 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.

viii. 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.

ix. 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.

x. An optional units element of ‘mdl:units_type’ describing unit of measure, if applicable.

xi. An optional element describing the read functionality used by the code generator to create the device driver.

xii. 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 function data type.

2.1.4.2 Name Type

This data type is an xs:simpleType with a restriction of string values only.

```
<!-- 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 data type.

2.1.4.3 Description Type

This data type is an xs:simpleType with a restriction of string values only.

```
<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 data type.

2.1.4.4 Address List Type

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

```
<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: The address list data type.

2.1.4.5 Length Enumeration Type

This data type is an xs:simpleType with a restriction of the the string values ‘Lower byte’, ‘Upper byte’ and ‘Full word’, which makes it an enumeration type.

```
<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: The length enumeration type.

2.1.4.6 Count Type

This data type is an xs:simpleType with a restriction of positive integer values only.

```
<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 count data type.

2.1.4.7 Format Enumeration type

This format enumeration data type is an xs:simpleType with a restriction of the the string values ‘INT8’, ‘UINT8’, ‘INT15’, ‘UINT16’, ‘INT32’, and ‘UINT32’, which makes it an enumeration type.

```
<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 enumeration type.

2.1.4.8 Block Label Type

This is an xs:simpleType comprising of 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: The block label data type.

2.1.4.9 Multiplier Type

This is an xs:simpleType comprising of a floating point value.

```
<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: The multiplier data type.

2.1.4.10 Units Type

This is an xs:simpleType comprising of a string value.

```
<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: The units data type.

2.1.4.11 Read 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="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: The read function data type.

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>
            </br></br>
            </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Figure 16: The write function data type.

3. CONCLUSION

The document describes the XML sschema for developing new device discovery protocols that can overlay legacy technologies and be incorporated into new automation devices. This will help in resolving current communication protocol standards used within buildings which support a device discovery procedure that 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 drier 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.

4. LICENCE