NIKON README

Tuesday, May 12, 2015 10:00:05 AM EDT

This document describes the data gathering for a Nikon CMM machine that updates a tab separated log file. Multiple adapters to Nikon software are possible, each contained within one MTConnect Agent.

The NIKON Agent contains back end adapters that read a log file generated from the CMM periodically (typically when an event occurs within the CMM). The log file contains events and not samples, but all the events are time stamped and in absolute order of occurrence.

The file is specified as a Window cross-platform file, so it must contain the PC or computer name. UNC is short for Universal Naming Convention and specifies a Windows syntax to describe the location of a network resource, such as a shared file, directory, or printer. The UNC syntax for Windows systems has the generic form:

\\ComputerName\SharedResource

In our case the SharedResource is a shared file that must be explicitly sharable. In order to use the UNC file, Microsoft File Operations: CreateFile, ReadFile and CloseFile are used as other generic C++ file operations did not work (but were originally tried.) UNC files on Windows seem to require Windows specific File operations. Note, the UNC file path must be accessible to other computers or it cannot be read.mInside the Agent are Adapters for each UNC file. Each Adapter runs as a thread, hence the distinction between 64 bit and 32 bit C++ solutions must be explicitly acknowledged in installing the binary exe. That is, 32-bit MTConnect agents do not on 64 bit platforms, although they may appear to.

In the Nikon file, it saves all events with each line within the UNC shared file. The delimiter between fields is “tab”. Each line contains an event describing a machine state transition, so that all the lines of the UNC file must be read to understand the current state (machine on is one event, and then run program is another event). Note, no error detection of runaway date or times is done.

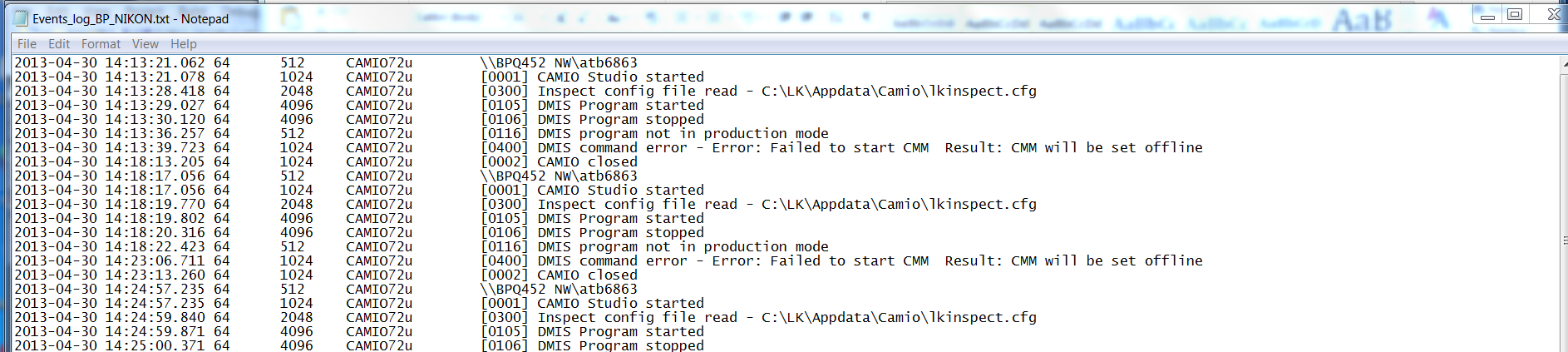
Below is a sample of the last line found in the Nikon shared file.

2013-04-30 14:13:21.078 64 1024 CAMIO72u [0001] CAMIO Studio started

There are five fields: timestamp, ??, ??, version, and a status event.

|  |  |
| --- | --- |
| Field | Example |
| timestamp | 01/23/2014 9:48:56 |
| ?? | 64 |
| ?? | 1024 |
| version | CAMIO72u |
| status | [0001] CAMIO Studio started |

Below is a snippet from a sample log file generated by Nikon camio:



These log fields, specifically the field 5 “status” offers sufficient information to develop a stack lite state model. Below is the translation of the Nikon status messages into MTConnect controller logic:

|  |  |
| --- | --- |
| State | Action |
| CAMIO Studio started | power=ON  controllermode=MANUAL  execution=IDLE |
| CAMIO Studio stopped | power=OFF  controllermode=UNAVAILABLE  execution= UNAVAILABLE |
| DMIS Program started | power=ON  controllermode=AUTOMATIC  execution=EXECUTING |
| DMIS Program stopped | power=ON  controllermode=MANUAL  execution=IDLE |
| DMIS command error - | power=ON  controllermode=MANUAL  execution=IDLE  error=message following dash |
| DMIS Program opened - | power=ON  controllermode=MANUAL  execution=IDLE  program=message following dash |
| Side effects | RPM and xyz move if automatic and executing |
|  |  |

Because of the deficiency of the MTConnect state logic, some side effects are generated to make the controller appear to be operating: positions for x,y,z and RPM change after every update if the controller is in automatic mode and executing.

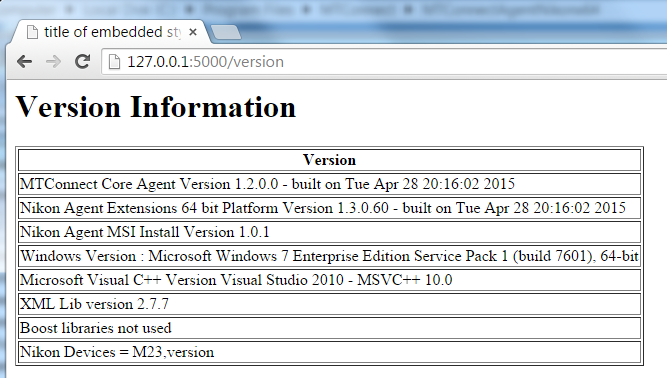
# Version Information from MTConnect Agent

The version information of the various MTConnect components is available through web browser access. It is recommended to use Google Chrome as it understand the XSLT formatting of the XML (Internet Explorer is baffled.)

The open source core MTConnect agent is downloaded from github and “frozen”. The version used is:

MTConnect Agent Version 1.2.0.0 - built on Sat Oct 12 13:30:24 2013

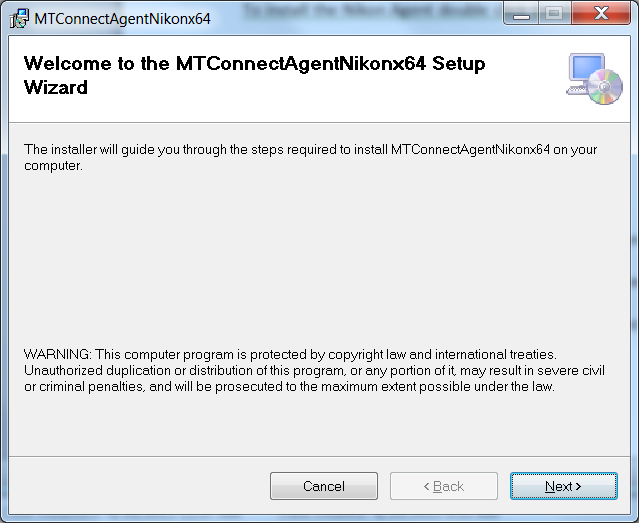
It is not perfect but the output shown below gives an indication of the software involved in the MTConnect Agent operation. It can be modified to include other version information, but requires a recompilation at this time.

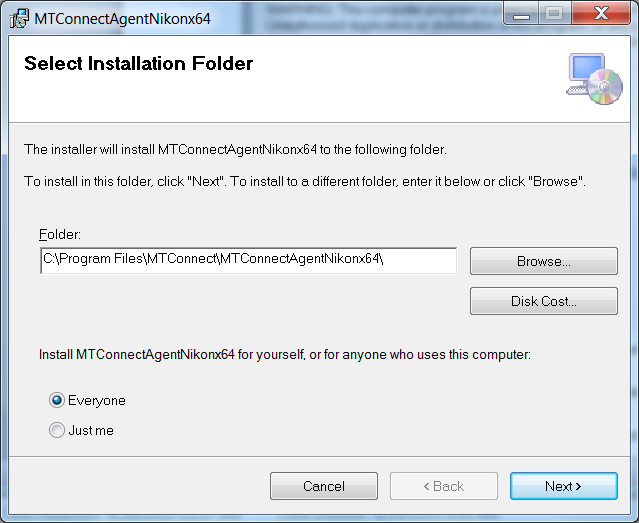


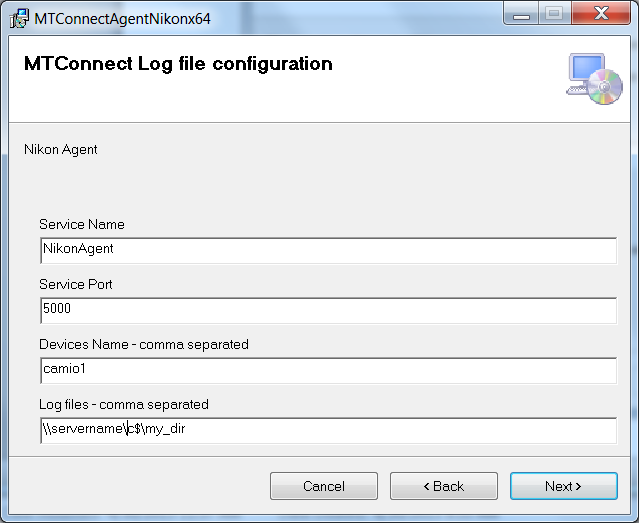
# Installation

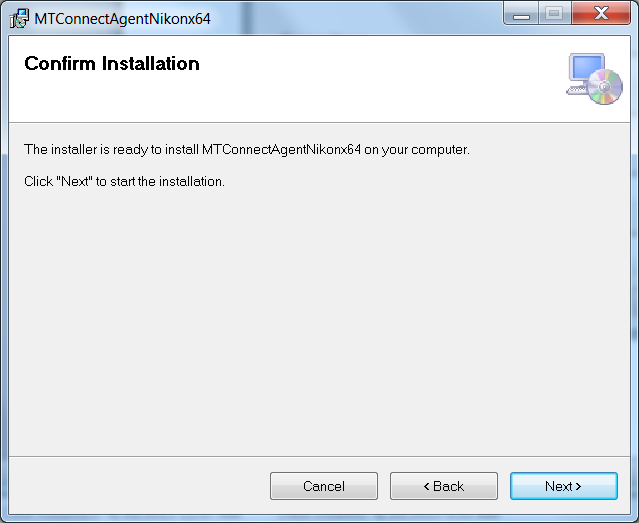
To install the Nikon Agent double click the (for 64 bit machines only).

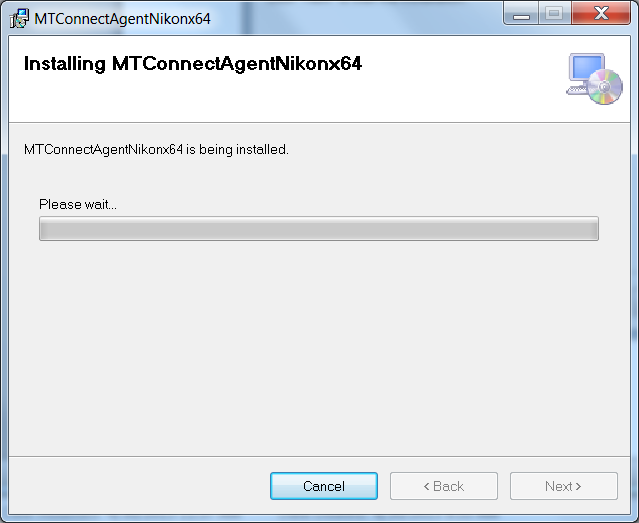




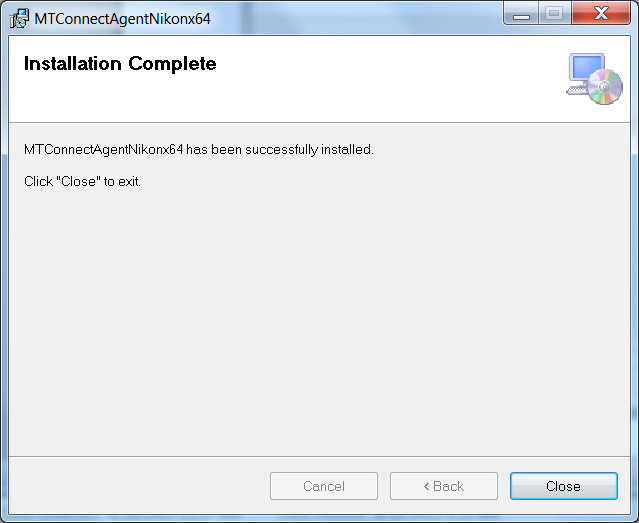


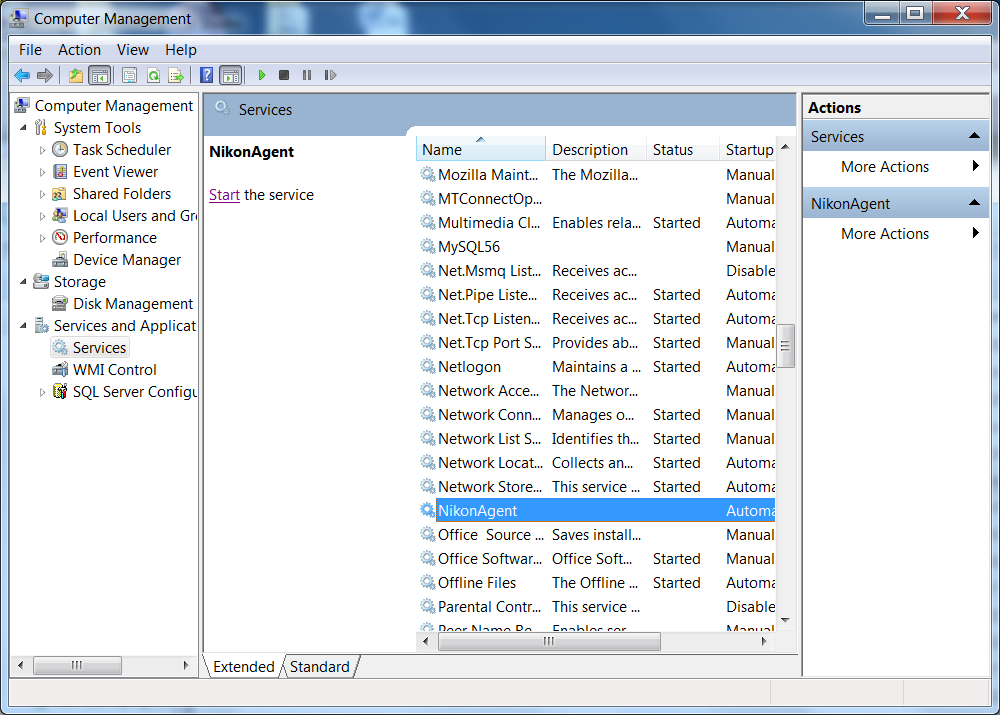






Acknowledge the installation permission challenge, then you should see the installation screen:





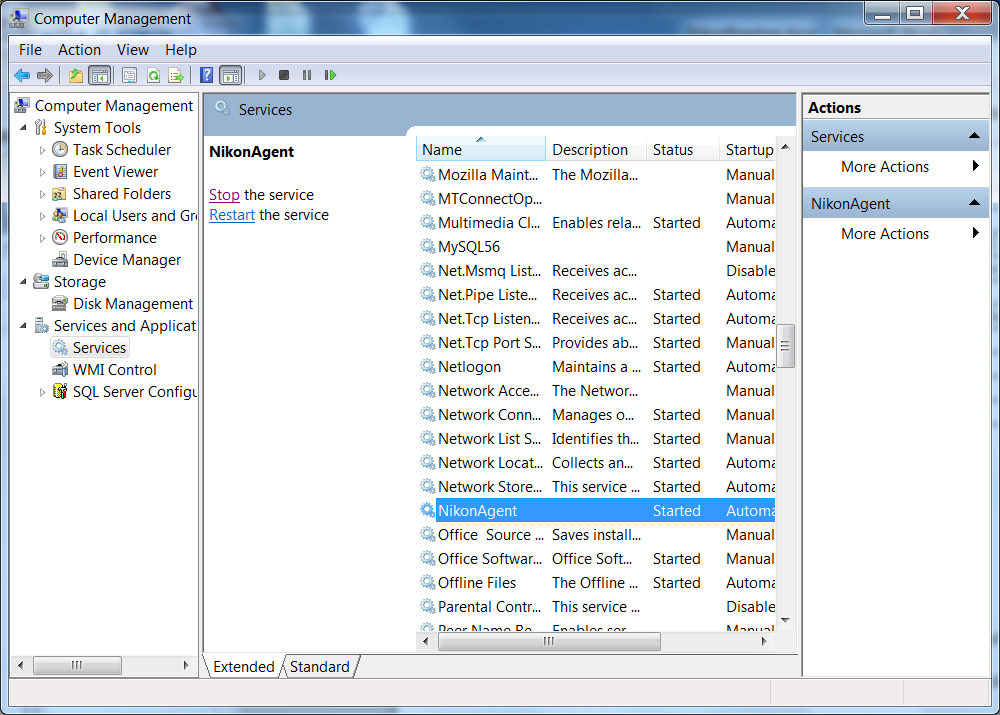
The agent will NOT start unless you tell it to START (unless you reboot).

# Uninstall

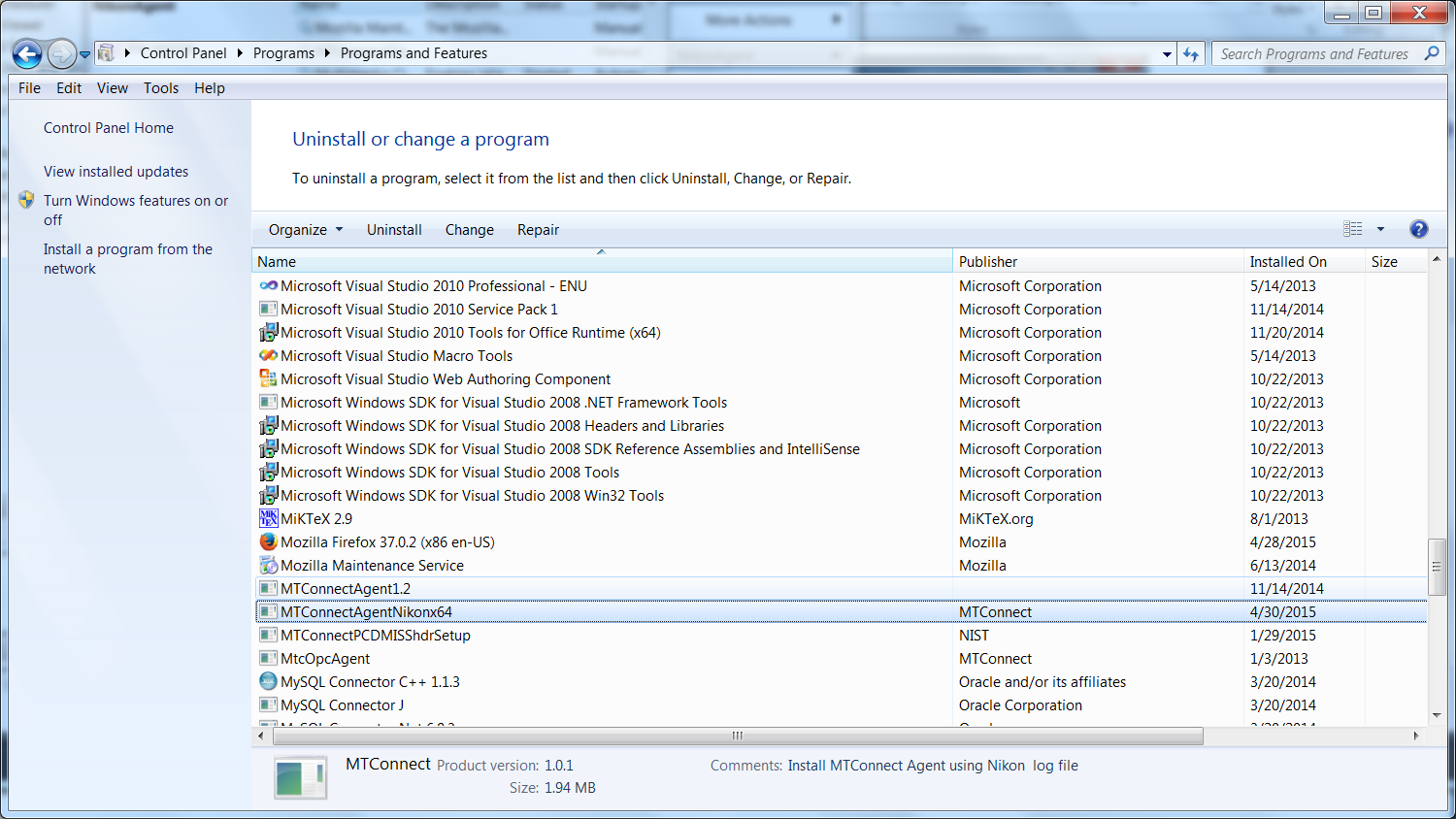
First make sure the Nikon Agent is stopped in the Service Control Manager:

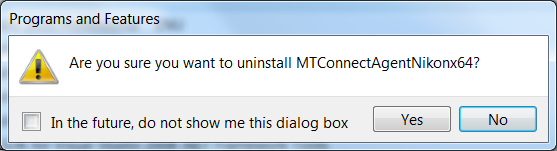
Right click on My Computer -> Select Manage -> Acknowledge UAC challenge

Select Services and Applications and then Services, scroll down to NikonAgent, and push Stop button.

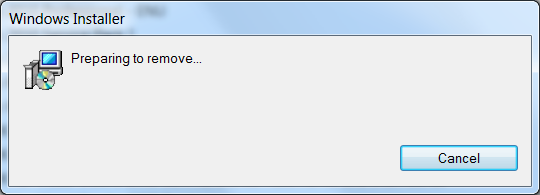


After you have stopped the Nikon Agent service, go into the control panel and uninstall the program: MTConnectAgentNikonx64





Yes!

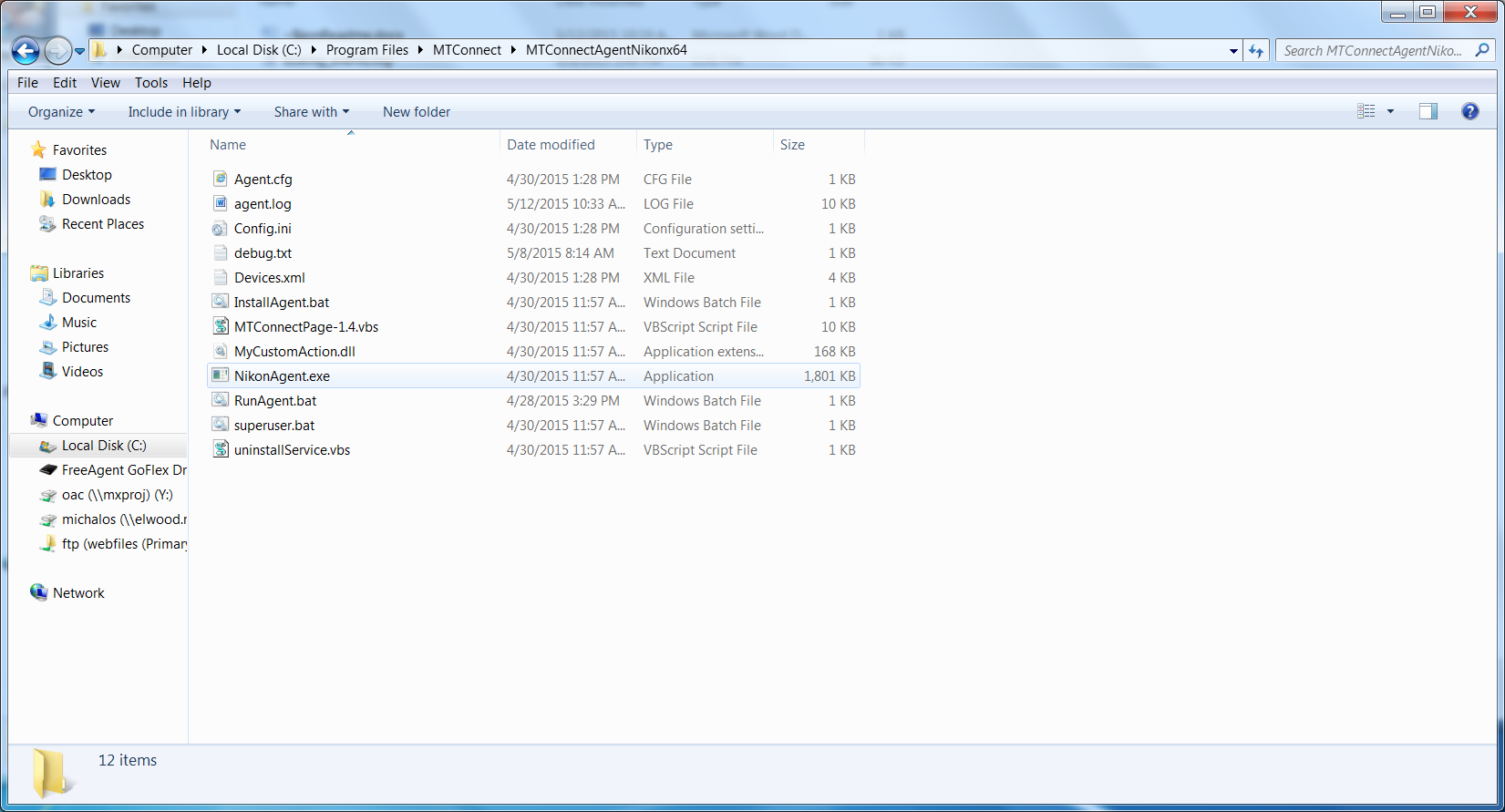


Please wait .. acknowledge UAC permission to uninstall challenge (you must be administrator or have administrator priviledges). It will uninstall and you may see a black console screen popup in the background momemtarily.

And then in the Service Manager click Action->Refresh, and the Nikon Agent service should be removed. The agent code in C:\Program Files\MTConnect\MTConnectAgentNikonx64 should also be removed.

# Configuration

The installation wizard installs the log file Agent into the folder: C:\Program Files\MTConnect\MTConnectAgentNikonx64 where x64 means a 64 bit installation platform, e.g., windows 7. The MTConnect Agent executable must be paired with the correct platform (32 or 64 bit).



In the folder, the vb script MTConnectPage-1.4.vbs in the can be used to verify that the log file Agent is working. MTConnectPage-1.4.vbs reads the data from the agent via <http://127.0.0.1:5000> and then formats the data. (assuming you have configure the agent port to 5000).

# Configuration

The Nikon MTConnect Agent uses the open-source MTConnect agent version 1.2 as the backbone to read http requests, generate XML responses, and update the underlying data. There is a NIST log file adapter addition to remotely read local or remote windows UNC files and then parse. The adapter can handle multiple device log file and update the core MTConnect agent.

The configuration is done at installation time during the wizard screens. The user must supply pairs of device names and UNC log file name, e.g., camio1,\\servername\C$\Program Files\Nikon\logfile.log From Wikipedia: The Microsoft Windows UNC, short for Universal Naming Convention or Uniform Naming Convention, specifies a common syntax to describe the location of a network resource, such as a shared file, directory, or printer. The UNC syntax for Windows systems has the generic form:

\\ComputerName\SharedFolder\Resource

Microsoft often refers to this as a "network path". You can use local paths, but if you are doing remote access of the log file, you must use the UNC convention.

The user can modify the config.ini file in the C:/Program Files/MTConnect/MTConnectAgentNikonx64 folder. However, the devices.xml and the agent.cfg files necessary for the core MTConnect agent are generated at installation time, and thus a reinstallation would be required to modify these files.

In the config.ini file, you can change the ServiceName, Agent port and query times of the log files. These changes will take if you stop/restart the Agent service or reboot the machine.

[GLOBALS]

ServiceName=NikonAgent

AgentPort=5000

QueryServer=10000

ServerRate=5000

Debug=0

ResetAtMidnite=false

MTConnectDevice=camio1

logging\_level=FATAL

[camio1]

ProductionLog=\\servername\c

Thus, the agent it reads the config.ini file for list of “MTConnectDevice”s under the Globals section. The configuration file also sets the Global flags: QueryServer, ServerRate, Debug, AgentPort(default 5000), and ResetAtMidnite.

# DIRECTIONS TO CONFIGURE NIKON LOG FILE AGENT.

Modify Config.ini in C:\Program Files\MTConnect\MTConnectAgentNikonx64directory

1. Stop Nikon agent, edit config.ini file, add new configuration:

[GLOBALS]

Config=NEW

1. Add new devices under [GLOBALS] section tag “MTConnectDevice” (spaces are stripped out)

MTConnectDevice=M1, M2, M3

1. Make sure there is an ini file “section” for each device (in this case M1, M2, M3) and ProductionLog tag that points to the UNC (Windows Universal Naming Convention) path to the log file as in:

[M1]

ProductionLog=\\grandflorio\c$\logfolder\Events\_log\_BP\_NIKON.txt

[M2]

ProductionLog=\\rufous\c$\logfolder\Events\_log\_BP\_NIKON.txt

[M3]

ProductionLog=\\synchro\c$\logfolder\Events\_log\_BP\_NIKON.txt

1. Start Nikon agent, the agent will detect a new configuration, and then write a new Devices.xml file to add the new devices.
2. If it works config.ini tag should say :”Config=UPDATED” if a problem tag will say: “Config=ERROR”

## Source Code Explained

void AgentConfigurationEx::initialize(int aArgc, const char \*aArgv[])

{

std::string cfgfile = Globals.inifile; // “Config.ini”

if(GetFileAttributesA(cfgfile.c\_str())!= INVALID\_FILE\_ATTRIBUTES)

{

config.load( cfgfile );

Globals.ServerName=config.GetSymbolValue("GLOBALS.ServiceName", Globals.ServerName).c\_str();

MTConnectService::setName(Globals.ServerName);

\_devices = config.GetTokens("GLOBALS.MTConnectDevice", ",");

}

Most importantly it reads the config.ini file for list of “MTConnectDevice”s under the Globals section.

The configuration file also sets the Global flags: QueryServer, ServerRate, Debug, HttpPort(default 5000), and ResetAtMidnite.

# Shift Change

std::vector<int> \_shiftchanges;

static int GetShiftTime(std::string s)

{

int Hour, Minute;

if(sscanf(s.c\_str(), "%d:%d", &Hour, &Minute)==2){}

else throw std::exception("Bad Shift time format - need hh:mm\n");

return Hour \* 60 + Minute;

}

// New shift changes - Mon 04/22/13 03:23:06 PM

shiftchanges = config.GetSymbolValue("CONFIG.SHIFTCHANGES", "06:00,15:00,23:00").c\_str();

std::vector<std::string> shifttimes =TrimmedTokenize(shiftchanges, ",", true);

for(int i=0; i< shifttimes.size(); i++)

{

\_shiftchanges.push\_back(GetShiftTime(shifttimes[i]));

}

COleDateTime today = COleDateTime(now.GetYear(), now.GetMonth(), now.GetDay(), 0, 0, 0);

COleDateTime tomorrow = COleDateTime(now.GetYear(), now.GetMonth(), now.GetDay()+1, 0, 0, 0);

//COleDateTime date2 = COleDateTime(now.GetYear(), now.GetMonth(), now.GetDay(), 0, 0, 0)

// + COleDateTimeSpan(1, 0, 0, 1);

//COleDateTimeSpan tilmidnight = date2-now;

// COleDateTimeSpan tilmidnight = date2-now;

//COleDateTime date2 = now + COleDateTimeSpan(0, 0, 2, 0); // testing reset time - 2 minutes

COleDateTimeSpan dt3=COleDateTimeSpan(1, 0, 0, 1);

COleDateTimeSpan dt4 = dt3; //(0, 0, \_shiftchanges[1], 1);

COleDateTimeSpan dt5= dt3; // (0, 0, \_shiftchanges[2], 1);

COleDateTimeSpan dt6= dt3; // (0, 0, \_shiftchanges[2], 1);

if( \_shiftchanges.size()<1)

dt3 = COleDateTimeSpan(1, 0, 0, 1);

if( \_shiftchanges.size()>=1)

dt3 = COleDateTimeSpan(0, 0, \_shiftchanges[0],0);

if( \_shiftchanges.size()>=2)

dt4 = COleDateTimeSpan(0, 0, \_shiftchanges[1],0);

if( \_shiftchanges.size()>=3)

dt5 = COleDateTimeSpan(0, 0, \_shiftchanges[2],0);

GLogger << FATAL << dt3.GetTotalMinutes() << std::endl;

GLogger << FATAL << dt4.GetTotalMinutes() << std::endl;

GLogger << FATAL << dt5.GetTotalMinutes() << std::endl;

COleDateTime date3 = today + dt3;

COleDateTime date4 = today + dt4;

COleDateTime date5 = today + dt5;

COleDateTime date6 = tomorrow + dt3;

COleDateTimeSpan tilnextreset = date6;

if(date6 > now)

tilnextreset=date6-now;

if(date5 > now)

tilnextreset=date5-now;

if(date4 > now)

tilnextreset=date4-now;

if(date3 > now)

tilnextreset=date3-now;