LOGFILE README

Monday, February 05, 2018 9:45:15 AM

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

The WIRING Agent contains back end adapters that read a log file generated from the 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 Wiring 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 Wiring shared file.

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

That's great detail if we needed detailed operation, but we are only looking at basic high level OEE. The original log file had a bunch of state/mode events with timestamps (dont have it on hand) - which is probably all we need. We can ignore the detailed operation (as these fields don't seem to have a timestamp.)

Here's all we need for Sidney's dashboard OEE:

|  |  |
| --- | --- |
| **State or tag** |  |
| **Down – machine off/on** | Assume no update in log file for 8 hours so we can look at file modification timestamp or the last time stamp. |
| **Start->stop** | Executing need this transition |
| **Stop->start** | Idle need this transition |
| **Hold/Unhold** | Paused (if available), when done resume executing. Optional. |
| **timestamp** | When event occurred: Note: can use log file modification timestamp – but not as easy to view. (Since we only read log file when it changes size). |
| **Mode** | Auto/manual if available, if not always auto. |
| **Program** | Name of program. |
| **Fault/Alarm** | Optional |
| **Operator** | Optional |
| **Operation** | Not necessary only looking for OEE |
| **Tooling** | Not necessary. |
| **xyz,rpm** | Can be faked. No movement if idle, moving if executing state. |

We can fake xyz movement and spindle for Sidney if we know the machine is doing something. Overrides for speed/feed can be set to 100 and ignored, sice they are not relevant unless you can change a wiring machine speed of operation.

FYI log file size can also effect performance – should be kept to < 10M. Maybe reset before starting daily operation.

Hope this makes sense.

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

|  |  |
| --- | --- |
| Field | Example |
| timestamp | 01/23/2014 9:48:56 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

These log fields, specifically the field 5 “status” offers sufficient information to develop a stack lite state model. Below is the translation of the Wiring 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.