Application and System Diagnostic Framework on CIP™

Joakim Wiberg
Manager Technology and Platforms
HMS Industrial Networks

Presented at the ODVA
2014 Industry Conference & 16th Annual Meeting
March 11-13, 2014
Phoenix, Arizona, USA

Abstract:

The requirements for diagnostics in industrial control applications have grown continuously over the last decade and have become a critical requirement for many industrial control applications. This paper describes a conceptual scalable and highly flexible diagnostic framework for application and system diagnostics, to be deployed in end devices and in diagnostic aggregator devices. Diagnostic aggregators make it possible to collect and store the diagnostic information for a subnet and provide that information to higher level supervisory diagnostic system tools sitting on upper level plant network(s). The proposed diagnostic framework is based on existing CIP™ communication mechanisms and also extends existing mechanisms to adapt the Heartbeat diagnostic notification to EtherNet/IP™.

High level alarms, warnings, and event flags are generated by the end device in Device Heartbeat notification message, with the more detailed diagnostic data of interest for the device logged and stored within the product. The diagnostic data stored within a product are accessible in a standard way using CIP™ communications and objects. This allows cross vendor platforms to be used to analyze and diagnose the information details after receiving a Device Heartbeat notification from an aggregator. Enhancements to the EDS make it possible for supervision systems to map and present internationalized text strings for all events with little resources needed in end devices. Higher level end devices with more memory resources have the option to store all the texts strings associated with events internally.

Keywords:

Diagnostic, Scaled Architecture, Information Modelling, EDS, Common Architecture, Event Logging

Introduction

For years, the EtherNet/IP™ Implementor Workshops have discussed diagnostics on different levels for EtherNet/IP™ and CIP™. The initial discussions where mainly focused on how to troubleshoot and diagnose Ethernet and TCP/IP networks, in order to detect problems with the communications and initial commissioning. As a continuation of this effort, it was also identified that CIP™ did not provide a common and generic way to log and present application diagnostic information and events.

In the European Series of the EtherNet/IP™ Implementor Workshops, the lack of a common and generic way to log and present application diagnostic information and events for EtherNet/IP™ and CIP™ was identified and discussed as a topic in several of the EtherNet/IP™ Implementor Workshop events. As a consequence of this, a working group was formed with the intention to develop and extend EtherNet/IP™ and CIP™ with functionality to allow devices to log and present diagnostic information in a generic way. The working group did some work in the area, but for a variety of reasons the effort fell short and no additions to EtherNet/IP™ and CIP™ where developed in order to address the identified issue.

Most users of industrial control applications expect diagnostics and event logging to be an integrated part of the devices being installed on the plant floor. Using diagnostics and event logging as a part of the automated production is important to increase the flexibility of the manufacturing system, maximize the investment turnover, and minimize the overall downtime. This can be archived when devices implement and provide a full set of diagnostic and event logging functionality along with a framework to report and publish the logged diagnostic information and events. Examples of diagnostic information that helps maximize the investment turnover and minimize the downtime could be a misaligned photo-eye reporting and therefore would need a service technician to correct its position or a motor that has been used at a certain load over a defined number of hours and therefore should be scheduled for service.

Devices collecting and reporting diagnostic information and events are by no means anything new. In fact, devices have reported diagnostics long before they became equipped with network connectivity. Though the real use and advantages of collecting and reporting diagnostics and events came along with the network connectivity, several fieldbus networks have since long had standardized and defined means to report diagnostic information and log events. Since this diagnostic information are reported and collected using a standardized function in the fieldbus communication, it is vendor neutral and allows it to be used in multi-vendor networks.

As industrial communication started adopting Ethernet and TCP/IP for control and to transport data, diagnostics have become yet even more important since this makes it easier gather the diagnostic information and tie that into higher layer supervision systems. The fieldbus networks already providing functionality to collect and report diagnostic information brought this functionality over when the Ethernet and TCP/IP counterparts where created and developed.

EtherNet/IP™ and CIP™ do not totally lack diagnostic functionality. Though the functionality that EtherNet/IP™ and CIP™ offer currently has some shortcomings. The diagnostic functionality that is defined as a part of the CIP™ Networks Library is limited and narrow. In this case the diagnostic functionality is bound to a specific type of product, e.g. an I/O block or a photoelectric sensor, but in these cases the functionality is very narrow and does not allow for different types of events and diagnostics to be reported. In the cases where enough diagnostic and events are provided, the functionality has been developed in a vendor specific way by each vendor. This make multi-vendor networks a big effort to build, and mixing devices, from different vendors, that have the best fit and functionality for that specific installation impossible. In those cases it is not possible to used one single engineering software tool to gather collect and present the diagnostics and event information. This is far from an ideal way of working since the service technician will have to learn and use multiple software tools, also the engineer building and commissioning the network will have to use more than one engineering tool to fully make use of the information from all devices.

Having a generic framework that allows for devices to provide diagnostics and event logging in a common way is important in order to create multi-vendor networks and installations and at the same time provide an elaborated diagnostics and information gathering functionality. This generic framework also is important to create interoperability making sure devices from different vendors play well together. Using defined means to log and present diagnostic information and events is important for end-users and engineers building the industrial control application since they then can choose the device with the best fit for their specific application. Also, it provides means for vendors to differentiate themselves from other vendors by implementing a wide and broad set of diagnostics and event logging functionality and not have to be tied and bound using their own tools to fully make us of the information.

It is of a high importance that a framework for diagnostics and event logging is flexible enough to make it possible to log any form of application and system diagnostic data that can be useful. Furthermore, it must not be limited to application and system diagnostic data, any form of information that could be of use for a device or an end user should be possible to be logged and presented in a generic way by the framework. Having a framework that is flexible enough this will allow for any type of information to be logged in the future. An example of events that could be logged that are not of application type are security related events, like when the normal operation is about to be compromised. It also could be possible to log information taken from the Energy objects when an energy consumption peak has been detected. Besides making the framework flexible enough to allow for logging of any type of diagnostics, events, and data, it must be well defined in order to maximized the interoperability and allow for multi-vendor installations.

This paper presents a diagnostic framework that allow vendors to implement any kind of device or application diagnostics, such as short circuits, misaligned photo eyes, over current detection, or operating thresholds meet. Because of its flexibility it also provide means to aid system and network diagnosis. However the framework does not define any attributes or figures that would guide a user through a system and network trouble shooting, this is outside the scope of the paper and needs to be developed on top of the functionality that the framework presented.

Throughout the document EtherNet/IP™ is used as an example network, the diagnostic framework and most things in the paper is not limited to EtherNet/IP™. It could be adapted to any CIP™ network technology. In fact some of the ideas already exists on DeviceNet™ and are based on the functionality of that network.

Conceptual overview

The remainder of this paper will present and focus on a suggested framework for the application and system diagnostics. It will present all components in the framework as a whole, discussing devices, aggregators, engineering stations used to present and display the diagnostics and events, and also other possible system components. Some design details will be described and discussed, though the focus of the paper is to give a high-level overview.

What components and devices would make use of the application and system diagnostic framework? Consider the simple network in Figure 1. This could be any EtherNet/IP™ network comprising a controller controlling, a set of I/O devices and AC drives, and also connected to the network is a HIM to present the current operation and to make changes to the operation. The PC connected to the network could be an engineering station to program the controller or a supervisor station used to monitor the operation in a control room. The device to the top right would represent something that normally not would be found on an EtherNet/IP™ network, a portable diagnostic tool.

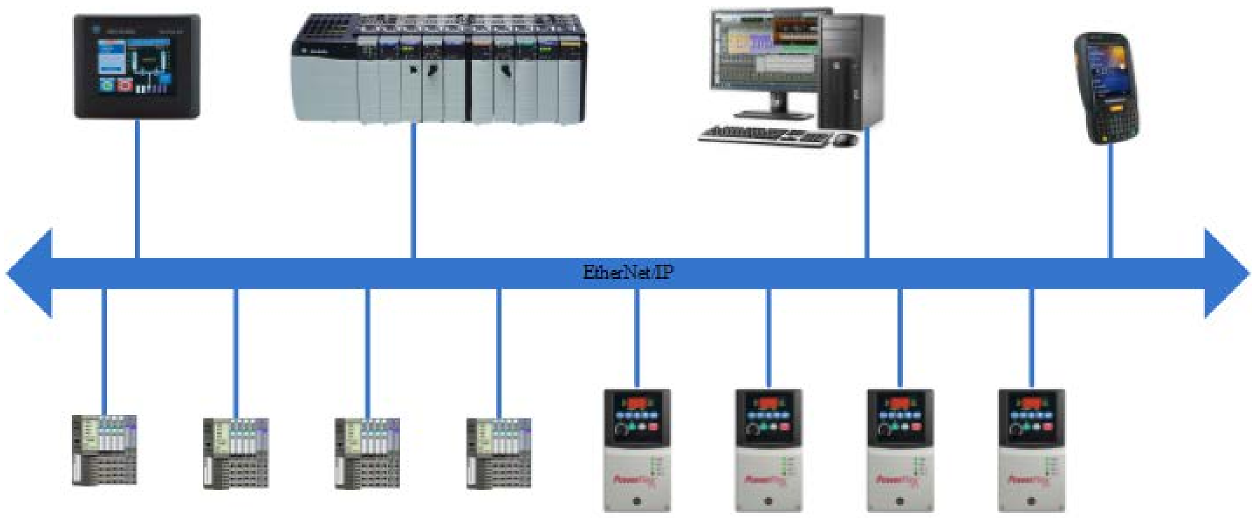


Figure 1 Simple Example Network

The I/O devices and the AC drives are the devices that would implement the diagnostic and event reporting functionality. When an event occurs or the device has something to report, it would log that information and make the information available through an object. The event will be tagged with information based on the type and importance. For example, a critical event would be a short circuit detected in one of the terminals on the I/O block, when the drive has reached a certain number of operation hours an information event would be logged. Once the diagnostic information has been logged the device will produce a Device Heartbeat message to report that it has new diagnostic information. This Device Heartbeat message is sent out in a way allowing any device to see and pick up the message.

When the HMI receives the Device Heartbeat message, it would use explicit message communication to read up diagnostic information from the device that produced the Device Heartbeat. Since the events have been tagged with information about their importance and type, the HMI could be set in a way to filter different events based on their importance and type. The operator of the machine might not be interested that the drive has passed a certain threshold for the number of operation hours and would need a service scheduled, but a short circuit might be of high importance since it might stop the operation.

The supervisor station might be interested in using the same information as the HMI, but also might add history and trending for relevant information. It also might be the station that informs the maintenance personnel to schedule a service for the motor that has passed the operational hour threshold.

A controller that has been set up to receive diagnostic Device Heartbeat messages could be configured to take action based on the event that occurred. An example could be the short circuit which might stop operation of that specific cell, but still maintain operation of other cells not affected by the short circuit event.

The portable diagnostic tool, e.g. a laptop or a dedicated hand-held, can be connected to the network when a service engineer would need to trouble shoot something, or during initial commissioning to make sure that the network is correctly installed and configured. This portable diagnostic tool would be able to pick up any Device Heartbeat and present the information without knowing anything about the network structure or layout or the specific device reporting the event. This would be accomplished by interrogation of the standardized diagnostic object where the events are logged in each device that has diagnostics to report.

Below Figure 2 shows a conceptual network layout where devices with an aggregation function, an Aggregator, resides on different sub-networks. Each Aggregator listens to Device Heartbeat messages on the local sub-network and collects diagnostic information from the devices on the sub-network.

The sub-networks could be actual physical networks structured as such, but it also could be a logical network to structure and assemble the diagnostic information in one or several devices. When the Aggregator is used to build up a physical network, it is located in a CIP™ Router that collects diagnostic data from downstream devices and presents the data to upstream devices. An Aggregator in a logical network could be a single port device that collects the data and presents the data on the same physical interface. The use of this kind of device is to have all diagnostic data collected and assembled in one single point on the network. This kind of device could be a Controller.

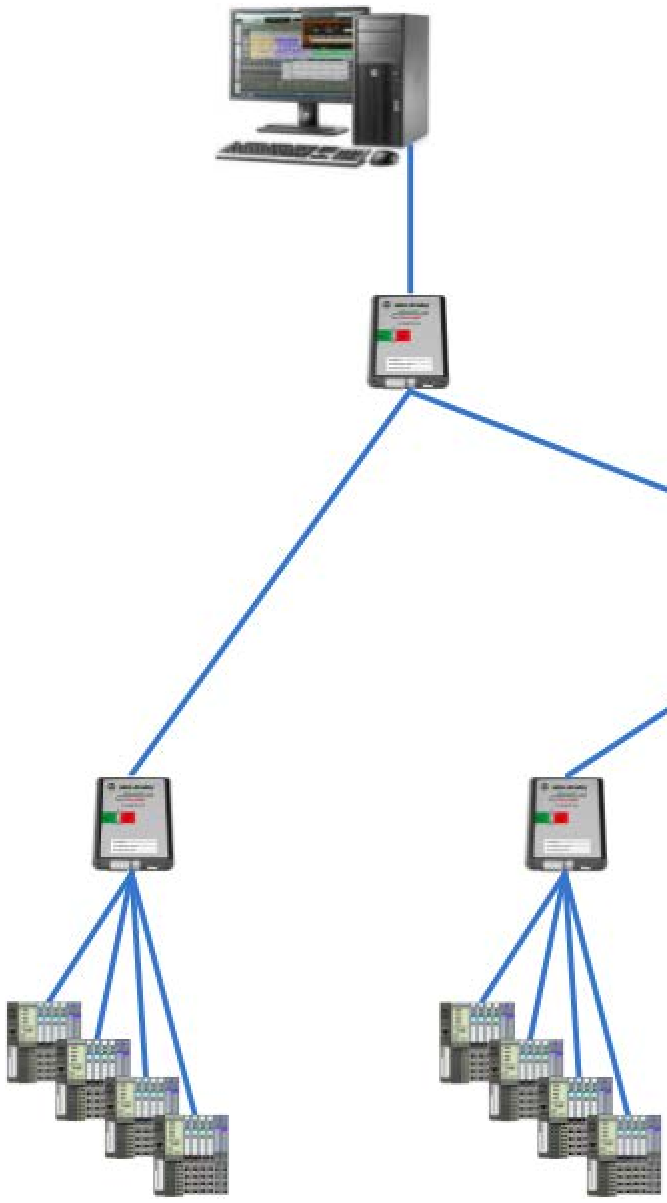


Figure 2 Conceptual Network Layout with Aggregators

The Aggregators also would act as Device Heartbeat producers when they have collected data from devices from where they aggregate data. Once the Aggregator has consumed a Device Heartbeat message from a device, it stores that information in an aggregated diagnostic object. The aggregated diagnostic object holds all Device Heartbeat messages, which the Aggregator has collected, together with some additional information. When the Aggregator has stored the information from the Device Heartbeat message it produces a Device Heartbeat message of its own allowing other Device Heartbeat consumers to see that it has new diagnostic information available.

Along with the information in the Device Heartbeat message, the Aggregator stores the path to the device that produced the Device Heartbeat message. Using this information, devices higher up in the hierarchy can drill down through the segmented network hierarchy. If the Aggregator is implemented in a CIP™ Router, it also makes it possible to reach to devices sitting on different CIP™ Networks.

Collecting data using Aggregators and having the local diagnostic from one sub-network concentrated in one single point makes it easy to structure the network. Also having the diagnostic data assemble in one single point a network makes it easier for devices sitting on higher layers to gather data from devices on lower layers.

Device Heartbeat messages

In order to have a way for a device on EtherNet/IP™ to distribute and publicize that there has been a change in its current state and that new diagnostic or event information exists to more than one recipient, an extension to the EtherNet/IP™ protocol needs to be developed. A Device Heartbeat message similar to the one available on DeviceNet™ needs to be defined. This message needs to be short, unsolicited, unconnected, and generated by the device. As shown in Figure 3, the Device Heartbeat message shall be sent by any device that has diagnostics or events to report, hence the unsolicited and unconnected nature it can be consumed by any device interested in doing so.

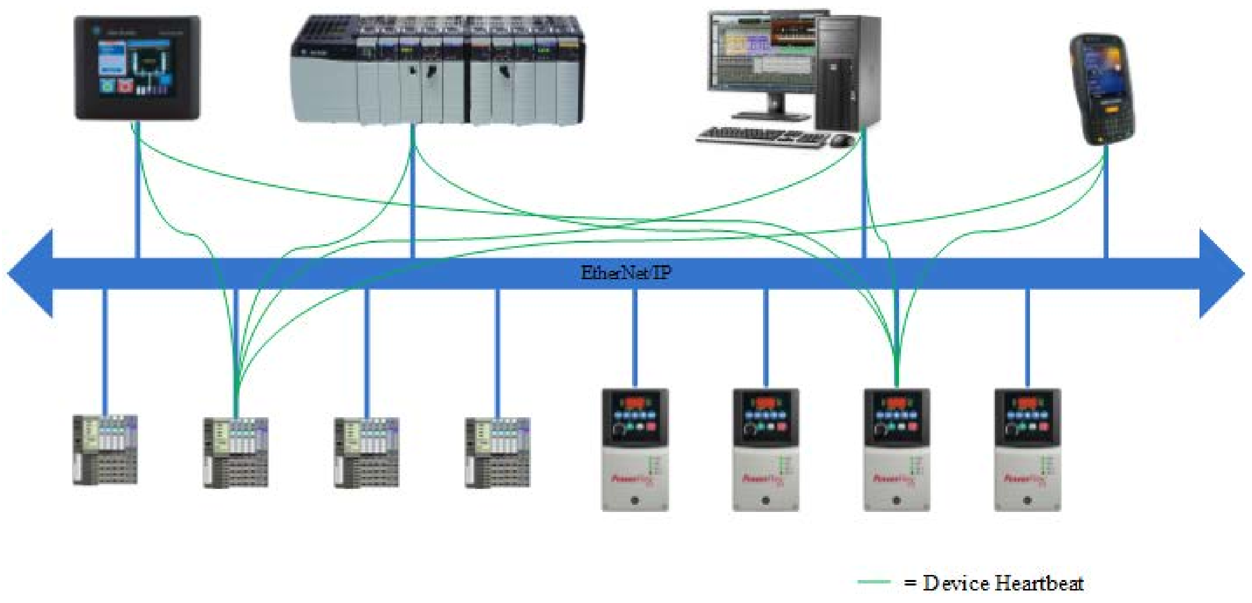


Figure 3 Device Heartbeat Messages

The Device Heartbeat could be sent as unicast, multicast, and broadcast. There are pros and cons with each of the three different delivering methods.

Unicast has the advantage that it reduces the bandwidth utilization and does not load the network with traffic on segments where there are no consumers of the message. The drawback that comes with unicast is that each and every potential consumer would have to be configured in the Device Heartbeat producer. This is a manual and rather cumbersome process, especially if the network in question contains a large number of nodes that will produce Device heartbeat messages. It also prevents easy usage of diagnostic devices that are plugged into the network temporarily to diagnose a potential issue.

Broadcast messages overcome the issue where each and every Device Heartbeat message producer has to be individually configured. However, it has the limitation that it “floods” the network with unnecessary traffic since each Device Heartbeat could be sent on every link in the system regardless if there is a consumer or not.

Multicast by default shares the same pros and cons as broadcast. But using IGMP snooping aware switches the unnecessary traffic would be reduced and only sent on links where there is an actual consumer of the Device Heartbeat message. For IGMP snooping to fully work, it is not only the switches that have to support IGMP, but the

Device Heartbeat consumers would have to implement the IGMP protocol. The Device Heartbeat consumers via IGMP would inform the switches that expects multicast on a specific address to be sent down its link.

The preferable deliverable mechanism for Device Heartbeat messages on EtherNet/IP™ would be to use multicast and this in conjunction with IGMP support in the Device Heartbeat consumers. The usage of multicast in combination with IGMP snooping aware switches and support for IGMP in Device Heartbeat consumers reduces the traffic to only be sent on ports where the messages will be consumed. Furthermore, if there is more than one consumer, which is a likely configuration in a larger network with hundreds of nodes of the Device Heartbeat messages, only one single message will be sent. This message will be consumed by all Device Heartbeat consumers which decreases the bandwidth utilization even further. In addition, the Device Heartbeat messages are sent at a slow background rate, so even in smaller systems without IGMP snooping aware switched the bandwidth utilization would be slow. Multicast combines the ease of ease of configuration with the possibility to optimal make use of the bandwidth by using the correct infrastructure components.

Using IP Multicast would require the Device Heartbeat messages to be sent over UDP. The Device Heartbeat messages would be sent on the UDP port registered for EtherNet/IP™ Encapsulation Protocol. For Device Heartbeat Consumers to distinguish between Device Heartbeats and other data sent on the same UDP port, a new encapsulation command needs to be defined. The actual Device Heartbeat data would be a part of a Common Packet Format following the Encapsulation Header and a new Item ID number would need to be defined for the Device Heartbeat Information. Table 1 shows how the Encapsulation Header and the Common Packet Format would be laid out.

Structure	Field Name	Data Type	Field Value
Encapsulation header	Command	UINT	Device Heartbeat
	Length	UINT	Length of the command specific data
	Session handle	UDINT	Any value (ignored by receiver).
	Status	UDINT	0
	Sender Context	ARRAY of octet	Value from request. Length of 8.
	Options	UDINT	0
Command specific data	Item Count	UINT	Number of target items to follow
	Target Items	STRUCT of	Device Heartbeat Information
		UINT	Item ID
		UINT	Item Length
		ARRAY of octet	Item Data

Table 1 Encapsulation Header

To confine the Device Heartbeat traffic to the local network, it is recommended that TTL scoping is used. In the IP header a field called Time To Live (TTL) is defined, see Figure 4. This field defines the lifetime of the frame and how far it can travel. Despite the name, it is really a count of the number of hops (transmission from one router to the next) the packet is allowed. The TTL field is decremented by one each time a packet leaves a router and a packet with a TTL of zero is discarded. Setting this TTL filed to a value of one (1) would limit to the scope of the local network.

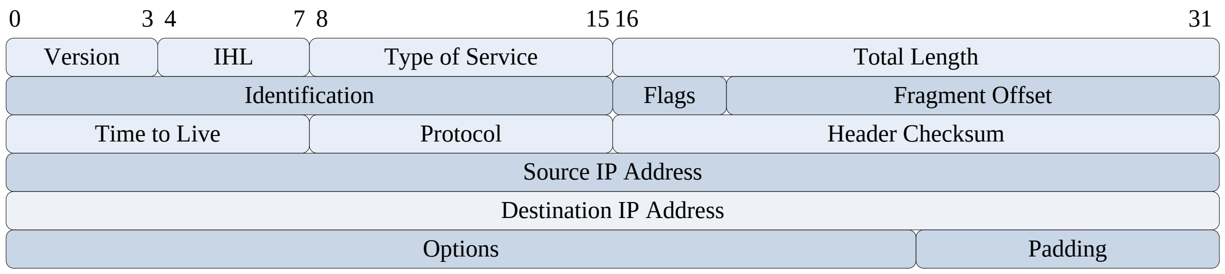


Figure 4 IP Header showing TTL

TTL scoping is used also used when sending EtherNet/P packets via IP multicast, in this case the TTL value is configurable to allow the traffic to traverse across IP multicast routers. There are use cases for Device Heartbeat messages to traverse IP multicast routers as well. The same attribute that configures the TTL value for EtherNet/IP™ packets could be used for Device Heartbeat, however since the Device Heartbeat messages and general EtherNet/IP™ multicast packets are used for vast different things it is more flexible to have a separate attribute that configures the TTL for Device Heartbeat messages. Table 2 shows how the Device Heartbeat TTL Value attribute would look implemented in the TCP/IP Interface Object [1].

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
<snip>							
13	Conditional[1]	Set	NV	Device Heartbeat TTL Value	USINT	TTL value for Device Heartbeat messages	Time-to-Live value for IP multicast packets. Default value is 1. Minimum is 1; maximum is 255
Table Footnotes							
1 Required if the device can produce Device Heartbeat messages							

Table 2 Device Heartbeat TTL Value Attribute in TCP/IP Interface Object

The usage of IP Multicast messages is similar to how the Device Heartbeat messages work on DeviceNet™. Although on CAN, every message can be seen by any node as it is a shared medium. To accomplish this on EtherNet/IP™ that uses Ethernet and the TCP/IP suite as a carrier for its messages, an IP Multicast message over IP needs to be used. A new IP multicast address, besides the ones already defined for Class 0 and Class 1 usage, would be allocated from the IPv4 Organizational Local Scope to be used for Device Heartbeat messages. EtherNet/IP™ today defines two mechanisms to allocate IP Multicast addresses, algorithm based on the device's IP address or manual configuration through an attribute in the TCP/IP Interface Object. For EtherNet/IP™ Class 0 and Class 1 data, a range of IP multicast addresses is allocated per device. For Device Heartbeat the normal case would be to use the same IP Multicast address for all devices. Therefore, a default IP IPv4 Organizational Local Scope needs to be defined. There are, however, cases when different groups of Device Heartbeat produces would be needed on the same sub-network. This makes it possible to virtually segment the network. To allow for this, a new attribute would be required in the TCP/IP Interface Object. Table 3 shows an example how this attribute could be defined.

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
<snip>							
14	Optional	Set	NV	Device Heartbeat Multicast Address	UDINT	The device's IP Multicast Address for Device Heartbeat Messages	IP Multicast Address (Class D). A value of 0 indicates that the default IP Multicast Address is used.
<snip>							

Table 3 Device Heartbeat Multicast Address Attribute in TCP/IP Interface Object

The Device Heartbeat message would also be sent as Change-of-State in order to speed up the delivery and notification when data changes or new diagnostic and event information has been produced. The main reason for this is that the heartbeat interval that is configured through the Heartbeat Interval attribute in the Identity Object is set in seconds. The current Heartbeat Interval configuration attribute does not contain any settings to configure an inhibit time for the Change-of-State productions. An inhibit time would need to be defined to prevent flooding the network with Device Heartbeat messages during short periods of many changes. However, it does not make sense to burden the user with this inhibit time as a configurable attribute. Instead, the inhibit time for Device Heartbeat messages will default to ¼ of the heartbeat interval. This is a change to the current definition of the Device Heartbeat Message production rule, which states that the Device Heartbeat at a maximum can be produced once per second.

Even though UDP can carry a lot more data than a single CAN frame, there is no reason stuffing the full diagnostic information within the Device Heartbeat message. Rather, it would be preferable to build on the information from the DeviceNet™ heartbeat message. The DeviceNet™ heartbeat message format carries some information (source MAC_ID and DeviceNet™ UCMM service code) that is not relevant for an EtherNet/IP™ device since that is the equivalent to the Encapsulation Header and Common Packet Format discussed above, so that will not be included. However, each Device Heartbeat in the structure will need some "path" information so that tools subscribing to the aggregation can drill down to the node reporting Change-of-State to obtain more detailed information about the diagnostic event(s).

This way, the diagnostic notifications would work in a similar fashion on both DeviceNet™ and EtherNet/IP™ allowing for generic CIP™ diagnostic tools to be used on both networks, and allow EtherNet/IP™ to DeviceNet™ Routers with the aggregation function to collect Device Heartbeats on DeviceNet™ and easily put them in the EtherNet/IP™ Device Heartbeat message that would be the basis of the aggregation structure on EtherNet /IP. To make it easier for Device Heartbeat consumers to detect changes a Heartbeat Sequence Count have added, the counter would be incremented by one when something in the Device Heartbeat message changes. Also, as UDP isn't a guaranteed delivery mechanism, messages can get lost or not be processed by the recipient if there is a heavily loaded network. Lost messages could be detected and dealt with using the Heartbeat Sequence Count. Removing and adding this information would give a proposed message format as in Table 4. The content of the remaining information is identical to the trailing information in the DeviceNet™ heartbeat message, as defined in [2], except from one member. There is a desire to define more flags to help identify different types of diagnostics and events. Therefore, the existing DeviceNet™ heartbeat message has been extended with one byte of reserved flags. Possible flags could be for security, maintenance, information event, warning events, and critical events. Since the extra byte of flags breaks word alignment, an extra reserved byte have been put in after the flag section to make the following data on an even word.

The MA flag is intended to be used when a condition in the device exists that is classified as maintenance would be required. Normally this event does not necessarily require stopping the machine and/or the process being controlled. A typical example could be a "life cycle counter exceeded" indication on a motor starter, a dirty lenses on a photo eye, and so forth.

Four vendor specific bits, VS0 through VS3, have been defined. Those bits are connected to one instance in the Diagnostic Object described below. When a device has some type of diagnostic to report that does not fit into one of the defined bit, it is still possible to report this kind of diagnostics using the Device Heartbeat message. Consumers of the Device Heartbeat can from the vendor specific bits get the same information as if it were a defined diagnostic type but interrogating the Diagnostic Object.

A new flag has been defined, AH (Aggregated Heartbeat), when this flag is set the Device Heartbeat messages have been produced by an Aggregator and isn't the original Device Heartbeat message. When the AH flag is set, the Device Heartbeat message contains additional information besides what is in a Device Heartbeat message produced by an end device. At the end, there are three optional fields that are included when the AH flag is set, including the path to the device who originally produced the Device Heartbeat and the instance number of the object in the Aggregator that produced the aggregated Device Heartbeat.

	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Heartbeat Sequence Count							
1								
2	Identity Object Instance ID							
3								
4	Device State							
5	Severity Level							
6	AH	Reserved	Reserved	Reserved	VS3	VS2	VS1	VS0
7	Reserved	Reserved	Reserved	MA	EV	SF	UF	DF
8	Configuration Consistency Value							
9								
10	Device Heartbeat Instance ID (included if AH flag is set)							
11								
12	Path Size (included if AH flag is set)							
13								
14	Padded EPATH (included if AH flag is set)							
N								

Table 4 Device Heartbeat Message structure

The Severity Level member holds a value defining the importance of the event and corresponds to the Severity Level member in the Diagnostic Object.

Not shown in Table 4 is that extra space would probably need to be allocated for a new alternative to the Configuration Consistency Value. The current definition of the Configuration Consistency Value is weak; therefore potentially the same value could be generated for different configurations. Also, there is no algorithm defined that will be used generating the Configuration Consistency Value, a more robust and well-defined algorithm would have to be defined. Preferably, an existing and proven solution like a Universally Unique Identifier (UUID) would be used. A UUID is larger than the two bytes used by the Configuration Consistency Value, and more space in the Device Heartbeat message would have to be allocated to transfer the UUID. A new attribute to hold the UUID would also have to be defined. This UUID attribute similar to the Configuration Consistency Value also would be placed in the Identity object.

Aggregator

As shown in Figure 5, a device with an aggregation function (Aggregator) would reside on each sub-network and listen for Device Heartbeats, collecting them into a structured array. Aggregators also could aggregate heartbeat structures from other aggregators lower in the architecture. At the upper layers, PCs or other devices with a maintenance and/or diagnostic interest can subscribe to these collected heartbeats via a Class 0 or Class 1 Change-of-State unicast connection.

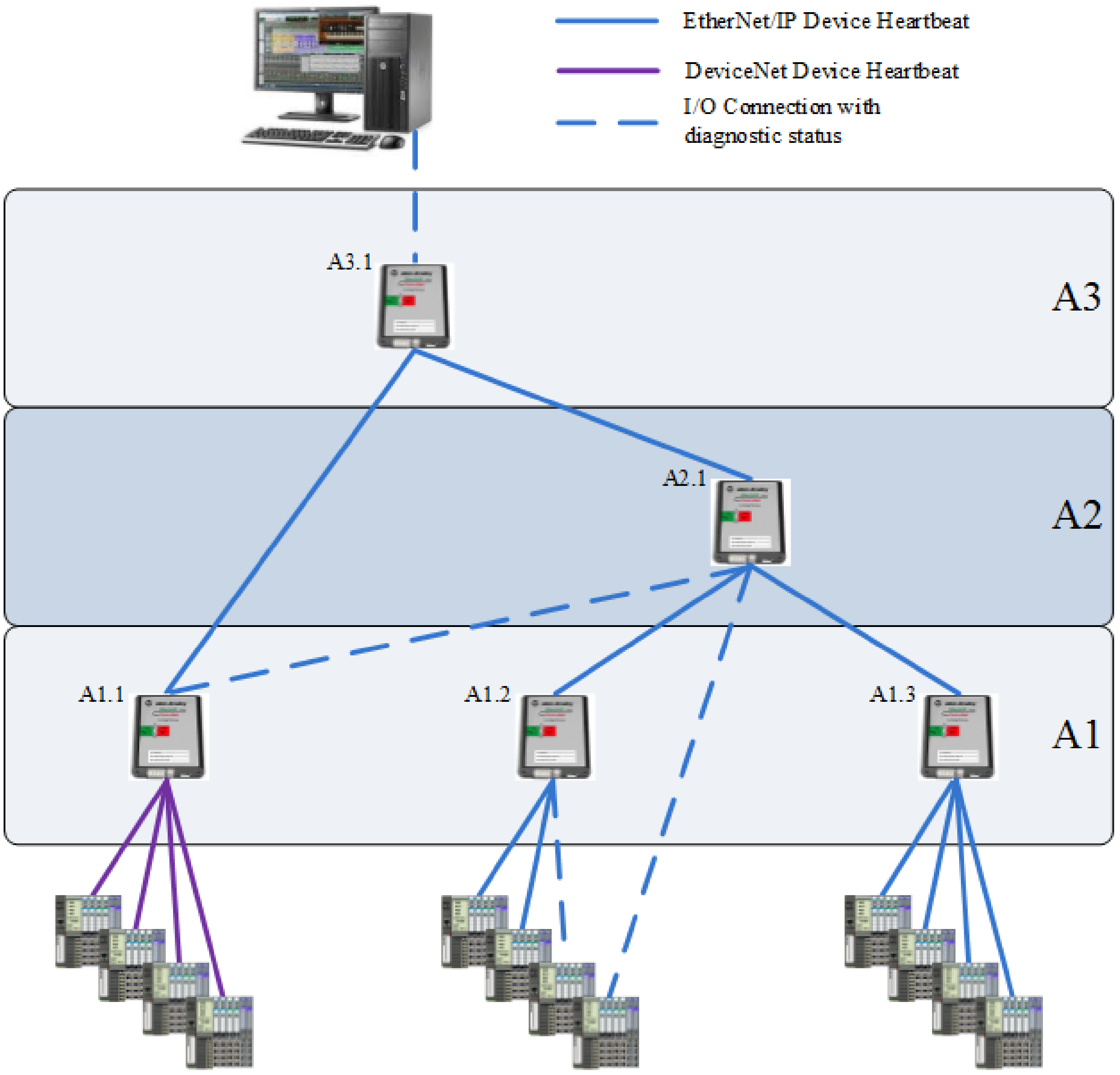

EtherNet/IP Device Heartbeat
DeviceNet Device Heartbeat
I/O Connection with diagnostic status
A3.1
A3
A2.1
A2
A1.1
A1.2
A1.3
A1

Figure 5 Aggregators in a layered network

The Aggregators on level A1 in Figure 5 collect the diagnostics and events from different sub-networks and aggregate the diagnostic information at subnet level. The A1 level Aggregator is normally the controller or PLC that controls the devices on the subnet or logical subnet. The diagnostic information can come from a Device Heartbeat as described above, or it also could be a part of the I/O connection between the controller and the device.

Aggregation can occur a layer above level A1 as well. In Figure 5, this corresponds to the A2 level. The normal operation for an A2 Aggregator is to collect the aggregated diagnostics from A1 Aggregators. But aggregators on the A2 level can reach below the subnet aggregation as necessary. If the A2 Aggregator and the device producing the diagnostics are on the same physical network, the Device Heartbeat messages or I/O connection can be directly brought into the A2 Aggregator. If the two are on physically different networks, the A2 aggregator can listen to aggregated Device Heartbeats as from A1.1 and A1.3, or as in A1.2, a CIP™ Router that supports bridged I/O connections. In the case of a CIP™ Router that supports bridged I/O, it creates a connection directly to the end device. A1.1 is an Aggregator that supports both DeviceNet™ and EtherNet/IP™, and aggregates Device Heartbeat messages from DeviceNet™ to the upper layer EtherNet/IP™ network. On the EtherNet/IP™ side, it can beside the Device Heartbeat also present the diagnostics via an I/O connection.

Aggregation of aggregators can roll up to a system level as in A3.1. From the rolled up system level aggregation in the A3 level, a PC running a diagnostic and analyzing tools their information about the whole system of sub-networks and devices below.

Device Heartbeat message would be put inside an assembly instance. This assembly instance would be required in some sense to be of a variable size since the EPATH member of the Device Heartbeat message can vary in length.

One concern that needs to be considered when building larger networks that aggregate a lot of data is the amount of traffic that could flow up to the top level PC running the diagnostic and analyzing tools. In large systems were there are few or almost no events in the end devices the amount of traffic at the top would be low, thus causing no issues. For larger systems with a lot of events coming and going, the traffic on the trunk line at the top, between the PC and top level Aggregator, would be high. Though the bottle-neck for the aggregated traffic would be the top level Aggregator's internal resources. The Aggregator would run out of resources and performance collecting the incoming Device Heartbeat messages, storing them in the Aggregator Object, see below, and send out the aggregated Device Heartbeat messages. The top level PC would not get starved keeping up collecting the aggregated Device Heartbeat messages, nor would the bandwidth between the two be insufficient. In the case when the top level Aggregator would run out of resources the network could be segmented differently, see Figure 6, by using more than one top level Aggregator. The top level Aggregators would then share the load collecting Device Heartbeat messages from the system, and therefore would address the starving issue when using just one Aggregator. The two top level Aggregators would be connected through a Gigabit switch making sure that the network bandwidth is sufficient between the PC and the top level Aggregators. The network can be structured in ways allowing numerous top level Aggregators making sure that this is not the bottle-neck, instead the top level PC potentially could be the weak link in the system. Though if this would be the case then the number of Device Heartbeats would be in the ten-thousand, or even hundred-thousand, range. In case of systems this large there would most likely be more than one top level PC running the diagnostic and analyzing tools, and therefore they also could share the load by collecting Device Heartbeat massages from different Aggregators.

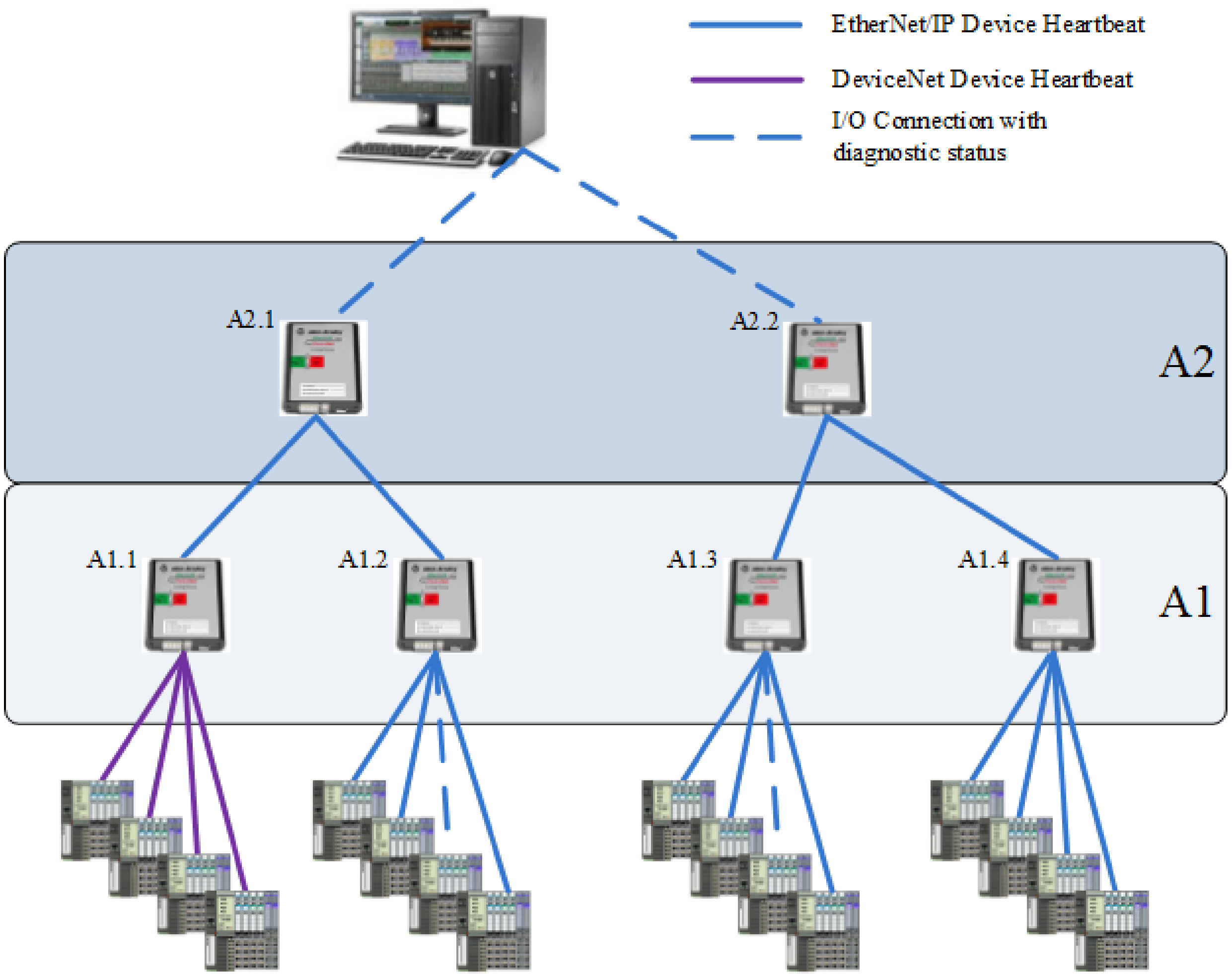

EtherNet/IP Device Heartbeat
DeviceNet Device Heartbeat
I/O Connection with diagnostic status
A2.1
A2.2
A2
A1.1
A1.2
A1.3
A1.4
A1

Figure 6 Aggregators used to share a high network load

The Aggregator collects and stores the Device Heartbeat messages in an object, the Aggregator Object. This is a rather simple object that in the instances just has one attribute which is the received Device Heartbeat. See Table 5. The Heartbeat Sequence Count is used by the Aggregator to determine if the received Device Heartbeat contains new information and, therefore, shall be stored in the object. The Aggregator either adds or prepends the EPATH to the Original Device Heartbeat producer with the IP address of the Device Heartbeat produced and the CIP™ entry port, before it is stored in the object. Each new Device Heartbeat message received by the Aggregator is stored as a new instance of the Aggregator Object. After the Aggregator has stored the updated Device Heartbeat messages in the object, it produces a new Device Heartbeat messages identical to the one stored in the object.

It is the responsibility of the tools higher up in the network hierarchy that consume the Device Heartbeat messages to delete the instances all the way down through all Aggregators when they have retrieved the diagnostic information from the original Device Heartbeat producer. In Figure 5, it is the upper layer PC that collects all the Device Heartbeat messages from the system that deletes the instance after it has drilled down to the end device and read the diagnostic information.

Attr ID	Need In Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	V	Device Heartbeat	STRUCT of:	The actual Device Heartbeat	See semantics section
				Heartbeat Sequence Count	UINT	Sequence counter to indicate changed in the Device Heartbeat	
				Identity Object Instance ID	UINT	Instance ID of the Identity Object who produced the Device Heartbeat	
				Device State	USINT	Attribute #8 of the associated Identity Object instance	
				Diagnostic Flags	WORD	Flags indicating the diagnostics available	
				Reserved	USINT	Reserved for future use	
				Configuration Consistency Value	UINT	Attribute #9 of the associated Identity Object instance	
				Original Device Heartbeat Producer Path	STRUCT of:	Path to the device who produced the Device Heartbeat	
					UINT	Size of Path (in words)	
					Padded EPATH	See Appendix C for format of this field	

Table 5 Device Heartbeat Object

When a tool running on an upper layer PC drills down through the network hierarchy, it makes use the Original Device Heartbeat Producer Path to traverse the different layers. Consider the simple system in Figure 7 that on the top have a PC running some sort of diagnostic tool going through two Aggregators to reach down to the end devices when reading the diagnostic data.

Once the DeviceNet™ device with MAC ID 11 has produced a Device Heartbeat, it will be consumed by the A1 Aggregator. A1 creates an instance in the Aggregator Object, add the Original Heartbeat Producer Path, and then sets the AH bit and then produces the aggregated Device Heartbeat on port 2. The Original Heartbeat Producer Path in A1 would look like:
Path Size: 01 00
Path:	03 0B

The aggregated Device Heartbeat will be consumed on port 4 by the A2 Aggregator, which prepends the Original Heartbeat Producer Path with the port number, where the Device Heartbeat was received, and the IP address of the A1 Aggregator, stored the aggregated Device Heartbeat message and then produced its own aggregated Device Heartbeat messages that the upper layer PC consumes. The Original Heartbeat Producer Path in A2 that include both the path on DeviceNet™ and EtherNet/IP™ looks like:
Path Size: 08 00
Path:	14 0B 31 39 32 2E 31 36 38 2E 30 2E 34 00 03 0B

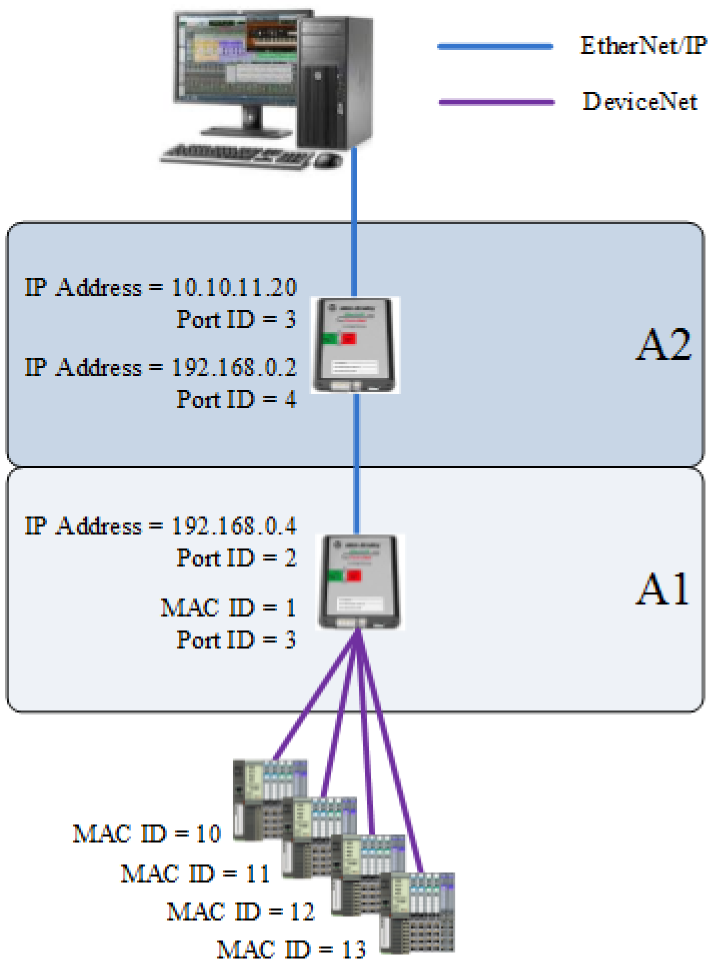


Figure 7 Example network with Port and Node IDs

The Aggregator Object also provides options to configure filtering of incoming Device Heartbeat message and the behavior on for creation and automatic deletion of instances. Table 6 shows the Class Attributes used to configure the above described options.

The Diagnostic Flag Mask is a simple acceptance mask where the value in the attribute is applied using a logical AND to the Diagnostic Flags in the Diagnostic Heartbeat message. If the results in any bit being set the received Device Heartbeat message is accepted and processed and if the result is all zeros then the Diagnostic Heartbeat message is discarded and no further processing is done. By default, this attribute shall be all one, meaning that all Diagnostic Heartbeat messages will be accepted.

Using the Severity Level Filter attribute it is possible to filter on the severity level of the events in Diagnostic Heartbeat messages. Using this attribute only events with the severity level equal to or higher (lower numerical values) than the value configured in the attribute will be accepted.

It is also possible to filter on IP addresses using the Device Heartbeat IP Address Mask attribute. This attribute is used as an acceptance filter for the destination IP in the Device Heartbeat messages received. The default value for this attribute is the default IP Multicast address defined for Device Heartbeat messages. Since the end users can configure the IP Multicast address to in Device Heartbeat messages from each device individually this creates a highly flexible way to create virtual diagnostic network within one physical network.

Since the instances are automatically created by the Aggregator when receiving and accepting a Device Heartbeat messages the number of instances might grow and at the end the Aggregator will run out of memory. Therefore, it is possible to configure, using the Device Heartbeat Storage Policy attribute, the behavior on how instances shall be handled and created when a new Device Heartbeat message has been received. There are three possible options:

1. Only keep one instance from each Device Heartbeat producer
2. Creating instances until a defined limit, when this limit is reached start overwriting the oldest instances.
3. Creating instances until a defined limit, when this limit is reached stop accepting Device Heartbeat messages.

Attr ID	Need In Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
<snip>							
8	Required	Set	NV	Diagnostic Flag Mask	WORD	Filter mask applied to Diagnostic Flags on received Device Heartbeat messages	See semantics section
9	Required	Set	NV	Severity Level Filter	USINT	Sets the minimum Severity Level of events that shall be accepted	See semantics section
10	Required	Set	NV	Device Heartbeat IP Address Mask	STRUCT of:	List of IP addresses that shall be masked	See semantics section
					UINT	Number of filtered addresses	
					UDINT	Device Heartbeat IP Address	
11	Required	Set	NV	Device Heartbeat Storage Policy	USINT	Defines how instances will be used to store Device Heartbeat	See semantics section

Table 6 Aggregator Object

Diagnostic Object

Any device implementing and supporting Device Heartbeat messages also shall implement means to log events and store a history of the logged events. This history of logged events shall be accessible from CIP™ using standard explicit messaging services. There is already an object defined in [3] called Event Log Object that is designed to provide an event indication and/or event logging for CIP™ nodes. This is a highly flexible object that can be used in many different ways to realize event logging and to store the logged events to form a history.

It would seem logical to use the exiting Event Log Object for the task to log and store event created in the device. There are, however, some issues with the Event Log Object and the way it is defined. It was defined to handle any kind on information that a CIP™ nod or application ever could log, therefore, the object is huge and complex to use. Another major issue using the Event Log Object is that it is very loosely defined where almost all attributes are optional, thus making generic and interoperable implementations almost impossible. Both the complexity and the lack of requirements in the definition and usage of the object could be corrected by updating [3], and by doing that making is possible to use the Event Log Object. This paper, however, explores a possibility defining a new object, the Diagnostic Object.

The Diagnostic Object is a small and light-weight object that any device easily could implement, yet it is designed to be flexible enough to handle any type of diagnostic and event logging that would be required within a CIP™ node or by the application. Based on the discussion in this paper, the thoughts and conclusions could also be used to update the Event Log Object by updating [3] with some requirements if the Event Log Object definition.

Table 7 shows the instance attributes from a suggested new Diagnostic Object. The object maintains one instance per bit, bit 15 excluded, from the flag word in the Device Heartbeat message, where bit 0 corresponds to instance 1, bit 1 to instance 2, and so on. Using this schema the client who receives a Device Heartbeat message can easily interrogate the correct instance within the Diagnostic Object.

When the list of events grows as new events are logged, it is possible for configure the behavior when the event list is full. For this purpose the List Full Action attribute is used. The configurable option is to halt the logging or scroll the list and push out the oldest event. The default value it to scroll the list.

When logging events, it is common that the same event gets logged over and over again. The object allows the end-user to define the behavior when these duplicate events are logged. Three configurable options are available: Ignore, Add, and Overwrite. Ignore is the default value and in this case duplicate events is not logged. In the Add case, duplicate events are added at the end of the event list as any new event. Using the Overwrite option the device will replace duplicate event stored in the event list with the newly logged event.

Each instance holds an array of logged events and related information. The related information might vary from device to device based on the capabilities and resources of the device. For example, a small device with limited code space and not real time clock might not have the space to store the textual strings for all possible events that it can log. It will also not be able to provide the time and date when the event was logged. Larger devices with more memory capabilities can store all the textual strings and might have a real time clock or implements IEEE-1588, thus being able to provide the time and date when the event was logged. The Event Code and Severity Level member cannot be excluded from the Event Information and must be implemented by all devices. In order to let clients know what information a device supports in the event list, it can read the Event List Contents attribute. This attribute is a bit field where the bits correspond to a member of in the Event Information structure.

The Severity Level member defines a set of discrete levels identifying the importance of the event. Using the Severity Type member, tools and applications can filter and group events based in the interest and importance. The severity levels defines in the Diagnostic Object are loosely based on the severity levels define for Syslog [4]. Table 8 shows the severity levels defined by the Diagnostic Object.

Attr ID	Need In Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	NV	Severity Type Description	SHORT_STRING	Textual representation of the Severity Type attribute	See semantics section
2	Required	Get	NV	List Max Size	UINT	Max number of entries that the Event List can contain	See semantics section
3	Required	Set	NV	List Full Action	USINT	Configures the action to take when a new event is detected and the log is full.	See semantics section
4	Required	Set	NV	Duplicate Action	USINT	Configures the action to take when a duplicate event is detected.	See semantics section
5	Required	Get	NV	Event List Contents	DWORD	Defines what structure members that are a part of the Event List	See semantics section
6	Required	Get	V	Event List	STRUCT of:	List of all logged events	See semantics section
				List Size	UINT	Number of entries diagnostic entries	
				Event Information	ARRAY of STRUCT of:	Array of diagnostic entries	
				Event Code	UINT	Identifier uniquely identifying this diagnostic event	
				Severity Level	USINT	The severity of the event	
				Event Code Description	SHORT_STRING	Textual representation of the diagnostic event	
				Time	DATE_AND_TIME	Data and time when the event was logged	

Table 7 Diagnostic Object

The Event List attribute is where all the information about the logged events are stored. The information stored for each event is a structure where members match the Event List Contents attribute. The attribute is a structure comprising a size member and an array of event information structures. The size member defines the number of entries that the Event Information array contains. Each time the device logs a new event, it is evaluated against the Duplicate Action attribute and it is determined whether the new event is to be added to the list or not. If the new event is accepted to be added to the event list, the List Full Action attribute is used to determine the behavior.

Code	Description
0	Emergency
1	Alert
2	Critical
3	Error
4	Warning
5	Information

Table 8 Severity Levels

New events that are logged and stored in the Event List are added to the end of the list to allow clients to easily work with the Event List the object implements the Get_Member and Remove_Member service. Beside those two member services, an object specific member service is provided by the object Get_Next_Unread_Member. See Table 9. The Get_Next_Unread_Member object specific service provides a mechanism to get the last unread event from the Event List attribute. The response parameters for the Get_Next_Unread_Member object specific service is one member of the Event Information member of the Event List attribute structure. In the case when there are no unread members in the Event Information, a success reply is returned with zero data length.

Service Code	Need in Implementation		Service name	Description of Service
	Class	Instance		
4Bhex	n/a	Required	Get_Next_Unread_Member	Returns the next member in the Event List Attribute that has not been read yet.

Table 9 Get_Next_Unread_Member Service

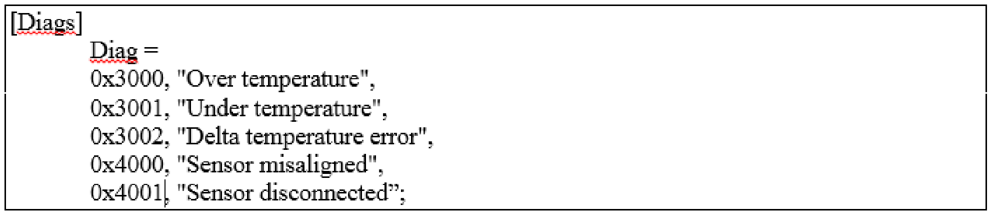
Many low end devices with limited resources still can report a wide range of diagnostic information. Storing all diagnostic textual strings in a low end device is many time not feasible or even possible. In this case, the device would implement the Event Code within the Event List attribute. Devices that don't implement the Event Code Description can still provide the till textual descriptions of the logged diagnostic information. This is done using EDS constructs using a Diag keyword. See Table 10 for a description for the format of the Diag keyword.

Field Name	Field Number	Data Type	Required/Optional
First Event Code	1	UINT	Required
First Event Code String	2	STRING	Required
Nth Event Code	3,5,7,…	UINT	Optional
Nth Event Code String	4,6,8,…	STRING	Conditional[1]

Table Footnotes
1 Required if preceding field is specified, not allowed if preceding field is not specified.

Table 10 EDS Diag Keyword

Figure 8 shows an example of how a Diag keyword would look like in an EDS file. The first field if the Event Code from the Event List attribute. A client reading the Event List attribute from a device not implementing the Event Code Description can simply parse the EDS file for the device in question and look up the Event Code from the Diag keyword and display the textual description provided in the EDS file.



```
[Diags]
        Diag =
        0x3000, "Over temperature",
        0x3001, "Under temperature",
        0x3002, "Delta temperature error",
        0x4000, "Sensor misaligned",
        0x4001, "Sensor disconnected";
```

Figure 8 EDS Example

Conclusion

This white paper discussed and presented an architecture for an application and system diagnostic framework on CIP™. Today, no such solution exists on CIP™ at this level presented herein. The framework allows for a highly flexible and generic functionality, which would be easy enough to implement in small and simple devices but still provide enough functionality to fulfill the requirements in high end device that can report a wide range of functionality.

Furthermore the framework is designed in a way, using the Aggregators devices, so it can be used from the smallest network up to larger networks with several of thousand devices reporting diagnostics. Within larger networks it is also possible to create “isolated” diagnostic segments, thus, controlling and optimizing the network bandwidth used for diagnostic traffic.

Since the functionality suggested and discussed in the paper, for the diagnostic framework, is designed around standardized CIP™ functionality this ensures the interoperability. This is a guarantee that devices from different vendors implementing the functionally would work without any issues in a mixed vendor installation.

The paper presents and discusses the features and functions needed in a diagnostic framework, and going into some specific design and implementation details. Hopefully, the information presented can be used as a starting point for the creation of diagnostic framework for CIP™.

References:

[1] ODVA, Inc. The CIP Networks Library, Volume 2: EtherNet/IP Adaptation of CIP, PUB00002
[2] ODVA, Inc. The CIP Networks Library, Volume 3: DeviceNet Adaptation of CIP, PUB00003
[3] ODVA, Inc. The CIP Networks Library, Volume 1: Common Industrial Protocol (CIP™), PUB00001
[4] R. Gerhards, Internet Engineering Task, RFC 5424, March 2009