5.12 Debug Messages

```
Mon Nov 03 16:38:35 1997 02:17:05.838 Citect Startup 5.00 Rev. 00  Service Pack C

Mon Nov 03 16:38:36 1997 02:17:06.278 (Id: 0000) (PORT_1 ) [s7_init: OK] Length 0
Mon Nov 03 16:38:36 1997 02:17:06.278 (Id: 0000) (PORT_1 ) [s7_set_window_handle_msg: OK] Length 0
Mon Nov 03 16:38:36 1997 02:17:06.298 (Id: 0000) (PORT_1 ) [s7_get_cref: OK] Length 0
Mon Nov 03 16:38:36 1997 02:17:06.298 (Id: 0000) (PORT_1 , Unit1) [s7_initiate_req: OK] Length 0
```

Port1, Unit1, initialise CP 5412 A2 card is successful.

```
Mon Nov 03 16:38:36 1997 02:17:06.298 (Id: 0000) (PORT_2 ) [s7_init: OK] Length 0
Mon Nov 03 16:38:36 1997 02:17:06.298 (Id: 0000) (PORT_2 ) [s7_set_window_handle_msg: OK] Length 0
Mon Nov 03 16:38:36 1997 02:17:06.298 (Id: 0000) (PORT_2 ) [s7_get_cref S7_ERR] Length 117
connection name in CRL not found:<a>make sure the CR-name is correct, and <a>the CP
descriptor references the correct CP.
Mon Nov 03 16:38:36 1997 02:17:06.308 Error: Channel offline, cannot talk
  UINIT    0015       PORT_2       UNIT_2                              16
  Generic 000021 Driver 00000020 (0x00000014)
```

Port2, Unit2, initialise CP 5412 A2 card is NOT successful. Reason: connection name has not been defined yet.

```
Mon Nov 03 16:38:36 1997 02:17:06.589 (Id: ffff) (PORT_1 , Unit1) [s7_get_initiate_cnf: OK] Length 0
Mon Nov 03 16:38:36 1997 02:17:06.609 (Id: 0002) (PORT_1 , Unit1) [s7_read_req: (M0,1) OK] Length 0
Mon Nov 03 16:38:36 1997 02:17:06.639 (Id: 0002) (PORT_1 , Unit1) [s7_get_read_cnf: OK] Length 0
```

Port1, Unit1, online test is successful. Normal operation starts.

```
Mon Nov 03 16:39:06 1997 02:17:36.241 (Id: 0003) (P
```

Consider a device which takes 100mS to read a single register and 150mS to read a block of 100 contiguous requests. If a registers 1 and 100 are required it is faster to read both registers in one go (ie read the block 1 - 100) taking 150mS than it would be to use two requests taking 100mS + 100mS.
The optimum size of a request for a given protocol needs to be determined by experiment (See Calculating the Blocking Constant in section 7) and then set as the blocking constant.

* Optimisation by the Citect Compiler

Citect assumes a simple model that has been found suitable with the majority of protocol for industrial devices. Citect specifies a devices available data into types, specified by the UnitType number and addresses specified by the UnitAddress number. This scheme is best suited for protocols which allow a number of registers contiguous in address to be read in one request. Citect blocks requests according to the UnitAddress. Variables of differing UnitType or differing RawType will not be optimised together. The compiler uses information from the specification file PROTDIR.DBF to determine what size of request to build. The value in the BIT_BLOCK field is the optimum size of a request in bits for a given protocol. The compiler assesses all the data required for a single client task (a page, a Cicode thread or another server such as alarms, reports or trends) and builds requests of the optimum size. Citect will never make a read request for a single bit of data. Digital read requests are normally blocked multiples of 16 bits starting on the 16 bit boundary. A driver may select 8 bit based blocking for digital read requests by setting the OPT_8_BIT_DIGITAL option in the PROTDIR.DBF file for the protocol.

* Optimisation by the Citect I/O Server

Citect is based on a true Client/Server architecture. The IO Servers must service requests from client tasks such as the Alarm Servers, Report Servers, pages displaying values, Cicode threads etc. The IO Server looks for opportunities to build further optimised requests from client requests. This is the same principal as the compile time optimisation but performed at runtime by the IO Server. The run-time system uses the Constants.Block value (in bytes) as the optimum size for a request. The user can modify this parameter from the INI file.

b.)   Problems might be caused by blocking.

As we can see from a.) that Citect will always try to read data in blocks, this works fine with most of PLCs since users of these PLCs cannot define sophisticated data types (structured data type) in their memory area. However, Siemens PLCs are the exception as users can configure their memory area into any structured data types.

Let assume for a moment that we have an S7-300 PLC. We have created a data block called DB1.
Here is what we have in the DB1.

Byte Address	DataType	Length in Bytes
0	int	2
2	real	4
6	byte	1
7	int	2
9	char	1
10	int	2

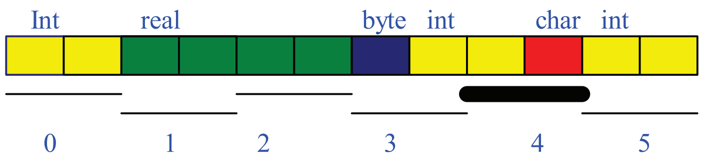
Assume again that Citect has the two integers displayed on a graphic page.
Here is the slow motion.

1. the graphic page is activated.
2. Citect wants three integers' values at addresses (DB1, 0) , (DB1, 7), and (DB1, 10)
3. I/O server works out the address and length and blocked them together.
   Request: datablock=1, start address=0, number=6, data type = integer
4. Sseven driver gets the request from the I/O Server and translated it into Siemens language

DB1, INT0, 5 and send it to the PLC.

~~5.~~ the PLC receives the request, and translated it into something like this. Citect wanted (5 - 0 + 1) * 2 = 12 bytes, starting from 0 in DB1. It responses with the data and specifies the length to be 6 (integers).

6. Sseven driver receives the data, and passes it on to the I/O Server and then Citect.
7. Citect retrieves the first integer from location 0, the second one from location 4 , and the third one from location 5.



8. It is obvious that the second integer at location 4 has an incorrect value.
9. If the data types are adjusted or padded properly, the user can get the correct reading, but the performance will suffer since there is a lot of unwanted data in the response packet.

Note this example is valid for other memory areas (M for instance) and It is valid for other data types (other than integer) as well.

c.) The correct way to configure a DB.

Let us re-organise the DB1 as following.

Address	DataType	Length in Bytes
0	int	2
2	int	2
4	int	2
4	real	4
8	byte	1
9	char	1

Now when Citect wants three integers, the I/O server will block it into DB1, address 0, number 3. And will displayed the values correctly, and the performance will not be reduced.

Let's extend this example a bit further.

Address	DataType	Length in Bytes
0	byte	1
1	byte	1
2	byte	1
3 (padding dumy)	byte	1
4	real	4
8	real	4
12	real	4
60	real	4
64	real	4
68	some	
xx	other	x
xx	data	x
xx	types	x
xxx	real	4
xxx	real	4

…

Minimum distance has to be greater than Constants.Block

Users can adjust/padding the data type onto the right boundary, this way, the Citect boundary check can be left on, if there is a bad boundary warning, you will know that something is not right.

If you want to add (append at the end) some real data type after all the work has been done, you need to make sure that the Minimum distance has to be greater than the value specified by **Block** parameter (Standard citect.ini file parameter). Otherwise you need to pad enough bytes to meet this requirement.

If you just starting the project. Make sure reserve some space for the future expansion.

Address	DataType	Length in Bytes
0	byte	1
1	byte	1
2	byte	1
3	reserved	1
4	reserved	1
5 (padding)	reserved	1
6	real	4
10	real	4
14	real	4
…	…	…
62	real	4
66	real	4
70	reserved	4
74	reserved	4
78	reserved	4
82	reserved	4
xx	some	x
xx	other	x
xx	data	x
xx	types	x
xx	reserved	x
xx	reserved	x
xx	reserved	x
xx	reserved	x