[DEVICES]

Machine=CNC1\_RESOURCE,CNC2\_RESOURCE,CNC3Old\_RESOURCE,CNC3New\_RESOURCE,CNC4\_RESOURCE

IP=127.0.0.1/CNC1\_RESOURCE,127.0.0.1/CNC2\_RESOURCE,127.0.0.1/CNC3Old1\_RESOURCE,127.0.0.1/CNC3New\_RESOURCE,127.0.0.1/CNC4\_RESOURCE

Machine=M1

IP=agent.mtconnect.org

[ALIASES]

controllermode=mode

path\_feedratefrt=Feed

Sload=SpindleLoad

Sspeed=Srpm

Frt=Feed

hlow=HydraulicLevel

hpre=HydraulicPressure

htemp=HydraulicTemp

clow=CoolantLow

clp=LogicProgram

[VALUEREMAPPING]

execution.READY=IDLE

execution.ACTIVE=RUNNING

mode.AUTOMATIC=AUTO

execution.EXECUTING=RUNNING

void CHtmlTableSnapshots::DbArchive()

{

CDBLogger db;

COleDateTime now = COleDateTime::GetCurrentTime();

std::string dbfile=\_archivepath + \_sMachine + "LogCurrent.mdb";

if( GetFileExists(dbfile) && ((GetFileSize(dbfile.c\_str())>> 20) > 20) )

{

std::string dbfile2 =\_archivepath + \_sMachine + "Log" + (LPCSTR) now.Format("%y\_%m") + ".mdb";

MoveFile(dbfile.c\_str(),dbfile2.c\_str()); // should "delete" dbfile

}

if(!GetFileExists(dbfile))

{

db.CreateLogDB(dbfile, \_fields, \_typedatum);

}

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

{

DataDictionary data=datum[i];

std::vector<std::string> values;

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

{

std::string value = data[\_fields[i]];

if(value=="UNAVAILABLE")

value = "";

if(value.find("--")!= std::string::npos)

value = "";

values.push\_back(value);

}

db.AddRow(dbfile,\_fields,\_typedatum, values);

}

}

class CMtConnectDboard : public MTConnectService

{

public:

CMtConnectDboard();

~CMtConnectDboard();

virtual void initialize(int aArgc, const char \*aArgv[]);

virtual void start();

virtual void stop();

virtual void logoff();

virtual void logon();

void serviceThread();

static HANDLE hBrowserMutex; ;

std::string html;

crp::Config config;

std::vector<CHtmlTableSnapshots \*> \_snapshots;

std::vector<std::string> sMachines;

std::vector<std::string> ipaddrs;

HANDLE m\_ThreadEndEvent;

boost::thread\_group group;

};

int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE /\*hPrevInstance\*/, LPTSTR lpstrCmdLine, int nCmdShow)

{

CMtConnectDboard myservice;

myservice.setName("MtConnectDboard");

myservice.main(\_\_argc, (const char \*\*) \_\_argv);

#include "ObjIdl.h"

extern CComPtr<IGlobalInterfaceTable> pGIT;

CComPtr<IGlobalInterfaceTable> pGIT;

CoCreateInstance(CLSID\_StdGlobalInterfaceTable,

NULL,

CLSCTX\_INPROC\_SERVER,

IID\_IGlobalInterfaceTable,

(void \*\*)&pGIT);

tWinMain(HINSTANCE hInstance, HINSTANCE /\*hPrevInstance\*/, LPTSTR lpstrCmdLine, int nCmdShow)

{

CMtConnectDboard myservice;

myservice.setName("MtConnectDboard");

myservice.main(\_\_argc, (const char \*\*) \_\_argv);

}

Next, read the configuration file, which is the module name + “.ini”.

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

The following parameters are read. CONFIG.ServiceName determines the name of the service in the Windows SCM. Global parameter DebugLevel is set via the ini file key value for CONFIG.Debug. Timestamping of debug output is also a global parameter determined by the ini file key value for CONFIG.Timestamp. Next a series of MTConnect streams parser Map (section head with a series of key-value parameters) that includes TagRenames and valuerenames. The parameter TagRenames renames some MTConnect into renames for the database, for example, controllermode is renamed into mode. Valuerenames is used to reassign the Section key value to another value. Next, saveddata is a key value pair (CONFIG.Data) which defines the tokens to save in the Archive database.

The Section DEVICES contains two key lists: IP and Machine which define the IP and machine device name for machine to use MTConnect archiving.

In order to log the MTConnect data from the status agents from Arena Example 6, we need to define the IP location and the machine names of the devices. This information is entered into the file: MTConnectArchiver.ini.

Machine=CNC1\_RESOURCE,CNC2\_RESOURCE,CNC3Old\_RESOURCE,CNC3New\_RESOURCE,CNC4\_RESOURCE

IP=127.0.0.1:5000,127.0.0.1:5000,127.0.0.1:5000,127.0.0.1:5000,127.0.0.1:5000

[CONFIG]

Type=IP

This configuration is then used by the MTConnect Archiver within its agent code to spawn off 5 adapters to read the devices status through MTConnect agent XML. This could be more efficient if the devices were always located on the same IP but this is not always the case. In fact, a hybrid approach that mixes simulation with real devices is possible using this archival mechanism.

The archival mechanism can read “live” XML data from an MTConnect agent stream, or it can read a comma separated values (csv) values. The type of access is defined by CONFIG.Type as either IP or CSV to signify the difference.

void CMtConnectDboard::serviceThread()

{

GLogger.Warning("CEchoService::serviceThread()\n" );

\_set\_se\_translator( trans\_func ); // correct thread?

try {

::Sleep(1000\*5);

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

{

CHtmlTableSnapshots \* snapshot = new CHtmlTableSnapshots();

std::string machine;

if( i>= sMachines.size())

machine="Machine name missing - please fix";

else machine=sMachines[i];

\_snapshots.push\_back(snapshot);

snapshot->Configure( config,

machine, // machine name

ipaddrs[i],

""); // ip address of agent

group.create\_thread(boost::bind(&CHtmlTableSnapshots::Cycle, snapshot));

::Sleep(1000);

}

group.join\_all();

}

catch(std::exception e)

{

AtlTrace("%s%s", "CEchoService::serviceThread() exception " , e.what());

}

catch(...)

{

AtlTrace("CEchoService::serviceThread() exception ");

}

// parser.Release();

if(m\_ThreadEndEvent != NULL)

SetEvent(m\_ThreadEndEvent);

//::CoUninitialize();

AtlTrace("CEchoService::serviceThread() Exit");

}

void CHtmlTableSnapshots::Configure(crp::Config &config,

std::string sMachine, // machine name

std::string ipaddr, // ip address of agent

std::string configname // section name to lookup configuration info

)

{

\_sMachine= sMachine;

\_ipaddr= ipaddr;

\_delay = config.GetSymbolValue("CONFIG.DELAY", L"2500").toNumber<int>();

\_colormapping = config.getmap("COLORCODING");

\_formats = config.getmap("FORMATS");

\_numberfields = config.GetSymbolValue("CONFIG.Numbers", L"").c\_str();

\_numberfields+=",Shift";

\_dateTimefields = config.GetSymbolValue("CONFIG.DateTime", L"").c\_str();

\_KPIList = config.GetTokens("CONFIG.KPI", ",");

\_fields = config.GetTokens("CONFIG.Fields", ",");

\_dbeventfields = config.GetTokens("CONFIG.DbEventFields", ",");

//???

\_headers= config.GetSymbolValue("CONFIG.SummaryHeader", L"").c\_str();

//\_sHeader = config.GetSymbolValue("CONFIG.Header", L"MTConnect Readings").c\_str();

\_typedatum.Mixin(MTConnectStreamsParser::\_saveddata, "adVarWChar");

\_typedatum.Mixin(\_fields, "adVarWChar");

\_typedatum.Mixin(config.GetTokens("CONFIG.Numbers", ",") , "adDouble");

\_typedatum.Mixin(config.GetTokens("CONFIG.DateTime", ",") , "adDBTime");

\_typedatum["Timestamp"]="adDate";

\_typedatum["Alarm"]="adVarWChar"; // should be in \_fields

\_typedatum["Shift"]="adInteger"; //

\_sCostFcns = config.getsection("COSTFCNS");

\_events = config.GetTokens("CONFIG.Events", ",");

// Archive information

\_archivepath = config.GetSymbolValue("CONFIG.ARCHIVEPATH", ::ExeDirectory()).c\_str();

\_ArchiveFields = config.GetSymbolValue("CONFIG.Archive", ::ExeDirectory()).c\_str();

// Shift information

std::string shiftchanges = config.GetSymbolValue("CONFIG.SHIFTCHANGES", L"00:00,08:00,16:00").c\_str();

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

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

{

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

// shift i+1 must be > shift i

}

\_delay = \_delay / 1000;

if(\_delay == 0)

\_delay=1;

std::vector<std::string> \_faults = config.GetTokens("CONFIG.Fault", ",");

\_alarmHandler.Configure(\_faults,\_archivepath,\_sMachine);

\_dbConnection = config.GetSymbolValue("CONFIG.DbConnection", "CMSD").c\_str();

//odbc.open("ConsolidatedData","root","bingo");

odbc.open(\_dbConnection,"root","bingo");

odbc.createSchema("Factory");

odbc.deleteTable("Factory", sMachine+"Events");

odbc.createTable("Factory", sMachine+"Events",\_dbeventfields, \_typedatum);

\_alarmColumns = TrimmedTokenize("Severity,RaisedBy,Description,Program,RaisedAt,ResolvedAt,TBF,TTR",",");

\_typedatum.Mixin(\_alarmColumns, "adVarWChar");

\_typedatum["ResolvedAt"]=\_typedatum["RaisedAt"]="adDate";

\_typedatum["TBF"]=\_typedatum["TTR"]="adTime";// 00:00:00

odbc.deleteTable("Factory", "Alarms");

odbc.createTable("Factory", "Alarms",TrimmedTokenize(\_alarmHandler.\_faultitems,","),\_typedatum);

\_programColumns= TrimmedTokenize("Timestamp,Machine,Shift,program,Duration,Machining,Faulted,RpmTotal",",");

\_typedatum["Faulted"]="adDBTime"; // seconds

\_typedatum["RpmTotal"]="adInteger"; // revolutions

odbc.deleteTable("Factory", "Programs");

odbc.createTable("Factory", "Programs",\_programColumns,\_typedatum);

}







