Parsing MTConnect XML in C++

John Michaloski

NIST

May 30, 2014

# Background

Understanding the need for a standard communication in manufacturing, the Association for Manufacturing Technology (AMT), sponsors of the International Manufacturing Technology Show (IMTS), originally sponsored MTConnect. AMT felt that to meet “Manufacturing in the Internet Age” the challenge was for AMT to get manufacturers to embrace interoperability standards so that customers could monitor various brands and models of CNC machines through a common interface. Synthesizing manufacturing data is difficult since all the necessary bits and pieces of production data can be in different file formats and representations. MTConnect handles production communication in a standard XML format via HTTP and is developing information models to standardize manufacturing concepts. In 2009, the MTConnect Institute, which is a not-for-profit 501(c)(6) organization, was established to further the development of the MTConnect Standard.



Figure 1 MTConnect Architecture

Figure 1 shows the basic system architecture of the MTConnect standard. An “MTConnect Device” is a piece of equipment organized as a set of components that provide data. The core of MTConnect is the “Agent”, which is a process that acts as a “bridge” between a factory device and a “Client Application”. The MTConnect Agent receives and stores single or a time series of data samples or events that are made available to the Client Application. Clients use the standard communication and information model to collect data that can then be turned into production knowledge. These client transport mechanisms can be personal computers (PC), CNC, machines, cell phones, devices, wireless devices, or any mechanism that supports communication via http.

MTConnect supports a hierarchical information model of factory equipment in which the data is organized into an XML tree-like structure. The tree structure allows representing information using parent/child relationships: each parent can have many children, but each child has only one parent. Thus the parent node <DEVICE> can support many child <DEVICES> nodes. This tree model is used exclusively to model MTConnect and the XML must conform to either the Streams or Probe XML Schema Definition (XSD) defined by MTConnect[[1]](#footnote-1).

Clients parse the MTConnect Agent XML data to retrieve information. Typically, a variant of DOM is used to read from a URL and then parse the XML (e.g., MSXML or Xerces).Figure 2 shows an overview of MTConnect Agents – it gathers data from one or more devices and has the ability to display via XML either the Electronic Data Sheet (describing the device) with an http probe or can stream XML data items that provides the latest device information. Readers are encouraged to visit the MTConnect Github repositories to learn more about MTConnect. When the Document Object Model (DOM) is used on an XML file, it parses the file, breaking the file out into individual elements (such as <DEVICE>, <DEVICES>), and attributes, and other XML items.

This document describes how to parse MTConnect Agent XML using the C++ programming language and the MSXML and Xerces tools.



Figure MTConnect Pseudo UML Overview

## XML Parsing Background

An XML file contains text with interspersed descriptions, called "tags." The XML file is organized in a hierarchy with an "open" tag at the beginning of the file, a "close" tag at the end and all the text elements in between. Tags begin with the less-than (<) character and end with the greater-than (>) character. Close tags have a </ character sequence. The basic building blocks of XML are elements and attributes. Elements describe data, whereas attributes provide additional definition about the element. An XML Schema (XSD) defines the valid content that elements and attribute may exhibit. Numerous toolkits exist to parsing XML (with or without using XML schemas) for most every language. Document Object Mode (DOM) and SAX are the primary methods for parsing XML.

DOM represents an entire XML Document into tree format with each element representing tree branches. DOM Parser creates an In Memory tree representation of XML file and then parses it. In DOM, there are no events triggered while parsing the XML. The entire XML is parsed and a DOM tree (with nodes being the XML branches) is generated. Once parsed a DOM standard API exists to access any element in the tree, so that the user can navigate the tree to access the data in the various nodes in the XML. .Parsing XML using a DOM parser is quite fast if XML file is small it is efficient. For most MTConnect clients, DOM is sufficient since the XML size is relatively small. The DOM API allows for easy and selective navigation and has the XPATH search mechanism integrated into most DOM toolkits which simplifies XML parsed tree navigating and branch extraction. All of the examples demonstrated in this document use DOM.

Unlike DOM, Sax does not loads the entire XML into memory before parsing it, nor does it creates any type of object from XML. In SAX, events are triggered when the XML is being parsed. When the parser is parsing the XML, and encounters a tag starting (e.g. <something>), then it triggers the tagStarted event (actual name of event might differ). Similarly when the end of the tag is met while parsing (</something>), it triggers tagEnded. Using a SAX parser implies you need to handle these events and make sense of the data returned with each event.

# Parsing MTConnect XML using C++ Xerces

Existing tutorials on Xerces exist on the internet. The following web sites offer background and explanation on basic usage of reading (and writing) of XML using Xerces. The sites include:

1. <http://www.yolinux.com/TUTORIALS/XML-Xerces-C.html>
2. http://www.codeproject.com/Articles/31088/Xerces-for-C-Tutorial-Using-Visual-C
3. <http://www.codeproject.com/Articles/32762/Xerces-for-C-Using-Visual-C-Part-2>
4. <http://www.ibm.com/developerworks/library/x-xercc/index.html>
5. <http://advancedcppwithexamples.blogspot.com/2012/10/xerces-for-c-using-visual-studio-2010.html>

## Installation

Xerces is platform neutral XML DOM parsing tool for C++. Xerces is a collection of software libraries for parsing, validating, serializing and manipulating XML. The library implements a number of standard APIs for XML parsing, including DOM, SAX and SAX2. The implementation is available in Java, C++ and Perl programming languages. Note, xerces XPath evaluation is only available only in Xerces version 3.0, such tthat, Xercses 2.0 would require another XPath software installation to be downloaded and used.

Below steps for Installing Xerces for Visual C++ 2008 (VC-9) are outlined below:

1. Download: xerces-c-3.1.1-x86-windows-vc-9.0.zip [PGP] [MD5] (2008 VC++)

xerces-c-3.1.1-x86-windows-vc-9.0.zip

from

<http://xerces.apache.org/xerces-c/download.cgi>

1. Unzip to C:\Program Files
2. Add to source include path: C:\Program Files\xerces-c-3.1.1-x86-windows-vc-9.0\include
3. Add library path: C:\Program Files\xerces-c-3.1.1-x86-windows-vc-9.0\lib
4. Copy dll to exe directory, either xerces-c\_3\_1.dll(Release build) or xerces-c\_3\_1D.dll (Debug), or add Xerces C:\Program Files\xerces-c-3.1.1-x86-windows-vc-9.0\lib to path environment variable.
5. Statically linking with Xerces will be explained.

## XML Sample

.

The following is a sample MTConnect version 1.2 compatible sample XML file that is can be used to test Xerces XML parsing.

<?xml version="1.0" encoding="UTF-8"?>

<MTConnectStreams xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:mtconnect.org:MTConnectStreams:1.2" xmlns:m="urn:mtconnect.org:MTConnectStreams:1.2" xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.2 http://www.mtconnect.org/schemas/MTConnectStreams\_1.2.xsd">

<Header lastSequence="74" firstSequence="1" nextSequence="75" bufferSize="131072" version="1.2.0.0" instanceId="1401459991" sender="mountaineer" creationTime="2014-0530T14:26:44Z"/>

<Streams>

<DeviceStream uuid="M1xxx" name="M1">

<ComponentStream name="SPINDLE" componentId="M1d1" component="Rotary">

<Samples>

<SpindleSpeed name="Srpm" sequence="73" timestamp="2014-05-30T14:26:43.0996Z" dataItemId="M1cs1">99</SpindleSpeed>

</Samples>

</ComponentStream>

<ComponentStream name="electric" componentId="M1elec" component="Electric">

<Events>

<PowerState name="power" sequence="74" timestamp="2014-05-30T14:26:43.0999Z" dataItemId="M1pwr">ON</PowerState>

</Events>

</ComponentStream>

<ComponentStream name="M1" componentId="M1id" component="Device">

<Events>

<Availability name="avail" sequence="44" timestamp="2014-05-30T14:26:32.0030Z" dataItemId="M1avail">AVAILABLE</Availability>

<AssetChanged sequence="6" timestamp="2014-05-30T14:26:31.0590Z" dataItemId="M1id\_asset\_chg" assetType="">UNAVAILABLE</AssetChanged>

</Events>

</ComponentStream>

<ComponentStream name="path" componentId="M1path1" component="Path">

<Samples>

<PathFeedrate name="path\_feedrateovr" sequence="59" timestamp="2014-05-30T14:26:39.0958Z" dataItemId="M1pfo1" subType="OVERRIDE">100</PathFeedrate>

</Samples>

<Events>

<Code name="alarm" sequence="54" timestamp="2014-05-30T14:26:36.0921Z" dataItemId="M1alarm">0</Code>

<Execution name="execution" sequence="53" timestamp="2014-05-30T14:26:36.0920Z" dataItemId="M1exec">EXECUTING</Execution>

<Code name="heartbeat" sequence="69" timestamp="2014-05-30T14:26:43.0991Z" dataItemId="M1heartbeat">2</Code>

<Message name="last\_update" sequence="51" timestamp="2014-05-30T14:26:36.0896Z" dataItemId="M1last\_update" nativeCode="5/29/2014 14:49">5/29/2014 14:49</Message>

<ControllerMode name="controllermode" sequence="52" timestamp="2014-05-30T14:26:36.0918Z" dataItemId="M1mode">AUTOMATIC</ControllerMode>

<Program name="program" sequence="60" timestamp="2014-05-30T14:26:39.0960Z" dataItemId="M1pgm">\_N\_1041\_MPF</Program>

<ToolId name="Tool\_number" sequence="61" timestamp="2014-05-30T14:26:39.0962Z" dataItemId="M1tid1">7</ToolId>

</Events>

</ComponentStream>

<ComponentStream name="X" componentId="M1x1" component="Linear">

<Samples>

<Position name="Xabs" sequence="70" timestamp="2014-05-30T14:26:43.0992Z" dataItemId="M1xp1" subType="ACTUAL">0.3</Position>

</Samples>

</ComponentStream>

<ComponentStream name="Y" componentId="M1y1" component="Linear">

<Samples>

<Position name="Yabs" sequence="71" timestamp="2014-05-30T14:26:43.0994Z" dataItemId="M1yp1" subType="ACTUAL">10.3</Position>

</Samples>

</ComponentStream>

<ComponentStream name="Z" componentId="M1z1" component="Linear">

<Samples>

<Position name="Zabs" sequence="72" timestamp="2014-05-30T14:26:43.0995Z" dataItemId="M1zp1" subType="ACTUAL">20.3</Position>

</Samples>

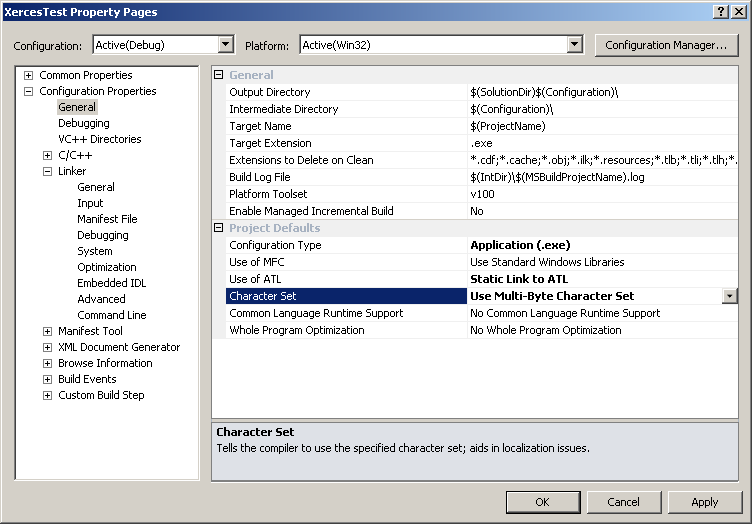
</ComponentStream>

</DeviceStream>

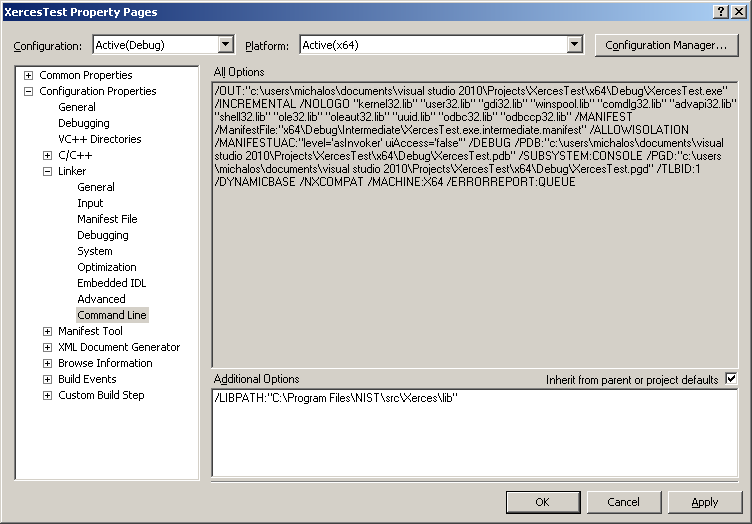
</Streams>

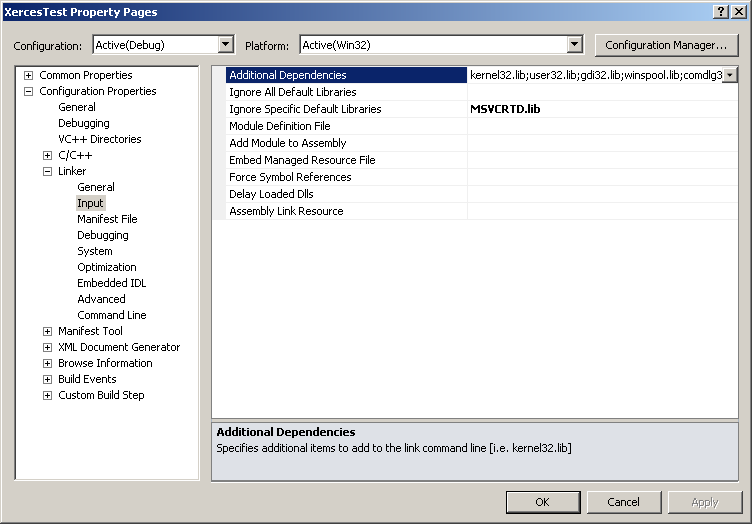
</MTConnectStreams>

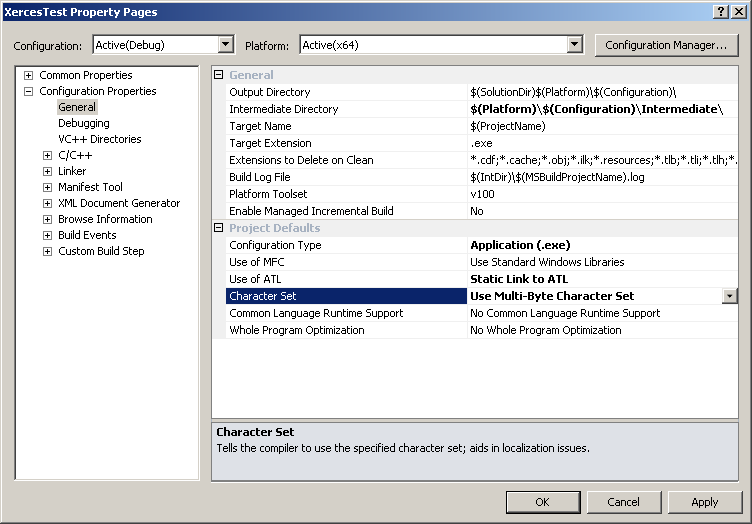
Note, only Win32 (32-bit) implementations are supported. There is no out of the box, 64-bit Xercs lib to link against, so it is not advised. It is also suggested that you use Multibyte characters instead of Unicode although both can be used. The Visual C++ 2010 setting for this is:



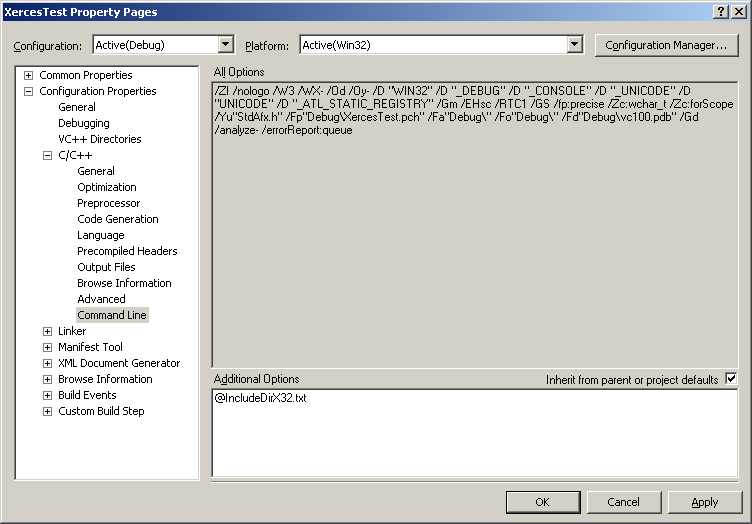
Above, the preprocessor definition, XERCES\_STATIC\_LIBRARY, tells Xerces DOM compiler that you want to link with a static library and do not wish to register a dll. The pragma “#pragma comment(lib, "xerces-c\_static\_3D.lib")” tells Microsoft Visual C++ 2010 to include and link uses the C++ xerces static library. Note 1) the library path of this library must be included and 2) the MSVCRT{D}.lib must not be included in the project, as shown below:







To make the include files found within the search path add the following command to the advanced line of the C++ Compiler settings:



Below the basics for setting up and parsing a given MTConnect file. The code illustrates basic setup for DOM parsing, without concern for namespaces, which are generally tricky.

std::string configFile = "C:\\Users\\michalos\\Documents\\Visual Studio 2010\\Projects\\XercesTest\\XercesTest\\Win32\\Debug\\TestData.xml";

XercesDOMParser \* m\_ConfigFileParser = new XercesDOMParser;

m\_ConfigFileParser->setValidationScheme( XercesDOMParser::Val\_Never );

m\_ConfigFileParser->setDoNamespaces( false );

m\_ConfigFileParser->setDoSchema( false );

m\_ConfigFileParser->setLoadExternalDTD( false );

m\_ConfigFileParser->parse( configFile.c\_str() );

Excerse provides underlying DOM (Document Object Model) that allows other XML facilities (besides parsing into the native C++ tree structure.) Foremost, is the ability to run XPATH queries on the XML tree. You are required to have version 3.0 to run XPath using Xerces!

XPath, the XML Path Language, is a query language for selecting nodes from an XML document. XPath 2.0 is the current version of the language, note Xerces version < 3.0 can only handle XPATH 1.0 queries. XPath is used to navigate through elements and attributes in an XML document. XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

|  |  |
| --- | --- |
| **Expression** | **Description** |
| *nodename* | Selects all child nodes of the named node |
| / | Selects from the root node |
| // | Selects nodes in the document from the current node that match the selection no matter where they are |
| . | Selects the current node |
| .. | Selects the parent of the current node |
| @ | Selects attributes |

Below is C++ code to build and run an Xpath query expression using Xerces to retrieve XPaths in the XML tree on the code. Of note, XMLString is a Xerces mechanism to handle string as 16-bit characters (i.e., not ascii) so that transcode function is needed to translate into/out of the Xerces representation. Also, XML namespaces are handled directly in the XPATH query – but are not necessary in MTConnect.

// XPATH Sample

std::vector<DOMNode\*> CXercesParsing::FindXPathMatches(XERCES\_CPP\_NAMESPACE::DOMDocument\* p\_DOMDocument, std::string element)

{

XMLCh\* xpathStr;

std::vector<DOMNode\*> nodes ;

try

{

xpathStr=XMLString::transcode(element.c\_str());

XERCES\_CPP\_NAMESPACE::DOMElement\* domroot = static\_cast<XERCES\_CPP\_NAMESPACE::DOMElement\*> (p\_DOMDocument->getDocumentElement());

XERCES\_CPP\_NAMESPACE::DOMXPathNSResolver\* resolver=p\_DOMDocument->createNSResolver(domroot);

XERCES\_CPP\_NAMESPACE::DOMXPathResult\* result=p\_DOMDocument->evaluate(

xpathStr,

domroot,

resolver,

xercesc::DOMXPathResult::ORDERED\_NODE\_SNAPSHOT\_TYPE,

NULL);

XMLSize\_t nLength = result->getSnapshotLength();

for(XMLSize\_t i = 0; i < nLength; i++)

{

result->snapshotItem(i);

DOMNode\* node = result->getNodeValue();

std::cout << XMLString::transcode( node->getTextContent() ) << std::endl;

nodes.push\_back( node );

}

result->release();

resolver->release ();

}

catch(const DOMXPathException& e)

{

XERCES\_STD\_QUALIFIER cerr << "An error occurred during processing of the XPath expression. Msg is:"

<< XERCES\_STD\_QUALIFIER endl

<< XMLString::transcode(e.getMessage()) << XERCES\_STD\_QUALIFIER endl;

}

catch(const DOMException& e)

{

XERCES\_STD\_QUALIFIER cerr << "An error occurred during processing of the XPath expression. Msg is:"

<< XERCES\_STD\_QUALIFIER endl

<< XMLString::transcode(e.getMessage()) << XERCES\_STD\_QUALIFIER endl;

}

std::string str = XMLString::transcode( xpathStr );

XMLString::release(&xpathStr);

return nodes;

}Parsing MTConnect XML using C++ MSXML

The following code uses XPATH and to read an MTConnect XML parsed tree. Get attributes indivually from a node is a handy utility, and has been coded up as GetAttribute. Of note, in Xerces often the node is not an element node so when it is cast to DOMElement it is null, and therefore has nothing much less no elements. The routine GetMTConnectData retrieves all the samples and events in an MTConnect XML parsed tree and returns name/value stl mapped pairs. This is useful for reading an entire tree and determining the current values.

std::string CXercesParsing::GetAttribute(DOMNode\* node, std::string attribute)

{

XMLCh\* xpathStr=XMLString::transcode(attribute.c\_str());

std::string text;

DOMElement\* currentElement = dynamic\_cast< xercesc::DOMElement\* >( node );

if(currentElement==NULL)

return text;

const XMLCh\* xmlch\_OptionA = currentElement->getAttribute(xpathStr);

text = XMLString::transcode(xmlch\_OptionA);

return text;

}

std::map<std::string,std::string> CXercesParsing::GetMTConnectData(XERCES\_CPP\_NAMESPACE::DOMDocument\* p\_DOMDocument)

{

std::map<std::string,std::string> data;

std::string items[2] = {std::string(".//Samples"), std::string(".//Events") };

for(int ii=0; ii<2 ; ii++)

{

std::vector<DOMNode\*> samples = FindXPathMatches(p\_DOMDocument, items[ii]);

for(int j=0; j< samples.size(); j++)

{

DOMNode\* pSampleHive = samples[j];

// Get each child

DOMNodeList\* children = pSampleHive->getChildNodes();

const XMLSize\_t nodeCount = children->getLength();

for(XMLSize\_t k=0; k< nodeCount; k++)

{

DOMNode\* pSample = children->item(k);;

if( pSample->getNodeType()==NULL && // true is not NULL

pSample->getNodeType() != DOMNode::ELEMENT\_NODE ) // is element

{

continue;

}

if(XMLString::transcode( pSample->getNodeName() )=="#text")

continue;

std::string name ;

std::string value;

name = GetAttribute(pSample, "name");

if(name.empty())

name = GetAttribute(pSample, "dataItemId");

if(name.empty())

continue;

value = XMLString::transcode(pSample->getTextContent());

data[name]= value;

}

}

}

return data;

}

## Installation

MSXML 6.0 is typically part of a windows distribution and does not need to be installed. It is not portable, and does not have a Linux port.

## XML Sample

.The following is a sample MTConnect XML file from

Code to parse this, and output some of the parameters and values to a stl map follows:

//

// MTConnectStreamsParser.cpp – Example MSXML parsing of MTConnect XML

//

// DISCLAIMER:

// This software was developed by U.S. Government employees as part of

// their official duties and is not subject to copyright. No warranty implied

// or intended.

#include "StdAfx.h"

#include "MTConnectStreamsParser.h"

#pragma comment(lib, "comsuppw.lib")

#pragma comment(lib, "Ws2\_32.lib")

#pragma comment(lib, "Iphlpapi.lib")

std::map<std::string, std::string> MTConnectStreamsParser::\_TagRenames;

std::vector<std::string>   MTConnectStreamsParser::\_saveddata;

std::map<std::string, std::string> MTConnectStreamsParser::\_valuerenames;

ptime GetDateTime(std::string s)

{

       // parse 2012-02-03T17:31:51.0968Z

       int Year, Month, Day, Hour, Minute, Second, Millisecond=0;

       if(sscanf(s.c\_str(), "%d-%d-%dT%d:%d:%d.%d", &Year, &Month, &Day, &Hour, &Minute,&Second, &Millisecond)==7){}

              //12/2/2009 2:42:25 PM

       else if(sscanf(s.c\_str(), "%d-%d-%dT%d:%d:%d", &Year, &Month, &Day, &Hour, &Minute,&Second)==6){}

       else if(sscanf(s.c\_str(), "%d/%d/%4d%d:%d:%d", &Month, &Day, &Year,  &Hour, &Minute,&Second)==6){}

       else throw std::exception("Unrecognized date-time format\n");

       return ptime( date(Year,Month,Day),

                  hours(Hour)  +

                  minutes(Minute) +

                  seconds(Second) +

                  boost::posix\_time::millisec(int(Millisecond)) );

       //return COleDateTime( Year, Month, Day, Hour, Minute, Second );

}

static \_bstr\_t checkParseError(MSXML2::IXMLDOMParseErrorPtr pError)

{

       \_bstr\_t parseError =\_bstr\_t("At line ")+ \_bstr\_t(pError->Getline()) + \_bstr\_t("\n")+ \_bstr\_t(pError->Getreason());

       //MessageBox(NULL,parseError, "Parse Error",MB\_OK);

       return parseError;

}

static        void dump\_com\_error(\_com\_error &e)

{

       ::AtlTrace("Error\n");

       //TRACE1("\a\tCode = %08lx\n", e.Error());

       //TRACE1("\a\tCode meaning = %s", e.ErrorMessage());

       \_bstr\_t bstrSource(e.Source());

       \_bstr\_t bstrDescription(e.Description());

       ::AtlTrace("\a\tSource = %s\n", (LPCSTR) bstrSource);

       ::AtlTrace("\a\tDescription = %s\n", (LPCSTR) bstrDescription);

}

MTConnectStreamsParser::MTConnectStreamsParser(void)

{

}

MTConnectStreamsParser::~MTConnectStreamsParser(void)

{

}

void MTConnectStreamsParser::Release(void)

{

       m\_pSchemaCache=NULL;

       m\_pXSDDoc=NULL;

       m\_pXMLDoc=NULL;

}

\_bstr\_t MTConnectStreamsParser::GetAttribute(MSXML2::IXMLDOMNodePtr node, \_bstr\_t attribute)

{

       \_bstr\_t text= L"";;

       CComPtr<MSXML2::IXMLDOMNamedNodeMap> attributes;

       node->get\_attributes( &attributes );

       if( attributes )

       {

              MSXML2::IXMLDOMNodePtr attr = attributes->getNamedItem(attribute);

              if(attr==NULL)

                     return text;

              if(attr->nodeValue.vt == VT\_BSTR)

                     return attr->nodeValue.bstrVal;

       }

       return text;

}

std::string MTConnectStreamsParser::ParseXMLDocument(\_bstr\_t xmlfile, \_bstr\_t xsdfile, \_bstr\_t xsdname)

{

       \_bstr\_t parseError(L"");

       try{

              IXMLDOMParseErrorPtr  pError;

              // load the XML file

              // \*\*\*\*\*\* you need to use IXMLDOMDocument2 interface \*\*\*\*\*\*\*\*\*

              HRESULT hr = m\_pXMLDoc.CreateInstance(\_\_uuidof(MSXML2::DOMDocument));

              m\_pXMLDoc->async =  VARIANT\_FALSE;

              hr = m\_pXMLDoc->load(xmlfile);

              //check on the parser error

              if(hr!=VARIANT\_TRUE)

              {

                     return (LPCSTR) checkParseError(m\_pXMLDoc->parseError);

              }

       }

       catch(\_com\_error &e)

       {

              dump\_com\_error(e);

              return  e.ErrorMessage();

       }

       return (LPCSTR) parseError;

}

// Replace into data with newer from data

static DataDictionary  Merge(DataDictionary &into, DataDictionary &from)

{

       DataDictionary newdata = into;

       for(DataDictionary::iterator it=from.begin(); it!=from.end(); it++)

       {

              into[(\*it).first]=(\*it).second;

       }

       return newdata;

}

std::string   MTConnectStreamsParser::GetMTConnectUrl(std::string ippath)

{

       std::string \_devicesfilename = "http://" + ippath + "/current";

       return \_devicesfilename;

}

std::vector<DataDictionary> MTConnectStreamsParser::ReadStream(std::string filename)

{

       DataDictionary data;

       std::vector<DataDictionary> datums;

       std::string ip;

       ip = filename.substr( 0, filename.find\_last\_of( '/' ) );

       ip = ip.substr( 0, ip.find\_last\_of( ':' ) );

       //if(!TestPing(ip))

       //     return datums;

       \_devicesfilename = "http://" + filename + "/current";

       // Not empty means error

       if(!ParseXMLDocument(\_devicesfilename.c\_str()).empty())

              return datums;

       data=ParseDataItems();

       datums.push\_back(data);

       m\_pXMLDoc=NULL;

       return datums;

}

DataDictionary MTConnectStreamsParser::ParseDataItems()

{

       MSXML2::IXMLDOMNodePtr root = m\_pXMLDoc->GetdocumentElement();

       MSXML2::IXMLDOMNodeListPtr nodes = root->selectNodes(\_bstr\_t("//DeviceStream"));

       DataDictionary data;

       try

       {

              for(int i=0; i< nodes->length; i++)

              {

                     MSXML2::IXMLDOMNodePtr pNode = NULL;

                     nodes->get\_item(i, &pNode);

                     \_bstr\_t items[3] = {\_bstr\_t(".//Samples"), \_bstr\_t(".//Events") , \_bstr\_t(".//Condition") };

                     for(int ii=0; ii<3 ; ii++)

                     {

                           MSXML2::IXMLDOMNodeListPtr samples = pNode->selectNodes(items[ii]);

                           for(int j=0; j< samples->length; j++)

                           {

                                  MSXML2::IXMLDOMNodePtr pSampleHive = NULL;

                                  samples->get\_item(j, &pSampleHive);

                                  // Get each child

                                  MSXML2::IXMLDOMNodeListPtr childs = pSampleHive->childNodes;

                                  for(int k=0; k< childs->length; k++)

                                  {

                                         MSXML2::IXMLDOMNodePtr pSample = NULL;

                                         ptime datetime;

                                         std::string name ;

                                         std::string value;

                                         std::string timestamp;

                                         std::string sequence;

                                         childs->get\_item(k, &pSample);

                                         name = (LPCSTR)  GetAttribute(pSample, "name");

                                         if(name.empty())

                                                name = (LPCSTR)  GetAttribute(pSample, "dataItemId");

                                         if(name.empty())

                                                continue;

                                         // Lookup any name remapping to shorten - e.g., Frt or path\_feedratefrt => Feed

                                         if(\_TagRenames.find(name)!= \_TagRenames.end())

                                         {

                                                name = \_TagRenames[name];

                                         }

#if 0

                                         // FIXME: is this necessary?

                                         // Check to see if data item to be saved

                                         if(std::find(\_saveddata.begin(), \_saveddata.end(), name) == \_saveddata.end())

                                                continue;

#endif

                                         value = (LPCSTR) pSample->Gettext();

                                         if(items[ii]== bstr\_t(".//Condition") )

                                                value =  std::string((LPCSTR) pSample->nodeName) + "."  + value  ;

                                         if(\_valuerenames.find(name+"."+value)!=\_valuerenames.end())

                                                value=\_valuerenames[name+"."+value];

                                         timestamp = (LPCSTR)  GetAttribute(pSample, "timestamp");

                                         sequence = (LPCSTR)  GetAttribute(pSample, "sequence");

                                         data[name]= value;

#if 0

                                         if(!timestamp.empty())

                                         {

                                                datetime=GetDateTime(timestamp);

                                                DataDictionary & dict(timeddata[datetime]) ;

                                                dict[name]=data[name];

                                                //for(DataDictionary::iterator it=dict.begin(); it!=dict.end(); it++)

                                                //{

                                                //     OutputDebugString(StdStringFormat("%s %s=%s\n", timestamp.c\_str(),

                                                //            (\*it).first.c\_str(), (\*it).second.c\_str()).c\_str());

                                                //}

                                         }

                                         streamdata[name]=StreamData(name,data[name], timestamp, sequence);

#endif

                                  }

                           }

                     }

              }

       }

       catch(...)

       {

              std::cout<< "MTConnectStreamsParser::ParseDataItems() Exception\n";

       }

       return data;

}

1. https://github.com/mtconnect/schema [↑](#footnote-ref-1)