

# PC-based Data Access

## *Protocol exploration*

Tue, Nov 12, 2002

This note explores an idea for access to controls data that is sourced via a PC. It explores possible protocols already supported within the front end, including those designed for access to data from SRMs or PLCs.

### *SRM data access*

Using arcnet, a request message is sent at the start of every 15 Hz cycle that asks for all the data sourced by an SRM. To guarantee correlated data, the system awaits subsequent replies that are delivered by the SRM after it collects all of its data for that cycle. Support for settings is handled by delivering an appropriate arcnet message.

This protocol is designed for arcnet, and it includes an Acnet header, which means it contains a layer of complexity that is not needed for simple access to data.

### *PLC data access*

Using ethernet, a local application manages the data acquisition by periodically, not necessarily at 15 Hz, requesting data to be copied out of memory areas—one for analog data and the other for digital data. Upon arrival, the local application delivers the data received into the data pool. Support for settings is handled via a message queue that communicates setting parameters to the local application, which then forwards memory-referencing setting messages to the PLC.

This `DirectNET` protocol is not well understood, as the company considers it somewhat private. But we have been able to support it well enough to access memory data from a PLC. It also includes a 16-bit CRC, which is probably not really needed.

### *PC data access*

Suppose a PLC-like model is used. A local application would collect data from the PC via some simple protocol over UDP. When the reply is delivered to the local application, the values would be distributed into the local data pool. The PC would maintain a kind of data pool, perhaps one analog and one digital, that it would update as it is convenient. It would also need to support request messages that prompt it to deliver the contents of that data pool. In this case, the analog and digital data pool data could be combined into a single message.

As for a simple protocol, consider the Classic protocol. Can some simple subset be used for this purpose? One-shot requests may be good enough. Multicast need not be supported. One could include access to memory data, where the memory addresses are interpreted as references to the analog and digital data areas that the PC maintains. Classic requests received by the PC that do not fit the supported subset can simply be ignored.

An advantage of using the Classic protocol is that it does not have to be invented, and it is well understood, having originally been designed for the same front ends. In addition, client support for this protocol is already built-in to the front-end software. Only support for a limited subset would have to be added to the PC side.

A local application can make a request to the PC for periodic replies, if desired. The PC would need to listen to port 6800, and it would need to have a native node#. The periodic replies can be monitored by the local application via `GetDat`, and it can then copy the data into the data pool. Should the PC “go away,” the system support would continue to repeat the original request every 2 seconds, hoping it will come up again. But the `REQM` local application will tear down the request if it cannot get a cancel response to a bogus reply.

Another way to approach it does not use quite so much of the built-in support for the Classic

protocol. The local application can make a Classic request from its own port, so that the replies will be delivered to the LA, without the system being more involved than that.

For setting support, we would need to again use the message queue scheme to bridge the gap from the system code, when it needs to make such settings, to the local application, so it can format a setting message for delivery to the PC.

Once some modicum of Classic support is in the PC, debugging services such as the Memory Dump page application may be useful to examine memory inside the PC. Or not.

Instead of using memory-based access to the PC, one might design a listype for the purpose. For example, the ident might target areas of memory without specifying a memory address. The areas could be small integers that have meaning to the PC support software.

To keep it as simple as possible, suppose the PC only supports one-shot requests. It receives a request message, builds a reply containing the requested data, and returns the reply. It should be able to parse multiple Classic protocol messages within a datagram, but if it cannot see its way to combine replies into a common datagram, that should be ok.

Example request message

```
0010 message size
node target node#
2006 request message, list#=6
0001 one-shot, one listype
xx00 listype#
nnnn #bytes requested
node target node#
indx index specifying which kind of data is to be returned
```

Example reply message for request that asks for 8 bytes to be returned

```
0010 message size
0000 node# not used
0006 reply message, list#=6
stat status word
data reply data (4 words)
data
data
data
```

As a model for a local application, consider the EPLC local application. It communicates with PLC hardware over ethernet. the protocol would have to change, of course, but the idea of specifying two areas of memory, one for analog and the other for digital data words, should apply.

Suggested list of parameters for the LA

<i>Prompt</i>		<i>Meaning</i>
ENABLE	B	LA enable bit#, as always
CYC UNIT		period in cycles in hi byte, unit# in lo byte
A REFADR		analog reference index
A WORDS		#words of analog data to collect
MAPCHAN	C	base analog channel#
D REFADR		digital reference index
D WORDS		#words of digital data to collect
MAPBIT	B	base digital Bit#
NODE		target node# of PC
STATUS	B	status of communications Bit#

Note that this list of parameters is very similar to that specified for EPLC. Instead of an IP address, here the node# is specified, plus a Bit# that can be monitored for detecting failure of communications. This is needed when formulating the Classic protocol request message.

#### LA operation

Periodically, as given by the hi byte of the "CYC UNIT" parameter, send a one-shot Classic request message that uses a new listype designed for this purpose. Use two requests, one for analog data and the other for digital data. The single listype will specify the appropriate number of bytes of data sought. Using `UDPWrite` will cause two datagrams to be sent using `UDPQueue` may permit concatenating the two messages into a single datagram.

When the reply data is returned, process it according to the list number used in the request. Install this data into the analog or digital data pools, as appropriate.

Each cycle, monitor a message queue called `PCDx`, where `x` is the value of the low byte of the "CYC UNIT" parameter word, in the range 0–9. The system code writes to such a message queue, using the parameters it has from the analog control field or from the `BADDR` entry. It will be necessary to allocate a new peculiar hi byte of an address, say `0x83`, to signify PC-based control. (Already, `0x80` means 1553, `0x81` means SRM, and `0x82` means PLC.)

The index values used for addressing PLC-based data are 16 bits. They are expected to cover two ranges only, one for analog and the other for digital.

To generalize things, we may include the concept of a data type, as done for the PLC support. Then the unit number will be in the hi nibble of the "CYC UNIT" word, and it will also be in the hi nibble of the second byte of the `BADDR` entry. This should permit heavy borrowing of the PLC support already in place within the system code. To actually make use of more than one data type specification, one needs another `LATBL` entry, just as one needs another to address a second PC.

When replies arrive, they can come at any time during the 15 Hz cycle. The LA should not send another request without giving some kind of timeout, even if a new request would be considered late by the hi byte of the "CYC UNIT" parameter. PC-based data is not expected to deliver correlated 15 Hz data, any more than PLC-based data is.

#### Classic setting message

```

000E
node
3002      setting message, 2-word ident
xx00      listype#
0002      2 bytes of setting data
node      target node#
indx      index

```

Or,

```

000C
node
3002
xx02
node
indx

```

This protocol leaves no room to specify a data type#. Either we abandon use of the type#, only allow settings using type#0, or we redesign the new listype to include another ident word. The easiest choice might be to forget the data type# option.

***What must the PC do to support all this?***

It needs to arrange to listen to UDP port 6800, to which Classic protocol messages will be sent. It parses the datagram it receives into messages and validates and processes each one. If it is a request message, it should format a reply message containing the data requested and return it to the node that sent the request. Because only one-shots will be used, there is not much to retain. If it receives a setting message, it should merely install the setting value. Routine monitoring of the data pool should show whether it was correctly received; i.e., the setting data should also be found in the data pool. This applies for both analog and digital data.

So, the PC must recognize one-shot requests and setting messages. It must return immediate replies that are sampled from its data pool memory in response to a request. This sounds simple.

***Why is this a good idea?***

It brings in the PC as a data source that is queried by the front end, rather than a client that routinely sends memory setting data to the front end. In this way, the front end can more easily analyze the health of the communications, since it the front end that may timeout the replies. The scheme borrows heavily on that already in place to support PLCs. The EPLC local application can be easily adapted to provide the needed protocol support.

It is a simple scheme that has a minimal model of communications, that between two memory areas in the PC, one used for analog and the other for digital data. The PC software, just as the ladder logic in a PLC, must manage these areas itself. The front end only samples what is there. Synchronization is not expected to be a problem, because PC-based data is not expected to be 15 Hz accelerator-synchronous.