TECHNICAL NOTE TNOI15

Title: G3 Writing Processing

Product(s): G3 HMI Series

CRIMSON 2.0 WRITE PROCESSING

Red Lion's G3 operator panels handle writes to external devices in a manner that is very different from that employed by our earlier Paradigm products. The previous techniques involved looking for changes in data items, and then writing those changes as they were identified. This had a downside, in that if two sequential changes occurred between comms scans, only the latter value would be written to the PLC. For most data items, this did not cause an issue, but it did produce difficulties in several distinct situations...

- A panel would sometimes be configured to write a bit in the PLC to an "on" state when a key was pressed, and to then write it to an "off" state when the key was released. If the release occurred too quickly, only the "off" state would be written, and the required effect would not be achieved.
- Some devices—typically servo or variable-speed drives—have registers that are used to issue commands. Distinct values must be written to these registers in a specified sequence in order to produce the desired behavior. Unless the writes were separated by enough time to allow each individual value to be written, the command sequence would not be transferred correctly.
- Some devices have write-only registers that can be changed by the remote device, but which the HMI needs to repeatedly set to the same value. For example, a command register may need to be set to '1' to clear an alarm state, and then set to '1' again to clear a subsequent alarm. Paradigm HMIs would not perform the second write unless the data was first set to another value.

Crimson solves all of these problems by performing what are known as transactional writes. This means that writes to remote data items within a single device are performed in exactly the order that they were requested by the controlling database. It also means that, *if the data item in question is set to Write Only*, repeated writes of the same value can be performed.

This process is managed by a queue that can hold up to 512 pending writes per devices. Writes will be added to the queue, and then processed in sequence by the comms drivers. If the queue grows beyond 100 items,

Crimson will invoke a technique called write throttling, adding a small delay after each write request until the queue is reduced to a manageable level. The system is designed to maintain the queue at 32 writes, and to turn off throttling if the queue empties below 16 entries.

If a device goes off-line, the write queue is emptied, with the written values being transferred into the data items within the terminal. This means that when the device comes back online, the last written values will be transferred, but any sequences contained in the queue will be lost. This compromise is required to ensure that the queue does not get out-of-control in off-line situations. If a specific sequences needs to be sent to a device when it comes back online, use a trigger based on **`IsDeviceOnLine`** to invoke the writes as required.

One artifact of the queued approach is sometimes seen in demo databases, where a register might be incremented on every display update. Crimson will write the changes to the remote device, but will ensure that each value is sent individually. In the comms driver is configured at a low Baud rare, the queue may grow if the driver is not able to keep up with the requests. The queue will stabilize as write throttling kicks-in, but the remote device will typically be a couple of dozen values behind those being displayed by the HMI.

The good news is that this is rarely as issue with real applications, as they do not typically perform writes at such a frequency. In fact, Crimson’s write processing makes nearly all databases behave more intuitively, with the required data being written to the selected devices in exactly the required order. Push button replacement is made much easier, and operations that require data items to be written and then strobed by a command register can now be performed without the need for Sleep statements.