MTConnect Agent Reset at Midnight

Monday, February 06, 2017

Partially explains the machinations of the reset at midnight process.

# Instructions

In config.ini **ResetAtMidnight** must be set to 1 for the resetting thread to process.

[GLOBALS]

Config=UPDATED

sleep=2000

**ResetAtMidnight=1**

Debug=0

MTConnectDevice=bpq516

ServiceName=NikonAgent

logging\_level=FATAL

QueryServer=10000

ServerRate=5000

SHIFTCHANGES=05:45,14:15,22:15

It must be spelled and capitalized correctly.

# Code Description

There is a reset at midnight flag declared in the Globals.h file. In the NikonAgent.cpp "initialize" method, parameter from the config.ini file are parsed, including the "reset at midnight" flag.

int ResetAtMidnight;

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

…

Globals.ResetAtMidnight=config.GetSymbolValue("GLOBALS.ResetAtMidnight", "0").toNumber<int>( );

If the Globals.ResetAtMidnight flag is true a diagnostic message should be logged, and timer should start countdown until midnight is reached, at which the resetthread will be spawned. Detection of the flag is done in the MTConnect agent start() method, which calls ResetAtMidnite().

void AgentConfigurationEx::start ( )

. . .  
 if ( Globals.ResetAtMidnight )

{

ResetAtMidnite();

}

ResetAtMidnite handles the initialization of the timer thread, calculation of the wait time, and posts a diagnostic message as to when and how long the next reset will be and will be done.

bool AgentConfigurationEx::ResetAtMidnite ( )

{

COleDateTime now = COleDateTime::GetCurrentTime( );

COleDateTime date2 = COleDateTime(now.GetYear( ), now.GetMonth( ), now.GetDay( ), 0, 0, 0) + COleDateTimeSpan(1, 0, 0, 1);

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

COleDateTimeSpan tilmidnight = date2 - now;

GLogger.Fatal(StdStringFormat("Agent will Reset from now %8.4f\n", ( tilmidnight.GetTotalSeconds( ) / 3600.00 ) ) );

\_resetthread.Initialize( );

\_resetthread.AddTimer(

(long) tilmidnight.GetTotalSeconds( ) \* 1000,

&\_ResetThread,

( DWORD\_PTR ) this,

&\_ResetThread.\_hTimer // stored newly created timer handle

);

return true;

}

The debug.txt log file should contain a diagnostic message describing the amount of time until the next reset, given as hours.minutes as a decimal number.

2017-02-06 12:23:09.0921 Start Nikon Agent

2017-02-06 12:23:09.0935 Main

2017-02-06 12:23:09.0937 initialize

2017-02-06 12:23:10.0112 Start

2017-02-06 12:23:10.0114 Agent will Reset at Monday, February 06, 2017 12:25:10

2017-02-06 12:23:10.0114 Agent will Reset 0.0333 hours::min from now

The testing of the reset is done after a 2 minute delay, so the time to reset depends on the current time plus 2 minutes, (not midnight). Actual reset at midnight would give a time a minute after midnight of the following day.

2017-02-06 13:06:06.0479 Start Nikon Agent

2017-02-06 13:06:06.0495 Main

2017-02-06 13:06:06.0495 initialize

2017-02-06 13:06:06.0667 Start

2017-02-06 13:06:06.0667 Agent will Reset at Tuesday, February 07, 2017 00:00:01

2017-02-06 13:06:06.0667 Agent will Reset 10.8986 hours::min from now

The timer thread is an implementation of a Visual C++ ATL CworkerThread Component with the Windows OS. A worker thread is commonly used to handle background tasks that the user should not have to wait for to continue using your application. In this case, the worker threads task is implementing background timer that will cause a stop and start of the MTConnect Agent service. The declaration of the CworkerThread thread is as follows:

////////////////////////////////////////////////////////////////////

bool ResetAtMidnite ( );

CWorkerThread<> \_resetthread;

struct CResetThread : public IWorkerThreadClient

{

HRESULT Execute (DWORD\_PTR dwParam, HANDLE hObject);

HRESULT CloseHandle (HANDLE) { ::CloseHandle(\_hTimer); return S\_OK; }

HANDLE \_hTimer;

}

\_ResetThread;

The Execute method within the CWorkerThread does the major lifing. It cancels the waitble timer (set when initializing the thread), and then sets up spawned process invocation to start a DOS command shell to use the service control command line interface (sc.exe) to stop and then start the current service.

////////////////////////////////////////////////////////////////////

HRESULT AgentConfigurationEx::CResetThread::Execute (DWORD\_PTR dwParam, HANDLE hObject)

{

static char name[] = "CResetThread::Execute";

AgentConfigurationEx \*agent = (AgentConfigurationEx \*) dwParam;

CancelWaitableTimer(hObject);

try

{

PROCESS\_INFORMATION pi;

ZeroMemory(&pi, sizeof( pi ) );

STARTUPINFO si;

ZeroMemory(&si, sizeof( si ) );

si.cb = sizeof( si );

si.dwFlags = STARTF\_USESHOWWINDOW;

si.wShowWindow = SW\_HIDE; // set the window display to HIDE

#ifndef RESETTEST

std::string cmd = StdStringFormat("cmd /c net stop \"%s\" & net start \"%s\"", Globals.ServerName.c\_str( ), Globals.ServerName.c\_str( ) ); // Command line

#else

TCHAR buf[1000];

GetModuleFileName(NULL, buf, 1000);

std::string exepath = buf;

std::string exe = ExtractFilename(std::string(buf));

std::string cmd = StdStringFormat("cmd /c taskkill /IM \"%s\" & \"%s\" debug", exe.c\_str(), exepath.c\_str() ); // Command line

#endif

\"%s\"", Globals.ServerName.c\_str( ), Globals.ServerName.c\_str( ) ); // Command line

if ( !::CreateProcess(NULL, // No module name (use command line)

const\_cast<char \*>( cmd.c\_str( ) ),

NULL, // Process handle not inheritable

NULL, // Thread handle not inheritable

FALSE, // Set handle inheritance to FALSE

0, // No creation flags

NULL, // Use parent's environment block

NULL, // Use parent's starting directory

&si, // Pointer to STARTUPINFO structure

&pi) ) // Pointer to PROCESS\_INFORMATION structure

{

AtlTrace("CreateProcess FAIL ");

}

::Sleep(5000); // make sure process has spawned before killing thread

}

catch ( ... )

{

agent->AbortMsg("Exception - ResetAtMidnightThread(void \*oObject");

}

return S\_OK;

}