Perfection in Automation
www.br-automation.com

AutomationCON 2013

ADVANCED MACHINE DIAGNOSTICS: SOFTWARE STRATEGIES FOR RAPID TROUBLESHOOTING

Agenda

Perfection in Automation
www.br-automation.com

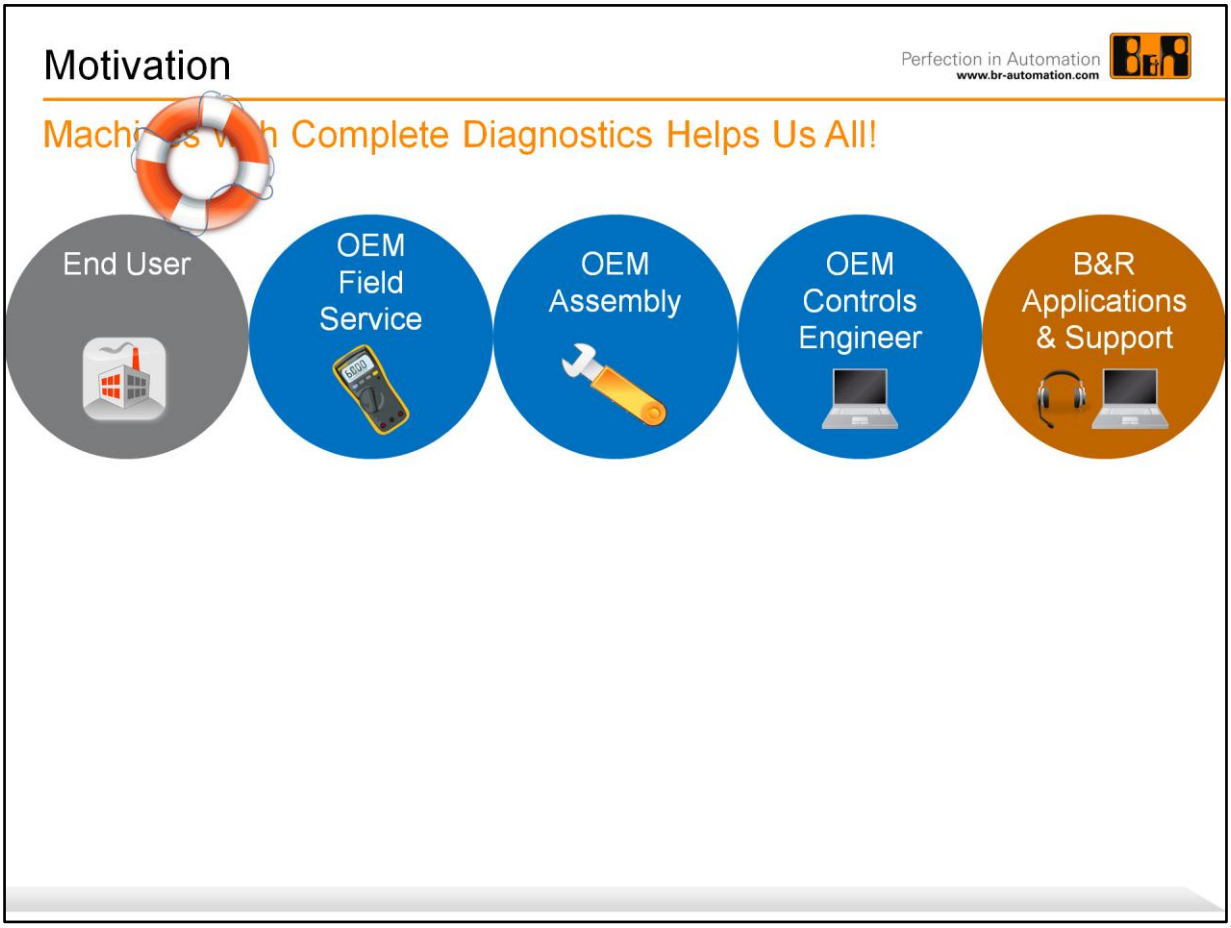
Topic	Presenter
Motivation for Advanced Diagnostics	Matt Adams
Best Practices	Matt Buck
Forcing	Matt Adams
Logging	Matt Buck
IEC Check	Matt Adams
Process Variable Functions	Matt Buck

Perfection in Automation
www.br-automation.com

Motivation for Advanced Diagnostics

Motivation for Advanced Diagnostics

The 21st century manufacturing environment is the most competitive it has been in the history of the world. Modern factories are measuring their overall equipment efficiency and have it displayed on large illuminated signs for everyone to see. Every single percent counts. Machine builders are exposed to and adopting the high performance controls equipment. These original equipment manufactures (OEMs) too are pressed on their efficiency to design, build, test and ship machines with shorter and shorter lead times. Both of these groups are demanding the same thing – smart machines that guide them to resolution when problems arise.



As a brief overview of our stake holders, end users in this group are maintenance personal on the factor floor, responsible for the OEE, or as they'd probably call it, keeping machines humming. If they have a problem with their machine, they'll try to fix it themselves. But if they don't understand why something isn't working or what an error message means they should do, they'll call the OEM for support.
Ideally, over the phone a service technician can guide them through, but if not, the OEM Controls Engineer will get involved. If they can't help, often the controls manufacturer is asked for assistance. In other words, snowballing support involving a lot of people.

While one could argue that the quantity of quality personnel in all of these groups is in short supply, ultimately our motivation today is going to be how the OEM Controls engineers can build tools into her machine that will help non-expert personnel diagnose and troubleshoot problems themselves. This means that as assembly technicians build the machine, they can check I/O wiring themselves without a laptop. It means that Field Service Technician can poke at variables, look at logs and follow clear instructions during their investigations. An End User will see error messages that also include a recommended action, so they don't have to guess and become frustrated.

Ultimately, the responsibility of implementing these tools is on of the controls engineers that are building software. My personal opinion is that they want to do this, but between a combination of not knowing what is truly important for diagnostics and not having enough time to explore this, they often implement diagnostics after a costly service trip. Our goal today is to provide controls engineers with ideas and best practices to make diagnostics as easy as possible. We all know that if a controls engineer is spending less time on support, she's spending more time on core tasks like R&D. Unapologetically, we hope she's experimenting with new B&R technologies, which will make the OEM's machine more competitive so they can gain market share, thus using more B&R product.

Perfection in Automation
www.br-automation.com

Best Practices

3 keys to good Machine Diagnostics

Perfection in Automation
www.br-automation.com

Best Practices

Logging – what has happened

Hardware Diagnostics – current status

Network Diagnostics – current status

There are 3 key focuses in good machine diagnostics; Logging, keeping track of what has happened on the machine; Hardware Diagnostics, the ability to see what hardware is there and what the status of the hardware is; and Network Diagnostics, so see what the network is supposed to be and what the status is of the networks.

Logging

Perfection in Automation
www.br-automation.com

Best Practices

Logging can provide substantial information in diagnosing issues on a machine.

It's best to try to break down the information into three areas

System Log	User Log	Debug Log
• Powerlink Sync Error	• Communication Error to I/O on Print Station #2	• BC0083 at Node 2 FailCycleCount=2 • BC0083 at Node 2 FailCycleCount=1

Logging can be used to provide substantial info in the diagnosing of the machine issues. Ideally there are three sets of data to be recorded. The system log, any issues that Automation Runtime or your Application automatically determine to substantial issues; a user log, to keep track of what issues the user sees and is able to fix, e.g. HMI E-stop pressed, release to run machine; and the Debug Log, a log for the OEM technician and engineers to use to diagnose problems that users can't fix.

System Log

Perfection in Automation
www.br-automation.com

Best Practices

Simple approach: Use the always present AR system log.

Better approach: Add an OEMsystem log. Use this log to record additional system related issues

The important thing is to make sure the user is informed of a system error and that he is instructed to contact the OEM if the issue persists.

The system log shows all of the system errors and warnings. If there is an error, it should be displayed to the user in the alarm box. This alarm could describe a deeper system problem as a general error to the user and to contact the OEM if the problem persists.

User Log

Perfection in Automation
www.br-automation.com

Best Practices

- Record all errors that are displayed to the user.
  - Example: “The E-Stop on the right side of the HMI has been pressed; release E-Stop to proceed.”
- Record/display system errors to the user in a way that they might understand.
  - Example: “ Error in Communications, if problem persists please contact OEM”
- Record hardware errors and any axis errors.

The user log book is intended to record all errors that are displayed to the user. For example, if the E-stop is pressed, this error should be logged every time as an event. It should also be displayed in the alarm box for the user to see and diagnose. The alarm should be displayed in a way that any user would be able to understand what the problem is and where to find it. For example, “The E-Stop on the right side of the HMI has been pressed; release E-Stop to proceed.” The other purpose of this logbook would be to display system errors to the user in a way that they might understand. For instance, if a system error states a device has not been initialized then the user logbook should display a general communication error to report to the user. This alarm should state to check connections. Another example that should be in this logbook is when any piece of hardware is removed or any axis errors.

Debug Log

Perfection in Automation
www.br-automation.com

Best Practices

Information for OEM technicians (not shown to user) :

- State changes
- Different commands that are used throughout the program.
- Parameter changes
- Function block errors
- Data read from a product scanner
- End of batch
- Start and finish of a operation

This logbook is information that does not need to be displayed to the end user but would be useful to debug the system if need be. This includes state changes and different commands that are used throughout the program. All of this information should be logged into a “Debugging Logbook” so that there can be more information for a service technician or engineer to diagnose the problem. This information does not need to be displayed to the end user but should be recorded in the background. Another example of an event that should be logged in this Debugging logbook is parameter changes. For example, if the user changes the axis limits, this should be logged. By logging these types of parameter changes, it will be easier to tell if the user has changed a parameter to put the machine into an unusable state.

Hardware Diagnostics

Best Practices

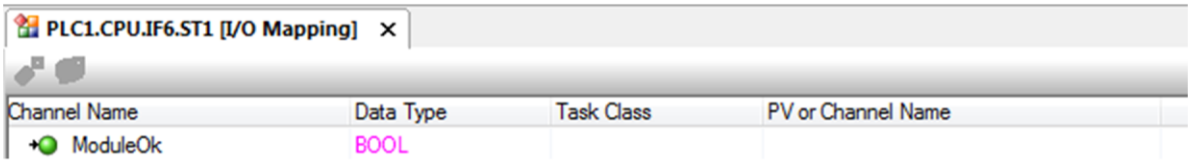
Helpful things to monitor, display on HMI or VNC, and/or log:

- Module OK bits
- Additional CPU information (Power On Cycles, Operating Hours, Battery Status, Environment Temperature)
- Module Information (Serial Number, Model Number)
- Additional I/O module Diagnostic information depending on the module

ModuleOK Bits

Perfection in Automation
www.br-automation.com

Best Practices



- Map the Module OK bit to a process variable
- If module OK is false, log to a logbook and display alarm to user, should also inhibit machine if necessary.
- If missing, error should indicate the hardware model, where it is in the machine, as well as the actual address of the module or OEM schematic location / OEM part number.

.

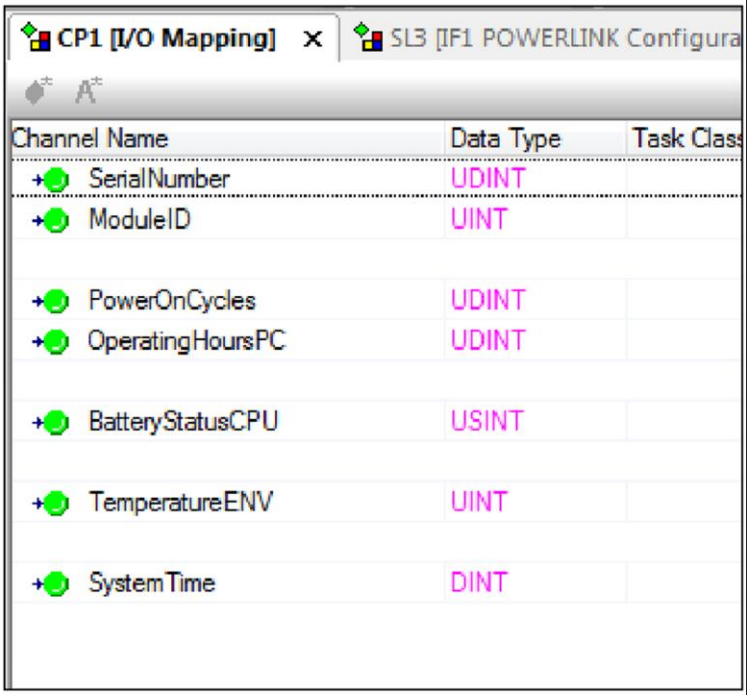
- On each module, a ModuleOK bit can be mapped to a process variable. This bit should be monitored on all modules.
- If the module is false on any module, an error should be logged in the user logbook and displayed in the alarm box. It should also inhibit a start of the machine until the user acknowledges or bypasses the error. If the error has not been fixed (e.g. a missing module has not been replaced, bypass bit is set), then the user should not be able to acknowledge the error.
- if missing, the error should display the material number of the piece of hardware (e.g. X20DO4322). The alarm should also indicate that the user should look at the System Diagnostic Manager (SDM) for the location of the module. Since the SDMhas a built in graphical representation of the hardware tree, the user will be able see where in the hardware tree the error occurred. Also giving the OEM's part number or schematic location may be helpful.

Additional CPU Information

Perfection in Automation
www.br-automation.com

Best Practices

- Battery could be backing up retained or permanent memory and any issues with the battery should be reported and logged



Here there are a few outputs that can be monitored but the most important is the BatteryStatus. Since the battery is the back up for the RAM that could contain many important reminent variables, if the batter y is
missing or low, an error should be reported to the user and logged in the user logbook.

Module Information

Perfection in Automation
www.br-automation.com

Best Practices

- Every module knows it's model number and serial number.

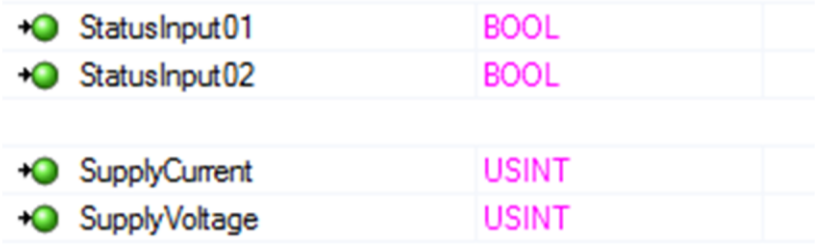
- May have to enable in the configuration view
- Some modules contain the firmware and hardware revisions.
- Logging this on boot up can be helpful to track hardware issues/replacement

Logging module information can identify if the user/operator/owner has replaced modules without informing the OEM point to possible neglect or damage the user has inflicted or a potential issue, for instance if a relay card keeps needing replacing, there may be in issue with solenoids drawing too much current or bad wiring.

Additional Diagnostic info (Power Modules)

Perfection in Automation
www.br-automation.com

Best Practices



- StatusInput01 – Bus Power Supply Warning
- StatusInput02 – I/O Power Supply Warning
- SupplyVoltage – Bus Voltage (in 0.1V)
- SupplyCurrent – Bus Current (in 0.1A)

StatusInput01 Bus Power Supply Warning-This bit tells whether the power supply has a low volt-age (Less than 4.7 V). This bit should be monitored and written to the user logbook with which module the error occurred on. A warning should also be displayed to the user for acknowledgement so that if it is a frequent occurrence the user can realize there might be a power supply issue going on in the background

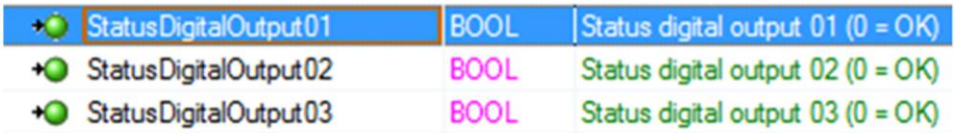
StatusInput02 I/O power supply warning- This bit tells if there is a voltage problem on the I/O Bus. This bit should be monitored and written to the user logbook with which module the error occurred on. A warning should also be displayed to the user for acknowledgement so that if it is a frequent occurrence the user can realize there might be a power supply issue going on

SupplyVoltage Bus Voltage- This gives the actual bus voltage that the module sees. This can be enabled by turning on the Voltage infor-mation in the I/O Configuration. Since it can be seen from SDM, there little need to monitor this data point. However, the volt-age information must be turned on in the I/O Configuration in order to see this value in SDM.

Additional Diagnostic info (DO Modules)

Perfection in Automation
www.br-automation.com

Best Practices



- StatusDigitalOutputXX – status of the output from 01-16 depending number of channels on the module

This bit should be monitored and written to the user logbook with which module the error occurred on. An error should also be displayed to the user for acknowledgement and it should say to check for wiring problems. Stopping the machine may be necessary if the output is essential.

Additional Diagnostic info (AI & AO Modules)

Best Practices

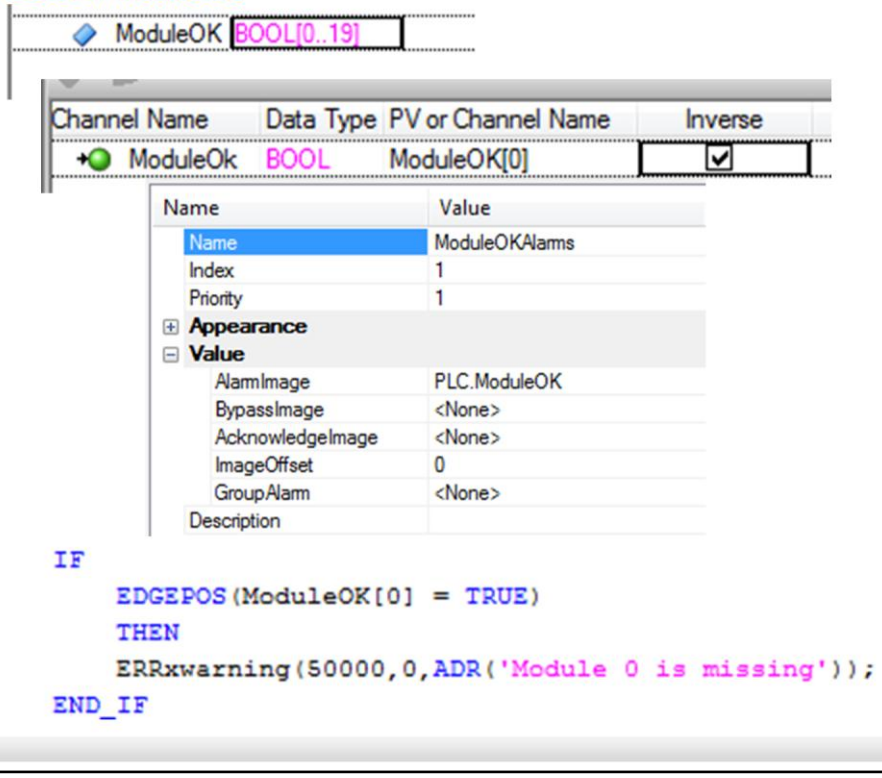
StatusInput01	USINT	Status of analog inputs

- StatusInput0X – Status of the analog input signal - this byte gives can tell the user if the input is above or below the user limit value as well as tell if there is a wire break.
- Bit 0 is low, Bit 1 is high, Both means wire break/not connected
- Note: this is already enabled by default.

Log and Alarm Example: Module OK bit

Perfection in Automation
www.br-automation.com

Best Practices




```
IF
    EDGEPOS(ModuleOK[0] = TRUE)
    THEN
    ERRxwarning(50000,0,ADR('Module 0 is missing'));
END_IF
```

I know that I keep saying log and show an alarm. For example, with ModuleOK bits, a good method of doing this is to create a array of Bools. Map the index to the ModuleOK of the module. Set the invert checkbox. Then attach the array to alarm image. This requires no supplemental programming code to implement. You can then use this to trigger a warning for the log. You can also run this in the init portion of code to check if any modules are missing at startup.

Network Diagnostics

Perfection in Automation
www.br-automation.com

Best Practices

For both X2X and POWELINK networks there are diagnostic I/O points to monitor.

Common sets for:
- POWERLINK Master
- POWERLINK slave
- X2X bus

May be helpful for there to be a Network Diagnostic Page

In the system, if there is an Ethernet Powerlink master. This master will have I/O points that can be monitored to detect and record problems on the EPL network. Concurrently each Bus Coupler can also monitor essential EPL network data and X2X network data that would be useful for debugging these communication systems. They can also be recorded to help diagnose intermittent communication faults.

POWERLINK Master

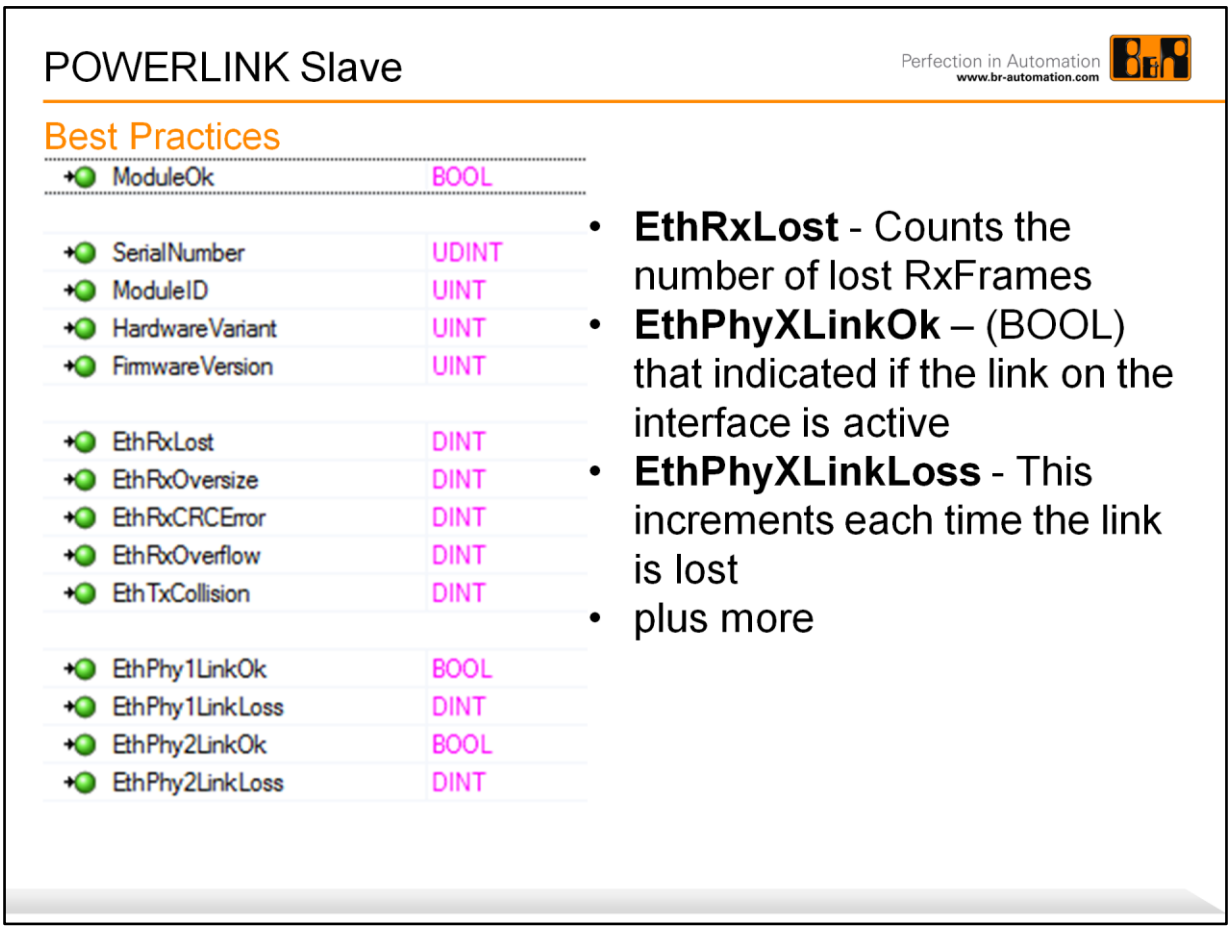
Perfection in Automation
www.br-automation.com

Best Practices

NodeSwitch	USINT
NodeNumber	USINT
CycleOk	BOOL
SyncOk	BOOL
LinkOk	BOOL
TimeSeconds	UDINT
TimeNanoseconds	UDINT
NettimeSoC	DINT
NettimeOffset	DINT
CycleCount	UDINT
FailedCycleCount	UDINT
CycleTimeViolationCount	UDINT
CycleIdleTime	UDINT
CycleCongestionCount	UDINT

- **CycleOk** – (BOOL) true if communication is working
- **SyncOk** – (BOOL) synchronized network is established
- **FailedCycleCount** – (UDINT) increments every time there is a failed bus cycle
- **CycleIdleTime** – unoccupied time during a POWERLINK cycle
- plus more …

CycleOk - Boolean that tells if the cyclic communication is working correctly - If this goes to false, then a warning should be logged. The machine should not stop (un-less the system automatically stops communication) but the user should be able to acknowledge the error.
SyncOk - Boolean that indicates that a synchronized powerlink network has been established - If this goes to false, then a warning should be logged in the user log. The machine should not stop (unless the system automatically stops communication) but the user should be able to acknowledge the error.
FailedCycleCount - This UDINT increments every time there is a failed bus cycle. - This value should be recorded and if there are more than 2 failed cycles per hour a warning should be written to the user log book and reported to the user. This warning does not need to stop the current process. The frequency that this value is checked can be changed depending on the situation as more machines are produced and more data is available for that particular machine.
CycleIdleTime - This value shows the unoccupied time during a Powerlink Cycle - This should be monitored and if the value gets too small (15% of the cycle time) this value needs to be written into the user log book.



CycleOk - Boolean that tells if the cyclic communication is working correctly - If this goes to false, then a warning should be logged. The machine should not stop (un-less the system automatically stops communication) but the user should be able to acknowledge the error.
SyncOk - Boolean that indicates that a synchronized powerlink network has been established - If this goes to false, then a warning should be logged in the user log. The machine should not stop (unless the system automatically stops communication) but the user should be able to acknowledge the error.
FailedCycleCount - This UDINT increments every time there is a failed bus cycle. - This value should be recorded and if there are more than 2 failed cycles per hour a warning should be written to the user log book and reported to the user. This warning does not need to stop the current process. The frequency that this value is checked can be changed depending on the situation as more machines are produced and more data is available for that particular machine.
CycleIdleTime - This value shows the unoccupied time during a Powerlink Cycle - This should be monitored and if the value gets too small (15% of the cycle time) this value needs to be written into the user log book.

X2X Bus

Perfection in Automation
www.br-automation.com

Best Practices

Channel Name	Data Type
CycleCount	DINT
BreakCount	DINT
SyncErrorCount	DINT
SyncBusyErrorCount	DINT
SyncNoRxErrorCount	DINT
SyncFormatErrorCount	DINT
SyncPendingErrorCount	DINT
AsyncErrorCount	DINT
AsyncBusyErrorCount	DINT
AsyncNoRxErrorCount	DINT
AsyncFormatErrorCount	DINT
AsyncPendingErrorCount	DINT
ModuleLostWhileOperational	DINT
ModuleNewWhileOperational	DINT

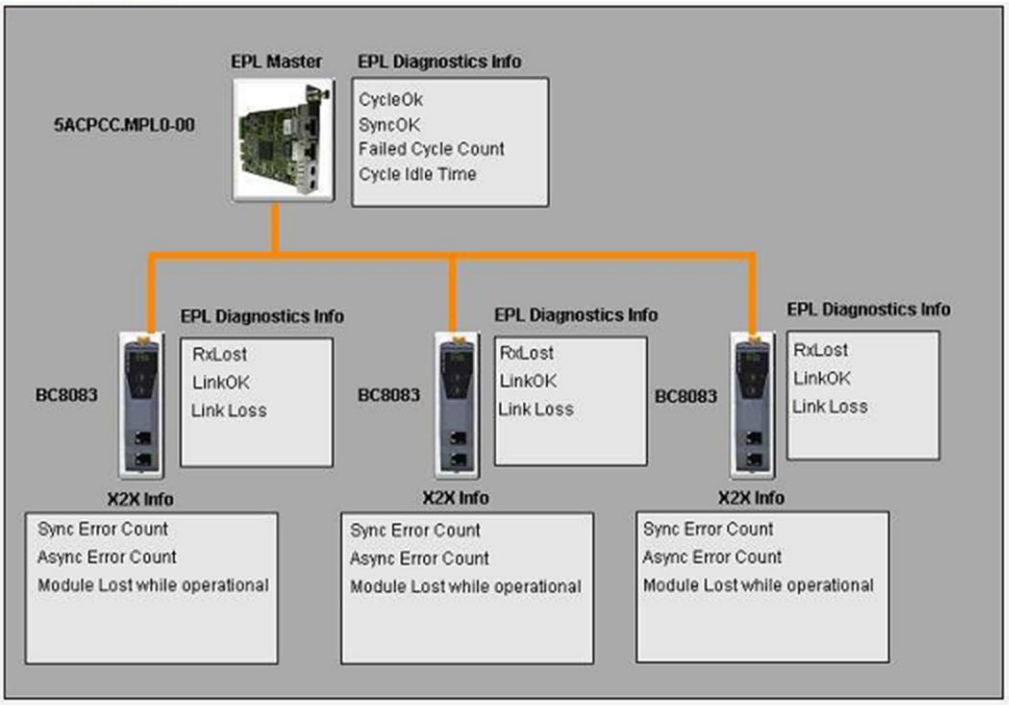
- **SyncErrorCount** – value of number of failed synchronous frames
- **AsycErrorCount** – value of number of failed asynchronous frames
- **ModuleLostWhileOperational** – increments anytime a module is failed or removed
- plus more …

SyncErrorCount This Value is the number of synchronous frames that have failed. This value should be monitored and if it increments then it should be logged and
a warning should be displayed
AsyncErrorCount This value is the number of asynchronous frames that have failed.
This value should be monitored and if it increments then it should be logged and a warning should be displayed
ModuleLostWhileOperational This increments each time a module fails or is removed This value should be monitored so that a user can quickly look at how many modules have failed on that particular bus. However an error doesn't need to be logged since the moduleOk's for every module should already be monitored.

Network Diagnostics Page

Perfection in Automation
www.br-automation.com

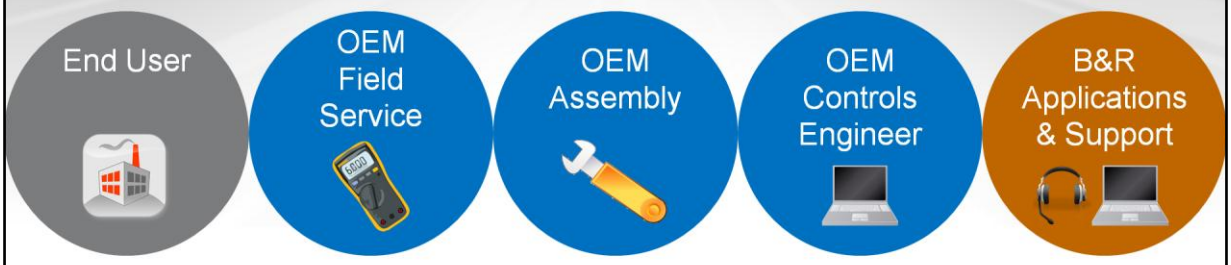
Best Practices



A graphical representation of the network diagnostics should be added and important network information should be easily accessible from this page in order to facilitate the processes of debugging network errors. A rough example what this page could look like can be seen in the figure. Note that the I/O points listed in the figure are the most important points to monitor. There are many other points that can be monitored for more detail on why the Error occurred and putting this information on this page should be considered.

Perfection in Automation
www.br-automation.com

Forcing and Monitoring I/O



Forcing and Monitoring I/O

Perfection in Automation
www.br-automation.com

Overview

- Forcing is a function of substituting the physical value of a discrete I/O point with a software value.
- Forcing is potentially dangerous! Implementing code to use forcing needs careful consideration for the safety of personnel and the machine itself.
- Forcing is a rapid way to troubleshoot externally connected hardware such as solenoids, lights or relays by testing them on demand.

Because I'm sure we have a variety of skillsets here in the room, when I say forcing, what I mean is that I'm not merely overwriting a variable that's mapped – I'm circumventing these values on a completely machine logic independent basis.

And when I say machine logic independent basis, this is really potentially dangerous. I highly recommend that forcing of I/O capability be kept behind OEM password protected controls.

And while I'm on this topic, I'd like to offer a couple of recommendations about passwords. The first is that passwords really ought to be dynamic, that is, they change over time. Inevitably, during troubleshooting, you allow users to get into certain password protected screens. While you're on the phone supporting them, you're doing hand-holding and the machine is unlikely to get damaged. However, I can tell you from experience that once that machine's password is out in the open, it is literally useless as a protection. There are machines with the OEM password written on a beautifully laminated label attached directly to the touch screen.

Creating your own dynamic password doesn't have to be more involved than some basic math. Add together the year, month and date. Multiply by 1000. Set the end time when automatic log-out will occur in 24 hour time format as the last 4 digits. That's the OEM password. The password to them is a number and to you it's great protection against liability and repair.

Example:

March 30ths, 2013:	2013 + 3 + 30 = 2046
Logout at 2:05 PM:	1405
OEM Password:	20461405

Forcing and Monitoring I/O

Perfection in Automation
www.br-automation.com

Strategy & Implementation

- The ability to force and monitor every I/O point without Automation Studio should be a standard feature on every machine!
- The key is to keep the capability implementation generic, simple to use, and hardware independent.
- Automation Studio provides the function blocks to do this in the AsIO library.

When I've worked with End Users at manufacturing plants, the conversation inevitably ends up that they want to have access to the source code of their machine. When I ask them why, I often get the same set of answers. A few want to be free to add future functionality, regardless of if the plant personnel are actually capable of doing so. Some want to be able to add buttons on the visualization or show additional data. But the most common response is, "Well, I want to be able to watch and force I/O if something breaks."

This in and of itself is not an unreasonable request, as it gets back to the heart of the issue is that plants want to solve their own problems, and we need to encourage it!

I think that more OEMs would implement this if they knew how, so we'll be spending some time here going over the library AsIO.

Forcing and Monitoring I/O

Perfection in Automation
www.br-automation.com

AsIO Function Blocks

AsIOListDP – Determines existing I/O datapoints
AsIOPVInfo – Collect variable name & context and invert status
AsIODPStatus – Determines an I/O data point's state & value
AsIOSetForceValue – Sets the force value for a datapoint
AsIOEnableForcing – Enables forcing on a datapoint
AsIODisableForcing – Disables forcing on a datapoint
AsIOGlobalDisableForcing – Disables forcing on all datapoints

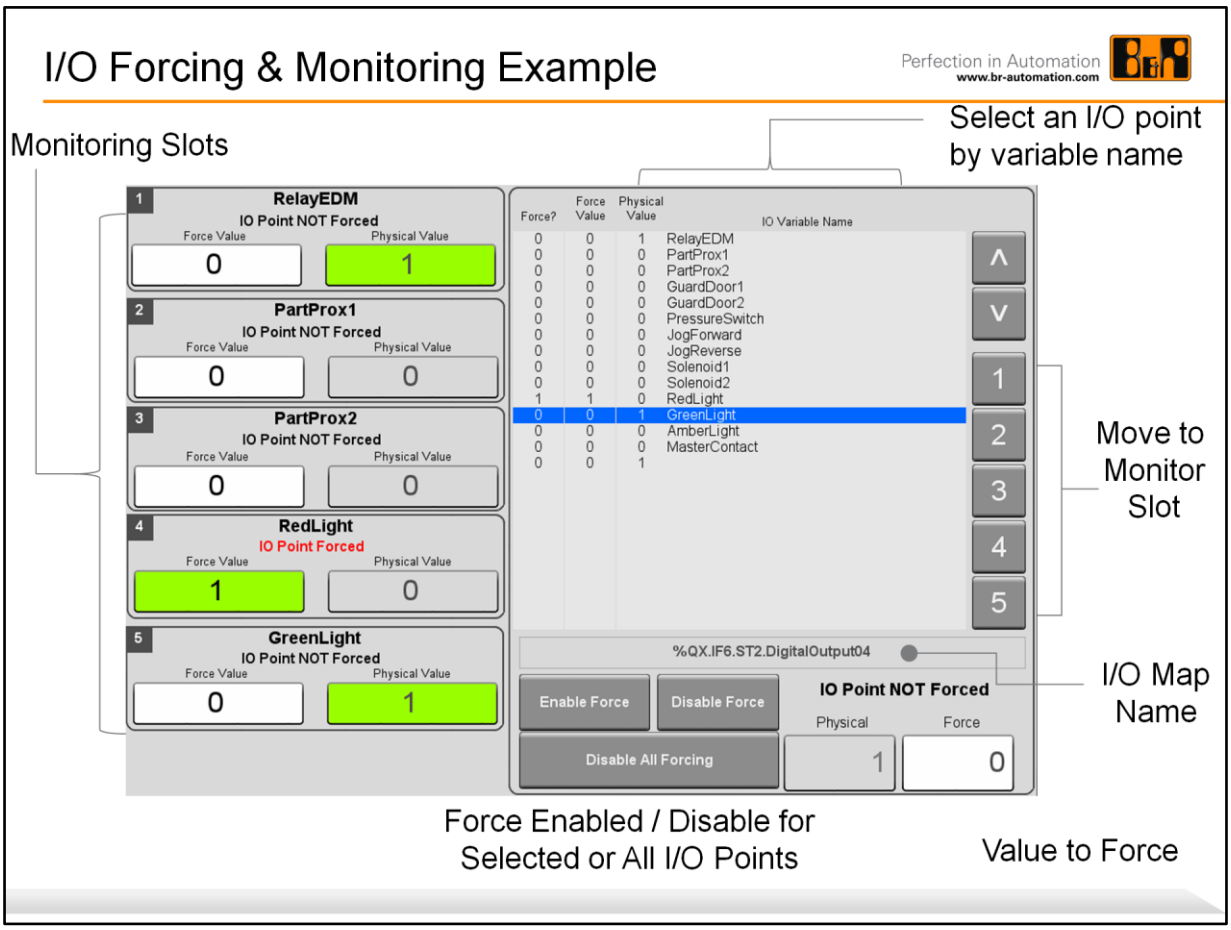
The List Datapoint library walks through the hardware tree and provides a list of all I/O datapoints. From there, we can feed these datapoints into IOPVInfo and find the variables which are mapped to the I/O Points, and then monitor the physical and forcing values with DP (Datapoint) Status

Monitoring I/O

Perfection in Automation
www.br-automation.com

AsIOListDP	AsIOPVInfo	AsIODPStatus
• %IX.IF6.ST1.DigitalInput01 • %IX.IF6.ST1.DigitalInput02 • %IX.IF6.ST1.DigitalInput03 • %IX.IF6.ST1.DigitalInput04 • %IX.IF6.ST1.DigitalInput05 • %IX.IF6.ST1.DigitalInput06 • %IX.IF6.ST1.DigitalInput07 • %IX.IF6.ST1.DigitalInput08 • %IX.IF6.ST2.DigitalOutput01 • %IX.IF6.ST2.DigitalOutput02 • %IX.IF6.ST2.DigitalOutput03 • %IX.IF6.ST2.DigitalOutput04 • %IX.IF6.ST2.DigitalOutput05 • %IX.IF6.ST2.DigitalOutput06	• Variable Name • Normal or Inverted • If I/O Point is Mapped	• Force value • Physical Value • Forced or Not Forced • Data type of the data point • Input or output channel

Here we'll review what's been mentioned before – you'll come across the complete list of I/O points from ListDP, and then can pass that I/O list string into PV Info and DP status.



Here is a generic concept of a visualization implementation using AsIO.

On the right side, you'll see a list of all I/O variables that have a datapoint mapped to them. I created a separate array of strings, and if there was a variable mapped to the I/O point from IO PV Value, I put the variable name in in the array. Then, I go through the array with IO DP Status and pull out the force and physical status.

Functionally, the user uses the arrow keys or by touching a variable name can can turn forcing on or off and set a force value on the bottom. The user also has the option to put a variable over on the left side in a monitoring slot. If they change the force value, it forces the I/O point.

The OEM customer I worked with uses a single assembly person to be able to walk around and test all of the sensors, guarding and other I/O on the machine. They do this by moving over the I/O points to the monitor slots and then turning their display towards them, and watching for the value / color change during their test, which may be just putting a wrench in front of a proximity sensor. This lets them test the status of their sensors without the need of a laptop, and without need to customize the visualization for all possible machine configurations. They can also test cabinet relays, solenoids, lights and other cabinet hardware by forcing outputs on or off.

Depending on the user permissions, they may not have permission to do forcing. Over the phone, the OEM can test hardware and diagnose problems like failed external relays by forcing on outputs.

Perfection in Automation
www.br-automation.com

Logging

Logging

Overview

- Logging is a powerful way to recreate user behavior, log critical errors and trap machine states.
- There are two methods of logging: AsArLog library and Error functions in SYS_lib.

Logging

Perfection in Automation
www.br-automation.com

SYS_lib Functions

ERRxfatal() – Logs an error and performs restart in service mode

ERRxwarning() – Logs a warning in the system logger.

	Level	Time	Error Number	ASCII Data	Binary Data
1	Error	2013-03-20 14:26:43...	55601	Var2 at deadly levels!	0000000500
2	Warning	2013-03-20 14:22:27...	55600	Var1 too high!	0000001000

Details

Name	Value
Level	Error
Date	20.03.2013
Time	2:26:43 PM,973000
Error Number	55601
OS Task	TC#4
Logger Module	System
Location	Online
Error Description	
ASCII Data	Var2 at deadly levels!
Binary Data	0000000500

```
IF Var1>200 THEN
    ERRxwarning(55600,Var1,ADR('Var1 too high!'));
    Var1:=0;
END_IF

IF Var2>200 THEN
    ERRxfatal(55601,Var2,ADR('Var2 at deadly levels!'));
    Var2:=0;
END_IF
```

These two functions are straightforward, as you can see from the example here.

They have a limitation that they log only to the system log, and you can only log 25 characters. This is a pretty short string. However, for messages like the ones that were mentioned earlier, like missed cycles or fail responses, a tiny log mentioning this can be very sufficient.

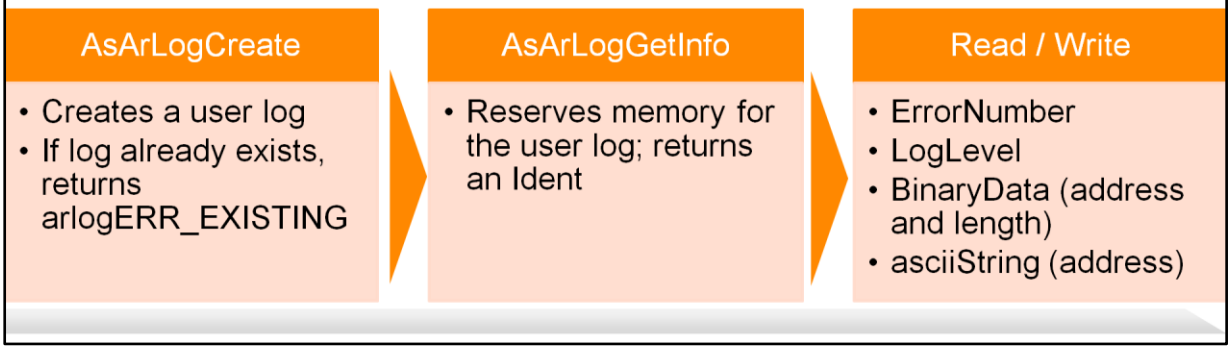
I'd recommend you be very weary about using a fatal error log. Almost all error states have a better strategy than merely putting the CPU into service mode.

Logging

Perfection in Automation
www.br-automation.com

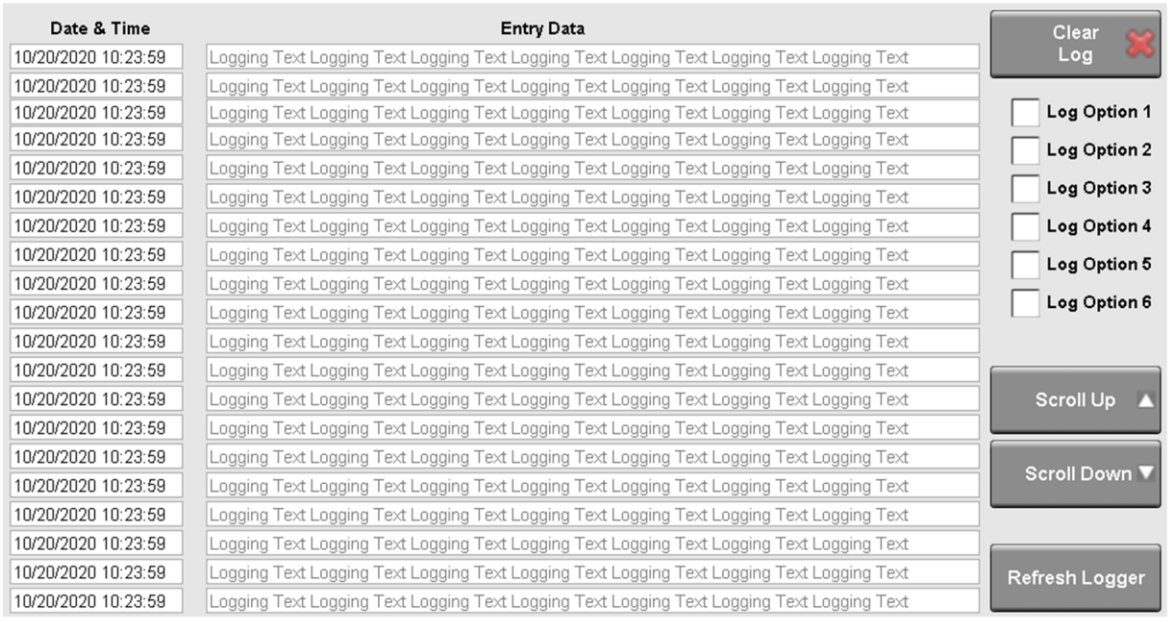
AsArLog Function Blocks

AsArLogCreate – Creates user log of specified size and name
AsArLogGetInfo – Gathers index and ident of a log
AsArLogDelete – Deletes a user log
AsArLogWrite– Writes a log entry to a user log
AsArLogRead– Reads an entry from user log
AsArLogClear– Clears all entries from user log



Here we have the collection of function blocks in the AsArLog library. There's a nice sample that you can add in that shows how these functions work together, but the basic strategy is that every time on boot up, you're going to use the LogCreate function, if it's successful or if the log already exists, you're going to use the LogGetInfo function to get an ident value so you can manipulate the log.

Log Viewer & Visualization Example



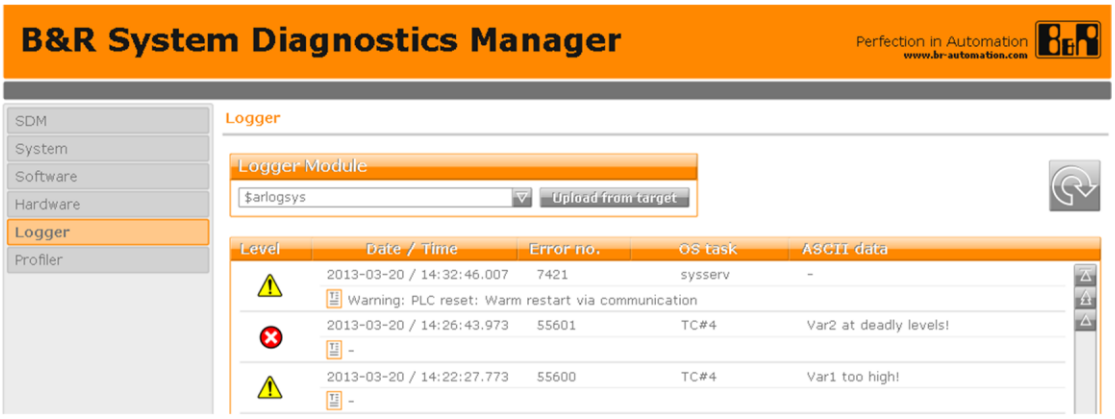
Here we see an example of how you can potentially ready out log entires with a date & time. There's also the option to only log certain types of events, such as machine states, I/O values or other critical data points. The AsArLog lets you display the logger values on a visualization in a way that is faster and easier to navigate than the System Diagnostic Manager tool that you can embed in your visualization.

Logging

Perfection in Automation
www.br-automation.com

Retrieving & Viewing Log Files from SDM

Log Files are available via the System Diagnostics Manager on any panel. They can be saved to a USB stick for retrieval and emailed for use in Automation Studio.



Log Viewing

Perfection in Automation
www.br-automation.com

Logs can be copied and pasted easily into Excel for advanced sorting.

Logger Entries: 34

	Level	Time	Error Number	Error Description	ASCII Data	Binary Data
1	Warning	2013-03-20 14:32:46.007000	7421	Warning: PLC reset: Warm restart via communication		0000000000

	A	B	C	D	E	F	G	H
1	Warning	2013-03-20 14:32:46	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
2	Error	2013-03-20 14:26:43	55601	TC#4	System		Var2 at deadly levels!	500
3	Warning	2013-03-20 14:22:27	55600	TC#4	System		Var1 too high!	1000
4	Warning	2013-03-20 14:21:48	55600	TC#4	System		Var1 too high!	29811496
5	Warning	2013-03-20 14:16:47	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
6	Warning	2013-03-20 14:00:40	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
7	Warning	2013-03-20 13:59:49	28700	sysserv	System	Function call error.	E_EXISTS	2147500040
8	Error / Syster	2013-03-20 12:36:49	9125	IOScheduler	System	TC#2 maximum cycle time violation		
9	Warning	2013-03-20 10:46:18	9222	ROOT	System	Warning: Warm restart after powerfail	Boot	0
10	Warning	2013-03-20 10:46:16	9200	ROOT	System	Warning: System halted because of power loss	Boot:Powerup	0
11	Warning	2013-03-19 13:34:35	9222	ROOT	System	Warning: Warm restart after powerfail	Boot	0
12	Warning	2013-03-19 13:34:32	9200	ROOT	System	Warning: System halted because of power loss	Boot:Powerup	0
13	Warning	2013-03-18 14:57:49	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
14	Warning	2013-03-18 10:00:25	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
15	Warning	2013-03-18 09:38:44	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
16	Warning	2013-03-18 08:17:46	9222	ROOT	System	Warning: Warm restart after powerfail	Boot	0
17	Warning	2013-03-18 08:17:44	9200	ROOT	System	Warning: System halted because of power loss	Boot:Powerup	0
18	Warning	2013-03-18 07:51:51	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
19	Warning	2013-03-18 07:17:23	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
20	Warning	2013-03-18 07:06:19	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
21	Information	2013-03-15 11:41:25	30303	DdX2XAcc.IF6	System	X2XLink firmware update.	IF6.ST2: firmware updated from V530 to V560	72603
22	Information	2013-03-15 11:41:23	30303	DdX2XAcc.IF6	System	X2XLink firmware update.	IF6.ST2: start firmware update from V530 to V560	72603
23	Warning	2013-03-15 11:41:03	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
24	Warning	2013-03-15 11:30:57	9222	ROOT	System	Warning: Warm restart after powerfail	Boot	0
25	Warning	2013-03-15 11:30:55	9200	ROOT	System	Warning: System halted because of power loss	Boot:Powerup	0
26	Warning	2013-03-15 11:23:55	7421	sysserv	System	Warning: PLC reset: Warm restart via communication		0
27	Warning	2013-03-15 11:12:15	9222	ROOT	System	Warning: Warm restart after powerfail	Boot	0
28	Warning	2013-03-15 11:12:15	27060	ROOT	System	NV memory (HDD/CF/RAM) has been exchanged	HDD/CF/RAM was change	0
29	Warning	2013-03-15 11:12:10	9200	ROOT	System	Warning: System halted because of power loss	Boot:Powerup	0
30	Information	2013-03-15 11:12:11	31280	ROOT	System	AR logger module created	base log module created	204800

Excel gives more opportunities for sorting, highlighting and comparing than Automation Studio.

Pressing Control-A to select All and Control-C to copy will allow you to then paste directly into Excel with Control-V

Perfection in Automation
www.br-automation.com

IEC Check

IEC Check

Perfection in Automation
www.br-automation.com

IEC Check is a library that checks for 5 mistakes:

- Dividing or MOD by Zero

```
Var1 := Var2 / (100-Var3);
```

- Null Pointers

```
Var1 := pVar;
pVar ACCESS ADR(ArrayVar[i]);
```

- Invalid Index in an Array

```
// ArrayVar has a range 0..5
Var1 := Array[6];
```

- Invalid Range of an Enumeration

```
// PriColors = Red, Yellow, Blue
PrimaryColorSelected:= 3;
```

- Invalid Value in a sub-range

```
// ValveAngle has range of 0..360
ValveAngle:= 400;
```

The first software tool we're going to talk about has some exposure among the general use of B&R engineers, but I tend to think it's not as understood as it could be.

IEC Check is a library that installs a series of functions to check for memory related errors during runtime. As you can imagine, this is a tool that is certainly a bit more narrow in scope if we were to look at our stakeholders in the previous screen. However, I wanted to talk about it a bit this morning, because a sizable quantity of the support I do revolves around these types of errors. What I hope to show is that with some customization, many of these complicated and sometimes cryptic errors can be rapidly diagnosed and corrected.

IEC Check helps with 5 mistakes: dividing or MOD by zero, reading or writing null pointers, calling an invalid index of an array, using an invalid range in an enumeration, or an invalid value in a sub range. In reality, the first three are the most common problem.

Just to touch on a couple of best practices or painful experience I hope you can avoid, I want to share a bit about dividing by zero. This is something that is simple to check for, and any time you perform a divide by a variable, a best practice is to make sure that the variable is not zero. Often times, the problem isn't actually dividing by zero, but dividing by a series of operations that result in zero, especially operations that involve integer math.

Null pointers are a little trickier – this is where you try to read or write to a pointer variable and you haven't pointed it at anything. Most of the time, this happen because you're conditionally assigning the pointer, and through a corner case, the pointer didn't get assigned.

Invalid Index in an array is when you have an array of 6 elements, say 0 to 5, and you point to index 6. What is interesting here is that without IEC check installed, you can often go along and manipulate

arrays out of range, but what you're really doing is manipulating adjacent variables in memory, or potentially other allocated memory.

Resetting via Node Switch

Perfection in Automation
www.br-automation.com

Side Note – Best Practice

Program a special number to cause a cold start or to restore default, good data.

Set node switches to particular value (like CC “Clear Corruption” or DD “Default Data”)

- FF and FE on some targets are reserved
- Using the same digit on both is easier on users.
- SYSreset() is the function that can cause a warm or cold restart.

I have seen situations where a certain users setting could slip through the cracks to cause a processor to be unusable. On the previous slide, we saw a divide by zero example. If Var3 was set to 100 and was retentive, then we would have a situation where you would not be able to get out of service mode. A good strategy to avoid this problem is to have a special node number which causes a Cold Start or a variable restore to known, good defaults.

The SYSreset function allows you to selectively command a cold start, which would restore variables to their initial values.

Example of Access Violation

Perfection in Automation
www.br-automation.com

If you spend enough time with arrays and loops, you may come across a service mode restart and a log entry like this:

	Level	Time	Error ...	OS Task	ASCII Data	Binary Data
1	Error	2013-03-13 10:25:45...	9101	DebugDprTask	:EXCEPTION_ACCESS_VIOLATION	248 019 000 000

IecCheck functions can help diagnose the problem.

- Causes a fatal error log (service mode restart)
- Value of variable that failed a check
- Task where check failed
- Which check failed

	Level	Time	Error Number	OS Task	Logger Module	ASCII Data	Binary Data
1	Error	2013-03-13 09:52:26,508000	55555	TC#4	System	RangeCheck > in task > AdvIndexHi	07 00 00 00

For the mere mortals that aren't perfect programmers, if you spend enough time with arrays and FOR loops, you'll have a situation where you'll wind up in service mode, and like a good controls engineer, you'll check the logger and get a cryptic error message. If you're a bad engineer, you might even be able to do a warm start and not have the problem show up again while you're developing. However, to be clear, it doesn't mean the problem is gone, it just means that it's waiting. It's waiting to appear again after the machine has shipped to one of the following conditions: Your largest most important customer, the most geographically remote customer, or the customer who loves to go on tirades about things they don't like.

The standard solutions to this is to installed the IEC Check functions with the default IEC Check library. This helps narrow down the cause, but depending on the program length, how many variables you have and the quantity of loops or pointers you're using, you might still have a tedious problem ahead.

The unfortunate problem with these errors is that by their nature, they can't be found at compile time, and often the back trace in the debugger can't get you there either. However, with a bit of customization, you can give yourself many more clues to rapidly pinpoint the bug.

IEC Check - Defaults

Perfection in Automation
www.br-automation.com

```
FUNCTION CheckBounds (* Check range for array access *)
    IF index < lower THEN
        CheckBounds := lower;
        MakeEntry(55555,index,'RangeCheck');
    ELSIF index > upper THEN
        CheckBounds := upper;
        MakeEntry(55555,index,'RangeCheck');
    ELSE
        CheckBounds := index;
    END_IF
END_FUNCTION

FUNCTION MakeEntry (* Makes a log book entry *)
    status_name := ST_name(0,ADR(taskname),ADR(group));
    strcpy(ADR(out_text),ADR(text));
    strcat(ADR(out_text),ADR(' > in task > '));
    strcat(ADR(out_text),ADR(taskname));
    MakeEntry := ADR(out_text);
    ERRxfatal(number,index,ADR(out_text));
END_FUNCTION
```

	Level	Time	Error Number	OS Task	Logger Module	ASCII Data	Binary Data
1	Error	2013-03-13 09:52:26,508000	55555	TC#4	System	RangeCheck > in task > AdvIndexHi	07 00 00 00

For our example of customization, we'll explore the CheckBounds function – here we see that if you're referring to an array index out of bounds, this CheckBounds function will tell you the value of the index and the task in the logger and put you into service mode.

IEC Check

Perfection in Automation
www.br-automation.com

Program length directly impacts IEC Check's usefulness

I've worked in programs with hundreds of lines of code and dozens of arrays and loops. While the default IEC check helps, it's not sufficient to rapidly diagnose the problem.

IEC Check Defaults

Perfection in Automation
www.br-automation.com

Default Logging

	Level	Time	Error Number	OS Task	Logger Module	ASCII Data	Binary Data
1	Error	2013-03-13 09:52:26,508000	55555	TC#4	System	RangeCheck > in task > AdvIndexHi	07 00 00 00

What's Not Logged

- Variable name
- Valid value range
- Line location in the task where check failed

IecCheck customization can help pinpoint the problem.

	Level	Time	Error ...	ASCII Data	Binary Data
1	Error	2013-03-14 14:23:13...	55555	Array Index 7 is outside [0..6] in task "AdvIndexHi" Section:16	007 000 000 000

What's lacking is the variable name, and valid value range, and where exactly in the program the problem occurs.

While we will not be able to get the variable name, we can use customization to get a more narrow scope of code and properties about the array to speed up troubleshooting our code.

IEC Check

Perfection in Automation
www.br-automation.com

An Example of IEC Check Customization

```
FUNCTION CheckBounds (* Check range for array access *)
    IF index < lower OR index > upper THEN
        itoa(lower, ADR(LowString));
        itoa(upper, ADR(UpString));
        itoa(index, ADR(IndexString));
        CheckBounds := upper;
        strcpy(ADR(text),ADR('Array Index '));
        strcat(ADR(text),ADR(IndexString));
        strcat(ADR(text),ADR(' is outside ['));
        strcat(ADR(text),ADR(LowString));
        strcat(ADR(text),ADR('..'));
        strcat(ADR(text),ADR(UpString));
        strcat(ADR(text),ADR(']'));
        MakeEntry(55555,index,text);
    ELSE
        CheckBounds := index;
    END_IF
END_FUNCTION
```

```
FUNCTION MakeEntry (* Makes a log book entry *)
    status_name := ST_name(0,ADR(taskname),ADR(group));
    PV_xgetadr(ADR('gIecChkSec'), ADR(pv_adr), ADR(data_len));
    IF pv_adr > 0 AND data_len > 0 THEN
        memcpy(ADR(SectionData),pv_adr,data_len);
        itoa(SectionData,ADR(SectionString));
    END_IF

    strcpy(ADR(out_text),ADR(text));
    strcat(ADR(out_text),ADR(' in task "'));
    strcat(ADR(out_text),ADR(taskname));
    strcat(ADR(out_text),ADR('" Section:'));
    strcat(ADR(out_text),ADR(SectionString));
    MakeEntry := ADR(out_text);
    ERRxfatal(number,index,ADR(out_text));
END_FUNCTION
```

	Level	Time	Error ...	ASCII Data	Binary Data
1	Error	2013-03-14 14:23:13...	55555	Array Index 7 is outside [0..6] in task "AdvIndexHi" Section:16	007 000 000 000

```
gIecChkSec := 1;
gIecChkSec := 2;
gIecChkSec := 3;
```

In my experience, most people don‘t realize that these IEC Check functions are open to editing. Naturally, with this great power comes great responsibility. What I want to expose you to here is what customization might look like and how it could help. What piques the interest when I talk about customization is how we can narrow down in code where a problem occurs. Here I’m creating a referencing global variable called IEC Check Section (gIecChkSec). While what is passed to the CheckBounds function is fixed, I can use the PV_xgetadr function to read the value of gIecChkSec in and include it in the logging.

Strategically, I can increment this variable as it goes through my code and at different sections, and when a fault occurs, I will have a narrower section of code to troubleshoot.

IEC Check Customization in Practice

Perfection in Automation
www.br-automation.com

Customization with section markers allow for rapid bug diagnosis

If I were to put these increments throughout my code, next time this error occurs, I can narrow down what would be too many lines of code into something manageable.

This section, combined with the fact that I know what the valid array range is, can accelerate troubleshooting dramatically.

IEC Check

Perfection in Automation
www.br-automation.com

Important Notes About Adding & Removing IEC Check:

- Once added to your project, even if not explicitly added to your configuration, the check functions are implemented.
- Disable in selective configurations with additional build option -D _IGNORE_CHECKLIB

I pulled this right from the help. Once you add this IEC check library to your project, all configurations upon a rebuild will have those functions implemented, even if you don't add this library into a specific software configuration. This was surprising to me. However, you can selectively ignore the check library within the build options command line message written here.

Perfection in Automation
www.br-automation.com

Process Variable Functions

The PV Functions

Process Variable Functions

There are 4 universal PV functions that work on all systems

- PV_xgetadr() – gets the address and the size of data in memory
- PV_xlist() – is used for getting data about all process variables on the system, one PV per call
- PV_ninfo() – gets the data type, length in bytes of the PV as well as the size of the array or number of elements in the structure
- PV_item() – used for getting the names of the elements of a structure.

Except for PV_xlist(), all require the name of the variable as an input.

PV Functions are powerful ways to gather data about variables. By supplying a string of the variable name and program, you can read back and display values inside your visualization.

To create a generic interface, you use these functions in conjunction to create tools that can remotely help troubleshoot problems without requiring Automation Studio.

PV Data types, Dimension

Process Variable Functions

There are 23 different data types that can be returned, (table in the help for PV_info, linked from PV_ninfo)

To Note:

- Defined for all 18 IEC data types supported by B&R (plus LWORD, LINT, ULINT)
- Data type 0 is returned for enumerations, structures, and derived data types
- Data type 15 is returned for arrays of structures, and the dimension parameter is the number of structures in the array, NOT the number of elements in the structure.

The datatype returned is an important part of creating a watch window. Since you'll be returned an address from PV_xgetadr(), you'll want to make sure any dynamic variables / pointers are of the correct data type.

Names of Process Variables

Perfection in Automation
www.br-automation.com

Process Variable Functions

- Global variables, enter the variable
  ```
  ADR('globalVar1')
  ```
- Local variables, **<Object-Name>:<PV-Name>**
  ```
  ADR('task:localVar2')
  ```
- Dynamic variables, use "*" in front of the variable
  ```
  ADR('*pGlobalDynVar1')
  ```
- Structures use a similar format
  ```
  ADR('task2:structure1.element1')
  ```
- More Info in Automation Studio Help
  - Programming>Variables and Data types>Variables>Names of Process Variables

The PV functions require that you pass the name of the variable as a string. Here are a few examples of what this would look like for global, local, dynamic and structure elements.

A full list of information is in the Automation Studio help.

Example:

Perfection in Automation
www.br-automation.com

Process Variable Functions

varUDINT UDINT[0..31]

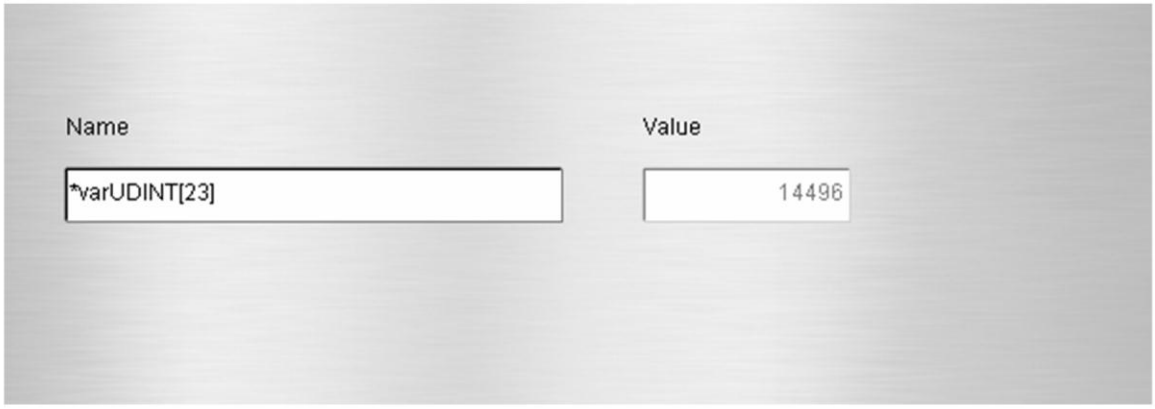
```
sts_xgetadr := PV_xgetadr(ADR('varUDINT'),ADR(address),ADR(data_len));
sts_ninfo := PV_ninfo(ADR('varUDINT'),ADR(data_typ),ADR(data_len),ADR(dimension));
```

Name	Value
address	46638332
data_len	128
data_typ	7
dimension	32

Example: Virtual Watch Window in HMI

Perfection in Automation
www.br-automation.com

Process Variable Functions



- Can be used to add a helpful watch window to your HMI or a VNC HMI for service techs to use without connecting Automation Studio up to the PLC

Here we can create a virtual watch window. What's important here is that the display value is of the same data type. It may be required that you have multiple display windows for the same data type, or that you perform type conversions. Mixing floats with integers or bools can be problematic!

Process Variable Function Warning

Perfection in Automation
www.br-automation.com

Process Variable Functions

To ensure that the performance of the system is not affected, these functions should only be used in initialization sub-programs or low priority, high tolerance task classes.