TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

{ DATUM |22002-04-09

DOKEGENSKAP "ArkivKod" * KOPPLAFORM |EE55

Kopia

Mottagare

PROVIEW/R REMOTE

User’s guide and object descriptions

Revision:	Date	Name	Comment
1.0	1996-11-28	Hans Werner	First version
1.1	1996-11-29	Hans Werner	Minor changes p 11, 15-17
2.0	1997-06-17	Claes Jurstrand	Converted to Word 7.0
2.1	1998-10-09	Claes Jurstrand	New protocols
2.2	2000-04-28	Claes Jurstrand	New protocol UDP/ip
2.3	2000-06-15	Claes Jurstrand	Minor changes p 12
2.4	2001-04-04	Claes Jurstrand	Minor changes UDP/ip and pwr 3.3 stuff
2.5	2001-09-04	Claes Jurstrand	New features in Remote DMQ
2.6	2001-12-19	Claes Jurstrand	Acknowledge feature in Remote UDP

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

1 Introduction ---- 4

1.1 RemTrans ---- 4

1.2 Remote I/O ---- 4

1.3 RemoteLogg ---- 4

1.4 Hierarchical structure ---- 5

1.5 Programs for Remote ---- 6

1.5.1 ELN systems ---- 6

1.5.2 VMS systems ---- 6

1.5.3 Lynx and Linux systems ---- 6

2 Transports ---- 8

2.1 Alcm (Transport type 1) ---- 8

2.2 PAMS (Transport type 2) ---- 9

2.3 3964-R VNET (Transport type 4) ---- 10

2.4 RK512 (Transport type 5) ---- 11

2.5 TCP/iP (Transport types 6 and 7) ---- 12

2.6 3964-R (Transport type 8) ---- 13

2.7 MODBUS (Transport type 9) ---- 14

2.8 ADLP10 (Transport type 10) ---- 14

2.9 Limab (Transport type 11) ---- 14

2.10 DMQ (Transport type 12) ---- 14

2.11 Serial (Transport type 13) ---- 15

2.12 UDP/iP (Transport type 15) ---- 15

3 Objects ---- 18

3.1 RemNode ---- 18

3.2 RemTrans ---- 23

3.3 LoggConfig ---- 29

3.4 MultiCast ---- 29

3.5 RemDi ---- 30

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

{ DATUM |22002-04-09

DOKEGENSKAP "ArkivKod" * KOPPLAFORM |EE55

3.6 RemDo ---- *30*
3.7 RemDv ---- *31*
3.8 RemAi ---- *31*
3.9 RemAo ---- *31*
3.10 RemCo ---- *31*
3.11 RemChanDi ---- *33*
3.12 RemChanDo ---- *34*
3.13 RemChanDv ---- *36*
3.14 RemChanAi ---- *37*
3.15 RemChanAo ---- *39*
3.16 RemChanCo ---- *41*
3.17 Buff256 ---- *42*
3.18 Buff1440 ---- *43*
3.19 Buff4096 ---- *43*
3.20 Buff32k ---- *43*

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

1 Introduction

The Remote concept in Proview is a way to standardize the methods of communication with other systems. It describes a number of transport programs and Proview objects used to implement a variety of different communication protocols and to handle incoming and outgoing messages.

Remote is designed to use different transport protocols such as TCP/IP or DMQ, and different hardware media such as ethernet or serial lines.

Some transports can only be used on ELN, others on both ELN and VMS. There are also some transports not (yet) converted to Alpha and LynxOS

The main purpose of Remote is to provide the programmer with an interface to communication

1.1 RemTrans

RemTrans handles messages to other systems. Different transport protocols handle the message adressing in different ways.

We can flag when we want to send a message, and we get a flag when a new message has arrived. RemTrans requires that we reset this flag when we have treated a new message so that the next message can be stored.

Messages can be sent or received by RemTransSend and RemTransRcv, or by DataArithm-objects in the PLC, or by C-programs.

RemTransSend- and RemTransRcv- objects are described in the NMps-manual.

1.2 Remote I/O

For some transports, we can also handle remote I/O from PSS7000.

We can get spontaneous IO-buffers, or request IO with a Poll message.

We can get DI, DO, ME, AI, AO, PI from PSS7000.

With DO, ME and AO we can choose if Proview is master of each signal, or if we only get the actual values. If Proview is master, and we detect a difference between the actual value and the mirrored value, we will send an update message to PSS7000.

1.3 RemoteLogg

RemTrans also includes logging of messages. This facility can also be used for other logg-functions that you want to realize.

For each loggfile you must specify a LoggConfig-object, where you state the logg number and the file name.

For each logg entry, you send a PAMS message to RemoteLogg with the logg file number and the ASCII string you want to logg.

The logg message should be sent to PAMS process no 90 with Class = 90, Type = 90.

In Proview version 3.3 QCOM replaces PAMS/DMQ as internal communication. RemoteLogg uses QCOM and an application should send a message to the QCOM-queue defined in the header file remote.h. The structure of the message is still the same.

The message structure should be:

```
short int     LoggNo;
char          ASCII_Buff[x];
```

1.4 Hierarchical structure

The Remote objects have to be configured in a certain hierarchical manner. Follow this example:

```
Node                        $NodeHier
. Nod1                      $Node
. . MultiCast   MultiCast
. . RemLogg     LoggConfig  Describes loggfile for Remote logging
. . Nod2        RemNode
. . . Mess1      RemTrans
. . . . Data      Buf256    Data-object or Userclass-object for message data
. . . Mess2      RemTrans
. . . . Data      Buf4096   Data-object or Userclasss-object
. . Nod3        RemNode     Transport with remote I/O
. . . IOBuff     Buff1440   I/O mirror-object not necessary, but convenient for
                            troubleshooting
. . . RemDi      RemDi      Hierarchical level for remote Di
. . . . DI001     RemChanDi Connects to Di-object in PlantHier
. . . . DI002     RemChanDi
. . . RemDo      RemDo      Hierarchical level for remote Do
. . . . DO043     RemChanDo Connects to Do-object in PlantHier
```

```
. . . . DO067     RemChanDo
. . . RemDv      RemDv      Hierarchical level for remote Dv
. . . . ME001     RemChanDvConnects to Dv-object in PlantHier
. . . RemAi      RemAi      Hierarchical level for remote Ai
. . . . AI001     RemChanAi Connects to Ai-object in PlantHier
. . . RemAo      RemAo      Hierarchical level for remote Ao
. . . . AO001     RemChanAo Connects to Ao-object in PlantHier
. . . RemCo      RemCo      Hierarchical level for remote Co
. . . . PI001     RemChanCo Connects to Co-object in PlantHier
. . . Mess1      RemTrans
. . . . Data      Buf4096   Data-object or Userclasss-object
. . . Mess2      RemTrans
. . . . Data      Buf256    Data-object or Userclass-object for message data
```

1.5 Programs for Remote

1.5.1 ELN systems

If you run under ELN, you should always include these lines in your EBUILD.DAT:

```
program SSABB_ROOT:<VAX_ELN.EXE>REMOTEHANDLER.EXE_ELN /warm_debug
/job_priority=15
program SSABB_ROOT:<VAX_ELN.EXE>RS_REMOTE_LOGG.EXE_ELN /warm_debug
```

There should also be some of the following lines or similar:

```
-- For transports using TCP/IP
internet /service /arc=256 /cto=30 /lt=180 /ikpt=180 /kpc=100 /telin
ip_device EZA0 /ndt=ethernet /ia=10.1.2.3 /am=255.0.0.0
-- Example of transport processes
program SSABB_ROOT:<VAX_ELN.EXE>REMOTE_TCPIP.EXE_ELN /norun/warm_debug
program SSABB_ROOT:<VAX_ELN.EXE>REMOTE_PAMS.EXE_ELN /norun/warm_debug
program SSABB_ROOT:<VAX_ELN.EXE>REMOTE_ALCM.EXE_ELN /norun/warm_debug
program SSABB_ROOT:<VAX_ELN.EXE>REMOTE_RK512.EXE_ELN /norun/warm_debug
program SSABB_ROOT:<VAX_ELN.EXE>REMOTE_3964R.EXE_ELN /norun/warm_debug
program SSABB_ROOT:<VAX_ELN.EXE>REMOTE_3964R_VNET.EXE_ELN /norun/warm_debug
program SSABB_ROOT:<VAX_ELN.EXE>REMOTE_MODBUS.EXE_ELN /norun/warm_debug
program SSABB_ROOT:<VAX_ELN.EXE>REMOTE_ADLP10.EXE_ELN /norun/warm_debug
-- Example of serial line configuration
device TTA /register=%O760440 /vector=%O150
terminal TTA0 /driver=DH /parity=even /noescape /noecho /passall /eightbit
terminal TTA1 /driver=DH /parity=even /noescape /noecho /passall /eightbit
```

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

1.5.2 VMS systems

If you run under VMS, you must specify $Appl-objects to start the applications SSABB_ROOT:[VAX_VMS.EXE]REMOTEHANDLER.EXE and SSABB_ROOT:[VAX_VMS.EXE]RS_REMOTE_LOGG.EXE

No $Appl-objects for the transport processes is necessary since the remotehandler will start them automatically. However, you must control that the transport application is copied to SSABB_ROOT:[VAX_VMS.EXE]

1.5.3 Lynx and Linux systems

If you run under LynxOS or Linux, the remotehandler should be started with one line in the startup file $pwrp_load/ld_appl_*sysname_busnumber*.txt:

```
remotehandler, remotehandler, noload, run, /pwr/exe/rs_remotehandler, 20,
nodebug, ""
```

No $Appl-objects for the transport processes is necessary since the remotehandler will start them automatically. However, you must control that the transport application is copied to $pwr_exe.

2 Transports

This section describes briefly the different protocols that are implemented as transport applications in the Remote environment.

2.1 Alcm (Transport type 1)

This is a DecNet-protocoll for communication with PSS7000. It works on both VMS and ELN.

It can handle remote I/O with or without poll, and of course RemTrans messages.

There is only one job for RemoteALCM and this job serves all of the remote nodes that uses the ALCM transport.

Remote ALCM also accept multicast-addresses. That means that we can listen to up to 8 multicast-addresses, besides your own DecNet-address.

The PSS7000 can send IO-messages without polling (or RemTrans messages), on a MultiCast-address to several systems.

Place one single MultiCast object under your $Node-object, on the same level as the RemNode object if you want to listen to multicast-addresses.

RemTrans-messages on ALCM must include a special header in the message data. This header is treated as an application header but is never the less very important for the protocol layer and must not be left out.

```
short int TransCode;       /* 2 Message with acknowledge
                              4 Message without acknowledgment */
short int Spare;
short int Length;          /* Number of words including this header */
short int Name[2];         /* Taskname or message name in RAD50 */
char      Data[x];         /* The rest of the message */
```

Warning! A value of ”0” in the Length field of the header can under certain curcomstances cause a system crash in the PSS7000-system.

For Common mirroring from PSS7000, there is a slightly different header:

```
short int     TransCode;   /* 1 - Common with acknowledgment
                              3 - Common without acknowledgment */
```

```
short int     Name[2];      /* Common name in RAD50 */
char          Data[x];      /* The rest of the message */
```

for the first part of Common. The later parts use the following header:

```
short int     TransCode;    /* 7 - Common with acknowledgment
                               8 - Common without acknowledgment */
short int     Name[2];      /* Common name in RAD50 */
short int     Offset;       /* Number of words from Common start */
char          Data[x];      /* The rest of the message */
```

The Common messages may be sent in several smaller parts. DataValid is set whenever some part of the Common has been updated.

Max length of messages is 1440 bytes. ALCM can only handle one ethernet package.

When we want to send a message (the DataValid is set), ALCM translates the Trans Name from the RemTrans object into the buffer, and send it to the node, specified in RemNode.Address[0 .. 1]. If the type is message with acknowledgement, and the message may be buffered, it is buffered in wait for acknowledgment, and resent if we get no ack.

No other messages may be sent before we have received the ack.

When ALCM receives a buffer it detects the sender, and search for corresponding RemNode.Address[0 .. 1]. It then checks the message type. It could be an acknowledge, IO-buffer or message buffer.

If it is a message buffer it translates the RAD50 name in the buffer and searches for coreresponding RemTrans.TransName. If found, and the buffer is small enough for the receiving data object, the message is stored into the data object, and the RemTrans.DataValid is set. The message may also be buffered, if the last message was not yet treated.

2.2 PAMS (Transport type 2)

Transport Remote PAMS can be used for RemTrans messages on both VMS and ELN.

The other node must be defined in PAMSINIT.TXT with the same PAMS group number that you specify in RemNode.Address[0]

There will be one Transport job for each process number that you specify in RemNode.Address[1]

Every transport can handle communication with several RemNodes.

Sending messages: (DataValid is set)

Remote PAMS will send a PAMS buffer to Group RemNode.Address[0],

and process no RemTrans.Address[2].

Data length will be RemTrans.DataLength number of bytes.

The message will be sent as Class = RemTrans.Address[0] and Type = RemTrans.Address[1].

If PAMS doesn't have contact with the other node the message will be buffered, if there are still free buffers for this message.

If Class = 101, we will, if possible, buffer the message in wait for an acknowledgment.

The acknowledgment consist of a PAMS buffer Class = 102, Type = same as message,

Data = first 4 bytes of message.

When Remote PAMS receives a buffer it will first search for the sending node (PAMS group) among the RemNodes. If its not an ack message, it will then search for the Class and Type at the RemTrans.Address[0] and [1]. If the data object for this message is big enough to contain the message, the message will be stored, and the DataValid flag will be set.

2.3 3964-R VNET (Transport type 4)

This is a serial protocoll for communication with PSS7000. It works only on ELN.

It can handle remote I/O with or without poll, and of course RemTrans messages.

There will be one Transport job for each RemNode.

You specify the serial terminal name, that we will use ourselves for the communication, in RemNode.NodeName.

RemTrans-messages on VNET must include a special header in the message data:

```
short int     Length;       /* Number of words in message  including this
                            header */
short int     Name[2];      /* Taskname or message name in RAD50 */
char          Data[x];      /* The rest of the message */
```

For Common mirroring from PSS7000, there is a slightly different header:

```
short int     Length;       /* Number of words in message  including this
                            header */
short int     Name[2];      /* Taskname or message name in RAD50 is xxxCOM */
short int     Offset;       /* Number of words from Common start */
char          Data[x];      /* The rest of the message */
```

The transport will send a message to the serial port, consisting of header + data.

There is no automatic fill in of data to the buffer.

If we don't have contact with the other node the message will be buffered, if there are still free buffers for this message.

When we receive a buffer, we search for RemTrans.Address[0] .. [1] that matches the Name in the message header. If the data object for this message is big enough to contain the message, the message will be stored, and the DataValid flag will be set.

The message may also be buffered, if the last message was not yet treated.

The Common messages may be sent from PSS7000 in several smaller parts. DataValid is set whenever some part of the Common has been updated.

2.4 RK512 (Transport type 5)

RK512 is a protocol for serial lines and is an extension of the Siemens 3964R-protocol. The major difference between 3964R and RK512 is the presence of a header and the reply telegram that should be sent after each received message. Transport RK512 can only be used on ELN.

You must define the serial device and the terminal when you build the system (EBUILD).

There will be one Transport job for each RemNode.

You specify the serial terminal name (ex. TTA0:), that we will use as communication line, in RemNode.NodeName.

RK512 header will not be included in the user data. Acknowledgments and reply messages will be handled automatically.

The transport will only handle 'AD'-type messages. High address byte is given in RemTrans.Address[0], and low address byte in RemTrans.Address[1].

If message is more than 128 bytes it will be divided into several telegrams. This will be handled by the transport.

E-messages (ask for specified data) is not handled by the RK512 transport.

Sending messages: (DataValid is set)

The transport will send a RK512 message to the serial port, consisting of header + data.

It will also receive a reply message.

If we don't have contact with the other node the message will be buffered, if there are still free buffers for this message.

When we receive a buffer, first we check the header to see that this is a AD message or a reply message. If it is a AD-message we search for RemTrans.Address[0,1] that matches the address bytes in the message. If the data object for this message is big enough to contain the message, the message will be stored, and the DataValid flag will be set.

2.5 TCP/iP (Transport types 6 and 7)

Note: From Proview 3.0 there is a remote transport for UPD/ip which uses connectionless datagram ip communication. This method works better for LAN-networks and is available on ELN, VMS and LynxOS.

Transport Remote TCP/iP can only be used for RemTrans messages on ELN.

You must include TCP/iP when you build the system (EBUILD).

There will be one Transport job for each RemNode.

You specify the port number, that we will use ourselves for the communication, in RemNode.Address[0].

If we use TCP/iP Client we must specify the iP-address of the other node in RemNode.NodeName. (Ex. 10.1.2.3) and the port number to contact in RemNode.Address[1].

TCP/iP messages will use a special header, that is not included in the user data:

```
char          RemId1;       /* STX (Hex 02) */
char          RemId2;       /* ETB (Hex 0F) */
short int     Length;       /* Number of bytes in message + 8 bytes for this
                            header */
short int     MessId1;      /* Message identity part 1 */
short int     MessId2;      /* Message identity part 2 */
char          data[x];      /* User Data */
```

All of the integers in the header will be sent as big endian values, that is most significant byte first, on the TCP/iP-link. The user data is the responsibility of the user to switch, if he wants integers to be sent with big endian. VAX uses little endian!

Sending messages: (DataValid is set)

The transport will send a message to the connected port, consisting of Remote header + data.

MessId in the header is taken from RemTrans.Address[0,1], byte-switchedto send as big endian.

If we don't have contact with the other node the message will be buffered, if there are still free buffers for this message.

If we have link up, and haven't sent anything for RemNode.IOCycleTime seconds, we will send a keep-alive message, that consists of only the TCP/iP header, 8 bytes without user data, with MessId1 = MessId2 = 0

When we receive a buffer, first we check the header to see that this is a correct RemTrans message.

Then we search for RemTrans.Address[0,1] that matches the byte-switched MessId. If the data object for this message is big enough to contain the message, the message will be stored, and the DataValid flag will be set.

If we haven't received any messages for RemNode.IOStallTime seconds, we will try to shut down and reestablish the connection.

There is a possibility to disable the usage of the remote header in both sending and receiving. In this case, the transport process avoids to complete the message with the remote header upon sending. When receiving a message, there is no way to tell which remtrans object that is the target and the transport process puts the message in the first available receiving remtrans object. To disable the remote header set bit 0 in RemNode.Address[3].

There is also a possibility to disable the sending of keepalive messages by setting bit 1 in RemNode.Address[3]. Note that the supervision of connection still is active which means that the connection is shutdown if no messages are received within the timeout period specified in RemNode.IOStallTime.

2.6 3964-R (Transport type 8)

3964R is a simple serial line communication protocol that is developed by Siemens. Transport 3964R can only be used for RemTrans messages on ELN.

You must define the serial device and the terminal when you build the system (EBUILD). The characteristics for the line (speed, number of bits, parity etc.) are also configured using EBUILD.

There will be one Transport job for each RemNode.

You specify the serial terminal name (ex. TTA0:), that we will use ourselves for the communication, in RemNode.NodeName.

Messages will be sent straight on without any header. ACK, NAK, DLE and BCC is handled according to the 3964R protocol.

Receiving messages:

There can be only one RemTrans object for receive-messages because of the lack of header in this protocol. Every received message will be put in the first found RemTrans-object below the RemNode-object. If the data object for this message is big enough to contain the message, the message will be stored, and the DataValid flag will be set.

Sending messages:

The transport will send a 3964R message to the serial without adding any header.

If we don't have contact with the other node the message will be buffered, if there are still free buffers for this message.

2.7 MODBUS (Transport type 9)

Transport Modbus on serial line can only be used for RemTrans messages on ELN.

You must define the serial device and the terminal when you build the system (EBUILD). The characteristics for the line (speed, number of bits, parity etc.) are also configured using EBUILD.

There will be one Transport job for each RemNode.

You specify the serial terminal name (ex. TTA0), that we will use for the communication, in RemNode.NodeName.

The format of MODBUS that is implemented is RTU. For identification of messages we use the fields known as *slave address* and *function code* in the MODBUS header. The RemTrans.Address[0] and [1] defines these fields in the Proview environment. Se MODBUS specifications documents for more information.

Receiving messages

When we receive a buffer, we search for RemTrans.Address[0] and [1] that matches the fields *slave adress* and *function code* in the message header. If the data object for this message is big enough to contain the message, the message will be stored, and the DataValid flag will be set.

Sending messages

Messages will be sent using the contents of RemTrans.Address[0] and [1] as the fields *slave address* and *function code* in the message header.

2.8 ADLP10 (Transport type 10)

ADLP10 is a special protocol that is used to communicate with a crane. It is a simple serial protocol and is only implemented for ELN.

This protocol is removed from remote in Proview version 3.0 and higher.

2.9 Limab (Transport type 11)

Limab is a special transport developed for special reasons to communicate with laser equipment from LIMAB. It is a simple serial protocol and is only implemented for ELN.

2.10 DMQ (Transport type 12)

DMQ (DEC Message Queue) is the new variant of PAMS. DMQ is the protocol we use in new installations of VMS on both VAX and ALPHA. DMQ-nodes can communicate with PAMS on a ELN-system.

As for Remote, RemoteDMQ is an extension of RemotePAMS using some new functions such as disk buffering etc. The basics are the same for these two transports.

2.11 Serial (Transport type 13)

RemoteSerial is an attempt to generalize the using of simple serial line communication protocols. Its useful when we have a one-way sending of messages from some equipment to the control system. You specify up to three termination characters in RemNode.Address[0] - [2]. These termination characters are used to detect the end of a received message.

Study the documentation of the protocol to find out how the messages are composed and which termination characters it uses.

2.12 UDP/ip (Transport type 15)

Transport UDP/ip can be used from Proview v2.7 on all supported platforms. UDP uses socket communication without connection (datagram). For supervising the link to the remote node we use keepalive messages and the IOStallFlag in the remnode object is set when the contact is lost.

For ELN systems you must include ip-services when you build the system (EBUILD). For VMS systems you must have the UCX software installed and configured.

There will be one Transport job for each RemNode.

You specify the port number, that we will use ourselves for the communication, in RemNode.Address[0].

You specify the iP-address of the other node in RemNode.NodeName. (Ex. 10.1.2.3) and the port number to contact in RemNode.Address[1].

UDP/ip messages will use a special header, that is not included in the user data:

```
char         RemId1;      /* STX (Hex 02) */
```

```
char         RemId2;      /* ETB (Hex 0F) in data message without acknowledge
                             ENQ (Hex 05) in data message with acknowledge
                             ACK (Hex 06) in acknowledge message */
short int    Length;      /* Number of bytes in message + 8 bytes for this
                             header */
short int    MessId1;     /* Message identity part 1 */
short int    MessId2;     /* Message identity part 2 */
char         data[x];     /* User Data */
```

All of the integers in the header will be sent as big endian values, that is most significant byte first, in the datagram. The user data is the responsibility of the user to switch, if he wants integers to be sent with big endian. VAX, Alpha and Intel all uses little endian!

Sending messages

The transport will send a message to the remote port, consisting of Remote header + data.

MessId in the header is taken from RemTrans.Address[0,1], byte-switched to send as big endian.

If MaxBuffers in remtrans-object > 0, the message is sent with type ”want acknowledge” and is stored in the retransmit queue for the remnode. The first entry in the retransmit queue is retransmitted with a time period equal to ErrTime in remnode-object. When a correspondning acknowledge message is received the message is deleted from the retransmit queue. This is done automatically by the transport process.

Receiving messages

When we receive a buffer, first we check the header to see that this is a correct RemTrans message.

Then we search for RemTrans.Address[0,1] that matches the byte-switched MessId. If the data object for this message is big enough to contain the message, the message will be stored, and the DataValid flag will be set.

The transport uses the value stored in RemNode.IOCycleTime (seconds) as cycle time for outgoing keepalive messages. A keep-alive message consists of only the TCP/iP header, 8 bytes without user data, with MessId1 = MessId2 = 0.

The transport uses the value stored in RemNode.IOStallTime (seconds) as timeout time for incoming keepalive messages. When the timeout expires, the RemNode.IOStallFlag is set and the connection is marked internally as lost. As soon as one keepalive message (or any other valid message) is arrived from the remote node the RemNode.IOStallFlag is cleared and the connection is marked as up.

The keepalive message traffic can be supervised in some meaning in RemNode.PollDiff. The value in this parameter is increased by one every time a keepalive message is sent and decreased by one every time a keepalive message is received. The increasing and decreasing is handled by the transport internally.

There is a possibility to disable the usage of the remote header in both sending and receiving. In this case, the transport process avoids to complete the message with the remote header upon sending. When receiving a message, there is no way to tell which remtrans object that is the target and the transport process puts the message in the first available receiving remtrans object. To disable the remote header set bit 0 in RemNode.Address[3].

There is also a possibility to disable the sending of keepalive messages by setting bit 1 in RemNode.Address[3]. Note that the supervision of connection still is active which means that the connection is shutdown if no messages are received within the timeout period specified in RemNode.IOStallTime.

3 Objects

This section describes the Proview-objects that should be used with Remote.

3.1 RemNode

Function

RemNode-objects are used to describe the communication transport protocol, and the adresses involved to communicate with external systems.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation only

NodeName

Type: *pwr_tString40*

Flags: *Param*

Different functionality, depending on transport.

Transport ALCM

System Name, ex **M4RSYS**

Transport TCP/ip and UDP/ip

Internet adress, ex. **1.2.3.4**

All serial line protocols

Serial portname, ex **TTA1:**

Other transports

Documentation only

Address[0 .. 3]

Type: *pwr_tUint16*

Flags: *Param*

Different functionality, depending on transport.

Transports PAMS and DMQ

Address[0] PAMS/DMQ node number for target.

Address[1] PAMS/DMQ processnumber for this node

Transport ALCM

Address[0] DECnet Area number for target

Address[1] DECnet Node number for target

Transport TCP/ip and UDP/ip

Address[0] Our own ip portnumber.

Address[1] Ip port number of the remote node.

Address[2] Not used

Address[3] Bitmask for configuration:

Bit 0: When set, disables remote header

Bit 1: When set, disables keepalive messages

3964R - VNET

Address[0] RAD50 for first 3 letters in system name.

TransportType

Type: *pwr_tUint32*

Flags: *Param*

Code that tells which transport this RemNode should use:

1 ALCM

2 PAMS

3 VNET (Not implemented)

4 3964R - VNET

5 RK512

6 TCP/iP Client

7 TCP/iP Server

8 3964R

9 Modbus

10 ADLP10

11 LIMAB

12 DMQ

13 Serial

14 Not used

15 UDP/ip

NumberOfRestarts

Type: *pwr_tUint32*

Flags: *Param*

Number of automatic restarts if transport dies.

Value of 0 means restart only at PLC-change.

If the same transport is used for different RemNode-objects (ALCM or PAMS) the highest value of NumberOfRestarts will be the valid number.

CycleTime

Type: *pwr_tFloat32*

Flags: *Param*

The cycle time (seconds) for searching RemTrans for send.

If the same transport is used for more than one Remnode (ALCM or PAMS) the fastest value of CycleTime will be the valid number.

ErrTime

Type: *pwr_tFloat32*

Flags: *Param*

Time in seconds for retransmit of buffered RemTrans message when contact was lost.

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

Also used for slow IO poll, when contact is lost.

IOCycleTime

Type: *pwr_tFloat32*

Flags: *Param*

ALCM, 3964R-VNET

Time in seconds for IO-poll

TCP/ip and UDP/ip:

Time in seconds for keep-alive messages, if no other messages are sent.

IOStallTime

Type: *pwr_tFloat32*

Flags: *Param*

ALCM, 3964R-VNET:

Time in seconds. If no IO response message are received within this time, action will be taken according to IOStallAction.

TCP/iP and UDP/ip:

Time in seconds for 'watch-dog'. If no messages arrive within this time, the link is declared down, and a new connection is established. (UDP doesn’t use connections, in this case the connection is marked as down internally).

IOStallAction

Type: *pwr_tUint32*

Flags: *Param*

Action when no IO messages are received

0. Mark IOStallFlag
1. Mark IOStallFlag and change to slow IO poll after ErrTime
2. Mark IOStallFlag and clear all remote Di, and all remote Dv and remote Do where PwrIsMaster is not set.

VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

3. Mark IOStallFlag, slow IO poll, and clear all remote input.

IODataArea

Type: *pwr_Objid*

Flags: *Param*

IO mirror object where IO messages are stored.

If missing, or if object too small, there will be no store of IO messages, but remote IO will work anyway.

NumberOfTrans

Type: *pwr_tUint32*

Flags: *Intern*

Number of RemTrans found when initiating this RemNode.

NumberOfDI

Type: *pwr_tUint32*

Flags: *Intern*

Number of RemChanDi found when initiating this RemNode

NumberOfDV

Type: *pwr_tUint32*

Flags: *Intern*

Number of RemChanDv found when initiating this RemNode

NumberOfDO

Type: *pwr_tUint32*

Flags: *Intern*

Number of RemChanDo found when initiating this RemNode

NumberOfAI

Type: *pwr_tUint32*

Flags: *Intern*

Number of RemChanAi found when initiating this RemNode

NumberOfAO

Type: *pwr_tUint32*

Flags: *Intern*

Number of RemChanAo found when initiating this RemNode

NumberOfCO

Type: *pwr_tUint32*

Flags: *Intern*

Number of RemChanCo found when initiating this RemNode

PollDiff

Type: *pwr_tUint32*

Flags: *Intern*

Incremented when we send IO poll or keepalive message

Decremented when we receive IO data or keepalive message

ErrTransCount

Type: *pwr_tUint32*

Flags: *Intern*

Number of RemTrans messages to this RemNode will illegal message address

Poll

Type: *pwr_tBoolean*

Flags: *Param*

If True, and remote IO for this node, we should send IO poll messages

IOStallFlag

Type: *pwr_tBoolean*

Flags: *Intern*

True if IOStallTime is gone since last IO data message

3.2 RemTrans

Function

The object describes a data message to or from another system. We can communicate with other Proview/R systems or with foreign systems in different ways, according to Transport type.

The RemTrans object must be hierarchically placed directly under the RemNode object. The message data is fetched from, or stored into an object that is the only child of the RemTrans object.

For certain messages, you can allow a limited number of buffered messages, both for sending and receiving.Buffered sending is used when we don't have connection to the remote node. Buffered receive is used if the final user of data is not fast enough to treat several incoming messages.

For certain transports you can use built in acknowledgement for messages.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation only

TransName

Type: *pwr_tString40*

Flags: *Param*

For most transports this is only documentation.

For ALCM sending messages you must store receiving task name (6 characters) here.

For ALCM receive you store message id (6 characters) here.

Address[0 .. 3]

Type: *pwr_tUInt16*

Flags: *Param*

Message address. Used different for different transports.

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

ALCM

Not used

PAMS

[0] - Class (0 - 255)

[1] - Type (0 - 255)

[2] - Processnumber for receiver (Only if we are sending)

DMQ

[0] - Class (0 - 255)

[1] - Type (0 - 255)

[2] - Processnumber for receiver (Only if we are sending)

[3] - Used in DMQ to control delivery mode for outgoing messages:

0 Use delivery modes WF_DQF and SAF

1 Use delivery modes NN_MEM and DISC

2 Use delivery modes WF_DQF and DISC

The WF_DQF delivery mode delivers the message to the remote recovery journal and blocks for verification or timeout.

The NN_MEM delivery mode delivers the message to the destination queue and doesn’t wait for any verification at all.

The SAF function puts the message in the local recovery journal if the link is down. When using SAF, DMQ handles the retransmission meaning that we (Proview) can forget the message and treat it as sent. This works on DMQ 3.2 but is not supported in the earlier version 3.0.

The DISC function tells DMQ to discard the message if it isn’t deliverable. Can be used with WF_DQF along with the remote buffering mechanism.

For a deeper description of delivery modes, consult the DMQ documentation.

3964R - VNET

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

[0] - First 3 letters of Trans name / Task name in RAD50

[1] - Next 3 letters of Trans name / Task name in RAD50

3964R

Not used.

RK512

[0] - Address High byte (0 - 255)

[1] - Address Low byte (0 - 255)

TCP/ip and UDP/ip

[0] - Message address part 1 (0 - 65535)

[1] - Message address part 2 (0 - 65535)

MODBUS

[0] - Slave address

[1] - Function code

Serial

[0] - Ascii number (decimal) for termination character #1

[1] - Ascii number (decimal) for termination character #2

[2] - Ascii number (decimal) for termination character #3

[3] - Used to set the use of 7 or 8 bit character length. Set a value of 1 to use 7 bits and a value of 0 to use 8 bits.

Direction

Type: *pwr_tUInt32*

Flags: *Param*

Direction for message

1 Receiving message

2 Sending message

DataLength

Type: *pwr_tUInt32*

Flags: *Param*

Number of Bytes in message data. For outgoing messages this parameter must be set, either statically in the development system or dynamically via an application, before sending the message. For incoming messages this parameter is set by the transport when each message is received and shows the received length of this message.

LoggLevel

Type: *pwr_tUInt32*

Flags: *Param*

Logg of messages can take place on different levels, when it is implemented in the system.

0. No logging
1. Logg of time, RemTrans objectname, and status when error has occured
2. Logg of time, RemTrans objectname, and status for every message
3. As above plus first 48 bytes of message data as ASCII hex dump
4. As 2 plus whole message data as ASCII hex dump

MaxBuffers

Type: *pwr_tUInt32*

Flags: *Param*

Max number of buffers for this message.Value of 0 means no buffering allowed. In Remote UDP, when the message is outgoing, a value > 0 indicates that the message should be sent with wanted acknowledge.

Buffers

Type: *pwr_tUInt32*

VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

Flags: *Intern*

Number of occupied buffers. For incoming messages this indicates the number of messages waiting to be taken by the PLC-process. For outgoing messages it tells the number of buffered messages waiting to be acked.

MaxLength

Type: *pwr_tUInt32*

Flags: *Intern*

Size of data receiving object, or data sending object.

RemTrans can't handle messages bigger than this. If there is a bigger message, this message is just discarded.

The parameter is updated by the transport and should be left blank in configuring.

LastsSts

Type: *pwr_tUInt32*

Flags: *Intern*

Init: *1*

Status of last treated message.

1 - OK

3 - buffered message OK

0 - Error when sending

2 - Too big data message

4 - Received message was lost. Couldn't buffer it.

6 - Waiting for acknowledgement on message with auto ack.

TransTime

Type: *pwr_tTime*

Flags: *Intern*

Time stamp when last received or sent/buffered a message

TransCount

Type: *pwr_tUInt32*

Flags: *Intern*

Number of messages OK since last restart

BuffCount

Type: *pwr_tUInt32*

Flags: *Intern*

Number of messages sent/received through buffering since last restart

LostCount

Type: *pwr_tUInt32*

Flags: *Intern*

Number of received messages that were lost because they couldn't be buffered.

ErrCount

Type: *pwr_tUInt32*

Flags: *Intern*

Number of lost send messages.

DataValid

Type: *pwr_tUInt32*

Flags: *Intern*

Flag for new data in data-object.

When we receive a new message, it is stored into the data-object. Then the DataValid is set. The next message can not be stored before the final user has treated this message, and then reset the DataValid signal. When using RemTransRcv in the PLC-program, this is handled automatically.

When the original sender want to send a new message, he fills in the data, and then sets DataValid. The Remote transport will

search for DataValid set, and send the message and then Reset the DataValid. Next message shouldn't be filled into the data object before the DataValid is reset. When using RemTransSend in the PLC-program, this is handled automatically.

3.3 LoggConfig

Function

The object describes the files for logging through Remote_Logg.

Attributes

LoggFile

Type: *pwr_tString80*

Flags: *Param*

File Name for logg.For example "3.9::PWRP_LOG:FIL.DAT"

NewVersion

Type: *pwr_tBoolean*

Flags: *Intern*

When this attribute is set, the loggfile will be closed, and a new version of the file will be opened.

Identity

Type: *pwr_tUInt32*

Flags: *Param*

The file number for logg. (Sent in PAMS-buffer to Remote_Logg along with buffer)

LoggCount

Type: *pwr_tUInt32*

Flags: *Intern*

Number of loggs in this file since the file was opened.

FileOpenCount

Type: *pwr_tUInt32*

Flags: *Intern*

Number of times this file was opened with new version.

3.4 MultiCast

Function

The object defines which DecNet addresses that are to be listened for by REMOTE_ALCM besides the ordinary address.

Attributes

Address[16]

Type: *pwr_tUInt8*

Flags: *Param*

[0] - Area number for Multicastaddress 1

[1] - Node number for multicastaddress 1

…

[14] - Area number for Multicastaddress 8

[15] - Node number for multicastaddress 8

3.5 RemDi

Function

Hierarchical level for all RemChanDi-Objects. Must be placed as direct child to the RemNode-object.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

3.6 RemDo

Function

Hierarchical level for all RemChanDo-Objects. Must be placed as direct child to the RemNode-object.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

3.7 RemDv

Function

Hierarchical level for all RemChanDv-Objects. Must be placed as direct child to the RemNode-object.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

3.8 RemAi

Function

Hierarchical level for all RemChanAi-Objects. Must be placed as direct child to the RemNode-object.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

Documentation

3.9 RemAo

Function

Hierarchical level for all RemChanAo-Objects. Must be placed as direct child to the RemNode-object.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

3.10 RemCo

Function

Hierarchical level for all RemChanCo-Objects. Must be placed as direct child to the RemNode-object.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

3.11 RemChanDi

Function

Defines the address for a binary input in PSS7000. Connect a Di-object to this object instead of a ChanDi, and you can use the Di in PLC, as if it was an ordinary internal binary input.

Attributes

Description

Type: *pwr_tString80*

Flags: Param

Documentation

SigChanCon

Type: *pwr_tObjid*

Flags: *CompileFixParam*

Name of Di-object that is connected.

Identity

Type: *pwr_tString40*

Flags: *Param*

Documentation

***ActualValue**

Type: *pwr_tBoolean*

Flags:

Pointer to actual value in DiValueBase

ConvOff

Type: *pwr_tUInt32*

Flags: *Param*

Channel number in PSS7000.

For PSS7000 DI000 should have value 0, DI067 shouldhave value 67, and so on.

For DS8 IO, DI001 should have value 0, DI067 should have value 66.

When Transport starts up, this Value is used to calculate BuffOff and ConvMask.

If you want to modify the value 'flying', all three values must be modified.

BuffOff

Type: *pwr_tUInt32*

Flags: *Intern*

Byte offset in IO message. Calculated from ConvOff at initiating the transport.

ConvMask

Type: *pwr_tUInt8*

Flags: *Intern*

Bit mask in IO message. Calculated from ConvOff at initiating the transport.

3.12 RemChanDo

Function

Defines the address for a binary output in PSS7000. Connect a Do-object to this object instead of a ChanDo, and you can use the GetDo in PLC, as if it was an ordinary internal binary output. If you specify PwrIsMaster as true, you can also use StoDo, SetDo and ResDo.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

SigChanCon

Type: *pwr_tObjid*

Flags: *CompileFixParam*

Name of Do-object that is connected.

Identity

Type: *pwr_tString40*

Flags: *Param*

Documentation

***ActualValue**

Type: *pwr_tBoolean*

Flags:

Pointer to actual value in DoValueBase

ConvOff

Type: *pwr_tUInt32*

Flags: *Param*

Channel number in PSS7000.

For PSS7000 DO000 should have value 0, DO067 shouldhave value 67, and so on.

For DS8 IO, DO001 should have value 0, DO067 should have value 66.

When Transport starts up, this Value is used to calculate BuffOff and ConvMask.

If you want to modify the value 'flying', all three values must be modified.

BuffOff

Type: *pwr_tUInt32*

Flags: *Intern*

Byte offset in IO message. Calculated from ConvOff at initiating the transport.

ConvMask

Type: *pwr_tUInt8*

Flags: *Intern*

Bit mask in IO message. Calculated from ConvOff at initiating the transport.

PwrIsMaster

Type: *pwr_tBoolean*

Flags: *Param*

If this flag is set, we can modify the DO in the subsystem. Otherwise we can just look at the value of the Do.

OldValue

Type: *pwr_tBoolean*

Flags: *Intern*

Last value from the subsystem. Used for detection of new value from Proview.

3.13 RemChanDv

Function

Defines the address for a binary flag MExxx in PSS7000. Connect a Dv-object to this object, and you can use the GetDo in PLC, as if it was an ordinary internal Dv. If you specify PwrIsMaster as true, you can also use StoDv, SetDv and ResDv.

The connection can't be done as an ordinary connection in the Configuration editor, but must be done under the Atribute editor.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

SigChanCon

Type: *pwr_tObjid*

Flags: *CompileFixParam*

Name of Dv-object that is connected.

***ActualValue**

Type: *pwr_tBoolean*

Flags:

Pointer to actual value in DvValueBase

ConvOff

Type: *pwr_tUInt32*

Flags: *Param*

Channel number in PSS7000.

For PSS7000 ME000 should have value 0, ME067 shouldhave value 67, and so on.

For DS8 IO, ME001 should have value 0, ME067 should have value 66.

When Transport starts up, this Value is used to calculate BuffOff and ConvMask.

If you want to modify the value 'flying', all three values must be modified.

BuffOff

Type: *pwr_tUInt32*

Flags: *Intern*

Byte offset in IO message. Calculated from ConvOff at initiating the transport.

ConvMask

Type: *pwr_tUInt8*

Flags: *Intern*

Bit mask in IO message. Calculated from ConvOff at initiating the transport.

PwrIsMaster

Type: *pwr_tBoolean*

Flags: *Param*

If this flag is set, we can modify the ME in the subsystem. Otherwise we can just look at the value of the ME.

OldValue

Type: *pwr_tBoolean*

Flags: *Intern*

Last value from the subsystem. Used for detection of new value from Proview.

3.14 RemChanAi

Function

Defines the address for an analog input in PSS7000. Connect an Ai-object to this object instead of a ChanAi, and you can use the Ai in PLC, as if it was an ordinary internal analog input.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

SigChanCon

Type: *pwr_tObjid*

Flags: *CompileFixParam*

Name of Ai-object that is connected.

Identity

Type: *pwr_tString40*

Flags: *Param*

Documentation

***ActualValue**

Type: *pwr_tFloat32*

Flags:

Pointer to actual value in AiValueBase

SensorPolyType

Type: *pwr_tUInt32*

Flags: *Param*

Type of conversion from raw value from IO message, into engineering actual value:

0. ActVal = raw
1. ActVal = coef0 + raw * coef1
2. ActVal = coef0 + raw *(coef1 + raw * coef2)
3. ActVal = coef2 * sqrt(coef0 + raw * coef1), 0 if sqrt of negativ value
4. ActVal = coef2 * sqrt(coef0 + raw * coef1), if positiv value for sqrt. ActVal = - coef2 * sqrt(-(coef0 + raw * coef1)) else.

SensorPolyCoef0

Type: *pwr_tFloat32*

Flags: *Param*

Coefficient coef0 in desacription of SensorPolyType

SensorPolyCoef1

Type: *pwr_tFloat32*

Flags: *Param*

Coefficient coef1 in desacription of SensorPolyType

SensorPolyCoef2

Type: *pwr_tFloat32*

Flags: *Param*

Coefficient coef2 in desacription of SensorPolyType

ConvOff

Type: *pwr_tUInt32*

Flags: *Param*

Channel number in PSS7000.

AI000 should have value 0, AI012 should have value 12, and so on.

3.15 RemChanAo

Function

Defines the address for an analog output in PSS7000. Connect a Ao-object to this object instead of a ChanAo, and you can use the GetAo in PLC, as if it was an ordinary internal analog output. If you specify PwrIsMaster as true, you can also use StoAo and CStoAo.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

SigChanCon

Type: *pwr_tObjid*

Flags: *CompileFixParam*

Name of Ao-object that is connected.

Identity

Type: *pwr_tString40*

Flags: *Param*

Documentation

***ActualValue**

Type: *pwr_tFloat32*

Flags:

Pointer to actual value in AoValueBase

OutPolyCoef0

Type: *pwr_tFloat32*

Flags: *Param*

Output Rawvalue = OutPolyCoef0 + ActualValue * OutPolyCoef1

OutPolyCoef1

Type: *pwr_tFloat32*

Flags: *Param*

Output Rawvalue = OutPolyCoef0 + ActualValue * OutPolyCoef1

OutMaxLimit

Type: *pwr_tFloat32*

Flags: *Param*

Maximum ActualValue to use in the conversion to raw value

OutMinLimit

Type: *pwr_tFloat32*

Flags: *Param*

Minimum ActualValue to use in the conversion to raw value

ConvOff

Type: *pwr_tUInt32*

Flags: *Param*

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

Channel number in PSS7000.

AO000 should have the value 0, AO004 should have the value 4, and so on.

OldValue

Type: *pwr_tInt16*

Flags: *Intern*

Last raw value from the subsystem. Used for detection of new value from Proview.

PwrIsMaster

Type: *pwr_tBoolean*

Flags: *Param*

If this flag is set, we can modify the AO in the subsystem. Otherwise we can just look at the value of the Ao.

3.16 RemChanCo

Function

Defines the address for a pulse input in PSS7000. Connect an Co-object to this object instead of a ChanCo, and you can use the Co in PLC, as if it was an ordinary internal pulse input.

Attributes

Description

Type: *pwr_tString80*

Flags: *Param*

Documentation

SigChanCon

Type: *pwr_tObjid*

Flags: *CompileFixParam*

Name of Co-object that is connected.

Identity

Type: *pwr_tString40*

Flags: *Param*

Documentation

***ActualValue**

Type: *pwr_tUInt32*

Flags:

Pointer to actual value in CoValueBase

***ExtendedValue**

Type: *pwr_tUInt32*

Flags:

Pointer to actual value in CaValueBas

NoOfBits

Type: *pwr_tUInt32*

Flags: *Param*

Number of bits (16 / 24) for counter card

ConvOff

Type: *pwr_tUInt32*

Flags: *Param*

Channel number in PSS7000.

PI000 should have value 0, PI012 should have value 12, and so on.

SyncRawValue

Type: *pwr_tUInt32*

Flags: *Param*

Value for initiating the card raw value. Function is not implemented.

CounterZeroFlag

Type: *pwr_tBoolean*

Flags: *Param*

Value for initiating the card. Function is not implemented.

CounterSyncFlag

Type: *pwr_tBoolean*

Flags: *Param*

Value for initiating the card. Function is not implemented.

3.17 Buff256

Function

The object can be used for any unspecified data storage, or as storage for a RemTrans message.

Attributes

Data[128]

Type: *pwr_tInt16*

Flags: *Param*

Data buffer 256 bytes.

3.18 Buff1440

Function

The object can be used for any unspecified data storage, or as storage for a RemTrans message.

Attributes

Data[720]

Type: *pwr_tInt16*

Flags: *Param*

Data buffer 1440 bytes.

3.19 Buff4096

Function

The object can be used for any unspecified data storage, or as storage for a RemTrans message.

Attributes

Data[2048]

Type: *pwr_tInt16*

Flags: *Param*

Data buffer 4096 bytes.

3.20 Buff32k

Function

The object can be used for any unspecified data storage, or as storage for a RemTrans message.

Attributes

Data[16384]

Type: *pwr_tInt16*

Flags: *Param*

TEKNIK
VTP/ { DOKEGENSKAP "Utfärdare" * KOPPLAFORM |CClaes Jurstrand
pnr 7023, tel { DOKEGENSKAP "tel" * KOPPLAFORM |555496, fax 54383

{ DATUM |22002-04-09

DOKEGENSKAP "ArkivKod" * KOPPLAFORM |EE55

Data buffer 32768 bytes.