[Comments]

NX Controllers best practices for longevity

1-Mar-22

NetLinx proves a useful and flexible environment to accomplish a variety of automation tasks, but as with any other programming environment some care must be taken to avoid overworking the controller.

The points detailed below are considerations that should be observed and fully understood before using the methods and NetLinx functions in your code.

Limit Writing to Internal File Storage

All NX Series Central Controllers employ a flash-based microSD card for file system storage. All SD cards have a high but finite number of write cycles before they begin to fail. If SD card writes are properly managed, the SD cards in NX Central Controllers can last the typical lifetime of the product, but excessive writes can shorten that time significantly. When the SD card fails it will need to be replaced to return the NX Central Controller to normal operation.

The advisory points below will help programmers avoid unnecessary writes to the SD card.

- **Use AMX_LOG instead of SEND_STRING 0:**
  A popular method of runtime debugging has been using SEND_STRING 0 to print a customized message to NetLinx Studio's Internal Diagnostic Messages, or to a TELNET or SSH session when you enable it with MSG ON DEBUG. This approach is still valid, however SEND_STRING 0 messages are also sent to the Audit Log. The Audit Log is always stored locally on the microSD. Excessive use of SEND_STRING 0 will cause unnecessary wear on the microSD.

  Using AMX_LOG will still print the custom messages, but they will not be recorded in the Audit Log for messages with a severity of AMX_INFO or AMX_DEBUG. If you wish to have messages appear in the Audit Log, use a severity level of AMX_ERROR or AMX_WARNING. The AUDIT_NETLINX_GENERIC_EVENT is also available for this purpose.

  So, instead of using:

  ```
  SEND_STRING 0,"'Got here.'"
  ```

  Use:

  ```
  AMX_LOG(AMX_ERROR,"'Got here.'")
  ```

  …for debugging messages that do not need to appear in the audit log.

  If you are using the Café Duet environment, use the log(int, string) method available in the Service class which is already a part of any Duet module. The traditional System.out.println() should be avoided as it will be recorded in the Audit Log just as SEND_STRING 0 is in NetLinx.

  So, instead of using:

  ```
  System.out.println("Got here.");
  ```

  Use:

  ```
  log(ERROR,"Got here.");
  ```

  …for debugging messages that do not need to appear in the audit log.

- **Use NetLinx functions FILE_WRITE & FILE_WRITE_LINE sparingly:**
  Effort should be taken to minimize the frequency of writing data to files on the SD Card. Writing data every loop should be avoided. Instead, monitor data for changes and/or cache data changes and write only changed data at timed intervals. Where possible, utilized Non-Volatile RAM (NVRAM) instead of file writes to the SD Card to store data, especially for data that changes regularly.

Avoid Consistently High CPU Usage:

The CPU usage is key in an efficient and stable system. A low CPU usage signifies the availability of the system to respond to events. A high CPU usage can cause the system to respond slowly and degrade the user experience.

While intermittent spikes of 80% or higher CPU usage is not uncommon, consistent high CPU usage can shorten the life of the CPU and the SD Card. We recommend that an NX controller should have a sustained CPU usage of no more than 25% at runtime. The current CPU Usage is available via a Telnet or SSH connection to the controller. Typing 'cpu usage' will collect the data and produce a 10 second average. CPU usage averaging 50% or higher should be reviewed and corrected to reduce the CPU's load.

Some common causes of high CPU usage:

- Using global variables in loops within DEFINE_PROGRAM – The DEFINE_PROGRAM section of NetLinx code will run whenever a global variable changes. Using a global variable as a loop counter within DEFINE_PROGRAM will cause it to run continually due to the global counter variable being incremented every loop. To avoid this, use a Timeline for feedback updates. https://help.harmanpro.com/993-when-does-define_program-run-(or-why-loops-in-mainline-are-bad)
- Short repeating timelines – Excessively fine Timeline intervals used for updating UI feedback can cause high CPU usage. An infinitely repeating Timeline of 100ms is a good starting place for responsive feedback. However, as the task of updating feedback becomes larger, the interval value may need to increase as CPU Usage is driven higher.
- Overloaded systems – Too many chatty devices requiring complex parsing will test the limits of the system. Try optimizing the data transfer at a system level (e.g. eliminate unnecessary messages, remove any unneeded data, send only data that has changed). Alternatively, move or spread the data parsing to other NX Central Controllers or other computing devices with the system.