APROL R 4.0 - Extended Engineering
F1 Drivers for B&R Connections

Perfection in Automation
www.br-automation.com

We reserve the right to change the contents of this manual without warning. The information contained herein is believed to be accurate as of the date of publication; however, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H. makes no warranty, expressed or implied, with regards to the products or the documentation contained within this book. In addition, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H. shall not be liable in the event of incidental or consequential damages in connection with or resulting from the furnishing, performance, or use of these products. The software names, hardware names, and trademarks used in this document are registered by the respective companies.

The following documentation ***F1 Drivers for B&R Connections***
(Version 4.06 / 25.08.2014) refers to APROL R 4.0

The ***Connectivity*** manual group contains the following manuals

F1 Drivers for B&R Connections

with the order number:

MAAPCONN40-ENG

Content
F1 Drivers for B&R Connections V4.06

APROL Documentation
F1 Drivers for B&R Connections V4.06

belongs to the manual set
Connectivity

Target group, conventions, and format

The target group for the manual **F1 Drivers for B&R Connections** is users that deal with the topic "interfaces" between the control computer and controller, and who are responsible for connecting other systems using the APROL standard driver.

In this documentation the following formatting is used:

Key	**[Esc]**-Key
Menu item	**"Module-Groups/open"**
Directory name	`HOME/ENGIN/HTML/049`

In this manual the following icons are used to highlight special information:


Listing


Listing


Listing


Tipp or suggestion


Note


Warning


Reference


This description is under construction at present.

Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.

This point is to be carried out by the user

List of necessary configuration steps	
Next step	
Configuration finished	

1 AnslDriver

1.1 Basic method of operation of the AnslDriver

.

1.1.1 General information about client redundancy

In the case of client redundancy there are normally two AnslDrivers started on two different control computers and they establish a connection to the same controller. One AnslDriver is the master (has process control) and the other is the slave.

The '-connectTimeout' option is used to switch between both AnslDrivers. The AnslDriver with process control (master) gives up its process control after it has expired. The slave registers this and turns into the driver with the process control. The '-connectTimeout' begins when the AnslDriver with process control can no longer reach the controller.

The '-slaveConnect' option is used when both AnslDrivers are started on one control computer. This can be done for testing purposes. The second AnslDriver is thus informed that it should start in slave mode.

1.1.2 Notes for operation of the driver

The AnslDriver monitors its configuration file for changes. If it recognizes a change in a configuration file it re-reads the corresponding file. The driver signs off objects which are no longer needed, and registers new ones. Existing objects without changes continue to be used without a new registration. The result is a reduction in the load of the entire controller communication during the download process.

It is possible to turn off this behavior with this if a complete new configuration is desired, instead of a part configuration. Upon detecting the change in a configuration file, the driver is completely re-started after a delay of 2 seconds, i.e. all objects are first signed off the controller and then registered again after the automatic restart.

Notes for the '-noOnlineReconfiguration' option:

The '-noOnlineReconfiguration' option deactivates the automatic reading of the configuration files if they have been changed in the file system.

The AnslDriver must be restarted if the option is set, so that its configuration files take effect after having been changed.

1.1.3 What are 'READ', 'WRITE' and 'SYNC' variables?

READ variables

READ variables are only transferred from the controller to the control computer. READ variabels are 'provided' by the driver.

Operating mode 'Event mode':

READ variables are registered on the controller in 'Event mode' and are monitored for changes there. A change in value is transferred from the controller to the control computer.

Special feature: Each event variables which is registered influences the response behavior of the controller. There are 6000 event checks per second per default, whereby each registration triggers one check (and therefore several checks if the same variable is registered many times). If there are less than 6000 variables registered, the checking of the individual variables increases respectively.

Disadvantage of this type of operation:
The checking of event variables is at the cost of CPU idle time, because the controller must do the work of checking.

Advantage of this type of operation:
Possibly quicker reaction times compared with the poll mode.

Operating mode 'Poll mode':

READ variables are queried cyclically by the driver in 'Poll mode' (the '-eventMode' option is not set). The driver must send a request telegram for each PV to the controller and it creates a response with the current value.

Advantage of this type of operation: Load reduction on the controller, because the cyclic value check is dropped.

Disadvantage of this type of operation:

- Slow response behavior
- The value detection of the individual variables is not deterministic, i.e. the current value of individual variables may have slightly different times.

Basically, **for both types of operation**, the time stamp of the time detection is currently created in the driver and the time stamp of the controller is not relevant. The event driver must still be used for high resolution events with the time stamp of the controller.

WRITE variables:

WRITE variables are only transferred from the control computer to the controller. They are either 'not supplied' or supplied by a control computer task.

WRITE variables are only transferred to the controller when necessary. The value in the process control system is compared with the value on the controller upon each connection establishment. If there is a deviation, the control computer value is sent to the controller.

The driver then only writes the variable to the controller if a change has taken place on the control computer, either due to a calculation in the control computer task ot the setting of a value in a faceplate.

SYNC variables:

SYNC variables are a combination of the above mentioned variable types. They are transferred, according to the type of operation, cyclically or via event monitoring from the controller to the control computer and kept there. If the value of such a variable is changed by an application on the control computer, the driver writes the variable to the controller.

SYNC variables are normally variables for CFC debugging.

The variables configured in the driver are only registered as objects on the controller when they are actually needed by the system. Variables which are marked as being

'Idle' in the system (see relevant documentation) are not registered on the controller and therefore do not need any resources. A registration takes place after the IDLE flag has gone and remains until the IDLE flag reappears. Remanent variables are handled as if they were never IDLE, i.e. they are always registered.

1.1.4 What is the benefit of the TCP/IP protocol which is used in comparison to the INA UDP/IP protocol?

The INA protocol uses the UDP/IP protocol, which is not connection orientated and does not monitor the connection via the operating system. This protocol does not ensure that telegrams are received by the partner. The protocol must create its own solution for receipts. A loss in connection is recognized through a missing connection monitoring telegram. A maximum of 240 bytes of reference data can be transferred per INA telegram.

ANSL communicates via the TCP/IP protocol. This protocol ensures that messages are 'received' with several receipts, whereby the telegram traffic increases on the network. An ANSL telegram segment can contain 1400 bytes of reference data, whereby several segments can constitute one telegram (they are only transferred separately). The TCP/IP connection monitoring does not suffice for APROL and this is the reason why a corresponding mechanism was created for a quick cyclic connection monitoring.

A much larger flow of data as INA is possible, because of the much larger telegrams.

1.1.5 How is the connection monitoring implemented, so that the large TCP/IP timeouts do not have a negative effect?

The telegrams are swapped cyclically (0.5 * ANSL_TIMEOUT) between both partners. If a telegram is not received within the ANSL_TIMEOUT then the connection is disconnected and must be re-established. This mechanism is necessary, because the connection monitoring of the operating system only offers reaction times in the scope of minutes. A connection monitoring telegram is not sent if another telegram with reference data has already been sent within this time period.

ANSL_TIMEOUT can be set for all customer connections. A sensible value is one that is not under two second, due to the nature of the system.

1.1.6 How are event variables updated on the controller and what effect does the client redundancy have on the event variables?

Also see the description of the READ variables in event mode.

In case a second driver is started on a redundant system and it works with the '-eventMode' and '-slaveConnect' options, its READ variables are already registered on the controller in an inactive state. In the worst case, the number of event variables is doubled, whereby the sampling rate (default 60000 PVs per second) is cut in half for each variable!

1.1.7 Which benefits does the poll mode have in comparison to the event mode?

The poll mode queries the value of a PV cyclically. The value is transferred even when there is no change in value.

Also see the description of the READ variables in poll mode.

1.1.8 Which system variables are basically made available and where do they get their information?

The system variables have the following syntax: 'S2A_<CTRL instance>_M_<name>'

Information from the driver (local):

Description:	Name:	Data type:
Number of connection attempts	_CntReCon	Integer
Configured READ PVs	_CntRead	Integer
Configured WRITE PVs	_CntWrite	Integer
Configured SYNC PVs	_CntSync	Integer
Configured EVENT PVs	_cntEvent	Integer
Currently active PVs	_CntPvAct	Integer
Number of cyclic updates	_CntReadCycl	Integer
Average value 'Events per second' from controller	_CntEvt_s	Integer
Average value 'Events per minute' from controller	_CntEvt_m	Integer
Number of write events sent to the controller	_CntWrtEvt	Integer
Active connection parameters	_connectString	String
Current connection status	_connectState	String
Last error	_DrvErr	Integer
Last error (text)	_DrvErrTxt	String

Information from the controller):

Description:	Name:	Data type:
Current load of the controller	_CtrlLoad	Integer
Current controller run mode	_CtrlMode	String
Controller node number	_CtrlNode	Integer
Configured controller host name	_CtrlHost	String
Configured controller IP address	_CtrlIP	String

Description:	Name:	Data type:
Type of controller	_CtrlType	String
'Short name' of the controllers	_CtrlShortName	String
Current battery status	_CtrlBatteryStatus	Integer
Current backplane battery status	_CtrlBatteryStatusBP	Integer

1.2 AnslDriver in CC

This description is under construction at present.

Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.

1.3 ApDrvAnsl diagnosis (ANSL cross-communication)

You can carry out an application-related diagnosis of the ApDrvAnsl configuration without involving an application. You can see how an application-related diagnosis (the use of diagnostic PVs in the driver configuration) can be carried out in the chapter for configuring a connection.

The ControllerManager Watch can be used for a manual diagnosis of the ApDrvAnsl. It is better to use two Watch windows and save them as a Watch group after they have been configured. The Watch group can then be loaded with the ApDrvAnsl context menu at any time, and shows all of the Watch windows in the last composition.

The names of the driver variables which can be used for diagnosis purposes all begin with 'ApDrvAnsl_Diag'.

The ApDrvAnsl_DiagHelp variable should be placed in the first Watch window and all elements set to a string output. A long cycle time should be set, e.g. 60 seconds, so that the system is not disturbed unnecessarily. The displayed variables are constant help texts, which are useful for the analysis in the second Watch window.

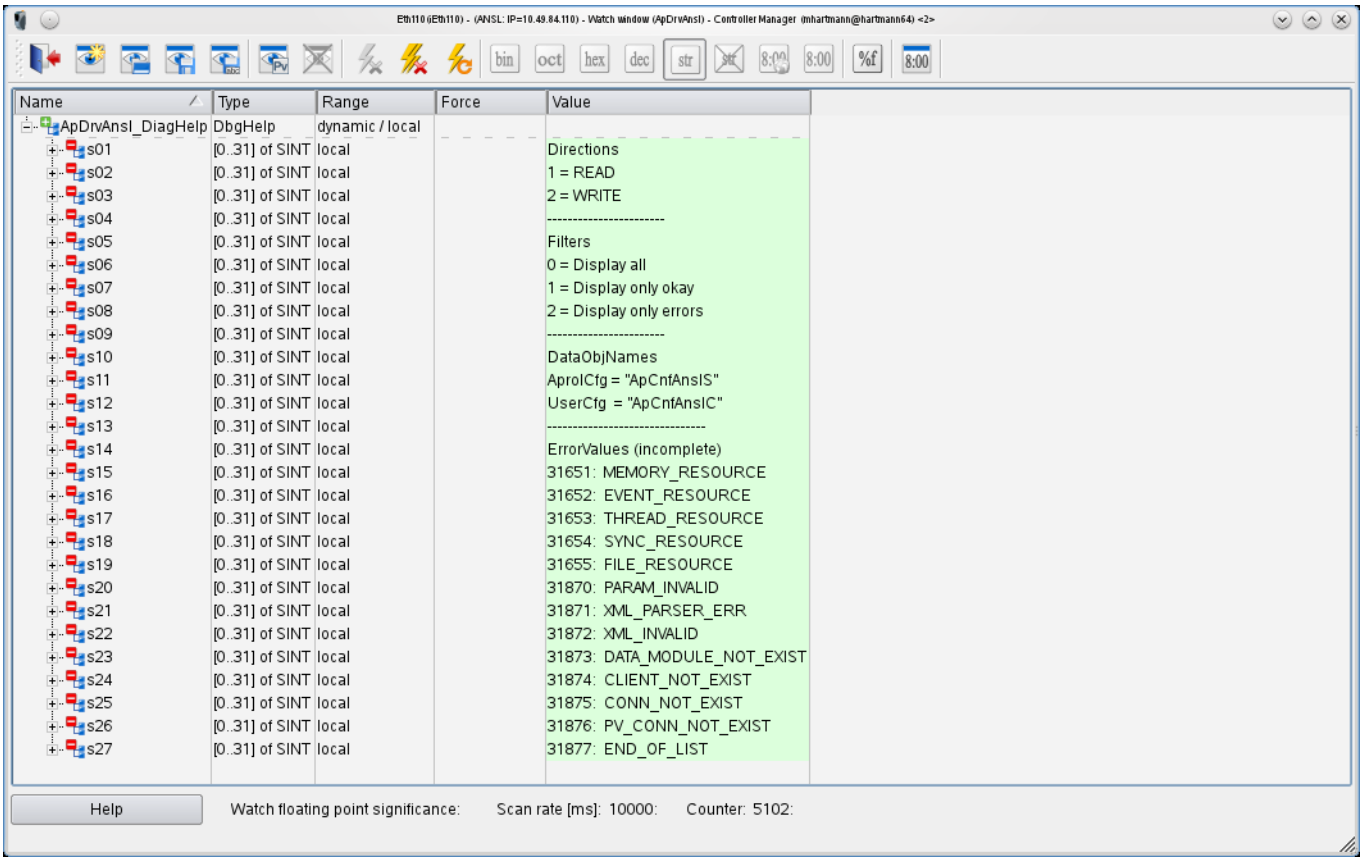


Illustration 1: Definition of the memory area for global remanent variables in the sysconf

The variables which are necessary for the diagnosis can be placed in the second Watch window. 6 independent variables are available, depending on the demands of the diagnosis.

Each of these three FUBs are programmed to access an element directly, or repeatedly through a filter which can be set.

ApDrvAnsl_DiagClient, ApDrvAnsl_DiagClientAll
ApDrvAnsl_DiagConn, ApDrvAnsl_DiagConnAll
ApDrvAnsl_DiagPv, ApDrvAnsl_DiagPvAll

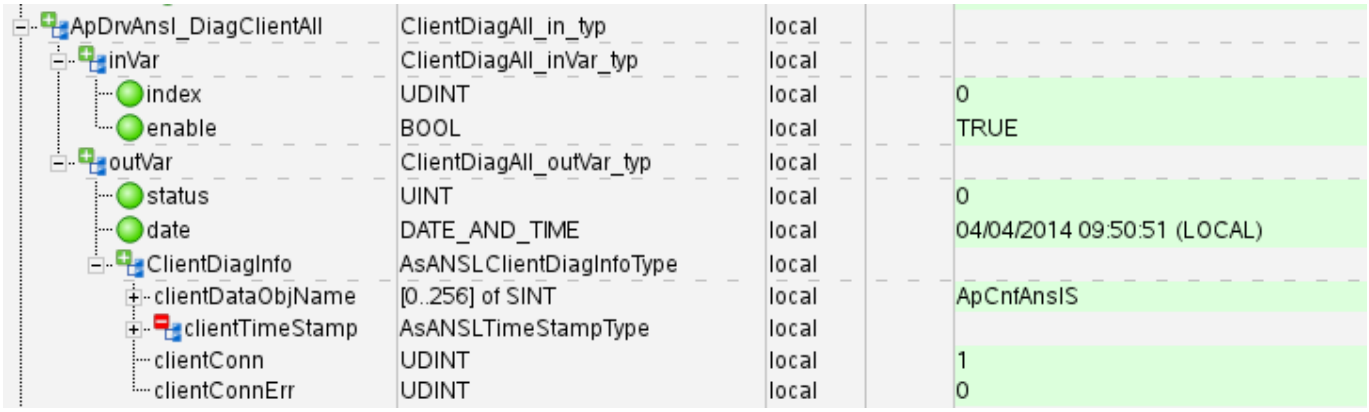
The 'All' variables use filter attributes which meet the search requirements of the elements, e.g. all elements with an error status.

Important:

Activating one or more diagnosis FUBs affects the timing of the driver. The diagnosis should therefore not be used in a critical environment. The ApDrvAnsl is operated in task class 8 and normally has a large tolerance, so that a cycle time violation is unlikely when using diagnosis FUBs, but is possible.

The diagnosis variables which can be written by the user in the Watch are not the variables which are transferred to the FUB afterwards. This avoids letting users destroying a variable (status range) in critical areas.

The variables consist of an input range (inVar) and an output range (outVar). The user fills out the inVar range and the result is shown in the outVar after processing.



ApDrvAnsl_DiagClientAll	ClientDiagAll_in_typ	local	
inVar	ClientDiagAll_inVar_typ	local	
index	UDINT	local	0
enable	BOOL	local	TRUE
outVar	ClientDiagAll_outVar_typ	local	
status	UINT	local	0
date	DATE_AND_TIME	local	04/04/2014 09:50:51 (LOCAL)
ClientDiagInfo	AsANSLClientDiagInfoType	local	
clientDataObjName	[0..256] of SINT	local	ApCnfAnslS
clientTimeStamp	AsANSLTimeStampType	local	
clientConn	UDINT	local	1
clientConnErr	UDINT	local	0

Illustration 2:

A 'date' variable is stored in each outVar range, is interpreted in the Watch, and corresponds to the value of clientTimestamp.timeStampSec. The nanoseconds there are not taken into account!

The dataObjName field should be filled out in almost every inVar range. The name of the configuration data module where the connections and PVs are to be diagnosed must be specified there. There are two data module names in an APROL environment: ApCnfAnslS is the automatically generated data module where the physical view-spanning controller-controller connections are configured (connections tab in the CaeManager controller view). ApCnfAnslC is the data module where the user-specific connections are configured (APROL connections tab in the CaeManager controller view).

Note:

The driver diagnosis carries on as long as at least one of the FUBs below has 1 set on the enable bit. Ensure that the enable bits of all FUBs are set back to 0 before the Watch is closed, in order to end all diagnosis activities.

The following diagnosis FUBs are available:

AsANSLClientDiag

With the AsANSLClientDiag function block, it is possible to detect the basic information of an ANSL client. The FUB configuration is made via the ApDrvAnsl_DiagClient variable.

The name of the configuration module is specified in the inVar range and the diagnosis is started by setting the enable bit.

The status field in the outVar range signals that the query was processed. A status value 0 means that the query was finished successfully and all other elements in the outVar structure contain useful information. The other elements do not contain useful information if there is an error (Status not equal to null); the status should be evaluated with ApDrvAnsl_DiagHelp in the other Watch window

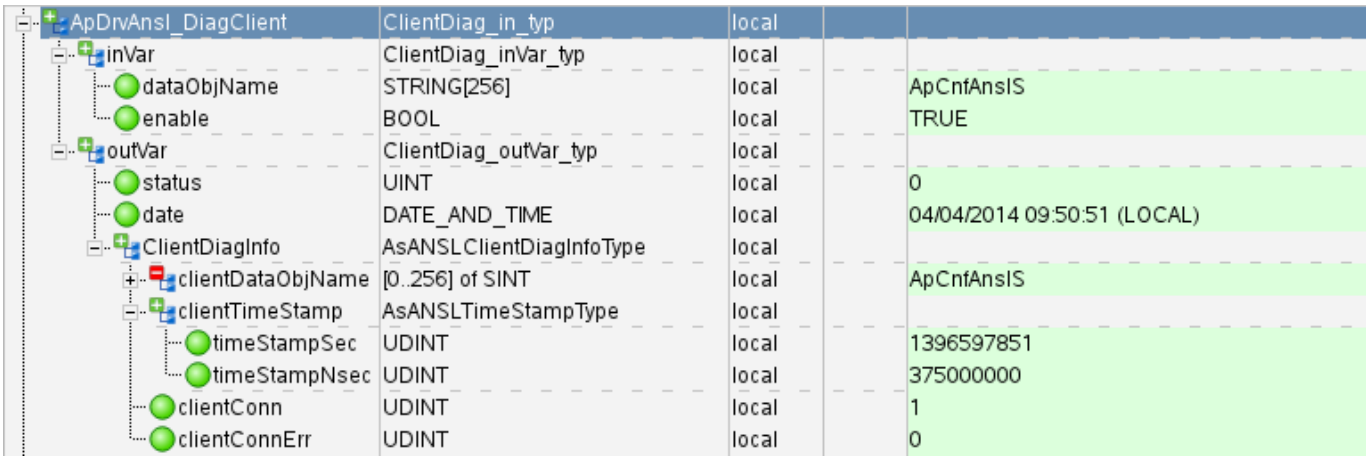


Illustration 3:

ClientDiagInfo:

Parameter	Description
clientDataObjName	Name of the data module
ClientTimeStamp	Time stamp of the last initialization in seconds and nanoseconds since 1.1.1970.
clientConn	Number of connections of this module
clientConnErr	Number of currently erroneous connections of this module

AsANSLClientDiagAll

This FUB is identical to AsANSLClientDiag. The ApDrvAnsl_DiagClientAll variable is used for configuration. A numerical value must be specified in the index field instead of the name of the data module. This value (0 or 1 as a rule) specifies which data module must be analyzed. The name of the data module which is found can be read on the outVar.ClientDiagInfo.clientDataObjName output variable. If an invalid index is entered, a status 31877 – END_OF_LIST is invoked and the other output variables do not have a usable content or that of the last call.

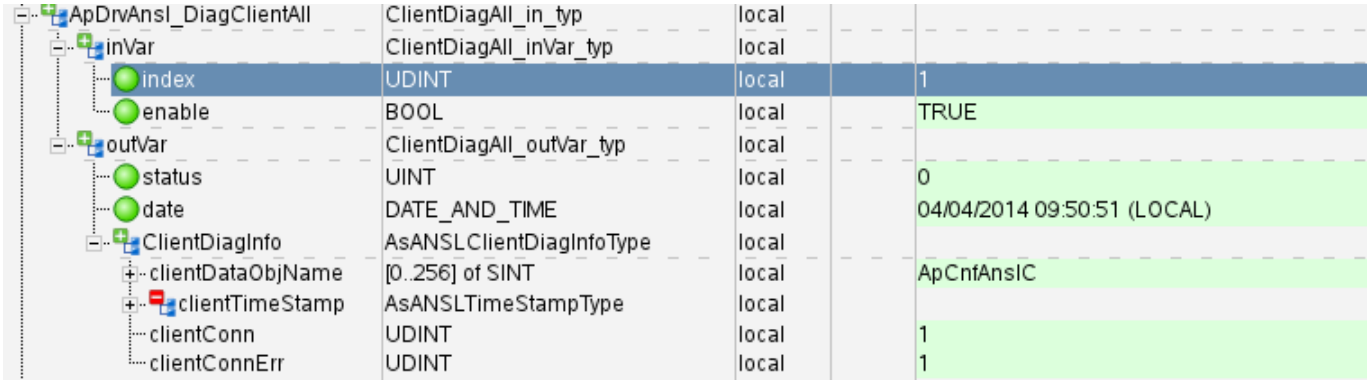


Illustration 4:

ClientDiagInfo:

Parameter	Description
clientDataObjName	Name of the data module

ClientTimeStamp	Time stamp of the last initialization in seconds and nanoseconds since 1.1.1970.
clientConn	Number of connections of this module
clientConnErr	Number of currently erroneous connections of this module

Note:

The index value is only used for repetitions over the data modules.

There is no fixed relation between the index value and the name of the data module. If a configuration module is removed from the controller, the index value of the remaining data modules may change!

AsAnslClientDiagConn

With the AsANSLClientDiagConn function block, it is possible to detect the basic connection information of an ANSL client. The FUB configuration is made via the ApDrvAnsl_DiagConn variable.

The name of the module, the name of the connection, and the enable bit must be set in the inVar range.

The status field in the outVar range signals that the query was processed. A status value 0 means that the query was finished successfully and all other elements in the outVar structure contain useful information. The other elements do not contain useful information if there is an error (Status not equal to null); the status should be evaluated with ApDrvAnsl_DiagHelp in the other Watch window.

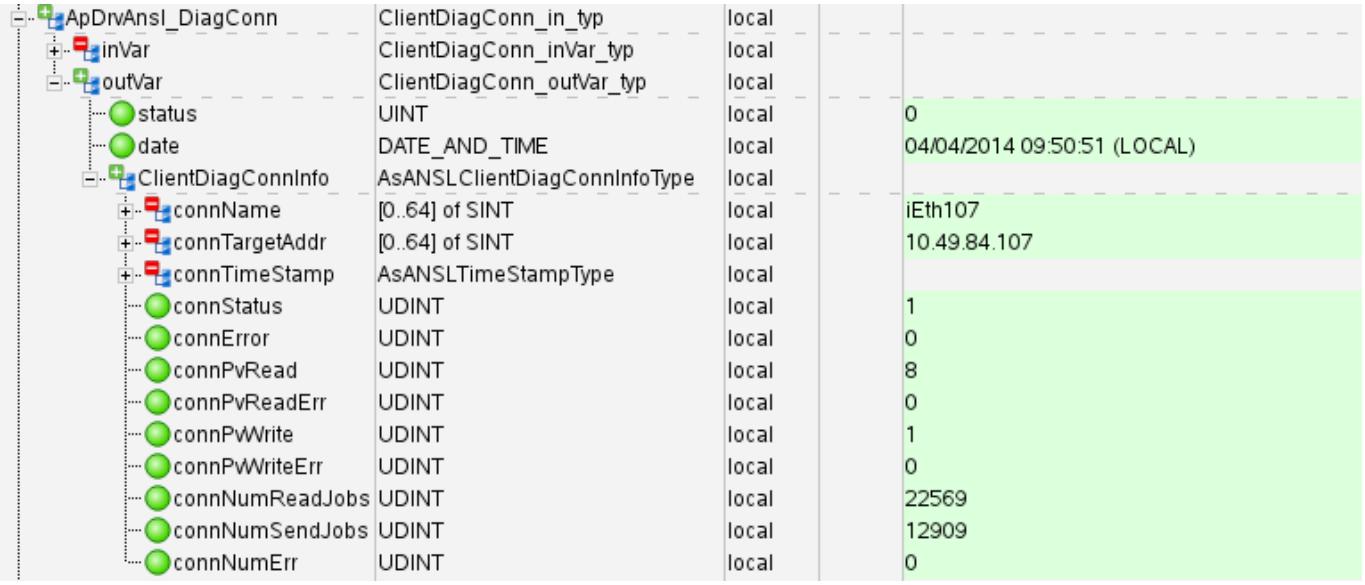


Illustration 5:

ClientDiagConnInfo:

Parameter	Description
connName	Connection name
connTargetAddr	Configured IP address or host name of the remote station
connStatus	Status of the connection 0 = not connected

	1 = connected
connError	Error codes of the connection 0 = No error The rror number used corresponds to the documented ANSL error numbers and is not part of this document.
ConnPvRead	Number of configured read variables
ConnPvReadErr	Number of existing erroneous read variables
ConnPvWrite	Number of configured write variables
ConnPvWriteErr	Number of existing erroneous write variables
ConnNumReadJobs	Number of read jobs executed since the connection was initialized
ConnNumWriteJobs	Number of write jobs executed since the connection was initialized
ConnNumErr	Number of lost connections /errors since the connection was initialized

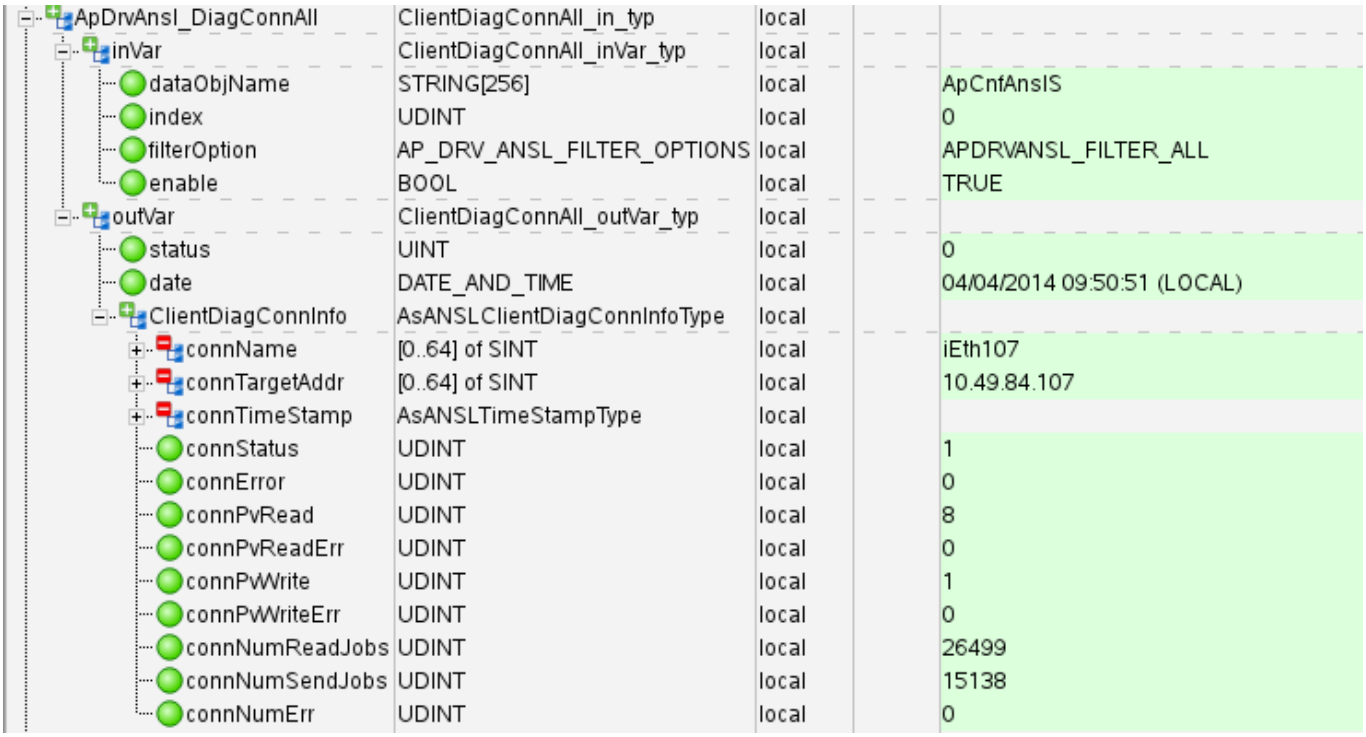
AsAnslClientConnAll

This FUB is identical to AsANSLClientConn. The ApDrvAnsl_DiagConnAll variable is used for configuration. A numerical value must be specified in the index field instead of the name of the connection. This value selects which connection must be analyzed. The name of the connection which is found can be read on the outVar.ClientDiagConnInfo.connName output variable. If an invalid index is entered, a status 31877 – END_OF_LIST is invoked and the other output variables do not have a usable content or that of the last call.

The filterOption specifies if the erroneous or correct connection should be taken into account. The filterOption field should have a string notation in the Watch. The meaning of the filter value can be understood better in this way, as when the decimal display is selected.

filterOption:

Value	Description
0	APDRVANSL_FILTER_ALL Display of all configured connections
1	APDRVANSL_FILTER_NOERROR Display of all connections which currently have no errors
2	APDRVANSL_FILTER_ERROR Display of all connections which currently have an error



ApDrvAnsl_DiagConnAll	ClientDiagConnAll_in_typ	local	
inVar	ClientDiagConnAll_inVar_typ	local	
dataObjName	STRING[256]	local	ApCnfAnsIS
index	UDINT	local	0
filterOption	AP_DRV_ANSL_FILTER_OPTIONS	local	APDRVANSL_FILTER_ALL
enable	BOOL	local	TRUE
outVar	ClientDiagConnAll_outVar_typ	local	
status	UINT	local	0
date	DATE_AND_TIME	local	04/04/2014 09:50:51 (LOCAL)
ClientDiagConnInfo	AsANSLClientDiagConnInfoType	local	
connName	[0..64] of SINT	local	iEth107
connTargetAddr	[0..64] of SINT	local	10.49.84.107
connTimeStamp	AsANSLTimeStampType	local	
connStatus	UDINT	local	1
connError	UDINT	local	0
connPvRead	UDINT	local	8
connPvReadErr	UDINT	local	0
connPvWrite	UDINT	local	1
connPvWriteErr	UDINT	local	0
connNumReadJobs	UDINT	local	26499
connNumSendJobs	UDINT	local	15138
connNumErr	UDINT	local	0

Illustration 6:

ClientDiagConnInfo:

Parameter	Description
connName	Connection name
connTargetAddr	Configured IP address or host name of the remote station
connStatus	Status of the connection 0 = not connected 1 = connected
connError	Error codes of the connection 0 = No error The error number used corresponds to the documented ANSL error numbers and is not part of this document.
ConnPvRead	Number of configured read variables
ConnPvReadErr	Number of existing erroneous read variables
ConnPvWrite	Number of configured write variables
ConnPvWriteErr	Number of existing erroneous write variables
ConnNumReadJobs	Number of read jobs executed since the connection was initialized
ConnNumWriteJobs	Number of write jobs executed since the connection was initialized
ConnNumErr	Number of lost connections /errors since the connection was initialized

Note:

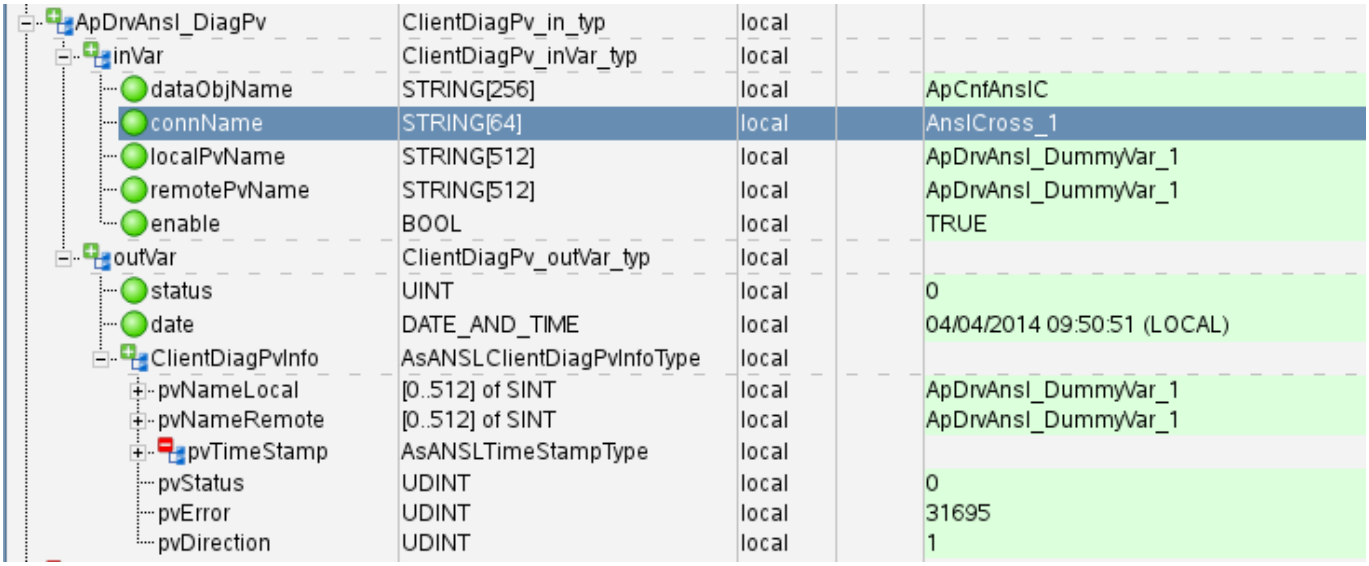
The index value is only used for repetitions over the connections.

There is no fixed relation between the index value and the name of the connection. If a configuration is changed on the controller, the index value of the last analyzed connection may change! There will also be a change if the filterOption field is changed.

AsANSLClientDiagPv

With the AsANSLClientDiagPv function block, it is possible to detect the information of an individual PV connection. The FUB configuration is made via the ApDrvAnsl_DiagPv variable. The name of the module, the name of the connection, the local name of the variable, the variable name on the remote station, and the enable bit must be set in the inVar range.

The status field in the outVar range signals that the query was processed. A status value 0 means that the query was finished successfully and all other elements in the outVar structure contain useful information. The other elements do not have information which can be evaluated if there is an error (Status not equal to null); the status should be evaluated with ApDrvAnsl_DiagHelp in the other Watch window.



ApDrvAnsl_DiagPv	ClientDiagPv_in_typ	local		
inVar	ClientDiagPv_inVar_typ	local		
dataObjName	STRING[256]	local		ApCnfAnslC
connName	STRING[64]	local		AnslCross_1
localPvName	STRING[512]	local		ApDrvAnsl_DummyVar_1
remotePvName	STRING[512]	local		ApDrvAnsl_DummyVar_1
enable	BOOL	local		TRUE
outVar	ClientDiagPv_outVar_typ	local		
status	UINT	local		0
date	DATE_AND_TIME	local		04/04/2014 09:50:51 (LOCAL)
ClientDiagPvInfo	AsANSLClientDiagPvInfoType	local		
pvNameLocal	[0..512] of SINT	local		ApDrvAnsl_DummyVar_1
pvNameRemote	[0..512] of SINT	local		ApDrvAnsl_DummyVar_1
pvTimeStamp	AsANSLTimeStampType	local		
pvStatus	UDINT	local		0
pvError	UDINT	local		31695
pvDirection	UDINT	local		1

Illustration 7:

ClientDiagPvInfo:

Parameter	Description
pvNameLocal	Name of the local variables
pvNameRemote	Name of the variables on the remote station
pvStatus	Status of the connection 0 = not connected 1 = connected
pvError	Error codes of the connection 0 = No error The error number used corresponds to the documented ANSL error numbers and is not part of this document.
pvDirection	Information if write or read variable 1 = read variable 2 = write variable

AsANSLClientDiagPvAll

This FUB is identical to AsANSLClientPv. The ApDrvAnsl_DiagPvAll variable is used for configuration. A numerical value must be specified in the index field instead of both variable names. This value selects which variable must be analyzed. The name of the variables which are found can be read on the outVar.ClientDiagPvInfo.pvNameLocal and outVar.ClientDiagPvInfo.pvNameRemote output variables.

The filterOption specifies if the erroneous or correct connection variables should be taken into account. The filterOption field should have a string notation in the Watch. The meaning of the filter value can be understood better in this way, as when the decimal display is selected.
If an invalid index is entered, a status 31877 – END_OF_LIST is invoked and the other output variables do not have a usable content or that of the last call.

filterOption:

Value	Description
0	APDRVANSL_FILTER_ALL Display of all configured variables
1	APDRVANSL_FILTER_NOERROR Display of all variables which currently have no errors
2	APDRVANSL_FILTER_ERROR Display of all variables which currently have an error

ApDrvAnsl_DiagPvAll	ClientDiagPvAll_in_typ	local	
inVar	ClientDiagPvAll_inVar_typ	local	
dataObjName	STRING[256]	local	ApCnfAnslC
connName	STRING[64]	local	AnslCross_1
index	UDINT	local	0
filterOption	AP_DRV_ANSL_FILTER_OPTIONS	local	APDRVANSL_FILTER_ALL
enable	BOOL	local	TRUE
outVar	ClientDiagPvAll_outVar_typ	local	
status	UINT	local	0
date	DATE_AND_TIME	local	04/04/2014 09:50:51 (LOCAL)
ClientDiagPvInfo	AsANSLClientDiagPvInfoType	local	
pvNameLocal	[0..512] of SINT	local	ApDrvAnsl_DummyVar_1
pvNameRemote	[0..512] of SINT	local	ApDrvAnsl_DummyVar_1
pvTimeStamp	AsANSLTimeStampType	local	
pvStatus	UDINT	local	0
pvError	UDINT	local	31695
pvDirection	UDINT	local	1

Illustration 8:

ClientDiagPvInfo:

Parameter	Description
pvNameLocal	Name of the local variables
pvNameRemote	Name of the variables on the remote station
pvStatus	Status of the connection 0 = not connected

	1 = connected
pvError	Error codes of the connection 0 = No error The rror number used corresponds to the documented ANSL error numbers and is not part of this document.
pvDirection	Information if write or read variable 1 = read variable 2 = write variable

Note:

The index value is only used for repetitions over the variables.

There is no fixed relation between the index value and the name of the variable. If a configuration is changed on the controller, the index value of the last analyzed variable may change! There is also a change if the filterOption field is changed or if the error status of the variable changes, so that the result changes the filtering.

2 dcs2000Driver

2.1 General information about dcsDriver driver package

*Information about the revision history can be obtained in the chapter Revision history in the **appendix**.*

The driver package described here is used for connecting B&R DCS2000 system controllers to APROL via an ARCNET connection. An ARCNET card **must** be used in the control computer.

The following cards can be used in the control computer (PC):

- SH ARC-PCI — PCI card
- SH ARC-PCIu — PCI card
- SH ARC-PCMCIA — PCMCIA card

A maximum of 3 PCI or 1 PCMCIA cards can be used per computer.

All cards will be delivered by *SOHARD AG*, in Fuerth.

In the engineering system, the dcsDriver is to be taken from the CC modules (DCS2000 connection) and used on the control computer (runtime system).

To connect DCS2000 systems, the user must be familiar with the documentation in the DCS2000 system and operator manuals. The user must know how to connect stations via *ARCNET*, and also be able to define routing parameters and be familiar with the structure of data points on the controllers!

2.2 Installation of the dcsDriver software

.

2.2.1 Installing the dcsDriver driver package

*The installation should take place on a PC with the **APROL** runtime system because the drivers and utilities are only needed there.*
*The installation can only be carried out by the **super-user (root)**.*

The installation procedure:

Step	Description
1	Insert the diskette and mount the disk drive to the `/media/floppy` directory.
2	Then go to the `/media/floppy` directory.

Step	Description
3	Install the required package(s) with the rpm command. Example: *rpm –i <RPM-FILENAME> --nodeps --force*

With the command ***rpm –e <PACKAGE_NAME>***, you can uninstall a package if necessary.

Please note the difference between RPM-FILENAME and PACKAGE_NAME!

Examples:

Install:

```
rpm -i APROL-DCS2000_ARCNET-1.X-Y.noarch.rpm --nodeps --force
```

Uninstall:

```
rpm -e APROL-DCS2000_ARCNET
```

After the installation, use ***umount*** to remove the disk drive from the file system and take the diskette out of the drive.

2.2.2 Delivery contents of the dcsDriver driver package

The following files are on the runtime computer after installation of the RPM package (with path):

File with path	Description
/etc/init.d/aprolArcnet	Start script for integrating the ARCNET card in the Linux computer. The configuration comes from /etc/rc.ARCNET, and is created using pccArcnetInstall (see below).
/lib/modules/*/kernel/drivers/aprol/aprolArcnetDriver.o /lib/modules/*/kernel/drivers/pcmcia/PCM20.o **Note:** The stars stand for the various kernel versions e.g. **2.4.20-4GB**	Various kernel modules for different Linux kernels. The hardware is accessed using these modules. aprolArcnetDriver supports the PCI cards and also provides functions for the PCMCIA connection. PCM20 extends the aprolArcnetDriver to include access of the PCMCIA cards.

File with path	Description
/opt/aprol/lib/libPccTelsys.so /opt/aprol/lib/libPccTelsys.so.1 /opt/aprol/lib/libPccTelsys.so.1.1p1-1 **Note:** The numbers after **so.** may be different that the numbers for your version. These numbers come from version numbers of the libraries.	Library with access functions for the DCS2000 system. This is a library file and the symbolic links required by the system.
/opt/aprol/cnf/dcsDriver/example/dcsDriver.cnf /opt/aprol/cnf/dcsDriver/example/dcsDriver.types /opt/aprol/cnf/dcsDriver/example/startup.cnf	Example files for configuring and starting the driver.
/opt/aprol/bin/dcsDriver	APROL driver for reading and writing the process variables
/opt/aprol/bin/dcsExport	Tool to read the list of all process variables from a controller. All PVs and their structure elements are prepared. The exported data can be imported in the APROL system using an import mechanism and the PVs can then be used in the charts.
/opt/aprol/bin/dcsDpDir	Tool used to read all data points on a controller.
/opt/aprol/bin/dcsDpRead	Tool used to read data points on a controller (that can be read!). This tool allows changes to event variables on the controller to be read cyclically and also displayed.
/opt/aprol/bin/dcsDpWrite	Tool used to write to data points on a controller (that can be written to!).
/opt/aprol/bin/aprolArcnetInstall **Note:** aprolArcnetInstall creates the file /etc/rc.ARCNET with your hardware settings.	Configuration tool that must be used to configure the device drivers and therefore also the ARCNET cards on the control computer.

2.2.3 Configuration after the installation

After the installation, the system must be configured with *aprolArcnetInstall* (Release < 2.4 *pccArcnetInstall*).

First, the major device number must be assigned. In most cases, the default major number is ok (set to 80 as default) and should not be changed. Then it is necessary to specify how many PCMCIA cards or PCI cards should be used.

Operating both card types together is not permitted, therefore the number of PCI cards can only be set if PCMCIA cards will not be used.

After this is done, the ARCNET address and the station name must be set for each card. After this is confirmed, the configuration file (`/etc/rc.ARCNET`) is created and the device driver is started.

You can use the "*dmesg*" command to view the console outputs in order to check if all cards were found.

An output showing an ARCNET card that was found looks similar to this:

```
Arcnet Driver for COM20020-PCI          V 2.0.0    Copyright 1995-2004 Bernecker + Rainer Industrie-Elektronik GmbH
register major number 80
supports 3 devices
send timeout set to 10000 milliseconds
found arcnet card (SH-ARC PCIu in slot 5) at address 0x00001490, irq = 18
```

The ARCNET cards are found in the order in which they are installed in the PCI slots: the smallest slot first, then the next slot, etc. The station addresses and the station names are allocated in this order.

2.3 Description and start options for the tools

.

2.3.1 General information about the start options for the tools

The start options for the tools can be separated into communication parameters and application-dependent parameters. The communication parameters are identical for all of the tools described here, the other parameters differ according to the specific applications. Each application can be started with the *–help* option in order to output a list of all start options that can be used including a short description.

2.3.2 Communication parameters

Communication between two DCS2000 partners takes place either directly or using one or more gateway controllers. The connection is defined using the station address of the partner and, when using gateways (not APROL gateways), using the routing for both directions. When communication parameters are mentioned in the following sections, then the following parameters are meant.

Parameter	Description
-board <BOARD>	The device driver used supports several ARCNET cards on a PC. They are numbered internally from 0 to n-1 (for n cards). The BOARD parameter defines the starting point for a communication line. If it is not used, then the first card (Board 0) is used for access. **Value range:** 0 - 2, max. of 3 cards
-dbServAddr <ADDRESS>	The ARCNET address for the communication end point (the controller). **Default value:** None This parameter always has to be specified.

Parameter	Description
-netPath <NETPATH>	Specifies the routing path for the connection from controller to the control computer (runtime system). **Example:** -netPath /n0/APROL The station named APROL must be able to be identified by the controller. The ARCNET card must be configured with aprolArcnetInstall APROL. Default value: Empty string
-routePath <ROUTEPATH>	Specifies the routing path for the connection to the controller. This parameter only has to be set if communication takes place via a gateway controller. Default value: Empty string
-os9Timeout<SEC>	The time in seconds that the application on the control computer waits for the confirmation from the controller that the telegram was received. After this confirmation, the respective request must be forwarded to the application on the controller, and the application may have to return a response telegram. Default value: 10 seconds
-appName <APPLICATION>	Name of the application as it appears in the process list on the controller. **Default value:** APROL
-systemName <SYSTEM>	Name of the Arcnet station in the network. It corresponds to the last part of the netPath, "APROL" in the example above. The station named "APROL" must be known on the controller. The Arcnet card must be configured as "APROL" with aprolArcnetInstall. Default value: APROL

2.3.3 The tools and their options

The additional tools for the ARCNET connection and their extra options are described in the following sections.

2.3.3.1 aprolArcnetInstall

Script for the installation of the device driver for *dcsDriver*.

Parameter	Description
-noStart	Configures the device driver that must be used for the ARCNET cards without restarting it. Normally, the corresponding device driver (kernel module) is started automatically after the configuration.

2.3.3.2 dcsDpDir

dcsDpDir is a command line program that is used to read all data points on a controller and output them using standard-out.

Parameter	Description
No further options	

2.3.3.3 dcsDpRead

dcsDpRead is a command line program that provides the possibility to read information (data) from individual PVs on the controller.

Parameter	Description
-dpName <PV name>	Provides the name of the PV to be read. **Example:** -dpName ANA_IN_01.VALUE
-ext	If this option is also used, additional process variable information is output. This is information about the decimal places and the item status.
-hex	The data bytes are output in hexadecimal.
-event	The data to be read is registered on the controller and value changes are automatically sent from there. In normal mode, data is requested from the application cyclically.

2.3.3.4 dcsDpWrite

dcsDpWrite is a command line program for entering and setting PVs, i.e. the opposite of ***dcsDpRead***.

Parameter	Description
-dpName <PV name>	Provides the name of the PV to be written. **Example:** -dpName ANA_IN_01.VALUE
-value <value>	Value the PV should be set to. See -mode for additional parameters or options.
-mode <char>	Specifies how the new value should be set. I > Increases the current value for the variable by the amount specified in "value" (see above) D > Subtracts "value" from the current value S > Sets the process variable on the controller to "value" T > Toggles the value A > Links the current value of the PV with "value" using an AND function O> Links the current value of the PV with "value" using an OR function X > Links the current value of the PV with value using an EXCLUSIVE OR function

2.3.3.5 dcsExport

This program should only be used for customer-specific projects.

Parameter	Description
-controllerName <NAME>	Sets the name of the controller to NAME. NAME is entered in the export where the name of the controller must be set.
-ignoreCnfPath	If this option is set, all configuration files are searched for relative to the current directory, otherwise they are searched for in the current APROL environment.
-typesFile <TYPES_FILE_NAME>	Specifies the name of the variable description file (also see section *Description of the Types file*)
-exportFile <EXPORT_FILE_NAME>	Specifies the name of the export file where the variables that are found are written.
-cnfFile <CNF_FILE_NAME>	Specifies the name of the configuration file to be created.
-ignoreString <IGNORE_STRING>	Replaces all the characters of the variable name that correspond to a character from the IGNORE_STRING with the character '_' (underline). This is necessary because certain characters such as decimal points are not permitted in the variable name.
-search <SEARCH_PATTERN>	Exports only the variable names that correspond to the SEARCH_PATTERN. All others are suppressed during output.

2.3.3.6 dcsDriver

dcsDriver is the driver that establishes the connection to the B&R controller (DCS2000 system) in the runtime system. This driver is active during runtime on the PCS.

Parameter	Description
-cfgFile <CNF_FILE_NAME)	Specifies the name of the configuration file for the driver. The entry is made as follows: <Controller name>/<CNF_FILE_NAME>
-typesFile<TYPES_FILE_NAME>	Specifies the name of the description file used to read the structure types for a variable.
-aliveTime <MILL_SEC>	Time (in milliseconds) in which the driver uses a GetStat telegram to cyclically check if the partner station is still available.
-respTimeout <RESP_TIMEOUT>	Maximum amount of time (in milliseconds) that the driver waits until it shows that the partner station is no longer available. If a response to the GetStat telegram is not received within RESP_TIMEOUT milliseconds, then it sets the corresponding error variable.

Parameter	Description
-restartTimeout <RST>	Time (in seconds) after which the driver automatically initiates driver redundancy switching if it cannot establish a connection to the partner station. **Value range:** 30 – 600 seconds, **Default:** 60 seconds
-setTz <TZ_STRING>	Sets the time zone for this driver to TZ_STRING. TZ_STRING is used to calculate switching over to daylight savings time and back.
-ignoreString <IGNORE_STRING>	Replaces all the characters of the variable name that correspond to a character from the IGNORE_STRING with the character '_' (underline). This is necessary because certain characters such as decimal points are not permitted in the variable name.
-d DEBUG_FILTER	Activates the driver in debug mode and outputs filtered texts. the respective text output can be switched on or off by setting or deleting the respective bit. The individual bits must be given as an ORed (OR operation) debug filter. Using the debug filter PV for the driver, the debug filter can also be changed after the driver is started, however it must be taken into consideration that some bits are only evaluated when the driver is started.

The following *debug filters* are available:

DEBUG_FILTER	Description
0x00000001	Output of error texts
0x00000002	Outputs normal messages
0x00000004	Output of the types file configuration
0x00000008	Output of the PV configuration
0x00000010	Show the internal connection list
0x00000040	Show open status messages
0x00000080	Show Iosys PV messages

2.4 Configuration of the PVs for the *dcsDriver*

2.4.1 Configuration of the PVs in the engineering system

The PVs with data that will be exchanged between the controller and the runtime system during PCS runtime are created in the CaeManager. The **Control Computer** (runtime system) part of the project is opened and the APROL connection (device-free connection) is used to generate the PVs required in the CAE system. Engineering can start after the generation.

During runtime, the dcsDriver driver expects two files that must also be created in the engineering system and provided to the runtime system and the driver when downloading. These are the files with the variable types (***types file***) and the c*onfiguration file* ***dcsDriver.cnf***

2.4.1.1 Description of the types file

A description must exist in the types file for each variable type used in the system. This 'types' file (default name = ***dcsDriver.types***) is available globally for all controllers in the DCS2000 system and must be in the following directory in the engineering system:

```
$(HOME)/ENGIN/PROJECTS/<Project name>.pgp/pgm/
                        PLS01/APROL/cnf/dcsDriver/
```

It can be created with a text editor that does not add special characters to the text (e.g. vi).

A description begins with the keyword VAR_TYPE and then the list of elements used inside curly brackets. The keywords "INTERN:" and "EXTERN:" have no meaning and only refer to the DCS2000 system manual description. Each element line consists of the element name, the PV type in the Iosys, the PV type on the controller (resulting valid value range, e.g. –128 to 127 for SINT type) and finally the direction of data flow. The '#' character indicates a comment; everything after this character is ignored.

Valid entries for IOS_TYPE:

STRING_TYPE for STRING variables in the Iosys

REAL_TYPE for FLOAT variables in the Iosys,

INT_TYPE for INTEGER variables in the Iosys.

Valid entries for variable types on the controller are:

BOOL,	Measuring range 0-1
BYTE,	Measuring range 0 to 255
SINT,	Measuring range -128 to 127
USINT,	Measuring range 0 - 255
WORD,	Measuring range 0 - 65535
INT,	Measuring range -32768 to 32767
UINT,	Measuring range 0 - 65535
DWORD,	Measuring range 0 to 4294967295
DINT,	Measuring range -2147483648 to 2147483647
UDINT,	Measuring range 0 to 4294967295
REAL	
TIME	
DATE	
DT	
TOD	
STRING	

The types file must be completely customized by the user to match the requirements. All variables that are not needed in the control system should be commented out for performance reasons!

```
VAR_TYPE DIG8
{
INTERNAL:
#                                NOTE       STRING_TYPE STRING  RW  # Long text
#                                LONG_NOTE  STRING_TYPE STRING  RW  # Description
#                                MAN     INT_TYPE    BOOL    RW  # Manual/Auto
                                 VALUE   INT_TYPE    BYTE    RW  # Measurement value
#                                M_VALUE INT_TYPE    BYTE    RW  # Substitute value
#                                I_VALUE INT_TYPE    BTYE    RW  # Initial value
#                                RANGE   INT_TYPE    BYTE    RW  # Steps
#                                VAL_TEXT   STRING_TYPE STRING  R   # Status text
#  PAR_TEXT[1]          STRING_TYPE   STRING  RW      # Status text
#  PAR_TEXT[2]          STRING_TYPE   STRING  RW      # Status text
#  PAR_TEXT[3]          STRING_TYPE   STRING  RW      # Status text
#  PAR_TEXT[4]          STRING_TYPE   STRING  RW      # Status text
#  PAR_TEXT[5]          STRING_TYPE   STRING  RW      # Status text
#  PAR_TEXT[6]          STRING_TYPE   STRING  RW      # Status text
#  PAR_TEXT[7]          STRING_TYPE   STRING  RW      # Status text
#  PAR_TEXT[8]          STRING_TYPE   STRING  RW      # Status text
#                                VAL_FLAG[1] INT_TYPE    BOOL    R   # State
                                 VAL_FLAG[2] INT_TYPE    BOOL    R   # State
#  VAL_FLAG[3]      INT_TYPE      BOOL    R     # State
#  VAL_FLAG[4]      INT_TYPE      BOOL    R     # State
#  VAL_FLAG[5]      INT_TYPE      BOOL    R     # State
#  VAL_FLAG[6]      INT_TYPE      BOOL    R     # State
#  VAL_FLAG[7]      INT_TYPE      BOOL    R     # State
#                                VAL_FLAG[8] INT_TYPE    BOOL    R   # State

EXTERNAL:
#                                &VALUE     struct                # Raw value
#  {
#                                VALUE      INT_TYPE    BYTE    RW  0
#  }
}

VAR_TYPE                         ANA4
{
INTERNAL:
#                                NOTE           STRING_TYPE STRING  RW
# Long text
#                                LONG_NOTE      STRING_TYPE STRING  RW
# Description
#  MAN                  INT_TYPE      BOOL    RW      # Hand/Auto
                                 VALUE      REAL_TYPE  REAL    RW  # Measurement value
#  M_VALUE              REAL_TYPE     REAL    RW      # Substitute value
#  MAX.VALUE            REAL_TYPE     REAL    RW      # Upper limit
#  MAX.FLAG               INT_TYPE      BOOL    R     # Upper limit
#                                MIN.VALUE      REAL_TYPE  REAL    RW
# Lower limit
#                                MIN.FLAG       INT_TYPE   BOOL    R
# Lower limit
#                                LIMIT.HH       REAL_TYPE  REAL    RW
# uppermost limit
#                                LIMIT.H    REAL_TYPE  REAL    RW  # upper limit
#                                LIMIT.L    REAL_TYPE  REAL    RW  # lower limit
#                                LIMIT.LL       REAL_TYPE  REAL    RW
# lowermost limit
#                                LIMIT.FLAG[1]  INT_TYPE      BOOL    R       # Messeage status
#                                LIMIT.FLAG[2]  INT_TYPE      BOOL    R       # Messeage status
#                                LIMIT.FLAG[3]  INT_TYPE      BOOL    R       # Messeage status
#                                LIMIT.FLAG[4]  INT_TYPE      BOOL    R       # Messeage status
#                                LIMIT.STATE INT_TYPE   BYTE    R   # Indicated state
#  CONV.RAW_1   REAL_TYPE REAL  RW  # Raw value 1
#  CONV.FINAL_1   REAL_TYPE REAL  RW  # Final value 1
#  CONV.RAW_2   REAL_TYPE REAL  RW  # Raw value 2
#  CONV.FINAL_2   REAL_TYPE REAL  RW  # Final value 2
#  DIM_TEXT   STRING_TYPE STRING  RW  # Unit
#  I_VALUE    REAL_TYPE REAL  RW  # Initial value

EXTERNAL:
#  &VALUE               struct        # Raw value
#  {
#     VALUE         REAL_TYPE     REAL    R       0
#  }
```

```
}
```

Excerpt from a types file for variable type DIG8

All characters after the '#' to the end of the line are considered comments.

2.4.1.2 Description of the dcsDriver configuration file

A configuration file must be available in the engineering system for each DCS2000 controller used in the system. This configuration file (default name = ***dcsDriver.cnf***) is to be created in the PCS for each controller in the DCS2000 system and must be in the following directory in the engineering system:

```
$(HOME)/ENGIN/PROJECTS/<Project name>.pgp/pgm/
      <Control computer name>/APROL/cnf/dcsDriver/<Controller name>/
```

It can be created with a text editor that does not add special characters to the text (e.g. vi).

Excerpt of a dcsDriver.cnf configuration file for station PU_01

```
#
# Configuration file for dcsDriver
# ================================
#
# Structure:
#
# SPS_NAME:VAR_NAME1 VAR_TYPEx
# SPS_NAME:VAR_NAME2 VAR_TYPEy
# SPS_NAME:VAR_NAME3 VAR_TYPEz
#
# SPS_NAME:VAR_NAME4 NO_TYPE       IOS_TYPE
# ...       ...
#
# with VAR_TYPE = element in types file
# or "NO_TYPE" for a single element without type description
#
# with IOS_TYPE  = INT_TYPE, REAL_TYPE, STRING_TYPE
#
#################################################################

PU_01:                          DIG_EIN_01 DIG8
PU_01:                          ANA_EIN_01 ANA4
```

All characters after the '#' to the end of the line are considered comments.

List of the process variables created, handling information and the direction of data flow

```
PU01_DIG_IN_01_VALUE, handled by driver, bidirectional variable (read and write)
PU01_DIG_IN_01_VAL_FLAG_2_, handled by the driver, read only
PU01_ANA_IN_01_VALUE, handled by the driver, bidirectional variable (read and write)
```

Variables that are only written are handled by the control computer task!

2.4.1.3 Operation of the dcsDriver driver

The driver reads the ***dcsDriver.cnf*** file in line for line, gets the structure description for each variable from the ***dcsDriver.types*** file and creates each element of the structure as a process variable in the Iosys. Invalid special characters such as ".", "&", or "/" are replaced by the "_"

character. The list of the characters to be replaced is found in the IGNORE_STRING (see above). The name of the controller is also included in order to differentiate between the process variables.

The driver is completely event-driven. It only writes variable changes to the controller, and only registers variables that are to be read on the controller. The controller decides itself when a value has changed and sends the current value to the control system.

Variables received from the controller are handled by the driver. Their value can only be set by the driver. If the variables are bi-directional then the surface only sends the desired changes to the driver, and it decides if the value can be written. Then the driver sets the value in the control computer and thus confirms the validity of the value.

Variables that are only written are handled by the control computer task. The control computer task sets the value in the control computer and it must be sent by the driver. If an error occurs, the driver indicates that the variable is invalid which causes it to be set again by the control computer task and therefore written again.

2.4.2 Configuration with *dcsExport*

The ***dcsExport*** tool is used when an existing and working controller from the DCS2000 system is connected to APROL. It is then used for support and fast configuration of the control computer and the ***dcsDriver***.

With ***dcsExport***, it's possible to read a list of data from the controller and export it in a file. The tool establishes the connection to the controller automatically.

In the CaeManager configuration editor, this file can be imported for the control computer under APROL *connections*. It is then a *device-free connection* element. After generating the control computer, the imported process variables are available for engineering; they are then gateway I/Os. At the same time, this tool can be used to create the configuration file for the ***dcsDriver*** (dcsDriver.cnf) between the control computer and the controller.

Before working with ***dcsExport***, a description file must be prepared and the desired structure elements must be declared; the *types file* described above.

dcsExport is an expansion of the ***dcsDpDir*** tool. After establishing a connection, all process variables on the controller are read and the types file is used to create an import list for the configuration editor and a configuration file for the ***dcsDriver***.

The configuration file for the dcsDriver must be copied to the engineering system in the driver directory so that it can also be used as target platform during download.

The corresponding directory is:

```
$(HOME)/ENGIN/PROJECTS/<Project name>.pgp/pgm/
   PLS01/APROL/cnf/dcsDriver/<Station name>/.
```

2.5 Debugging and error handling

2.5.1 The dcsDriver status variables

To control or analyze the *dcsDriver*, each driver creates a number of status variables that can be used with *pio* or *IosEv* for the evaluation. The status variables can also be stored in a CFC and shown in a process graphic for monitoring.

The names of these status variables include the name of the driver (start option *–appName*).

Status variable name	Description
APP_NAME_pidPID_debugFilter	This variable always mirrors the current debug settings. When set, these settings are changed and it's possible to influence the debug outputs for the driver after starting in debug mode. **ATTENTION:** If you forget to set the debug settings back to 0, the hard drive can become full if the messages are being written to a file!
APP_NAME_reconnect	This variable triggers a controller reconnect. 1 = Reconnect, The driver resets the variable to 0.
APP_NAME_connStatus	Variable that mirrors the current connection status for the controller. This variable is only set by the active driver (if driver redundancy is implemented). The following values occur: 0: the driver is shutting down 1: the driver is booting up 2: the driver has read in its configuration 3: the driver is establishing a connection 4: the driver is connected to the controller

2.5.2 dcsDriver error numbers and error messages

An error message for an application, e.g. *dcsDpDir*, is output on the console. This type of error message consists of an error text with "ret =", "errno=" and "errValue=".

It's possible to determine the cause of the error using of the returned values and the error text. Unfortunately not all cases are documented, but the most important error messages should be described here:

Return value	Description
pccInitDbserv failed	
... ret = -1	The connection to the device driver could not be opened. Check for the cause using dmesg, maybe the card cannot be found or too many applications are started. Errno may provide help, also see /usr/include/asm/errno.h
... ret = -2	The connection to the station could not be reserved. Is an identical application already running?
... ret = -3	An *OPEN request* could not be sent or a response was not received within the timeout time or a negative response was received. Are the settings for the netPath correct?
... ret = -4	An ATTACH request could not be sent or a response was not received or a negative response was received.

Return value	Description
... ret = -5	An expected *OPEN REMOTE request* was not received within the reaction time. Are the settings for the Routing-Path correct?
... ret = -6	An expected ATTACH REMOTE request was not received within the reaction time.
... ret = -7	An *Init* for the *dbServ* on the controller could not be carried out. Either writing failed or there was no reaction from the partner station or the partner station sent a negative response.
pccArcnetRemoteService failed	
... ret = -1	A send telegram could not be sent successfully
... ret = -3	Timeout when waiting for a response telegram
pccArcnetService failed	
... ret = -1	A write telegram could not be sent
... ret = -3	Timeout when waiting for a response telegram

3 dcsEventDriver

3.1 General information - dcsEventDriver

The *dcsEventDriver* is used for establishing an event-driven connection from the APROL system to the B&R controller according to the DCS2000 system *EVT.DRV concept*. The *DCS2000 manual* is used as the foundation of this concept.

The driver consists of two parts; one part runs on the control computer and handles the connection to the APROL system, the other part runs on the controller and makes it possible to connect to the task on the controller. Communication between the parts of the driver takes place using the TCP/IP protocol and is based on Ethernet.

To connect DCS2000 systems, the user must be familiar with the documentation in the DCS2000 system and operator manuals.
Additionally, the user should be familiar with the structure of the data points on the controllers!

3.2 Configuration of the dcsEventDriver

The configuration of the *dcsEventDriver* takes place in the *CaeManager* within the framework of the CC modules.

The following tables contain a short description of the driver options:

Option	Value range	Description
-cfgFile		Specifies the name of the configuration file for the driver. The path consists of `...RUNTIME/cnf/dcsEventdriver/[<Driver name>]/file`.
-genImport		If a file is specified here, then the current PV configuration is written to it when the driver is started. This can also be read as a device-free link in APROL. The file can be found in the configuration file path.
-iosys	port [0 -15]	This option determines the program connection to the specified Runtime System with Iosys and its port. The Iosys port must always correspond to the Iosys port specified when the control computer master or slave is configured. When using a redundant Runtime System, the name and port number of the redundant computer must be separated from the first computer using a comma. This entry is usually taken automatically from the content of the *Communication* field, meaning that changes don't normally need to be made here.

Option	Value range	Description
-n		This allows you to specify a driver name. It is incorporated into the configuration files when forming the path. (see *–cfgFile*)
-controllerIpAddr		Entry for the IP address of the controller.
-controllerPort		Specifies the TCP/IP port number of the controller.
-restart	[0 – 20]	This option makes it possible for the application to be restarted automatically using the APROL startup mechanism if the application has been closed externally. This option is **not** activated by default. If the specified value is exceeded, automatic restarts no longer take place. This mechanism is only switched back to active after manually resetting it with the StartManager or carrying out a new download.
-self		The Self ID is a unique identifier for the program instance and is specified by the system, which increments this two-digit number for each instance. The Self ID identifies the instance using a character string which is attached to the name of the application in the operating system process list when the application has been started. If several instances of this application run on the same computer, these instances can be told apart using the different Self IDs. The Self ID can also be overwritten with a name so the string is more descriptive.
-typesFile		Specifies the name of the TYPE file where the DCS2000 types are defined. The path is the same as for -cfgFile.

The driver is configured on the control computer using a *cfg* file; the controller part is configured dynamically when establishing a connection with the control computer.

An example configuration file (APROL.cfg) can be found in directory `/opt/aprol/doc/packages/dcsEventDriver/.`

3.2.1 Structure of the configuration file on the control computer

The structure of the configuration file is clarified using the following example:

```
# configId: 995275737
# StatusVar: 1
# setTime: 40
# upBuffer: event_up
# downBuffer: downBuffer
#
# Name          IOType  AnlNr UstNr Art DpNr TypeEVT

AnlNr           R       1     1     1   1    ANA
Spannung        W       1     1     2   2    ANA4
Zähler          S       1     1     2   2    ZA1
```

The entries in the configuration file have the following meanings:

Configuration file entry	Description
configID	The configID must change for each new configuration.
StatusVar	StatusVar is used to determine if a status variable is created for each data point. 0 = no status variables 1 = a status variable is created for each data point that is made up of 'Name' + '_STAT'.(e.g. Voltage_STAT)
setTime	The value setTime specifies if a time adjustment should be made from the driver to the controller: -1 time not supplied 0 ... 32000 time correction in ms
upBuffer / downBuffer	Specifies the name of the communication buffer on the controller. The names can have a maximum of 32 characters.
Name	Name includes the data points in the APROL system
IOType	The IOType can be R = read W = write S = read/write
SysNo	System number
SstNo	Substation number
Type	Substation type
DpNo	Data point number
TypeEVT	The TypeEVT must be found in the DCS type file that is described in the following section.

An example DCS type file (APROL.types) can be found in directory `/opt/aprol/doc/packages/dcsEventDriver/`.

Example of a DCS type file:

```
#Name of the DCS type
VAR_TYPE ZA1
{
# WertName              PV_Type      I/O  off DCS-Wert
#
  ZL                    REAL_TYPE    R     0  UINT24 # Comment
  RESET                 INT_TYPE     W     3  UBYTE  #
  LAST                  REAL_TYPE    R     4  UINT24 #
  OG                    REAL_TYPE    S     7  UINT24 #
  GMLD                  INT_TYPE     R    10  UBYTE  #
}
```

The entries in the DCS type file have the following meanings:

ValueName
If a DCS type has multiple values, then the DCS value name is added as a suffix after a "_" (e.g. *Counter_RESET*)

The following are possible ***PV_types***:
REAL_TYPE
INT_TYPE

The ***I/O*** entry specifies the direction data is transferred:

Entry	Description
R	read only
W	write
S	read/write

off specifies the position of the value for a structure

The following ***DCS*** values exist

Type	Length	Signed Yes/No
FLOAT	4	0
ULONG	4	1
LONG	4	0
UINT24	3	1 ! is for the 3 byte UCHAR
UINT	2	1
INT	2	0
UBYTE	1	1
BYTE	1	0

3.3 Interfaces

The **controller side** is specified using the ***EVT.DRV concept***. Each data point is uniquely labeled with a *SYS number*, *SST number*, *Data point number* and the *Data point type*. The length of the data point data is also known. One read and one write buffer are available for exchanging data. This buffer is only one entry in size, so only one data point can be exchanged per double cycle time.

On the APROL side, the interface is specified by the connection to the Iosys.

3.4 Procedure from the configuration to active connection

After starting the driver, the DCS2000 system types are read in (i.e. the actual event configuration is read in). This is used to create the control computer structure.

A data point can also contain a structure, so it is possible for a data point to have several PVs. Each PV is registered on the Iosys – or created if it does not yet exist.

Then a connection to the controller is established. When the connection is active, the *configID* (configuration identification) is compared with the ID on the controller.

If the IDs are the same, normal operation starts. If the IDs are different, a new configuration is sent to the controller. If a new configuration is sent, the old event buffer is deleted and all data points are read in a complete read procedure. Even data points that are only designed to be written are read first and then written to the control computer (to APROL). Normal operation starts for all data points that respond to the read procedure. Then all controller events are transferred to the control computer. If the connection is not active, the data is placed in a buffer (400 entries).

The other direction is a little bit different. All Iosys events are sent to the controller using two buffers. The first buffer is on the controller and can hold 400 entries. If the buffer is filled to more than 90 percent during a "slow collector task", the rest of the events are buffered on the control computer. If the buffer status is < 10 percent, it is filled up again from the control computer buffer. The controller sends the data point to the other tasks using the event buffer.

If a write acknowledgement is received for this task, then it is sent from the controller to the control computer with the value. The control computer driver "remembers" the acknowledged value for the data point. If the connection to the controller is broken now, the control computer driver returns the last acknowledged value to the Iosys for each Iosys change. When the connection is reestablished, the procedure continues as described above.

3.5 dcsEventDriver status messages

The following states can be read from the status PV for a data point:

Bit	State
bit 0	Substation on the data point is not functioning
bit 1	Data point is not functioning
bit 2	Connection to the APROL controller is faulty

A PV with the name "*drivername_STAT*" is also created. This PV provides information about the driver status on the PCS and on the controller.

Bits 0-15 provide information about the PCS status.

Bit	State
bit 0	Driver has a connection to the Iosys
bit 1	Read valid DCS types
bit 2	Read valid configuration

Bit	State
bit 3	Connection to the controller exists
bit 4	A new configuration was created the last time the driver was started
bit 5	The old configuration was used the last time the driver was started
bit 6	The controller driver is configured and started
bit 7	Shows when an event is received from the controller (toggle)
bit 8	Shows when an event is sent to the controller (toggle)

Bits 9-15 are not used.

Bits 16 - 31 show the controller status (only valid if bit 3 is set)

Bit	State
bit 16	downBuffer is not enabled (no task on the other side)
bit 17	The write buffer for DownEVT on the controller is filled to over 90% (bit is cleared when under 10 %)
bit 18	UpEVTs on the controller have been lost. (cleared when a new connection is established)
bit 19	downBuffer not found on the controller.
bit 20	upBuffer not found on the controller.

Bits 21-31 are not used.

4 Dispatcher

4.1 General information about the Dispatcher

4.1.1 Functionality of the Dispatcher

The Dispatcher serves as an engineering simplification, and has been developed for certain project specific characteristics. One characteristic is for example that the execution of the same functions in connection with several similar pieces of equipment, which are not in use at the same time. In this case the Dispatcher can be used, by a simple change of a name, to switch the logic of all of the connections from one piece of equipment to another.

In order to realize such demands with the Dispatcher it is possible to connect two process variables with another in a logic controlled way for each task. The variable that is to be allocated can be chosen using a variable name. The name to be selected is made available to the job by another process variable. Several jobs per configuration can be edited simultaneously with the Dispatcher. Several jobs can be influenced by a selection variable by grouping in the configuration. With the appropriate validation pattern in the configuration, the Dispatcher can check a selection name, and impede the connection to the resulting process variable.

If a connection is established between two process variables, the value of the selected process variable is made available in pure read-only process, e.g. in a configured input variable. Only reading, or additionally writing can be explicitly allowed for this job.

The driver configuration is carried out in the control computer in the APROL connections tab. The Dispatcher is configured here with the usual dialog.

The Dispatcher is a pure control computer driver, and therefore can only connect process variables that are available on the control computer. As with all of the available system drivers in the APROL system, the Dispatcher supports process redundancy, and can be deployed in a control computer redundancy cluster.

Example of a Dispatcher configuration

Group configuration	**G1**
Group	Tank
Selection validator	T[0-9]
Job type:	Read task
Selection variable	T_NAM
Access variable	T_ZUG

Task configuration	**A1**	**A2**
Task ID	ANST_V	RÜCK_V
IEC type	BOOL	BOOL

Prototype	AV_@	RV_@
Selection validator	<group settings>	<group settings>
Job type:	Write task	Group settings
Selection variable	---	---
Access variable	A_ZUG	R_ZUG
Input variable	---	R_INP
Output variable	A_OUT	---

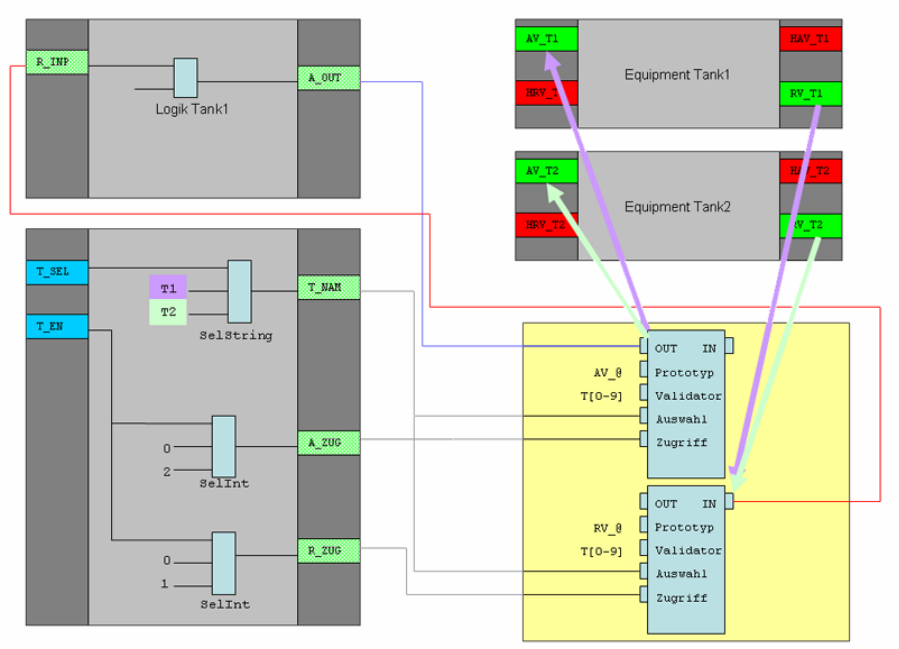


*Illustration **9**: Functionality of the APROL Dispatcher*

4.1.2 Requirements / Limitations

The following requirements are to be observed by the user:

- Changing process variables that are already supplied by other APROL processes is not supported by the Dispatcher!
- A job can only write to one process variable, which already exists due to other configurations in the APROL system, and is labeled as remanent. This is possible via a device-free connection in the control computer. Already existing connectors that are on the controller can be used. We recommend the use of gateway variables on the border of CFCs, as it is only possible to stipulate a clear direction of data using the device-free connections, and it is only there that all of the benefits such as import, export, etc. are available.
- A task can only read process variables, which have already been created by other configurations in the APROL system. If the variable is labeled as "not valid" in the Iosys, this leads to an error message.
- The Dispatcher is solely suitable for use on the control computer.
- Different pieces of equipment, which are controlled by a logic from the Dispatcher, cannot be processed simultaneously.
- At present, a maximum of 5000 jobs are allowed.

Due to the virtual addressing of the connection for input and output variables, the following limitations are to be observed during the project analysis and monitoring.

- A sensible plan debugging of the job variables in use is not possible!
- A signal tracing of the job variables is not possible!
- A documentation of the connection paths is not possible!
- When using connectors as the selected process variables, a documentation of these variables is not possible!

4.1.3 Dispatcher delivery contents

The software package is delivered as an RPM package. The following files are created when installing the package:

File name	Description
/opt/aprol/bin/Dispatcher	Application Dispatcher, which is started on the runtime system.
/opt/aprol/doc/packages/Dispatcher/ver.txt /opt/aprol/doc/packages/Dispatcher/version.txt	Version information about the drivers contained.
/opt/aprol/cnf/Dispatcher.cnf	Example configuration

4.2 Configuration of the Dispatcher on the control computer

4.2.1 Dispatcher start options

The configuration of the Dispatcher's start options takes place in the usual manner, in the scope of the CC modules.

Starting a driver beyond the control of the AprolLoader is only allowed to be done by an experienced APROL user! The system can be put into an unsafe state with a false configuration, or drivers that are running twice.

The following table contains a list of the start options and the respective descriptions:

Option	Value range	Description
-cfg <name>		The name of the driver's configuration file is set with this option. This file must be located in the directory of the entry -plc. Default value: driver.cfg

Option	Value range	Description
-iosys <server1:port, server2:port>	port [0 -15]	This option determines the program connection to the specified Runtime System with Iosys and its port. The Iosys port must always correspond to the Iosys port specified when the control computer master or slave is configured. When using a redundant Runtime System, the name and port number of the redundant computer must be separated from the first computer using a comma. This entry is usually taken automatically from the content of the *Communication* field, meaning that changes don't normally need to be made here.
-plc < driver-name >		This parameter defines the environment where the driver should be started. The driver searches for its configuration file in the *…/RUNTIME/cnf/Dispatcher/<driver-name>* directory. At the same time, the driver is registered in the system with this name for the Iosys and forms the name of its status variables (see below) via *driver-name*. This parameter must be set! Default value: CFG1
-restart <value>	[0 – 20]	This option makes it possible for the application to be restarted automatically using the APROL startup mechanism if the application has been closed externally. The value entered defines the maximum amount of restarts since the last start of the instance via the AprolLoader. This option is not activated by default. If the specified value is exceeded, automatic restarts no longer take place. This mechanism is only switched back to active after manually resetting it with the **StartManager** or carrying out a new download.
-retrytimer <time [ms]>		With this option the time in [ms] is set, in which a re-connect is attempted after an error (for example with an unsuccessful validator check). Default value: 1000
-self <id>		The *Self ID* is a unique identifier for the program instance and is specified by the system, which increments this two-digit number for each instance. The *Self ID* identifies the instance using a character string which is attached to the name of the application in the operating system process list when the application has been started. If several instances of this application run on the same computer, these instances can be told apart using the different *Self IDs*. The *Self ID* can also be overwritten by a named string to make it more descriptive.

4.2.2 Configuration of the Dispatcher

The Dispatcher configuration is takes place in the APROL system in the APROL connections tab. A configuration must be created for each Dispatcher that is started.

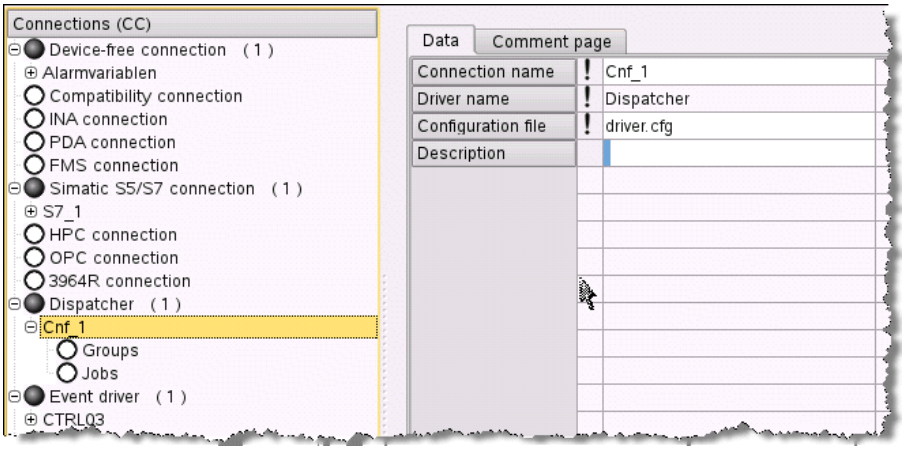


*Illustration **10**: Configuration view of a Dispatcher*

A short description, and a comment about the configuration can be made. Furthermore, the name of the configuration file used is stated. As is usual with the connections, a preview of the configuration that has been made can be created here.

Individual or grouped jobs can be configured underneath this entry.

4.2.3 Configuration of the Dispatcher jobs

A job configures exactly one dynamic connection between a process variable defined by name and a Dispatcher input or output variable.

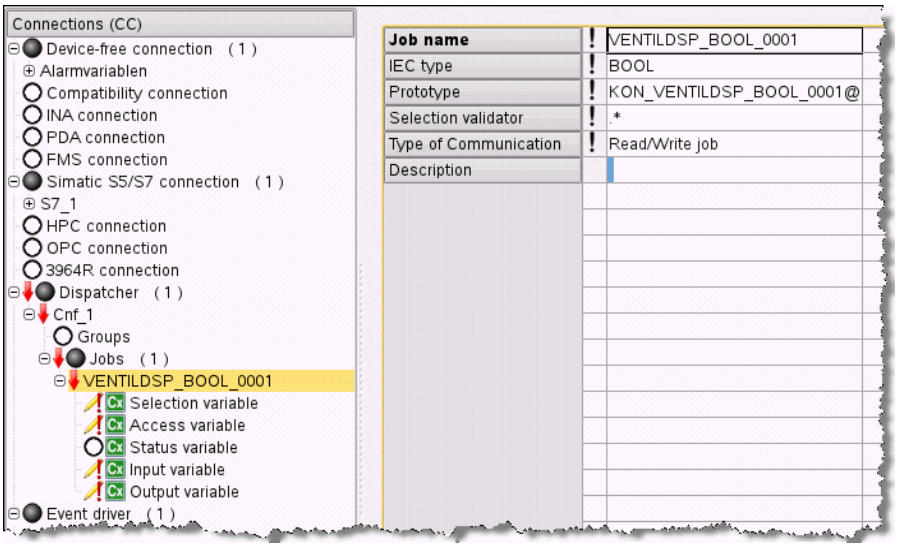


*Illustration **11**: Configuration view of a job*

The following attributes and variables must be configured in each job for this purpose.

Attribute	Example	Description	Required
Job name	Disp_Cnf_1_1	This is the name of the task. The debug output, for example, can be filtered according to this.	yes

Attribute	Example	Description	Required
IEC type	BOOL	The IEC type of the selected process variable is entered here. This type must correspond to the input and output variables.	yes
Prototype	VENTIL_@	Name pattern of the process variables to be selected. The place holder "@" that is to be stated in the pattern will be replaced later by the content of selection variable.	yes
Selection validator	MSR_[0-9][0-9]	This value is used for validation of the selection variable. This validator can be a "regular expression", and is checked before the connection establishment. **This attribute can be omitted underneath a configuration and is then taken from the group configuration!**	yes
Job type:	Write task	This setting defines the task's data direction. There is a choice between – Read task – Write task – Read/write task – Group setting (only exists below a group) The configuration of the respective in/output variable is released depending on these settings. **This attribute can be omitted underneath a configuration and is then taken from the group configuration!**	yes
Description		A short description of the job can be entered here.	no

Variable	Direction	IEC type	Description
Selection variable	Output	STRING	The process variable to be selected is defined with this process variable. The resulting name is made up of a combination of the **content** of the selection variable and the attribute prototype. The content must match the selection validator attribute.

Variable	Direction	IEC type	Description
			This variable can be omitted underneath a configuration and is then taken from the group configuration!
Access variable	Output	INT	The access to the selected process variable is controlled with the value of this process variable. The following values are possible: 0 = Job inactive, no connection to the selected process variable. 1 = Read connection and value of the selected process variable. If the value read is "not valid", an error message is generated. 2 = Additionally to 1, if the task type contains write, the value of the output variable is written in the selected process variable. If the selected process variable has not been created by another project part, this leads to an error message. **This variable can be omitted underneath a configuration and is then taken from the group configuration!**
Input variable	Input	<job type> INT,REAL,...	The value read from the selected process variables is written in this variable, which is usual situated on the chart's input border. The value is only actualized when the access variable has a value >0. The IEC type corresponds to the task IEC type. **Only configured with** – **Read task** – **Read/write task** **possible.**
Output variable	Output	<job type> INT,REAL,...	The value of this variable is read from the chart's output border when the access variable is set to 2, and written to the selected process variable, which is generally in the chart's input border. The value is actualized until the access variable has a value <2.

Variable	Direction	IEC type	Description
			The IEC type corresponds to the task IEC type. **Only configured with** – **Write task** – **Read/write task** **possible.**
Status variable	Input	INT	This state variable detects the state of the job, and can adopt the following values: 0 = "no error" Errorless state 1 = „compile of regular expression failed“ Validation of the regular expression show errors, meaning that an invalid regular expression has been configured 2 = "execute of regular expression failed“ The content of the selection variable does not match the selection validator 3 = "duplicate dispatcher” There are simultaneously several active tasks for one and the same selected process variable. 4 = "wrong dispatcher state” This marks the desired loss of a pending write task when changing the access variable quickly to a read-only access. 5 = "wrong command state” The value of the access variable is outside the valid range [0, 1, 2] 6 = "wrong IOSYS type” The type of the connected variable (input, output variable) does not match the configured type of the selected variable. 7 = "IOSYS value is not valid” The value of the process variable that has been selected for reading is not valid. 8 = "dispatcher pv is not

Variable	Direction	IEC type	Description
			remanent” The selected process variable has not been created by any CAE project part. 9 = "dispatcher pv is not sourced” The process variable that is to be written cannot be supplied by the Dispatcher, i.e. another process is already supplying this variable 10 = "no dispatcher pv in IOSYS found” The selected process variable could not be connected 11 = "no inpv in IOSYS found” The input variable could not be created, e.g. because there is no memory available to the system in the Iosys.

4.2.4 Configuration of the Dispatcher groups

The group configuration simplifies the creation of many tasks, which for example have the same choice validator and the same choice variable.

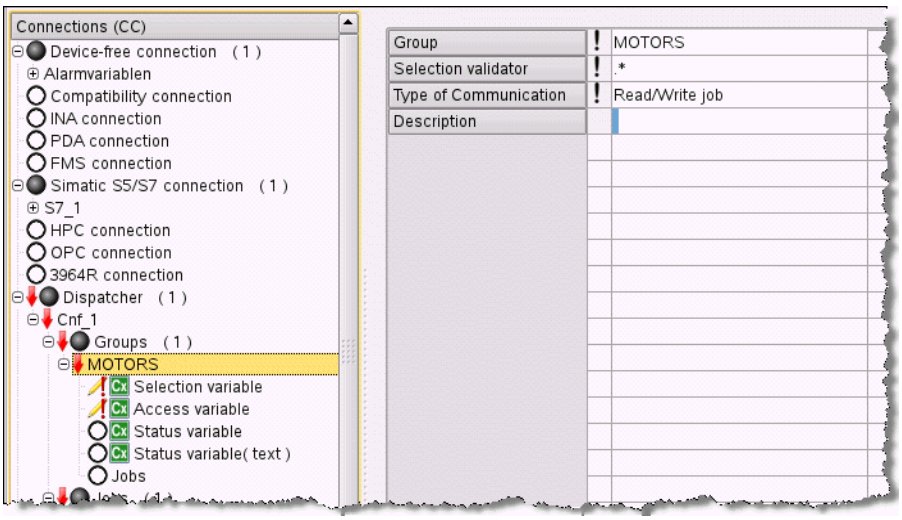


*Illustration **12**: Configuration view of a group*

The choice validator and the task type, as well as the choice variable and the access variable for all of the sub-tasks, can be stipulated in the group configuration. The attributes and variables that are named in the group can then be omitted in the tasks. When necessary, the group information that is in these tasks can be overwritten.

Attribute	Example	Description	Necessary
Group	Grp_1	Name of the group. The debug output, for example, can be filtered according to this.	Yes
Selection validator	MSR_[0-9][0-9]	This value is used for validation of the selection variable. This validator can be a "regular expression", and is checked before the connection establishment. **This attribute can be overwritten in the tasks that lie under this group.**	Yes
Type of Communication	Write task	The data direction of all tasks under this group can be pre-defined with this setting. There is a choice between – Read task – Write task – Read/write task The configuration of the respective in/output variable under this group is released depending on this setting. **This attribute can be overwritten in the tasks that lie under this group.**	
Description		A short description of the job can be entered here.	no

Variable	Direction	IEC type	Description
Selection variable	Output	STRING	The process variable to be selected is defined with this process variable. The resulting name is made up of a combination of the **content** of the selection variable and the attribute prototype. The content must match the selection validator attribute. **This attribute can be overwritten in the tasks that lie under this group.**

Variable	Direction	IEC type	Description
Access variable	Output	INT	The access to the selected process variable is controlled with the value of this process variable. The following values are possible: 0 = Job inactive, no connection to the selected process variable. 1 = Read connection and value of the selected process variable. If the value read is "not valid", an error message is generated. 2 = Additionally to 1, if the task type contains write, the value of the output variable is written in the selected process variable. An error message is generated when the selected process variable is not "remanent". **This variable can be overwritten in the tasks that lie under this group.**
Status variable	Input	INT	The status of the group can be detected with this status variable. The variable can adopt the following values: 0 = All group tasks OK 1 = At least one group task erroneous
Status variable (text)	Input	STRING	The status of the erroneous group entries can be accessed as a text string here. If more than one task is erroneous then the status of each is output cyclically.

4.3 Possible diagnostics when implementing the Dispatcher on the control computer

As with all processes in APROL, the Dispatcher also writes its start/stop messages, and eventual errors and warnings, in the local system messages of the respective APROL server.

The evaluation takes place in the normal way on the APROL system with '**APROL / Diagnostic / System messages (local)'**, or on an external APROL system with '**APROL / Reports / Detail report / System messages (central)',** if forwarding has been configured.

In order to obtain more detailed information about the Dispatcher and its configured tasks during a start-up it is possible to extend the Dispatcher's tasks with a dialog. For this purpose, the Dispatcher can be started on the console of the respective APROL runtime system with the following additional option

-d_local <debugFilter>.

The debug filter can be set to the integer value of the <debugFilter> with the APROL tool *pio* via an Iosys variable (DebugFilter variable) that has the name

DRIVER_<Driver name>_<Process ID>_debugFilter.

The <debugFilter> is a bit mask with which the different outputs can be controlled.

A further Iosys variable with the name
DRIVER_<Driver name>_<Process ID>_debugRegex
can be created for a special filtering in the DBG_ONLINE mode. When using a regular expression with this variable, the active outputs in the DBG_ONLINE mode can be filtered for certain groups and/or tasks.

Additionally, all DBG_ERROR and DBG_ONLINE outputs are written in a process variable with the name
DRIVER_<Driver name>_<Process ID>_errorTxt. The error text can be read from this variable with IosEv.

The following filtering is controlled with the <debugFilter>:

debugFilter	**Description**
1 (0x01) (%0000 0001)	DBG_ERRORS The extended error and warning messages are output here. Example: "no persist flag for pv VENTIL_T1" "wrong iosys type for pv VENTIL_T2"
2 (0x02) (%0000 0010)	DBG_NORMAL This filter provides messages, which can be helpful for analysis in the start-up phase. Example: "ACCESS PV: VENTIL_T1" "RELEASE PV: VENTIL_T1"
4 (0x04) (%0000 0100)	DBG_IOSYS The value and status changes of the configured Iosys variables are shown with this. Example: "changed_cmdgrp for CMD_TANK1 (-VS—P)" "changed_cmdpv for VENTIL_T1 ((-VS—P)"
8 (0x08) (%0000 1000)	DBG_ONLINE All of the individual task variables are recorded with regard to their connection and value changes with this filter. With the above mentioned Iosys variable "DRIVER_<name>_<pid>_debugRegex", the output can be limited to certain tasks and/or groups. **Because these tasks can have a large scope, this operation should only be carried out for short periods, or with a limited group/task filter, otherwise an overflow of the SysLogging could occur.**
16 (0x10) (%0001 0000)	DBG_CONFIGURATION The start configuration that was found by the driver is output with this.

4.4 Driver configuration example (Dispatcher)

```
<?xml version="1.0" encoding="UTF-8" ?>
<Dispatcher>
<Comment>
##############################################################
## APROL R 3.2-0127
## Kopplungstyp:  Dispatcher (Dispatcher)
## Kopplungsname: CFG1
## Version: V1.2
## Erzeugt: 06.09.2007 16:49:40 CEST   aprol
##############################################################
</Comment>
<Group name="ANTRIEBE" validator=".*" req_type="RDWR">
<SelPv>ANTRIEBE_SelPv</SelPv>
<CmdPv>ANTRIEBE_CmdPv</CmdPv>
<StatePv>ANTRIEBE_StatePv</StatePv>
<StateTxtPv>ANTRIEBE_StateTxtPv</StateTxtPv>
<GDsp id="MOTORDSP_BOOL_0001" prototype="KON_MOTORDSP_BOOL_0001@"  type="BOOL" req_type="GROUP" >
<Validator>.*</Validator>
<SelPv>MOTORDSP_BOOL_0001_SelPv</SelPv>
<CmdPv>MOTORDSP_BOOL_0001_CmdPv</CmdPv>
<StatePv>MOTORDSP_BOOL_0001_StatePv</StatePv>
<InPv>MOTORDSP_BOOL_0001_InPv</InPv>
<OutPv>MOTORDSP_BOOL_0001_OutPv</OutPv>
</GDsp>
<GDsp id="MOTORDSP_BYTE_0001" prototype="KON_MOTORDSP_BYTE_0001@"  type="BYTE" req_type="GROUP" >
<Validator>.*</Validator>
<SelPv>MOTORDSP_BYTE_0001_SelPv</SelPv>
<CmdPv>MOTORDSP_BYTE_0001_CmdPv</CmdPv>
<StatePv>MOTORDSP_BYTE_0001_StatePv</StatePv>
<InPv>MOTORDSP_BYTE_0001_InPv</InPv>
<OutPv>MOTORDSP_BYTE_0001_OutPv</OutPv>
</GDsp>
<GDsp id="MOTORDSP_DATE_0001" prototype="KON_MOTORDSP_DATE_0001@"  type="DATE" req_type="GROUP" >
<Validator>.*</Validator>
<SelPv>MOTORDSP_DATE_0001_SelPv</SelPv>
<CmdPv>MOTORDSP_DATE_0001_CmdPv</CmdPv>
<StatePv>MOTORDSP_DATE_0001_StatePv</StatePv>
<InPv>MOTORDSP_DATE_0001_InPv</InPv>
"driver.cfg" 414L, 18167C
```

*Illustration **13**: XML code of the configuration file*

The following table gives an overview of XML tags and the equivalent variable identifiers on the surface.

XML tag	Variable
<SelPv>	Selection variable
<CmdPv>	Access variable
<StatePv>	Status variable
<StateTxtPv>	Status variable (text)
<Validator>	Selection validator
<InPv>	Input variable
<OutPv>	Output variable

4.5 Dispatcher status variables

This description is under construction at present.

*Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.*

5 EventDriver

5.1 General information about the EventDriver

Please read the information about process bus redundancy in chapter Process bus redundancy for Ethernet connections in this manual!

Please also note the information in chapter Technical notes about the use of controllers.

5.1.1 Using the EventDriver

Each value change or status change (event) for a process variable within the process control system is assigned a time stamp (date, time). This time stamp is assigned by the control computer (runtime system) using its system time. This type stamp is used when archiving the data.

The *EventDriver* described here is used to evaluate event type process variables on B&R controllers; **the time stamp is created on the respective controller**.

The expansion of the communication level with this driver has the big advantage that time-critical processes on the controller can now be analyzed. It is possible to follow signals in a time pattern of approx. 10 ms and check if e.g. a programmed signal sequence with certain timing is being executed properly during runtime.

The connection to the control computer (communication) is handled exclusively via Ethernet using the TCP/IP protocol. Using this driver does not exclude the usage of other drivers on this controller.

5.1.2 EventDriver operation

The driver works together very closely with the modules on the controller. For optimal use, a short description of the operation of this driver, **which only used read access**, is provided:

The driver inspects a maximum of 2000 PVs per task cycle.

If all PVs change at the same time, the software creates a load of approx. 25 % of the CPU time (CP360). **The system load is less with powerful CPUs**.

If more than these 2000 PVs were checked, it could result in a cycle time overrun because of other tasks that are running at the same time. The driver processes the PV list like a ring buffer due to the other cyclic tasks, i.e. it starts at the beginning again after the end of the list has been reached.

This results in a guaranteed resolution of

T=[(PV_ANZAHL - (MAX_PV -1)) / MAX_PV] * 10 milliseconds

with MAX_PV = 2000

The expression in [....] is a whole number division without remainder, therefore **T** is always a **whole** number.

The driver does not send its event buffer to the control computer (runtime system) cyclically with the latest value change, instead **it always sends it when it has at least 68 entries (80%) or when it has at least one entry and the last send was at least 100 milliseconds earlier**. This results in a runtime delay for data transfer that is a maximum of 100 milliseconds, which is added to the runtime delay for the control computer applications.

The clocks on the controllers and the control computer must be synchronized using standard APROL mechanisms so that the timing for a group of events (values) can be analyzed accurately!

5.1.3 Technical note about the usage on the controller

The **EventDriver** for controllers **uses a buffer system that is comprised of 100 buffers, which can each contain a maximum of 85 event entries**. With this there is the chance of catching a data-loss if a connection fails.

A telegram which is to be sent contains a completely filled event buffer in order to achieve a high data throughput.

The EventDriver can manage a maximum of 4096 event PVs.

An event entry is either created by the EventDriver itself, or by the so-called Event link mechanism (see below) of a task (the EventDriver's task class).

The transfer of event buffers **is solely carried out by the EventDriver**.

The EventDriver sends an event buffer **when**

no buffer has been sent for at least 10 cycles, and the buffer contains at least one entry,

Either one telegram can be received per task cycle from the control computer, or one buffer containing change events sent, in order to keep the system load low. A maximum of 1000 value checks are carried out in this case. If there is no communication, then all of the 2000 checks are carried out.

As each telegram which the control driver sends must be confirmed in the control computer by the EventDriver, and a simultaneous receiving and sending **is not possible**, there is a resulting steady load of 85/2 events, i.e. 42 events per task cycle. The sum of the EventDriver and EventLink events are contained in this number!

If more than 42 events are continuously created, then a data-loss occurs due to buffer overflows.

5.1.4 Schematic overview for EventDriver

The driver's buffer system consists of three queues.

- The **first** (**Empty queue**) contains all of the currently unused and partial filled event buffers.
- The **second** (**send queue**) contains all of the completely filled, but not yet sent event buffers. These should be sent as quickly as possible to the control computer.
- The **third** (**Unconfirmed queue**) contains all of the event buffers which have been sent to the control computer, but have not been acknowledged by it.

The accumulating data is written to the next free event buffer. If this buffer is full it is removed from the free buffers, and moved to the send queue. Data can still be sent in the case that an acknowledgement telegram has not already been processed by the control computer:

The first send buffer is sent when it exists. If the send queue is empty then the first partly filled buffer can be sent. Pre-requisition for this is that it has a content of at least 80 percent, or that no data have been sent for a period of > 100 ms.

A sent buffer is moved to the 'unconfirmed queue'. It stays there until the confirmation from the control computer arrives. It is then released for re-use afterwards.

If the connection to the control computer is lost in the meantime then all of the 'unconfirmed buffers' are transferred again after a reconnection. In this way a loss of data should be avoided. These data are eventually sent to a redundancy driver, if it connects to the controller because of the APROL redundancy mechanism.

If new events are created and there is no free buffer then the oldest 'unconfirmed buffer' is used. In certain circumstances, this buffer's data may have already been transferred to the control system but not yet acknowledged.

If there are also no 'unconfirmed buffers' available then the oldest send data is overwritten.

In this case, all 85 entries of the respective buffer are lost! In this case there will be an unavoidable loss of data!

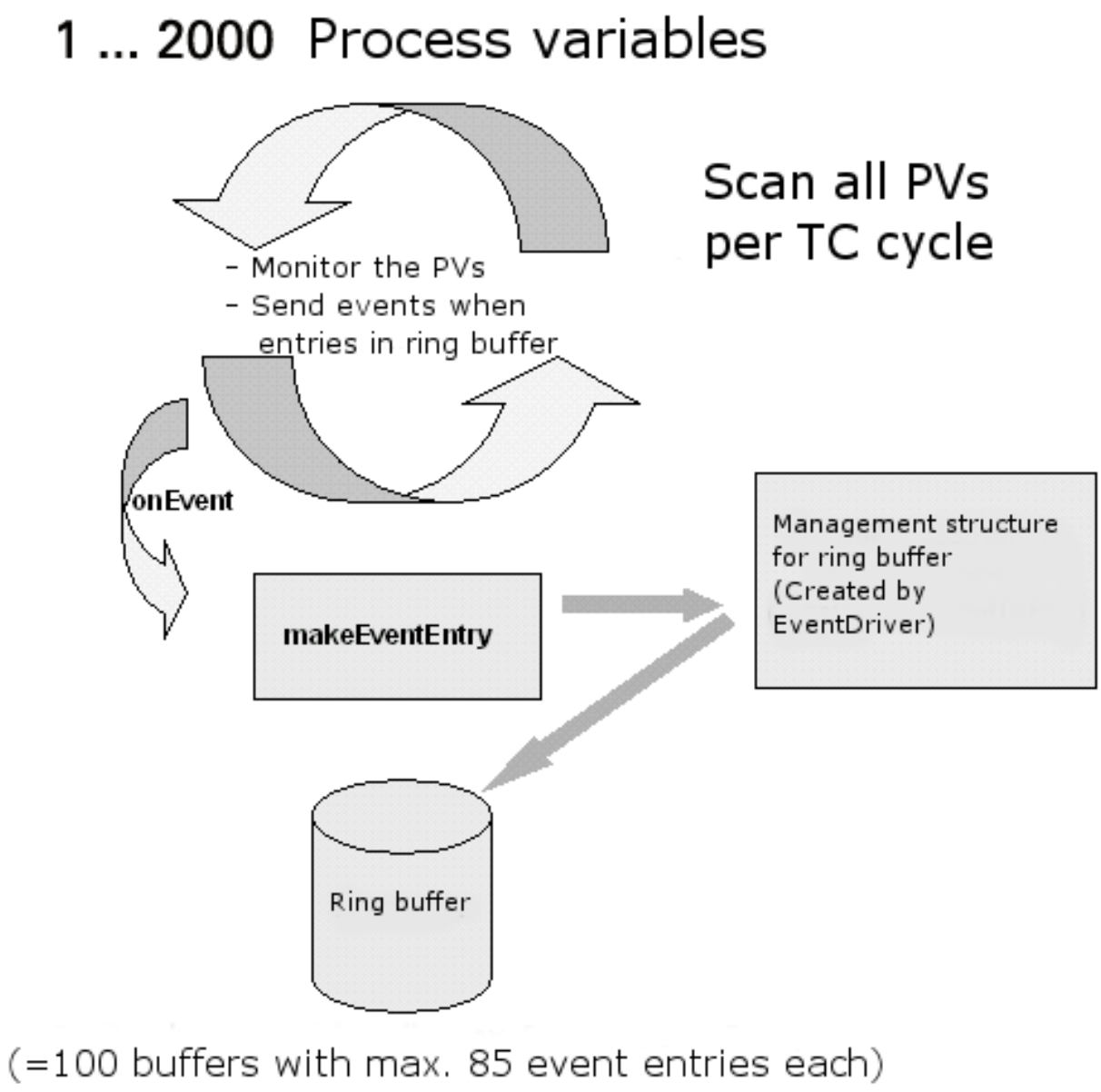


Illustration 14: EventDriver functionality

5.2 Contents of the RPM package

5.2.1 Description of the files included in the package

The RPM package is identified as the 'APROL EventDriver'.

It contains the following described files:

File name	Description

File name	Description
ApDrvEvt.br	The module monitors the process variables on the controller which have event time stamps that are taken from the controller. The controller module is configured by the control computer driver. It runs cyclically in task class 1 (10 milliseconds) on the CPU. In each cycle, the values of the variables are compared with the states from the last cycle. If a minimum deviation from the last value exists, an entry is made in the event buffer. **The event buffer is sent to the control computer if at least 68 entries are in the event buffer, or after 100 milliseconds at the latest**. **A maximum of 2000 variables are compared per task cycle**. If more variables are configured, then it will continue in the next cycle where it left off in the last cycle. It is clear that the timing resolution becomes coarser when more variables have to be compared. The module on the controller works with interlinked lists and requires 32 bytes of memory per process variable. Directory `/opt/aprol/br/aprol/Vxxx/i386/module/`
EventDriver	The control computer driver for receiving process variables from the controller. After establishing a connection, the control computer driver and the controller compare their configurations and create a new configuration is they are different. If the configurations on both sides are identical, the controller can send the contents of its event buffers to the control computer. When a new configuration is made, all existing entries in the event buffer on the controller are lost. Directory: `/opt/aprol/bin/`
libEventDriver.so	Shared library used by the APROL driver on the control computer to communicate with the controller. Directory: `/opt/aprol/lib/`
EventDriver.README	README file with a description of the RPM package. Directory: */opt/aprol/doc/packages/EventDriver/*

The following files – which are not described in detail – are also on the APROL computer after the installation of the software package:

```
/opt/aprol/cnf/EventDriver/examples/ApEvtLink
/opt/aprol/cnf/EventDriver/examples/ApEvtLink/Link.cfg
/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx
/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx/ApDrvLink.c
/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx/ApEvtLink.c
/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx/ApEvtLink.h
/opt/aprol/doc/packages/EventDriver
```

5.3 Configuration

5.3.1 Configuration of the EventDrivers

The following modules (and conditions) are required for operation of the EventDriver:

- ethsock.br
  AR-OS system module
- A valid TCP/IP configuration
- sys_lib.br
  AR-OS system module
- ApDrvEvt.br
  AR-OS system module

If these modules are not installed on the controller, then the individual modules must be transferred to the CPU using the engineering system. The data module must be generated before transferring it to the respective controller!

The ncessary TCP/IP configuration is made with the help of the **CaeManager**.

*In order to test if the TCP/IP connection is operating on the controller without problems, send a **ping** instruction to the corresponding controller address. If the **ping** instruction is not answered, then the configuration of the driver and the hardware used for the connection must be checked!*

5.3.2 Configuration of the driver in the APROL system

The control computer driver for receiving process variables from the controller is **configured** in the scope of the CC modules.

The launching options of the EventDriver can be found in the manual 'X99 CC Modules', chapter Launching Options EventDriver

An event variable type can be of the type 'Hardware I/O'!

To create an event variable, it's necessary to activate the "Event Variable" type when configuring the I/O module!

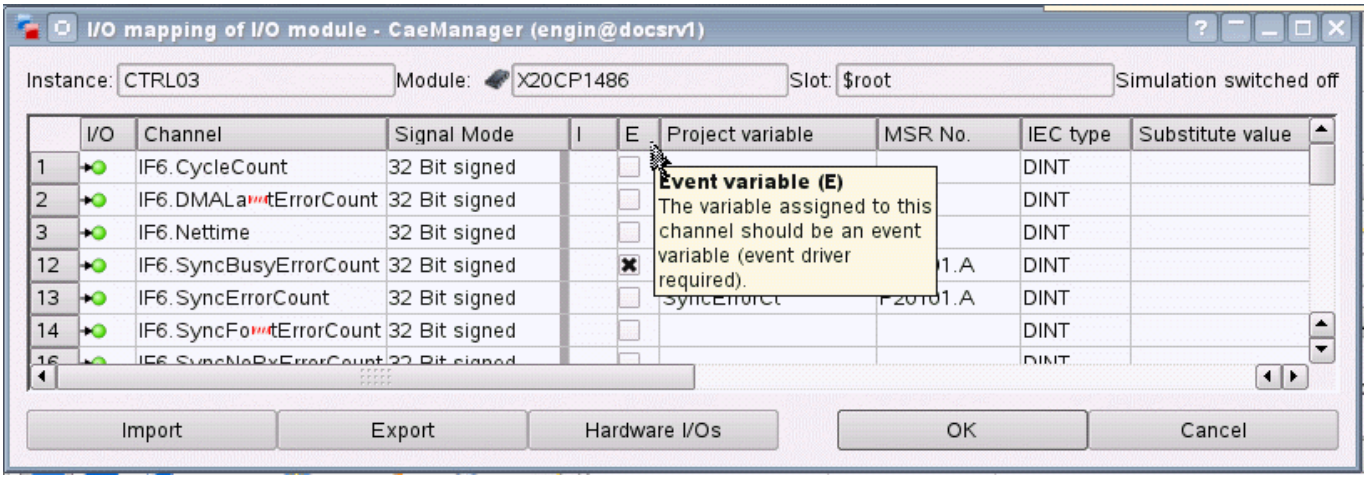


Illustration 15: Creating an event variable

Variables that are recognized as event type process variables are no longer in the configuration file for the cyclic driver during runtime on the runtime system.

The 'event' type process variables that should be transferred must be use used as 'global variables' in the CFCs.

5.4 EventDriver status variables

The names of the EventDriver status variables begin with the prefix 'S2E_'.

The string "*DRIVER_*", together with the "*-controller"* start option entry, is used. For example, if the driver is started with *-controller controller1*, then a variable with the prefix *DRIVER_controller1* is created.

Name of the status variable	**Description**
<Prefix>_connStatus e.g. *DRIVER_PLC1_connStatus*	**Integer type** Value 0: Boot phase of the driver Value 1: Connection not yet established Value 2: Connection to the controller established
<Prefix>_numEventsLost	This variables shows how many events have been lost since the control computer driver was started. The variable is increased with each event that could not be recorded on the controller because of lack of space. The last time that an event was lost can be detected with the changed time stamp of the variable (e.g. with IosEv).
<Prefix>_reduConfigured	Value 0: No redundant IP address configured Value 1: Redundant IP address configured
<Prefix>_connUsed	IP address currently being used (the address is being used even if a connection is not currently active)
<Prefix>_debugFilter	Variables that can be switched on or off using '**pio'** debug outputs. They are output to **stderr** using **fprintf** and, if necessary, can be read from there by redirecting the data. Error outputs are also written to the Syslog container.

5.5 Event variables in external tasks

Please note the information in chapter *Technical notes about the use of controllers.*

5.5.1 Transferring event variables with their own time stamp

The following section describes how event variables are transferred from external tasks to the control computer.

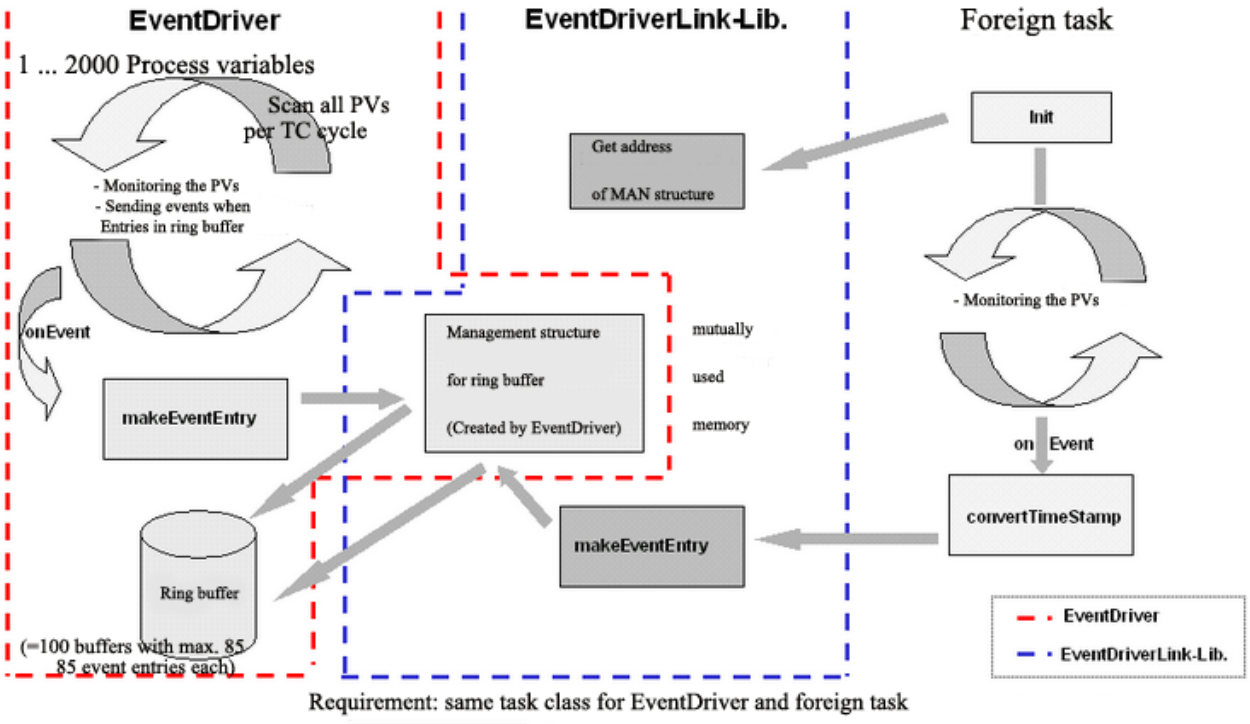


Illustration 16: Schematic diagram of the ApEvtLink module

In the following description, it is necessary that the external task is created with *Automation Studio*.

Controller global variables cannot be used!

In the example, the file

`/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx/ApDrvLink.c`

will be used to clarify the use of the link module.

The files:

`/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx/ApEvtLink.c`
`/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx/ApEvtLink.h`

must be integrated in the external task!

The software package provides the application programming with two routines that must be used in the software created. First, the routine ***ApEvtLink_Initialize*** is used to initialize a local variable with the name ***evtLink*** and type ***T_AP_EVT_LINK*** (also see include file ApEvtLink.h). This routine should be called from the Init SP for the application task and must be repeated until ***evtLink.pEvBufPtr*** has a value other than 0. This is the case when the Init UP has successfully finished processing the APROL event driver. In this way, it is clear that the APROL event driver always has to be loaded to the controller before the application task!

The second routine ***ApEvtLink_MakeEventEntry (DINT pvId, T_PLC_TIME *pPlcTime, UDINT *pValue)*** creates an event entry in the ring buffer for the APROL event driver. The transfer includes a unique variable identification number that must be created by the user and configured in the Gateway editor, the address of a time structure with a time stamp that must be entered by the user as a "local time" and the address of a 4 byte variable that contains the variable value. The user does not have to take the byte order into consideration because the routine places the bytes in the order that is needed. Take note that Define BR_controller_IX86 must be set for Intel CPUs!

The list of the H and C files as well as an example application is also provided.

Because of the locking mechanism used, the external task and EventDriver must run in the same task class.

If these instructions are not adhered to, there may be occurences ranging from errors in the controller memory mangement to a CPU crash.

The transfer of the values to the event driver requires a 4 Byte variable value (Float, INT32, UINT32, DATE), so that a conversion must eventually take place in the user task.

5.5.2 Configuration of the user variables with the Gateway editor

The configuration of the link variables for the APROL event driver consists of a list of variables (one per line) each having four entries:

```
PV_NAME     PV_ID     CONTROLLER_TYP    IEC_TYP
```

The entry **PV_NAME** specifies the name of the PV in the Iosys. The PV is handled by the driver, i.e. writing to the PV on the controller is not possible and also does not make sense!

The **PV_ID** is sent to the control computer driver by the controller driver for identification. The internal ID in the control computer driver is generated from the sum of the LINK_OFFSET (see above) and the configured PV_ID. The sum ID must be given to the routine ApEvtLink_MakeEventEntry, not the PV_ID listed in the configuration file. This method has the advantage that the PV can be easily moved within the user task (e.g. using Define function). Take note that the IDs for APROL event variables are assigned dynamically and therefore it may be necessary to move them if more than 32767 APROL event variables should be used!

The **VARIABLE_TYPE** entry specifies how the variables are to be interpreted by the control computer driver and which Iosys data type should be used. Three variable types can be defined:

- In the Iosys, FLOAT variables use the DOUBLE data type.
- In the Iosys, INT32 variables use the INTEGER data type
- In the Iosys, UINT32 variables use the INTEGER data type.
- DATE on the controller corresponds to a UINT32, is depicted as a data type INTEGER in the Iosys.

The desired IEC data type must be set for IEC_TYP. This ensures that a variable of the name with the corresponding IEC type can be used in the CFC.

This entry does not appear in the configuration file of the driver and cannot be seen in the configuration preview! The driver can carry out a limit value check, i.e. it is possible that a USINT has the value 65535 assigned! It is the responsibility of the user to adhere to the limits!

5.5.3 Testing the software

The configuration can be checked with the *IosEv* tool.

```
IosEv –pv PV_NAME
```

or

```
IosEv –mask PV_NAME
```

shows a corresponding list of variables and incoming value changes including time stamp. When the controller time changes, the time stamp must change as well.

5.5.4 Limitations

The following limitations need to be taken into account:

✔	The event task on the controller is only configured using the variable address and variable type. Event variables always have to be controller global variables. Only global variable addressed can be accessed by the control computer without communication with the controller.
✔	It is currently not possible to define a hysteresis. The minimum value change that generates an event on the controller is permanently set to 0.
✔	One connection is possible per controller. Redundancy or controller cross communication is not implemented.
✔	A check is not made to determine if the memory addresses to be transferred are actually being used on the controller. The values shown are not valid if communication is activated without the respective tasks running on the controller.

5.5.5 ApEvtLink.h

```
#ifndef AP_EVT_LINK_H
#define AP_EVT_LINK_H

#ifndef PLCTASK
#define PLCTASK
typedef void * PTR;
#endif

#include <PccEventDriverTypes.h>
#include <PccEventDriverLib.h>

#ifndef BR_PLC_IX86
#ifndef BR_PLC_M68K
#error  Missing target information - define either BR_PLC_IX86 or BR_PLC_M68K
#endif
#endif

#ifdef BR_PLC_IX86
#define swapLong(x)  ((((x)<< 24 ) & 0xFF000000 ) |   \
                      (((x)<<  8 ) & 0x00FF0000 ) |   \
                      (((x)>>  8 ) & 0x0000FF00 ) |   \
                      (((x)>> 24 ) & 0x000000FF ))

#define swapShort(x) ((( x << 8 ) & 0xFF00 )      |   \
                      (( x >> 8 ) & 0x00FF ))

#define ntohs htons
#define ntohl htonl
#define htonl(x) swapLong(x)
#define htons(x) swapShort(x)

#define T_TIME_ENTRY  T_PLC_TIME

#else

#define ntohs(x) x
#define ntohl(x) x
#define htons(x) x
#define htonl(x) x

#define T_TIME_ENTRY  T_PLC_TIME_SG3

#endif

typedef struct
{
    T_PLC_EVENT_BUFFER **pEventBufferPtr;
    T_PLC_EVENT_BUFFER **pSendQueuePtr;
    UINT               *pSendQueueLen;
    T_PLC_EVENT_BUFFER **pFreeBuffersPtr;
    T_PLC_EVENT_BUFFER **pUnconfirmedPtr;

    UINT            *pNumEventsLostPtr;

    BOOL            isInitialized;
    UDINT           lastError;
    UINT            maxEventEntries;

} T_AP_EVT_LINK;

#define AP_EVT_LINK_ERR_BASE       50000
enum
{
  AP_EVT_LINK_NO_ERR = AP_EVT_LINK_ERR_BASE,
  AP_EVT_LINK_SIZE_ERR,
  AP_EVT_LINK_CONNECT_FAILED,
};

/* Funktions-Prototypen */
BOOL ApEvtLink_Initialize ( void );
UINT ApEvtLink_ConnectPv ( UDINT *varAddr, char *varName, UDINT expectedLen );
void ApEvtLink_MakeEventEntry ( DINT pvId, T_TIME_ENTRY *pPlcTime, UDINT *pValue );

#endif  /* AP_EVT_LINK_H */
```

5.5.6 ApEvtLink.c

```
#include <plc.h>
#include <plctypes.h>

#ifdef APROL
#include <iectypes.h>
#endif

#include <sys_lib.h>

#ifdef AS_PROJECT                   /* Set define AS_PROJECT if you are using an Automation Studio project.
                                       Disable this define if you are using this file as an APROL CaeManager
                                       library import */
#include <ApEvtLink.h>
#endif

static T_AP_EVT_LINK evtLink;       /* This variable is only visible in this library. It is not visible
                                       in a controller watch! */

BOOL ApEvtLink_Initialize ( void )  /* Should be called from the INIT section of the task / function block */
{
    BOOL initializeSuccess;

    evtLink.isInitialized = 0;

    initializeSuccess = ( ApEvtLink_ConnectPv ((UDINT *) &evtLink.pSendQueuePtr,
                          "ApDrvEvt:pSendQueue", 4 ) == AP_EVT_LINK_NO_ERR );
    if ( initializeSuccess )
    {
        initializeSuccess = ( ApEvtLink_ConnectPv ((UDINT *) &evtLink.pSendQueueLen,
                              "ApDrvEvt:sendQueueLen", 2 ) == AP_EVT_LINK_NO_ERR );
    }

    if ( initializeSuccess )
    {
        initializeSuccess = ( ApEvtLink_ConnectPv ((UDINT *) &evtLink.pEventBufferPtr,
                              "ApDrvEvt:pEventBuffer", 4 ) == AP_EVT_LINK_NO_ERR );
    }

    if ( initializeSuccess )
    {
        initializeSuccess = ( ApEvtLink_ConnectPv ((UDINT *) &evtLink.pFreeBuffersPtr,
                              "ApDrvEvt:pFreeBuffers", 4 ) == AP_EVT_LINK_NO_ERR );
    }

    if ( initializeSuccess )
    {
        initializeSuccess = ( ApEvtLink_ConnectPv ((UDINT *) &evtLink.pUnconfirmedPtr,
                              "ApDrvEvt:pUnconfirmed", 4 ) == AP_EVT_LINK_NO_ERR );
    }

    if ( initializeSuccess )
    {
        initializeSuccess = ( ApEvtLink_ConnectPv ((UDINT *) &evtLink.pNumEventsLostPtr,
                              "ApDrvEvt:pccEventDriverNumEventsLost", 2 ) == AP_EVT_LINK_NO_ERR );
    }

    if ( initializeSuccess )
    {
        evtLink.maxEventEntries = MAX_SEND_EVENTS;
        evtLink.isInitialized = 1;
    }

    /* Must be called again from CYCLIC if return value is FALSE */
    return initializeSuccess;
}

UINT ApEvtLink_ConnectPv ( UDINT *varAddr, char *varName, UDINT expectedLen )
{
    /* This routine gets the address of variables from ApDrvEvt and checks
     * if len is as expected
     */

    UDINT pvLen;

    evtLink.lastError = PV_xgetadr ( varName, varAddr, &pvLen );
    if ( evtLink.lastError == 0 )
    {
        if ( pvLen == expectedLen )
            return AP_EVT_LINK_NO_ERR;        /* Got address and len check successful */
        else
        {                                     /* Fails due to different length */
            evtLink.lastError = AP_EVT_LINK_SIZE_ERR;
            return AP_EVT_LINK_SIZE_ERR;
        }
    }
                                              /* Variable not found. Is ApDrvEvt already
                                               * on target?
                                               */
    return AP_EVT_LINK_CONNECT_FAILED;
}

void ApEvtLink_MakeEventEntry ( DINT pvId, T_TIME_ENTRY *pPlcTime, UDINT *pValue )
```

```
{
    /* This routine makes an event entry in ApDrvEvt's event buffer and updates
     * it's internal management variables
     */

    if ( evtLink.isInitialized )
    {
        T_PLC_EVENT_BUFFER *pEvBuffer = *(evtLink.pEventBufferPtr);
        int numEventsLost = 0;
        int numEvents = 0;

        if ( ! pEvBuffer )
        {
            if ( *(evtLink.pFreeBuffersPtr))
            {
                pEvBuffer = *(evtLink.pFreeBuffersPtr);
                *(evtLink.pFreeBuffersPtr) = pEvBuffer->next;
            }
            else if ( *(evtLink.pUnconfirmedPtr))
            {
                pEvBuffer = *(evtLink.pUnconfirmedPtr);
                *(evtLink.pUnconfirmedPtr) = pEvBuffer->next;
                numEventsLost += ntohs (((T_EVENT_IND *) &pEvBuffer->data[HDR_SIZE])->numEvents );
            }
            else if ( *(evtLink.pSendQueuePtr))
            {
                pEvBuffer = *(evtLink.pSendQueuePtr);
                *(evtLink.pSendQueuePtr) = pEvBuffer->next;
                numEventsLost += ntohs (((T_EVENT_IND *) &pEvBuffer->data[HDR_SIZE])->numEvents );
            }
            else
                numEventsLost++;

            if ( pEvBuffer )
            {
                if ( pEvBuffer->next )
                    pEvBuffer->next->prev = NULL;

                pEvBuffer->next = 0;
                pEvBuffer->prev = 0;

                ((T_EVENT_IND *) &pEvBuffer->data[HDR_SIZE])->numEvents = 0;
                *(evtLink.pEventBufferPtr) = pEvBuffer;
            }
        }
        else
            numEvents = ntohs (((T_EVENT_IND *) &pEvBuffer->data[HDR_SIZE])->numEvents );

        if ( numEventsLost )
            *(evtLink.pNumEventsLostPtr) = numEventsLost + *(evtLink.pNumEventsLostPtr);

        if ( pEvBuffer )
        {
            T_EVENT_IND *pInd = (T_EVENT_IND *) &pEvBuffer->data[HDR_SIZE];
            T_EVENT_ENTRY *pEntry = (T_EVENT_ENTRY *) &pEvBuffer->data[HDR_SIZE+sizeof ( T_EVENT_IND)];

            pEntry+= numEvents;

            pEntry->pvId = htonl ( pvId );
#ifdef BR_PLC_IX86 /* SG4 */
            pEntry->time.tv_sec = htonl ( pPlcTime->tv_sec );
            pEntry->time.tv_usec = htonl ( pPlcTime->tv_usec );

#else /* SG3 */

            memcpy ( &pEntry->time, pPlcTime, sizeof ( T_TIME_ENTRY ));
            memcpy ( &pEntry->value, pValue, 4 );
#endif

            *((unsigned int *) pEntry->value) = htonl ( *((unsigned int *) pValue ));
            pInd->numEvents = htons (( 1 + numEvents ));

            if (( 1 + numEvents ) == MAX_SEND_EVENTS )
            {
                T_PLC_EVENT_BUFFER *pSend = *(evtLink.pSendQueuePtr);
                *(evtLink.pEventBufferPtr) = 0;
                if ( pSend )
                {
                    while ( pSend->next )
                        pSend = pSend->next;

                    pSend->next = pEvBuffer;
                    pEvBuffer->prev = pSend;
                }
                else
                {
                    *(evtLink.pSendQueuePtr) = pEvBuffer;
                }

                *(evtLink.pSendQueueLen) = 1 + *(evtLink.pSendQueueLen);
            }
        }
    }
}
```

5.5.7 ApDrvLink.c

```
#include <plc.h>
#include <plctypes.h>

#ifdef APROL
#include <iectypes.h>
#endif

#include <sys_lib.h>

#define PLCTASK
typedef void * PTR;

#include <ApEvtLink.h>

#ifdef BR_PLC_IX86
#include <astime.h>
#endif

#ifdef  LOG_ERRORS
#define LOG_ERROR(line, format, args...)              \
          {                                           \
            char buffer[80];                          \
            sprintf ( buffer, format, args );         \
            ERRxwarning ( 50000, line, buffer );      \
          }
#else
#define LOG_ERROR(line, format, args...);
#endif

_LOCAL UINT myTaskCounter;
_LOCAL UINT numEventsPerLoop;
_LOCAL UINT myLinkOffset;
_LOCAL BOOL enabled;
_LOCAL BOOL initialized;

_INIT void ApEvtLink_Init ( void )
{
  myTaskCounter = 0;
  numEventsPerLoop = 25;
  myLinkOffset = 50000;

  enabled = 0;
  initialized = 0;

  initialized = ApEvtLink_Initialize ();
  LOG_ERROR(__LINE__, "isInitialized = %d", evtLink.isInitialized );
}

_EXIT void ApEvtLink_Exit ( void )
{
}

_CYCLIC void ApEvtLink_Cyclic ( void )
{
  if ( ! initialized )
    ST_tmp_suspend ( 0 );

  if ( enabled )   /* Set enabled true in watch to create event entries */
  {
      if (( myTaskCounter = (++myTaskCounter % 100 )) == 0 )
      {
        UDINT value;
        int i;

#ifdef BR_PLC_IX86
        DTStructure utcTime;
        UtcDTStructureGetTime_typ UtcDTStructureGetTime_Var;
        T_PLC_TIME plcTime;

        UtcDTStructureGetTime_Var.pDTStructure = (UDINT) &utcTime;
        UtcDTStructureGetTime_Var.enable = 1;
        UtcDTStructureGetTime(&UtcDTStructureGetTime_Var);

        plcTime.tv_sec = htonl ( DTStructure_TO_DT ((unsigned long) &utcTime ));
        plcTime.tv_usec = htonl ( utcTime.millisec );

        value = plcTime.tv_usec;
#else
        T_PLC_TIME_SG3 plcTime;
        RTCtime_typ currentTime;

        RTC_gettime(&currentTime);
        plcTime.year         = currentTime.year - 1970;
        plcTime.mon          = currentTime.month;
        plcTime.day          = currentTime.day;
        plcTime.hour         = currentTime.hour;
        plcTime.min          = currentTime.minute;
        plcTime.sec          = currentTime.second;
        plcTime.milliseconds = htons ( currentTime.millisec );

        value = plcTime.milliseconds;
```

```
#endif


        for ( i=0; i<numEventsPerLoop; i++ )
        {
          value = myLinkOffset+i;

          if ( i == 2 || i == 5 )
          {
            float f = myLinkOffset + i;
            memcpy ( &value, &f, 4 );
          }

          ApEvtLink_MakeEventEntry ( myLinkOffset+i, &plcTime, &value );
        }
      }
  }
}
```

5.5.8 Example configuration for the APROL EventDriver

```
#
#
# PvId      VarName       VarType
#
# 0x0000    TestVar0      INT32
# 0x0001    TestVar1      UINT32
# 0x0002    TestVar2      FLOAT
#
#
# PvIds are used internally together with linkOffset,
# so that PvId = PvId + linkOffset is used internally!!!
#
# VarName specifies the PV names in the Iosys.
#
# VarType is used for evaluating values, casting may be necessary on the controller!!!
# Valid VarTypes are FLOAT, INT32 and UINT32. In the Iosys, they are
# FLOAT, INT and FLOAT, (internally, FLOAT is type double) because only
# double variables can represent the full value range for UINT32.
#
##############################################################################

0x0000      TestVar0      INT32
0x0001      TestVar1      UINT32
0x0002      TestVar2      FLOAT
0x0003      TestVar3      INT32
0x0004      TestVar4      UINT32
0x0005      TestVar5      FLOAT
0x0006      TestVar6      INT32
0x0007      TestVar7      INT32
0x0008      TestVar8      FLOAT
0x0009      TestVar9      INT32
0x000A      TestVar10     INT32
0x000B      TestVar11     FLOAT
0x000C      TestVar12     INT32
0x000D      TestVar13     INT32
0x000E      TestVar14     FLOAT
0x000F      TestVar15     INT32
0x0010      TestVar16     INT32
0x0011      TestVar17     FLOAT
0x0012      TestVar18     INT32
0x0013      TestVar19     INT32
0x0014      TestVar20     FLOAT
0x0015      TestVar21     INT32
0x0016      TestVar22     INT32
0x0017      TestVar23     FLOAT
0x0018      TestVar24     INT32
0x0019      TestVar25     INT32
0x001A      TestVar26     FLOAT
0x001B      TestVar27     INT32
0x001C      TestVar28     INT32
0x001D      TestVar29     FLOAT
0x001E      TestVar30     INT32
```

The PvId can be entered in hexadecimal (see example) and also in decimal!

5.5.9 Creating a task with Automation Studio

The short overview in the following section explains how to create a task with *Automation Studio* (Version 2.3) using the example test file *ApDrvLink.c*:

- Create a new project in *Automation Studio* with the name *ApDrvLnk*.
- Create a new cyclic task (**TC1**) after uploading the hardware.
- Add the files *ApDrvLink.c*, *ApEvtLink.c* and *ApEvtLink.h*.
- Add the library *sys_lib* as well as the files *sys_lib.h* and *sys_lib.a*.
- If you are using an Intel CPU:
  In the compiler options under "**Project/Settings**", the *DBR_controller_IX86* setting must be added.
- Add the files *controller.h* and *controllertypes.h* from your *Automation Studio* environment.
- Select "**Build**" (F7) to create the cyclic task.
- Replace the file *ApDrvLink.c* with your application and add the necessary calls.

1. ApEvtLink_Initialize( ) should not be called in the INIT_SP in order to guarantee that the EventDriver is finished with its initialization.
*2. Call this routine repeatedly in the cyclic section until all values in the structure have a value "**not equal to 0**"!*

5.5.10 Using the driver with a C block

Transferring data with a time stamp that has been set explicitly is also possible via a C block from a CAE library. In this case, the same basic requirements and limitations apply as described in the previous chapter.

The above mentioned *ApEvtLink.h* and *ApEvtLink.c* files are integrated as library-global project parts in a library in order to allow a library block to address the necessary driver routines. For this, proceed as follows:

- Eventually create a new library group, e.g. with the name 'GlobalLibs', in the CaeManager
- Select the group and choose
  File => New => Library-Global Header
  from the menu. Give the new project part a name, e.g.
  '**GlobalEvtLink_H**'
- Activate the 'Code for the controller' in the project part's master data; deactivate 'Code for the control computer'
- Select the 'Program' tab. Select 'Import from file' from the context menu (right mouse-click in the editor window)
- Navigate to
  `/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx`
  in the file selection dialog, and choose the file '*ApEvtLink.h*' to be 'opened'. Its content is then imported into the new project part.
- Save the created library-global header module and compile it.

The integration of the driver C routine is carried out in the same way:

- Select the 'GlobalLibs' group and choose
  File => New => C Module
  from the menu. Give the new project part a name, e.g.
  '**GlobalEvtLink_C**'
- Activate the 'Code for the controller' in the project part's master data; deactivate 'Code for the control computer'
- Select the 'Program' tab. Select 'Import from file' from the context menu (right mouse-click in the left window) of the 'C Code' editor area.
- Navigate to
  `/opt/aprol/cnf/EventDriver/examples/ApEvtLink/V02xx`
  in the file selection dialog, and choose the file '*ApEvtLink.c*' to be 'opened'. Its content is then imported into the new project part.
- Complete the following lines in the 'Header' area (right editor window):
  #include "GlobalEvtLink_H.h"
- Save the created library-global C module and compile it.

The routines for using the driver for the library's C function blocks are now available.

The basic structure of the program code of a C block is similar to that of the above mentioned *ApDrvLink.c* file. A function block for use on the B&R SG4 platform is created in the following manner.

Select the **'Declarations'** tab after creating the block (only for a controller, master data) and create the following entries:

Includes & Defines for the local header file
#include <astime.h> #include <asarcfg.h>

Includes & Defines for the local C file
#include "GlobalEvtLink_C.h"

Local variables of the block instance		
Name	**Category**	**Data type**
LinkInit	IEC type	BOOL
UtcDTStructureGetTime_Obj	IEC structure	UtcDTStructureGetTime
utcTime	IEC structure	DTStructure

Further column values for all entries:

Usage : only CTRL

Remanent : OUT

Default value : 0

Dimension :

The driver is initialized in the **'Init Code'** tab:

Init Code
LinkInit = ApEvtLink_Initialize(void); return (0);

The basic structure of the C code may look as follows:

```
REAL realVal, nullVal = 0.0;                /* Example declaration of a user variable
                                               Declare your desired variables instead
                                            */
T_PLC_TIME plcTime;

if ( ! LinkInit )
{
                                            ST_tmp_suspend ( 0 );    /* Stop task if driver is not initialized
                                               Alternatively you can try to reinitialize with
                                               LinkInit = ApEvtLink_Initialize( void );
                                            */
}
else
{
                                            /* Place your program code here */

                                            /* Example to read the current time of the controller */

                                            UtcDTStructureGetTime_Obj.pDTStructure = (UDINT) &utcTime;
                                            UtcDTStructureGetTime_Obj.enable = 1;
                                            UtcDTStructureGetTime( &UtcDTStructureGetTime_Obj );

                                            /* Converts a DTStructure to seconds since 1970-01-01 (UTC) */
                                            plcTime.tv_sec = DTStructure_TO_DT ( (unsigned long) &utcTime );
                                            plcTime.tv_usec = utcTime.millisec * 1000;

                                            /* Example to write a value with timestamp to the event link buffer */

                                            ApEvtLink_MakeEventEntry ( VarOffset + VarID, &plcTime, (UDINT *) &realVal );

}
```

The two variables '*VarOffset*' and '*VarID*' that are marked in color are configuration data that are held as best as block pins:

Block inputs		
Name	**Data type**	**Default value**
VarOffset	UINT	50000
VarID	UINT	

'The default value for the variable offsets must be the same as the '-linkOffset' parameter of the driver instance in the control system catalog. Furthermore, the '-useLinkDir' driver parameter must be activated; its default value is "." (point).

The driver does not address variables with their name, but with their configured ID in the 'APROL couplings / EventDriver / <Controller instance>':

List of PVs			
PV name	**ID of the system var.**	**Data Type (controller)**	**IEC Data type**
Temperature	0	FLOAT	LREAL
Printing	1	UINT32	UINT

If you want to address the 'Druck' variable, the 'VarID' in the block code is '1'.

The IEC data type is relevant for use in the **CFC**. You can access the PV with the name 'Druck' from the list of gateway variables on the input border of a CFC; the data type there is REAL.

You will only obtain the explicitly set time stamp of the variable event by accessing the border directly. Each block that is placed in between is an allocation to a task, and results in the local time stamp!

The CFC, in which a block for the use of the Event link mode is placed, must be allocated to a controller task with the class 1, parallel to the 'ApDrvEvt' driver.

6 InaDriver

6.1 General information about the InaDriver

APROL drivers convert data types when instance transitions take place, if necessary automatically. The data type and respective size depends on the respective control computer / controller.

The values transferred are limited to the minimum, or maximum values of the corresponding target system.

Please take note that there can be a limitation in the value range, or the accuracy.

*An overview page for the IEC types can be reached in the **CaeManager** with the menu item '**Help/IEC type info**'!*

***Different data sizes** between control computer and controller **are marked in the help page with a yellow triangle**.*

The software package works with various I/O modules (e.g. Profibus cards, Ethernet cards, serial interfaces). This documentation provides a description of the hardware, the installation and configuration routines in the *"InaDriver*" software package for APROL running in LINUX and also notes concerning various additional packages. The package consists of the kernel drivers that handle access of the hardware, various utilities for configuration and network analysis and also the APROL driver for communication with the I/O modules. The installation of the software, the configuration of the hardware and the network parameters for a few selected network settings are also described. This can be used as an aid when solving problems if errors occur.

The program package requires a LINUX kernel with Version 2.2.0 or higher. The kernel must be compiled and started with the option "enable loadable module support" and without the option "set version information on all symbols for modules". Please check your SuSE LINUX documentation to see how a corresponding LINUX kernel is structured.

Please read the information about process bus redundancy in chapter** Process bus redundancy for Ethernet connections **in this manual!

6.2 Description of the PC hardware

6.2.1 Softing PROFIboard

One I/O module that can be used is the PROFIBUS card from ***Softing***. This is an ISA card that requires a 4 byte IO area, a free interrupt line as well as a 16 KB address area in the first MB on your PC in order to operate. The device driver supports parallel operation of two PROFIBUS

cards on a PC that can share a common interrupt. A detailed specification of the hardware can be found in the ***PROFIBUS FMS for softing PROFIboards*** section.

6.2.2 B&R PC Profibus card

The second PROFIBUS card to be used is a B&R company product. This is an ISA card that requires a 16 KB address area in the first Megabyte of your PC in order to operate. The device driver supports parallel operation of two PROFIBUS cards. A detailed specification of the hardware can be found in the ***APROL Profibus Package for B&R Hardware*** documentation.

6.2.3 PC Ethernet card

Any PC Ethernet card that can run in Linux can be used for the *InaDriver;* it should be installed and configured according to the Linux installation manual.

An overview of the hardware types that are supported can be found in the SuSE installation manual. If Ethernet is correctly detected by the kernel, corresponding messages are generated when the system is booted.

For example:

eth0: Digital DS21143 Tulip rev 65 at 0xa800, 00:00:1C:B5:F4:D1, IRQ 10.
eth0: EEPROM default media type Autosense.
eth0: Index #0 - Media MII (#11) described by a 21142 MII PHY (3) block.
eth0: MII transceiver #5 config 3000 status 7829 advertising 01e1.

After successfully configuring the kernel, the parameters for this Ethernet device can be set using the ***YaST*** program. After the parameters have been set for the device, the ***ifconfig*** program returns the configuration of the Ethernet device.

For example:

eth0 Link encap:Ethernet HWaddr 00:00:1C:B5:F4:D1
inet addr:192.168.2.35 Bcast:192.168.2.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:15041342 errors:0 dropped:0 overruns:0 frame:0
TX packets:15484661 errors:0 dropped:0 overruns:0 carrier:0
collisions:1705 txqueuelen:100
Interrupt:10 Base address:0xa800

6.2.4 Serial interfaces

In order to use a serial interface for the *InaDriver*, it must be detected and supported by the Linux system. It is important that other processes (e.g. modem driver) do not access this interface. If the kernel supports the serial interface, which is normally the case, then the operating system generates for example the following message when booting:

ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A

Then the serial interfaces ttyS00 and ttyS01 are available on the PC (COM1 and COM2 in DOS). Setting the parameters for the interfaces is described in the installation manual.

6.3 Installation of the PC software

The software is installed using the ***YaST*** installation tool for the SuSE Linux version. Now insert the APROL installation CD in your CD drive and choose the "settings for installation" menu item after starting the ***YaST*** installation tool. The software is installed using the ***YaST*** installation tool for the SuSE Linux version. If you are installing via FTP or NFS, please check the SuSE installation manual.

To install the software, go to "Define/start installation" and select the "Change/create configuration" menu item.

One of the following packages must be installed depending on the I/O module used:

InaDriver as serial driver:	Series 5dr - InaDriver
InaDriver as Ethernet driver:	Series 5dr - InaDriver
InaDriver as Profibus driver with B&R PC card:	Series 5dr - InaDriver and BuRProfiboard
InaDriver as Profibus driver with SOFTING Profiboard:	Series 5dr - InaDriver and PROFIboard

If you have decided to use the driver via Profibus, you will be asked to make some system settings after the installation (see respective documentation for detailed information).

6.4 Configuration of the controller hardware

6.4.1 System settings

In the controller operating system, corresponding configuration entries must be made for the various 2000 series hardware modules. Operating system (controller software) version 2.20 or higher is required.

In the system configuration, the following entries must be checked and adjusted if necessary:

- Connections:
  Number of parallel connections (sum of client and server connections)
- Communication Channels :
  5 + number of parallel connections + 1 for each PB card being used.
- Interfaces:
  Each InaDriver interface must be enabled in the system settings for the communication, with the exception of the online interface on the CPU, which is always enabled.

Example:.
Ethernet interface IF2 on the interface module for a CPU is enabled as follows:

Interface=IF2; Typ: Ethernet; Slot: Local; Subslot: SS1.

Ethernet interface IF4 on the interface module is enabled as follows:

Interface=IF4; Typ: Ethernet; Slot: Local; Subslot: no SS.

6.5 Installation of the controller software

6.5.1 Description of the individual modules

In order for the controller to operate as an InaDriver server, the corresponding modules must be "burned" on the CPU according to the application (serial, Profibus, Ethernet).

Ethernet:

- tcpipcfg.br
  User configuration (data module); contains the IP address, network mask, etc.
- fbtpip.br
  Routing module for INA communication
- tcpipmgr.br
  TCP/IP protocol software

Profibus:

- fbpb.br
  Routing module for INA communication

Serial:

- No additional modules needed

6.6 Description of the utilities (InaDriver)

6.6.1 InaCmd

The ***InaCmd*** utility is used to perform actions with the help of the APROL driver described in chapter Configuration of the APROL driver. This tool establishes a connection to the APROL driver via the Iosys and uses the connection made by the APROL driver to transmit INA services. For this to work, the Runtime system must be fully configured and already started. One advantage is that additional connections to the controller are not needed and the Runtime system does not have to be stopped when using serial communication so that another tool can establish a connection to the controller via the serial interface.

The program is required for performing the download to the controller in the APROL system. The Iosys environment variable must be set to a valid value (Iosys="ComputerName":"IosysPort") so that the program can connect to the APROL driver's Iosys.

The following section offers a brief description of the individual services and options.

The following options are supported:

Option	Short description	Description
-help	Shows help info	

-driver DRV	Connects to the DRV driver	The program attempts to establish a connection with the DRV driver. DRV is the value used to call the driver when using the switch –n.
-download FILE	Controller task download	The program loads a controller task to the CPU connected to the APROL driver. Generally, the entire path plus file name must be entered for FILE, because the APROL driver opens the BR file. The BR file must be located on a file system which the driver can access directly.
-burn	"Burning" the controller task	If this option is used, then the controller task is saved in the USER_FLASH (cold restart safe), otherwise the task is saved in the USER_RAM, which means that it is battery buffered but is deleted at cold restart.
-init MODE	Resets the CPU	The program executes a reset command. Possible reset modes: NORMAL: Warm restart (restart) TOTAL: Cold restart (RAM deleted, restart) DIAGNOSE: Diagnostics start (RAM deleted, restart in diagnose mode, no user applications are started).
-dom_info MODULE	Module info for a controller task	A variety of information is provided about type, size and other properties of the controller module.
-delete MODULE	Deletes the module	MODULE is deleted from the CPU.
-stop_drv MODULE	Stops module communication	The APROL driver stops communication for the MODULE controller task.
-start_drv MODULE	Starts module communication	The APROL driver starts communication for the MODULE controller task, if contained in the driver configuration.
-meminfo	Shows the memory usage info for the CPU	Depending on the type of memory, this information includes the start address, total length and available length.

Example of module info:

```
MODULE INFO:

1.  DOM INDEX / STATE:          0x00000723 /        6
2.  PI  INDEX / STATE / COUNT:  0x00000581 /        3 / 1
```

```
3.  ADDRESS / LENGTH:           0x0024ed94 /    80872
4.  NAME:                       ST_ALLG
5.  VERSION                     0.000
6.  CREATION DATE:              31.07.2000 12:02:38
7.  MODIFICATION DATE:
8.  CREATED BY:
9.  MODIFIED BY:
10. TASK CLASS:                 3
11. INSTALL NO:                 128
12. PV INDEX:                   0x0c00
13. PV COUNT:                   1376
14. LOCAL ANA-ADDR:             0x00f17230
15. LOCAL DIG-ADDR:             0x00000000
16. MEM LOCATION:               3
17. MODULE TYPE:                17


Description:
1. Domain invocation index (module) with status  (0:NON_EXISTENT 1:EXISTENT
   2:LOADING 3:INCOMPLETE 4:COMPLETE 5:READY 6:IN_USE )
2. Program invocation index (task) with status (0:NON_EXISTENT
   1:UNRUNNABLE 2:IDLE 3:RUNNING 4:STOPPED 5:STARTING 6:STOPPING 7:RESUMING
   8:RESETTING) and program invocation with value>0 when domain is used
3. Physical start address of the module and length of the module in bytes
4. Module name
5. Module version
6. Creation date
7. Name of the person who programmed the module (optional)
8. Name of person making changes (optional)
9. Date when changes were made (optional)
10.Task class (0:NOT_VALID 1:TC1 2:TC2 3:TC3 4:TC4 17:SS1 18:SS2 19:SS3
   20:SS4 21:INTERRUPT 22:EXCEPTION)
11.Start sequence within a task class (optional)
12.Start index for PV - Table for this module
13.Number of PVs in this module
14.Start address of local analog memory
15.Start address of local digital memory
16.Location where the module is saved (2:USER_FLASH 3:USER_RAM 4:MEMCARD 5:FIXRAM
   65:DRAM)
17.Modultyp (u.a. 17: controller task 28 : system task module 22: exception task 65:
   data module)
```

6.6.2 CfgInaDriver

The ***CfgInaDriver*** utility combines many of the functions of the ***InaCmd*** utility in a menu-guided interface. The ***CfgInaDriver*** requires the ***ncurses*** package from Linux. This usually comes already installed.

The ***CfgInaDriver*** is controlled by selecting menu items instead of using start options. A base menu is provided where the individual commands can be selected. This works the same way as the ***InaCmd*** tool. A connection is established to the APROL driver and the services are delivered via this program. After starting the ***CfgInaDriver*** tool, the user is asked to enter the name of the APROL driver. The name of the APROL driver is entered here the same as the *–driver* switch in the ***InaCmd*** program.

The following section offers a brief description of the individual services:

Menu item	Description
RESET controller	Triggers a reset on the CPU connected to the APROL driver. NORMAL (warm restart), TOTAL (cold restart) and DIAGNOSE are the possible reset modes.
UPLOAD TASK	Uploads a controller module. The module that should to be processed must be specified, and the filename must be entered for the file where the module will be saved on the hard drive (the file is saved in the driver's working directory if the file name is entered without a directory).

DOWNLOAD TASK	Loads a controller task to the CPU connected to the APROL driver. Generally, the entire path plus file name must be entered for FILE, because the APROL driver opens the BR file. The BR file must be located on a file system which the driver can access directly. You can also define whether or not to save the module (i.e. it is saved in the USER_FLASH by setting the BURN item). Otherwise, it is placed in the USER_RAM. The domain and program index is shown after a successful action.
READ LOGBOOK	Reads the previous 10 entries from the CPU logbook. Error number, error information, the affected task and the event date are output. If available, the corresponding error text is also displayed.
MODULE INFO	Reads the module info for a specific controller module. See InaCmd.
SET TIME	Sets the date and time on the controller. The date and time from the PC where the APROL driver was started is used as reference.
CPU INFO	Outputs CPU-specific information such as the operating system version, type of CPU or AWS, start and status of the CPU, as well as the battery status.
LIST MODULES	Outputs all of the modules on the controller. MODULE INFO provides more detailed information.
SHOW MEM INFO	Depending on the type of memory, this information includes the start address, total length and available length.
LIST FORCED VARS	The physical addresses of the forced PVs and the corresponding task classes are output.

6.6.3 InaConnect

Unlike ***InaCmd*** and ***CfgInaDriver***, the ***InaConnect*** utility is a program, which establishes a connection to the controller on its own. As a result, this program can also be used to integrate individual controllers in a network, or to perform actions without having to start an APROL Runtime system. The program can be used two ways:

1. As a command shell (ideal for diagnostics and error analysis).
2. Direct command execution (ideal for use in scripts)

Generally, the individual services are called up in the command shell and directly. There are two types of options when calling up the program: options that define the connection type and connection parameter and the options that described the individual services.

The following section describes how to define a connection:

The respective connection parameters must be set according to the medium used to establish the connection with the controller:

Ethernet:

- Specification of the partner IP address (not the name – names are not resolved) or specification of the defined node number used for the partner station and the IP broadcast address of the network, if it is not 255.255.255.255 (when multiple networks are defined, each network has its own broadcast address).

- Specification of the station's own node number, so that the software (when there is more than one connection to a partner station) can correctly assign the packages (the number used should be unique in the respective network).
- Specification of the partner station's socket port number; the port for Ethernet communication on controllers is configured with 0x2b97 by default. This is why the port number specification can be considered optional.

Serial:

- The serial interface name is the only parameter. The DOS/Windows names are entered (COM1, COM2 etc.), which corresponds to the respective Linux devices /dev/ttySx (default settings for the interface are: baud rate=57600 , data bits=8 ; parity=2 [EVEN] ; stop bit=1).

Profibus:

- The user must decide which Profibus card should be used, the Profibus card from the manufacturer Softing or the B&R Profibus card.
- Which Profibus card should be accessed. By default, the first PB card of a type is accessed, but up to 3 boards per type are supported.
- Specifies the partner's station address. When using the Softing card, a specific LSAP (Local Service Access Point) can be defined (optional).

General:

- When the bus is under high load, it is possible to increase the timeout for the response to the request, to prevent disconnection.
- The B&R-2000 series can be routed from the connected CPU to other CPs.

  Example: There is a connection between a PC and an interface card. The routing text "*CP*" can be used if a connection is to be established with this controller's central CPU.

  When routing from a central CPU to the parallel processor on slot 2, the routing string could be called "*PP2*".

If a service is also specified directly with the correct connection parameters, then an attempt is made immediately when starting the program to establish a connection and the service is transmitted. The program is exited after successful processing. The program starts in the command shell if a service is not specified. The selected options for establishing a connection replace the default values upon startup.

The following table provides an overview of the currently available connection parameters and implemented services. Some of these are predefined as option when starting the program and/or shell command:

Option	Command	Description	Default
-help / -h	?	Displays the available options and commands.	

Option	Command	Description	Default
-medium	o	Selects which medium should be addressed: -Ethernet medium or o 1 as command -Profibus medium or o 2 as command -serialPort medium or o 3 as command	-
"Ethernet"	"o 1"	If Ethernet is selected, then the following parameters can be set:	
-socket No	3:socketNo	Port for INA communication with the partner station (controller)	0x2b97
-ip IP	4:ipAddress	IP address of the partner station	-
-node NODE	5:node	Node number of the partner station	-
-mynode NODE	6:myNode	Own node number (important for response telegrams)	-
-bcast BCAST	7:bcastAdr	Broadcast address when working with node number and the broadcast address is not 255.255.255.255	-
"Profibus"	"o 2"	If Profibus is selected, then the following parameters can be set:	
-softing	3:useSofting	Determines if the Softing Profiboard or the B&R Profibus card should be used for access	B&R card
-pbAddr ADDR	4:pbAddress	Specifies the partner's station address	-
-pbBoard No	5:pbBoard	Specifies which Profibus card in the PC should be used for access (0-2)	0
-pbLsap LSAP	6:pbLsap	When using the Softing card, a special LSAP can be used in addition to access via station addresses (optional).	-
"serialPort"	"o 3"	If serialPort is selected, then the following parameters can be set:	
-comport COM	3:comIf	Specifies the serial interface in DOS/Windows – COMx type which corresponds to the Linux devices ttySx - 1.	COM1
	4:comBd	specification of Baud rate (110, 300, 600, 1200, 2400, 3600, 4800, 9600, 19200, 38400 ,57600 ,115200)	57600
	5:comPa	Parity (0 =NO, 1=ODD , 2=EVEN)	2
	6:comIt	Timeout (0 – 60000 ms)	14
	"o [1|2|3]"	Connection parameters	
-setRT VAL	1:rtValue	Timeout for response telegrams	165

Option	Command	Description	Default
-route ROUTE	2:routeInfo	Routing information e.g. "CP"	-
	"c"	Attempts to establish a connection using the defined parameters	
	"d"	Disconnects	
	"s"	Displays the connection status CONN_establish_connection CONN_stable_connection CONN_lost_connection CONN_disconnecting CONN_disconnected	
-download FILE -burn	"p"	Downloads a B&R module, with specification about whether or not to "burn" the module	
-upload Module -file FILE	"g"	Uploads a B&R module from the controller, with a specified file name where the module should be saved on the hard drive	
-reset MODE	"r"	Resets the CPU connected to the program. Possible MODE (TOTAL, NORMAL, DIAGNOSE)	
-logbook NUM	"l"	Reads the CPU's error logbook with specification about the number of entries that should be read (max. 39)	
-cpuinfo	"i"	Reads CPU information (software version, type, etc.)	
-meminfo	"m"	Reads the CPU's memory information	
-dom_info MODUL	"lm"	Outputs module information. All modules are output if "*" is entered. If a specific module is defined, then detailed information is provided about that module.	
-delete MODUL	"dm"	Deletes a module on the controller	
	"pl"	Outputs a list of global PVs or a specific PV. The search can be filtered by specifying the start index and the range. The PV start index for a task can be determined using the command in the "Task".	
	"fl"	Outputs a list of the forced addresses and task class	
	"fn"	Forces a PV to a whole-number value by specifying its address and task class	

Option	Command	Description	Default
	"ff"	Un-forces a specific PV by specifying its address and task class, and all forced PV are deleted from the force table	
-gettime	"gt"	Outputs the date and time of the CPU as well as the PC	
-settime	"st"	Sets the PC date and time on the controller	
	"cm"	Deletes memory areas (Caution: all is deleted). To do this, the controller must be in Diagnose mode (r DIAGNOSE). This can only be performed via a serial connection. After deleting, the controller should be reset with a total-Init (r TOTAL)	
	"tki"	Supplies the status of the individual task classes on the connected CPU with name, number and status (STOP, RUNNING, IDLE)	
	"tkr"	Starts a task class (i.e. if a task class was stopped with tks "TC", then it is restarted with tkr "TC"). This means that all of the tasks in this task class are started again	
	"tks"	Stops a TC task class. All tasks in this task class are stopped and can be restarted using tkr "TC"	
	"tr"	Starts a specific task that is currently stopped	
	"ts"	Stops a specific task that has been started and is located on the CPU	
	"q"	Disconnects and stops the program	
-d LEVEL	"v"	Sets the debug level (1:error ; 2:transactions ; 4: data).	
	"t"	Displays the debug level	

6.7 Configuration of the APROL driver (InaDriver)

The APROL driver can be configured in two ways when starting. Part of the configuration is transferred to the driver in the form of start options (also see section *Description of the driver launching options*), the process variables are configured using a configuration file (also see the chapter *Configuration file of the APROL driver*).

6.7.1 Description of the InaDriver's start options

The following section offers a description of the start options for the driver. The start options can be combined in any way. Some of the options require additional parameters and some can simply be called as they are. When parameters are required, symbolic parameter names are defined that must be replaced by their values. Some of the parameters, such as specification of the connection name, are mandatory parameters and others are optional:

Option	Description
-D	Displays the date the driver was created and is then exited (optional parameter).
-n DRIVER_NAME	Sets the name of the driver in the Iosys to DRIVER_NAME. The status variables for the connection states and error indicators are generated with this name (optional parameter, default: InaDriver).
-run_task	The driver assumes that the configured task on the controller is in a "running" state and immediately begins communication after starting. This option is only used in APROL and has no effect on FMS operation (optional parameter).
-ignoreCnfPath	The driver searches for its configuration file relative to the present position instead of in the default directory (optional parameter).
-set_id CLIENT_ID	This option sets the client ID in the Iosys to the value CLIENT_ID. Which application has set the value of a variable can be detected with the output of IosEv using the client ID. (Mandatory parameter, Default: 31263)
-l LOGFILE	The driver creates a log file with the name LOGFILE. Start parameters and status outputs are written here and can be read during operation (optional parameter).
-i CAE_CONFIG_FILE	The driver in the APROL operating mode bases its configuration on the file CAE_CONFIG_FILE. Also in this case, the location of the configuration file refers to the default configuration directory if the -ignoreCnfPath option isn't used. Otherwise, it is relative to the start position of the driver.

Option	Description
-d DEBUG_FILTER	The driver is started in the debug mode using the -d option (i.e. it provides permanent status information instead of going into the background). The information is output either to the log file or to the screen via stderr. To avoid having all of the output on the screen, you can tell the driver which information to provide and which to ignore by specifically setting individual bits in the debug filter. The set bits in the debug filter allow the following output: 0x00000001: Output of error messages 0x00000002: Output of messages 0x00000004: Output of configuration messages at startup 0x00000002: Configuration of the Iosys objects 0x00000010: Online configuration messages 0x00000020: Iosys event messages 0x00000040: Messages from active read telegrams 0x00000080: Messages about connections 0x00000100: Messages about controller objects 0x00000200: Hex dump for incoming and outgoing data 0x00001000: PV forcing 0x4xxxxxx: INAFrame debug 0x8xxxxxx: INAComm debug
-f DEBUG_FILTER	The driver enables the debug information output as described above. This option starts in the background despite the output (optional parameter).
-delay DELAY_FACTOR	The driver evaluates all time settings in the configuration file as a multiple of the DELAY_FACTOR in milliseconds. The default setting for DELAY_FACTOR is 200 ms. The lowest possible setting for the DELAY-FACTOR is 100 milliseconds (optional parameter, default: 200).
-r	The driver is started in redundancy mode. (optional parameter)
-slaveNoConnect	If the driver was started in redundancy mode, then the slave usually establishes a connection, so that direct communication will continue when a redundancy switch takes place. It can be useful to stop this when resource problems occur (optional switch).
-forcePrefix PREFIX	Usually, only PVs that start with the prefix RV_ can be forced. This is a default in the APROL engineering system. This option allows you to modify this behavior.
-readHigh	By default, read telegrams have a low priority while write, diagnostics and download telegrams have a high priority. With this switch it is possible for read telegrams to also become high-priority telegrams.
-medium MEDIUM	Chooses the medium over which the driver should communicate (serialPort, Ethernet, Profibus).
	"Ethernet"
-node NODE	Specifies the node number of the partner station (if –ip is not used, then this switch is mandatory).

Option	Description
-bcast BROADCAST_ADDR	Sets the broadcast address. If the broadcast address is not 255.255.255.255, then the broadcast address valid for this network must be defined.
-ip DEST_IP_ADDR	Specifies the IP address of the partner station (if –node is not used, then this switch is mandatory).
-mynode NODE	Assigns your own node number. If several drivers or programs have established a connection to one controller, then a unique number must be set here.
-socket SOCKET	Specifies which port should be used to communicate to the controller (default: 0x2b97)
	"Profibus"
-softing	Determines if the Softing Profiboard or the B&R Profibus card (default) should be used for access.
-pbAddr ADDR	Specifies the partner's station address
-pbBoard BOARD	Specifies which Profibus card in the PC should be used for access (0-2)
-pbLsap LSAP	When using the Softing card, a special LSAP can be used in addition to access via station addresses (optional).
	"serialPort"
-comport COMPORT	Specifies the serial interface in DOS/Windows – COMx type which corresponds to the Linux devices ttySx - 1.
-setRT RTVAL	Timeout for response telegrams (default : 165)
-route ROUTE	Routing information e.g. "CP"

6.7.2 Configuration file of the APROL driver (InaDriver)

A configuration file is used to configure the driver's tasks and the process variables that will be used as well as their positions. The name and position of the configuration file are specified in the driver at startup using the ***"-i FILENAME"*** option. The position is relative to the default setting for driver configurations in APROL. More information can be found in the respective documentation. This default setting can be ignored by using the ***"-ignoreCnfPath"*** option and FILENAME describes the absolute position of the configuration file.

The driver configuration consists of a list of PVs. Each individual line is made up of the PV name, the corresponding task, the physical address, the PV type, the type of task and optional measurement range specifications. Comment lines begin with the # character and run to the end of the line.

A driver configuration generally looks something like this:

```
# FILE: T_RIO.br: 13 sections (Ver: 0.0)
# TK:   1
# Prio: 128
# PVs:  36
# IOs:  4
#
# Name                          Task    PhysAddr    Type      Req
```

```
L2003_RunningLightOff_1_LL_INT    T_RIO  0x7c070000  UINT32     W
L2003_RunningLightOff_1_P_ON      T_RIO  0x71060003  BIN0       W
L2003_RunningLightOff_1_P_OFF     T_RIO  0x71060002  BIN0       W
L2003_HltFbAdjuTime_1.TOn         T_RIO  0x70070004  UINT32     S
L2003_HltFbAdjuTime_1.TimeUnit    T_RIO  0x70070008  INT16      S
L2003_HltFbAdjuTime_1.TOff        T_RIO  0x7007000a  UINT32     S
L2003_RunningLight.tout           T_RIO  0x7007000e  UINT32     S
L2003_RunningLight.step           T_RIO  0x70070012  UINT16     S
L2003_RunningLight.time_out       T_RIO  0x70070014  UINT32     S
L2003_RunningLight.Enable         T_RIO  0x70070018  BIN0       S
L2003_RunningLight.Q1             T_RIO  0x70070019  BIN0       S
L2003_RunningLight.Q2             T_RIO  0x7007001a  BIN0       S
L2003_RunningLight.Q3             T_RIO  0x7007001b  BIN0       S
L2003_RunningLight.Q4             T_RIO  0x7007001c  BIN0       S
L2003_RunningLight.Q5             T_RIO  0x7007001d  BIN0       S
L2003_RunningLight.Q6             T_RIO  0x7007001e  BIN0       S
L2003_RunningLight.Q7             T_RIO  0x7007001f  BIN0       S
L2003_RunningLight.Q8             T_RIO  0x70070020  BIN0       S
L2003_RunningLight.Q9             T_RIO  0x70070021  BIN0       S
L2003_RunningLight.Q10            T_RIO  0x70070022  BIN0       S
L2003_RunningLight.Q11            T_RIO  0x70070023  BIN0       S
L2003_RunningLight.Q12            T_RIO  0x70070024  BIN0       S
L2003_Lauflicht.Q13               T_RIO  0x70070025  BIN0       S
L2003_Lauflicht.Q14               T_RIO  0x70070026  BIN0       S
L2003_Lauflicht.Q15               T_RIO  0x70070027  BIN0       S
L2003_Lauflicht.Q16               T_RIO  0x70070028  BIN0       S
L2003_SR_1_QN                     T_RIO  0x71060001  BIN0       R
L2003_SR_1_Q                      T_RIO  0x71060000  BIN0       R
L2003_SR_1.S                      T_RIO  0x7007002a  BIN0       S
L2003_SR_1.R                      T_RIO  0x7007002b  BIN0       S
L2003_SR_1.Q                      T_RIO  0x7007002c  BIN0       S
L2003_SR_1.QN                     T_RIO  0x7007002d  BIN0       S
RV_DO14_4                         T_RIO  0x610a0013  BIN0       S
RV_DO14_3                         T_RIO  0x610a0012  BIN0       S
RV_DO14_2                         T_RIO  0x610a0011  BIN0       S
RV_DO14_1                         T_RIO  0x610a0010  BIN0       S
HA_DO14_4                         T_RIO  0x71060013  BIN0       S
HA_DO14_3                         T_RIO  0x71060012  BIN0       S
HA_DO14_2                         T_RIO  0x71060011  BIN0       S
HA_DO14_1                         T_RIO  0x71060010  BIN0       S

# FILE: Task3_1.br: 13 sections (Ver: 0.0)
# TK:   3
# Prio: 128
# PVs:  10
# IOs:  4
#
# Name           Task    PhysAddr    Type      Req    Mba Mbe  Na Ne

TK_4_DIV_REAL    Task3_1 0x78070132  FLOAT      R     0   2000 0  100
TK_3_DIV_REAL    Task3_1 0x7807012e  FLOAT      R  -10    10   0  100
TK_2_DIV_REAL    Task3_1 0x7807012a  FLOAT      R   0     2000 0  200
TK_1_DIV_REAL    Task3_1 0x78070126  FLOAT      R  -10    10  -10 10
ST_R1_AI05_02    Task3_1 0x66f100ee  UINT16     S
ST_R1_AI05_01    Task3_1 0x66f100ec  UINT16     S
RV_R1_AO07_10    Task3_1 0x6603000e  UINT16     S
RV_R1_AO07_09    Task3_1 0x6603000c  UINT16     S
RV_R1_AI05_02    Task3_1 0x66010008  UINT16     S
RV_R1_AI05_01    Task3_1 0x66010006  UINT16     S
HA_R1_AO07_10    Task3_1 0x7807247c  FLOAT      S
HA_R1_AO07_09    Task3_1 0x78072478  FLOAT      S
HA_R1_AI05_02    Task3_1 0x7807005c  FLOAT      R
HA_R1_AI05_01    Task3_1 0x78070058  FLOAT      R
```

In principle, the number of process variables in a task and the number of tasks is unlimited. The only limitation to the configuration can result from the amount of memory used by the driver and the general limitation when generating a task.

PV_NAME:
The PV name is the name of the PV on the controller and the name of the Iosys object on the PC. According to task generation, a PV cannot be longer than 32 characters on a 2000 system.

Task:
The PVs are organized in tasks. Individual tasks can then be activated and deactivated. If a task is not available on the controller, then there is no communication with its PVs. This means that e.g. write events are not executed and the Iosys PV is reset to its previous value.

PV_ADDR:
The physical address of the PV is entered because the Ina driver performs physical access, (i.e. communication is not made via the PV names). For example when entering 0x78070058, the leading byte describes the PV type (e.g. hardware-IO or global PV) and the rest is the actual address (i.e. 0x08070058).

PV_TYPE:
The PV type determines what kind of variable must be found. Among other things, the setting for the PV type affects the default measuring range of the variables.

The following PV types are recognized:

PV type	PV length	Default measurement range
BIN0 to BIN7	1 bit	0 to 1
INT8	1 byte	-128 to 127
UINT8	1 byte	0 to 255
INT16	2 Bytes	-32768 to 32767
UINT16	2 Bytes	0 to 65535
INT32	4 Bytes	-2147483648 to 2147483647
UINT32	4 Bytes	0 to 4294967295
FLOAT	4 Bytes	-3E38 to 3E38
STRINGxx	XX bytes	No default measurement range, maximum 64 characters long

According to the configuration, the following **Iosys_types** are created in the Iosys:

Integer (whole numbers) with all PV types BINx, INTx and UINTx

Real (floating decimal numbers) with PV type S5FLOAT and FLOAT and with all whole number types if scaling was configured

String type with a maximum of 64 characters with PV type STRINGxx

Types of tasks:
The driver recognizes four different types of tasks:
SYNC Req=S,s,
READ Req=R.r,
WRITE Req=W,w,
IWRITE Req=I,i.

These tasks types are differentiated by their direction of communication:

READ tasks read cyclic variables from the controller and update the process variables in the Iosys. If the variables are changed (written) by another application, then the values are overwritten the next time the driver reads the values.

WRITE tasks write data to the controller as soon as the process variable is changed. Each PV must be read just once and compared with the value in the Iosys to align the control computer. If the PV has a different value than the Iosys, then the PV on the controller is set to the Iosys value (write task). Variables that for whatever reason could not be written properly are reset to the previous value.

IWRITE tasks behave mostly like ***WRITE***, but each PV must be read just once to align the control computer with the controller. This read process generally occurs once after each time the connection is established. If process variables are changed before the alignment has been made, then the variables are reset to the previous value.

SYNC tasks allow communication in both directions. This is a combination of cyclic READ procedures with event-controlled WRITE tasks. Variables can only be written to the controller if they have been read at least once. If value changes are made before they can be transferred to the controller, then driver undoes the changes and the previous value is set on the controller.

MBA and MBE or SPS_LOW and SPS_HIGH:
Measurement range for the variables on the controller. The measurement range is used for scaling to determine the multiplication factor and is also used to prevent operating errors. It only works in the direction from the control computer to the controller. If a variable value is entered in the control computer that is outside of the permissible measurement range (directly or as a result of scaling), then the driver resets the variable to the limit value and the limit value is sent to the controller. A value from the controller that is outside the limits is transferred to the control computer as is. SPS_LOW and SPS_HIGH are optional. If not otherwise specified, the default values from the PV type are used. Both of these values must be explicit if scaling is required!

NA and NE or LS_LOW and LS_HIGH:

LS_LOW and LS_HIGH are the control computer's limit values. Together with SPS_LOW and SPS_HIGH, they are used to compress or stretch (scaling) a variable. They are optional. When these values are not specified then there is a 1:1 scale. They can only be specified when SPS_LOW and SPS_HIGH have also been set.

6.8 InaDriver status variables

Visualization of the status of a controller's CPU

The status of a CPU in the visualization is the combination of the reason for booting (CpuBootTyp PV) and the state of the processor (CpuSt PV).

Reason for Initialization	Processor state	Status of the CPU for the display
Warm restart	Service	Service
Cold restart	Service	Service
Watchdog	Service	Service
Diagnostics	Service	Diagnostics
Error	Service	Service
Boot	Service	Boot

Reason for Initialization	Processor state	Status of the CPU for the display
-	Run	Run
-	Stop	Stop

The driver creates the following status variables in the Iosys for displaying and analyzing errors. These are generated and supplied automatically.

Note about the syntax of the status variables:

<CTRL>	Instance description of the controller (The instance is transferred on the driver with "-plc").
<CC>	Instance description of the control computer
<INST>	Instance description of the *InaDriver* ("-self" parameter)
<IP>	IP address of the driver
<PID>	Process ID

*All **counter** status variables have the following basic behavior:*
The counter counts until an overflow. It begins again at '0' after a stopping or starting the APROL system.

Variable name	Meaning	Iosys data type	Rec. IEC data type	Values
S2I_<CTRL>_Cnt Cyc	Watchdog for cyclic operation This counter is incremented by one with each run of the driver. a run describes the reading of all the driver's read PVs (so the polling of the read PVs, event PVs is not regarded)	Integer	DINT	
S2I_<CTRL>_Cnt Evt	Number of currently active event variables from the driver	Integer	DINT	
S2I_<CTRL>_Cnt Evt_m	Number of event PVs in the last minute Each event that is sent from the controller as an event variable is counted.	Integer	DINT	
S2I_<CTRL>_Cnt Evt_s	Number of event PVs in the last second Each event that is sent from	Integer	DINT	

Variable name	Meaning	Iosys data type	Rec. IEC data type	Values
	the controller as an event variable is counted.			
S2I_<CTRL>_CntEvtId	Highest ID that was sent back from the controller to a registered variable	Integer	DINT	
S2I_<CTRL>_CntPvAct	Amount of the driver's active variables These are all of the driver's registered, and communicated, variables. All types of variables (read, write, debugging, and event PVs) are counted.	Integer	DINT	
S2I_<CTRL>_CntRd_m	Amount of variables read in the last minute Each reading of a read variable is counted, regardless if its value has changed or not.	Integer	DINT	
S2I_<CTRL>_CntReqBad	counter, which is incremented by one with each negatively answered task (indication of network problems)	Integer	DINT	
S2I_<CTRL>_CntReqGood	Counter that is increased by one after each positive task response	Integer	DINT	
S2I_<CTRL>_CntWr	Number of write tasks sent to the controller No distinction is made about in which memory area the variables are held in the CPU (LOCAL; GLOBAL).	Integer	DINT	
S2I_<CTRL>_CntWr_m	Number of variables written in the last minute Each writing of a write variable to the controller is counted.	Integer	DINT	
S2I_<CTRL>_CntWrLoc	Number of write tasks sent to the controller and lie in the LOCAL area. (This is a part of the DRIVER_<CTRL>_wr_counter)	Integer	DINT	

Variable name	Meaning	Iosys data type	Rec. IEC data type	Values
S2I_<CTRL>_CpuSt	Display of the state of the controller (CPU)	Integer	INT	0 = RUN 1 = SERVICE 2 = STOP -1 = Information not available
S2I_<CTRL>_DrvConPara	Current connection configuration used by driver.	String64	SSTRING	Example: "`/IF=ETH /PORT=0x2b97 /IPADDR=rpCTRL10 /MYNODE=210`"
S2I_<CTRL>_DrvConSt	Display of the connection status of the driver.	Integer	INT	7 = being established 8 = is established 9 = lost 10 = being disconnected 11 = is disconnected
S2I_<CTRL>_DrvConTxt	Displays the connection status	String64	SSTRING	CONN_establish_connection = being established CONN_stable_connection = is established CONN_lost_connection = lost CONN_disconnecting = being disconnected CONN_disconnected = is disconnected
S2I_<CTRL>_DrvErrTxtCtrl	Displays the previous error The error text refers to problems, which occur during communication with the controller.	String64	SSTRING	See error message example: „Read request failed!“
S2I_<CTRL>_DrvErrTxtIos	Error text of the error that occurred last. This error text describes problems, which have mostly to do with values and states of Iosys PVs.	String64	SSTRING	See error messages Example: "unable to write : A02A00_PLST32_Rec_FrmRec"
S2I_<CtrlInst>_IntDbgMask	Variable for reading and setting the driver's current debugging filter	integer	DINT	0 = no debug information
S2I_<CtrlInst>_SrcNodeMa	The INA node number that is current used by the driver	Integer	INT	

Variable name	Meaning	Iosys data type	Rec. IEC data type	Values
	(actual node number from -*mynode* or -*mynodeslave*) <PID> is respectively replaced by the driver's actual process ID. (with 'Ma' for the configured master, and 'Sl' for the configured slave)			
S2I_<CtrlInst>_SrcNodeSl	The INA node number that is current used by the driver (actual node number from -*mynode* or -*mynodeslave*). <PID> is respectively replaced by the driver's actual process ID. (with 'Ma' for the configured master, and 'Sl' for the configured slave)	Integer	INT	

Additional status variables that display the memory load, the time, and the status of the controller. Also, the number of reconnects to the controller is output.

Variable name	Meaning	Iosys data type	Recommended IEC data type	Values
S2I_<CTRL>_CntRecon	Number of successful reconnects (restoration of the connection) Remark: The counter is initialized during a restart or change in process redundancy.	INT	DINT	-1: no connection established yet 0: (one) connection could be established since process start 1 ... n: successful restoration of the connection
S2I_<CTRL>_CpuArVer	AR version of the controller	STRING	SSTRING	(e.g.: R295)
S2I_<CTRL>_CpuBatt	Battery status	INT	INT	Bit0=1: Battery OK, Bit1=1: Battery OK;

Variable name	Meaning	Iosys data type	Recommended IEC data type	Values
S2I_<CTRL>_CpuBootTyp	Booting reason (warn start, cold start...) with	INT	INT	1 = Warm start, 2 = Cold start, 4 = Watchdog, 32 = Diagnosis, 64 = Error, 128 = BOOT
S2I_<CTRL>_CpuNodeNo	Network node number of the processor	INT	INT	
S2I_<CTRL>_CpuSt	Processor state	INT	INT	0 = RUN, 1 = SERVICE, 2 = STOP
S2I_<CTRL>_CpuTm	Seconds since 1.1.1970; local time of the processor, resolution 1 second, with the uncertainty of the communication	INT	DINT	
S2I_<CTRL>_CpuTyp	Description of the processor	STRING	SSTRING	(E.g. CP382, CP260, ...)
S2I_<CTRL>_MemDRamFlags	Validity of the DRAM information (can be used for evaluation of the size and free PV)	INT	INT	Bit0=1: Start address valid, but is not displayed as PV Bit1=1: Size PV valid Bit2=1: Free PV valid Bit3=1: 'Free block size' valid, but is not displayed as PV.
S2I_<CTRL>_MemDRamFree	free length of the memory area	INT	DINT	Length in bytes

Variable name	Meaning	Iosys data type	Recommended IEC data type	Values
S2I_<CTRL>_MemDRamSize	Complete length of the memory area	INT	DINT	Length in bytes **Not supported on the most targets (e.g. SG4 and SGC); he value -1 or. 0xffffffff is delivered in this case and is to be seen as invalid although the respective 'Bit 1' of the 'Flags PV' is set.**
S2I_<CTRL>_MemSysRomFlags	Validity of the SYSROM information (can be used for evaluation of the size and free PV)	INT	INT	Bit0=1: Start address valid, but is not displayed as PV Bit1=1: Size PV valid Bit2=1: Free PV valid Bit3=1: 'Free block size' valid, but is not displayed as PV
S2I_<CTRL>_MemSysRomFree	free size of the memory area	INT	DINT	Length in bytes
S2I_<CTRL>_MemSysRomSize	Complete length of the memory area	INT	DINT	Length in bytes **Not supported on the most targets (e.g. SG4 and SGC); he value -1 or. 0xffffffff is delivered in this case and is to be seen as invalid although the respective 'Bit 1' of the 'Flags PV' is set.**

Variable name	Meaning	Iosys data type	Recommended IEC data type	Values
S2I_<CTRL>_MemUserRomFlags	Validity of the USERROM information (can be used for evaluation of the size and free PV)	INT	INT	Bit0=1: Start address valid, but is not displayed as PV Bit1=1: Size PV valid Bit2=1: Free PV valid Bit3=1: 'Free block size' valid, but is not displayed as PV
S2I_<CTRL>_MemUserRomFree	free length of the memory area	INT	DINT	Length in bytes
S2I_<CTRL>_MemUserRomSize	Complete length of the memory area	INT	**DINT**	Length in bytes **Not supported on the most targets (e.g. SG4 and SGC); he value -1 or. 0xffffffff is delivered in this case and is to be seen as invalid although the respective 'Bit 1' of the 'Flags PV' is set.**

Further internal variables of the InaDriver:

These variables are solely used for internal communication, and are not allowed to be used in the application.

These PVs ensure the communication between the driver and the service tools.

Amongst others, the driver is told during the download of tasks via **ControllerManager** which task are to be loaded. Thus, the driver does not communicate with this task's PVs during the download.

Variable name	Meaning	Iosys data type	Recommended IEC data type	Values
<CC>_AppInaDriver_<INST>	System variable Details can be found in chapter *System variables* in the manual "D1 System Manual".	integer	INT	

Variable name	Meaning	Iosys data type	Recommended IEC data type	Values
<CC>_AppInaDriver_<INST>_UsedMem	System variable Details can be found in chapter *System variables* in the manual "D1 System Manual".			
<CC>_AppInaDriver_<INST>_ReduActiv	System variable Details can be found in chapter *System variables* in the manual "D1 System Manual".	Integer	INT	
ReduApp_InaDriver_<IP>_PID_<PID>	Internal PV for detecting the client redundancy	Integer	---	1
ReduGrp_InaDriver_<INST>	Internal PV for detecting the client redundancy	String64	---	actual client master as text
S2I_<CTRL>_IntCnfReq	Variable for the online driver configuration	Messagebox	---	
S2I_<CTRL>_IntCnfResp	Variable for the online driver configuration	Messagebox	---	
S2I_<CtrlInst>_SrcNode_<PID>	The INA node number that is currently used by the driver (actual node number from *srcNode* or *srcNodeSlave*). <PID> is respectively replaced by the driver's actual process ID.	Integer	INT	0-254

6.9 Error

6.9.1 Error numbers and messages (InaDriver)

The utilities and the APROL driver use different error numbers and handling mechanisms. The APROL driver works with error texts written to the error variable.

For communication, the utilities and the driver use libraries whose error numbers are the same for all utilities.

The library error numbers:

Error No.:	Error detection	Error text
4000	ERR_APIF_CPIN_ENCODE_TEL	Error from coding function
4001	ERR_APIF_CPIN_DECODE_TEL	Error from decoding function
4002	ERR_APIF_CPIN_UNEXPECTED_TEL	Unexpected telegram
4003	ERR_APIF_CPIN_DEFAULT	Default error
4010	ERR_APIF_RS_ENCODE_TEL	Error from coding function
4011	ERR_APIF_CPIN_DECODE_TEL	Error from decoding function
4012	ERR_APIF_RS_UNEXPECTED_TEL	Unexpected telegram
4013	ERR_APIF_RS_DEFAULT	Default error
4014	ERR_APIF_RS_NO_RIGHTS	Password incorrect
4020	ERR_APIF_CMEM_ENCODE_TEL	Error from coding function
4021	ERR_APIF_CMEM_DECODE_TEL	Error from decoding function
4022	ERR_APIF_CMEM_UNEXPECTED_TEL	Unexpected telegram
4023	ERR_APIF_CMEM_DEFAULT	Default error
4024	ERR_APIF_CMEM_TYPE	Memory type not available or cannot be deleted
4025	ERR_APIF_CMEM_DIAG	Controller must be in diagnose mode
4030	ERR_APIF_MEMI_ENCODE_TEL	Error from coding function
4031	ERR_APIF_MEMI_DECODE_TEL	Error from decoding function
4032	ERR_APIF_MEMI_UNEXPECTED_TEL	Unexpected telegram
4033	ERR_APIF_MEMI_DEFAULT	Default error
4040	ERR_APIF_LIMO_ENCODE_TEL	Error from coding function
4041	ERR_APIF_LIMO_DECODE_TEL	Error from decoding function
4042	ERR_APIF_LIMO_UNEXPECTED_TEL	Unexpected telegram
4043	ERR_APIF_LIMO_DEFAULT	Default error
4044	ERR_APIF_LIMO_OBJ_NOT_EXISTS	Object does not exist
4050	ERR_APIF_DELM_ENCODE_TEL	Error from coding function
4051	ERR_APIF_DELM_DECODE_TEL	Error from decoding function
4052	ERR_APIF_DELM_UNEXPECTED_TEL	Unexpected telegram
4053	ERR_APIF_DELM_DEFAULT	Default error
4054	ERR_APIF_DELM_OBJ_NOT_EXISTS	Object does not exist
4055	ERR_APIF_DELM_PI_NOT_STOPPED	PI must be stopped
4056	ERR_APIF_DELM_DEINSTALL_FAILED	Object cannot be uninstalled?
4060	ERR_APIF_DLM_ENCODE_TEL	Error from coding function
4061	ERR_APIF_DLM_DECODE_TEL	Error from decoding function
4062	ERR_APIF_DLM_UNEXPECTED_TEL	Unexpected telegram
4063	ERR_APIF_DLM_DEFAULT	Default error

Error No.:	Error detection	Error text
4064	ERR_APIF_DLM_CANCEL	DL interrupted by USER
4065	ERR_APIF_DLM_NO_ACCESS	Download disabled for USER
4066	ERR_APIF_DLM_NOT_USED_66	Not used
4067	ERR_APIF_DLM_BRM_INVALID	No BR module (2b97)
4068	ERR_APIF_DLM_BRM_CS	Defective BR module checksum
4069	ERR_APIF_DLM_BRM_INSTALL	BR module install error
4070	ERR_APIF_DLM_BRM_LENGTH	Incorrect length for BR module
4071	ERR_APIF_DLM_TARGET_SIZE	Not enough target memory available
4072	ERR_APIF_DLM_BURN	Error burning the BR module
4073	ERR_APIF_DLM_NO_BRNC	NC Manager not installed
4074	ERR_APIF_DLM_BRNC_ERROR	Error from the NC Manager download function
4080	ERR_APIF_ULM_ENCODE_TEL	Error from coding function
4081	ERR_APIF_ULM_DECODE_TEL	Error from decoding function
4082	ERR_APIF_ULM_UNEXPECTED_TEL	Unexpected telegram
4083	ERR_APIF_ULM_DEFAULT	Default error
4084	ERR_APIF_ULM_CANCEL	UL interrupted by USER
4085	ERR_APIF_ULM_OBJ_NOT_EXISTS	Module doesn't exist (incorrect name)
4086	ERR_APIF_ULM_OBJ_STATE_CONFLICT	Module state != READY
4087	ERR_APIF_ULM_OBJ_NO_ACCESS	Uploading the module disabled
4088	ERR_APIF_ULM_NO_ACCESS	Upload disabled for USER
4090	ERR_APIF_TKIN_ENCODE_TEL	Error from coding function
4091	ERR_APIF_TKIN_DECODE_TEL	Error from decoding function
4092	ERR_APIF_TKIN_UNEXPECTED_TEL	Unexpected telegram
4093	ERR_APIF_TKIN_DEFAULT	Default error
4100	ERR_APIF_TKR_ENCODE_TEL	Error from coding function
4101	ERR_APIF_TKR_DECODE_TEL	Error from decoding function
4102	ERR_APIF_TKR_UNEXPECTED_TEL	Unexpected telegram
4103	ERR_APIF_TKR_DEFAULT	Default error
4104	ERR_APIF_TKR_PI_RUNNING	TC already running
4105	ERR_APIF_TKR_PI_NOT_EXISTS	TC does not exist
4110	ERR_APIF_TKS_ENCODE_TEL	Error from coding function
4111	ERR_APIF_TKS_DECODE_TEL	Error from decoding function
4112	ERR_APIF_TKS_UNEXPECTED_TEL	Unexpected telegram
4113	ERR_APIF_TKS_DEFAULT	Default error
4114	ERR_APIF_TKS_PI_SUSPENDED	TC already stopped
4115	ERR_APIF_TKS_PI_NOT_EXISTS	TC does not exist

Error No.:	Error detection	Error text
4120	ERR_APIF_TR_ENCODE_TEL	Error from coding function
4121	ERR_APIF_TR_DECODE_TEL	Error from decoding function
4122	ERR_APIF_TR_UNEXPECTED_TEL	Unexpected telegram
4123	ERR_APIF_TR_DEFAULT	Default error
4124	ERR_APIF_TR_PI_RUNNING	Task already running
4125	ERR_APIF_TR_PI_NOT_EXISTS	Task does not exist
4130	ERR_APIF_TS_ENCODE_TEL	Error from coding function
4131	ERR_APIF_TS_DECODE_TEL	Error from decoding function
4132	ERR_APIF_TS_UNEXPECTED_TEL	Unexpected telegram
4133	ERR_APIF_TS_DEFAULT	Default error
4134	ERR_APIF_TS_PI_SUSPENDED	Task already stopped
4135	ERR_APIF_TS_PI_NOT_EXISTS	Task does not exist
4140	ERR_APIF_TSSC_ENCODE_TEL	Error from coding function
4141	ERR_APIF_TSSC_DECODE_TEL	Error from decoding function
4142	ERR_APIF_TSSC_UNEXPECTED_TEL	Unexpected telegram
4143	ERR_APIF_TSSC_DEFAULT	Default error
4144	ERR_APIF_TSSC_OBJ_NOT_EXISTS	Task does not exist
4150	ERR_APIF_LIGV_ENCODE_TEL	Error from coding function
4151	ERR_APIF_LIGV_DECODE_TEL	Error from decoding function
4152	ERR_APIF_LIGV_UNEXPECTED_TEL	Unexpected telegram
4153	ERR_APIF_LIGV_DEFAULT	Default error
4160	ERR_APIF_LIFV_ENCODE_TEL	Error from coding function
4161	ERR_APIF_LIFV_DECODE_TEL	Error from decoding function
4162	ERR_APIF_LIFV_UNEXPECTED_TEL	Unexpected telegram
4163	ERR_APIF_LIFV_DEFAULT	Default error
4170	ERR_APIF_RV_ENCODE_TEL	Error from coding function
4171	ERR_APIF_RV_DECODE_TEL	Error from decoding function
4172	ERR_APIF_RV_UNEXPECTED_TEL	Unexpected telegram
4173	ERR_APIF_RV_DEFAULT	Default error
4174	ERR_APIF_RV_INVALID_OBJ_ID	Invalid PV ID
4175	ERR_APIF_RV_OBJ_NOT_EXISTS	PV does not exist
4176	ERR_APIF_RV_OBJ_WR_LEN	Incorrect PV length
4177	ERR_APIF_RV_OBJ_NIL_PTR	Dyn. PV is not used (NULL-Ptr.)
4180	ERR_APIF_WV_ENCODE_TEL	Error from coding function
4181	ERR_APIF_WV_DECODE_TEL	Error from decoding function
4182	ERR_APIF_WV_UNEXPECTED_TEL	Unexpected telegram
4183	ERR_APIF_WV_DEFAULT	Default error
4184	ERR_APIF_WV_INVALID_OBJ_ID	Invalid PV ID
4185	ERR_APIF_WV_OBJ_NOT_EXISTS	PV does not exist

Error No.:	Error detection	Error text
4186	ERR_APIF_WV_OBJ_WR_LEN	Incorrect PV length
4187	ERR_APIF_WV_OBJ_NIL_PTR	Dyn. PV is not used (NULL-Ptr.)
4190	ERR_APIF_FVON_ENCODE_TEL	Error from coding function
4191	ERR_APIF_FVON_DECODE_TEL	Error from decoding function
4192	ERR_APIF_FVON_UNEXPECTED_TEL	Unexpected message
4193	ERR_APIF_FVON_DEFAULT	Default error
4194	ERR_APIF_FVON_INVALID_OBJ_ID	Invalid PV-ID (Data type, Baseptr, ...)
4195	ERR_APIF_FVON_INVALID_TK	Invalid task class
4196	ERR_APIF_FVON_FTAB_OVERFLOW	No free entry in the force table
4200	ERR_APIF_FVOFF_ENCODE_TEL	Error from coding function
4201	ERR_APIF_FVOFF_DECODE_TEL	Error from decoding function
4202	ERR_APIF_FVOFF_UNEXPECTED_TEL	Unexpected telegram
4203	ERR_APIF_FVOFF_DEFAULT	Default error
4204	ERR_APIF_FVOFF_INVALID_OBJ_ID	Invalid PV-ID (Datentyp, Baseptr, ...)
4205	ERR_APIF_FVOFF_INVALID_TK	Invalid task class
4206	ERR_APIF_FVOFF_PV_NOT_FORCED	PV is not forced
4210	ERR_APIF_DIAGIN_ENCODE_TEL	Error from coding function
4211	ERR_APIF_DIAGIN_DECODE_TEL	Error from decoding function
4212	ERR_APIF_DIAGIN_UNEXPECTED_TEL	Unexpected telegram
4213	ERR_APIF_DIAGIN_DEFAULT	Default error
4220	ERR_APIF_DIAGLM_ENCODE_TEL	Error from coding function
4221	ERR_APIF_DIAGLM_DECODE_TEL	Error from decoding function
4222	ERR_APIF_DIAGLM_UNEXPECTED_TEL	Unexpected telegram
4223	ERR_APIF_DIAGLM_DEFAULT	Default error
4224	ERR_APIF_DIAGLM_NOT_INITIALIZED	Diagnose task not loaded
4230	ERR_APIF_DIAGDM_ENCODE_TEL	Error from coding function
4231	ERR_APIF_DIAGDM_DECODE_TEL	Error from decoding function
4232	ERR_APIF_DIAGDM_UNEXPECTED_TEL	Unexpected telegram
4233	ERR_APIF_DIAGDM_DEFAULT	Default error
4234	ERR_APIF_DIAGDM_NOT_INITIALIZED	Diagnose task not loaded
4235	ERR_APIF_DIAGDM_INVALID_MODUL	DIAG module index invalid, list not yet read ?
4240	ERR_APIF_DIAGEX_ENCODE_TEL	Error from coding function
4241	ERR_APIF_DIAGEX_DECODE_TEL	Error from decoding function
4242	ERR_APIF_DIAGEX_UNEXPECTED_TEL	Unexpected telegram
4243	ERR_APIF_DIAGEX_DEFAULT	Default error
4244	ERR_APIF_DIAGEX_NOT_INITIALIZED	Diagnose task not loaded

Error No.:	Error detection	Error text
4250	ERR_APIF_RLB_ENCODE_TEL	Error from coding function
4251	ERR_APIF_RLB_DECODE_TEL	Error from decoding function
4252	ERR_APIF_RLB_UNEXPECTED_TEL	Unexpected telegram
4253	ERR_APIF_RLB_DEFAULT	Default error
4260	ERR_APIF_LHWC_ENCODE_TEL	Error from coding function
4261	ERR_APIF_LHWC_DECODE_TEL	Error from decoding function
4262	ERR_APIF_LHWC_UNEXPECTED_TEL	Unexpected telegram
4263	ERR_APIF_LHWC_DEFAULT	Default error
4264	ERR_APIF_LHWC_NO_DATA	No HWC determination carried out on controller (e.g. in Diag.Mode)
4270	ERR_APIF_GT_ENCODE_TEL	Error from coding function
4271	ERR_APIF_GT_DECODE_TEL	Error from decoding function
4272	ERR_APIF_GT_UNEXPECTED_TEL	Unexpected telegram
4273	ERR_APIF_GT_DEFAULT	Default error
4280	ERR_APIF_ST_ENCODE_TEL	Error from coding function
4281	ERR_APIF_ST_DECODE_TEL	Error from decoding function
4282	ERR_APIF_ST_UNEXPECTED_TEL	Unexpected message
4283	ERR_APIF_ST_DEFAULT	Default error
4284	ERR_APIF_ST_INVALID_TIME	Invalid time
4290	ERR_APIF_RMEM_ENCODE_TEL	Error from coding function
4291	ERR_APIF_RMEM_DECODE_TEL	Error from decoding function
4292	ERR_APIF_RMEM_UNEXPECTED_TEL	Unexpected telegram
4293	ERR_APIF_RMEM_DEFAULT	Default error
4300	ERR_APIF_WMEM_ENCODE_TEL	Error from coding function
4301	ERR_APIF_WMEM_DECODE_TEL	Error from decoding function
4302	ERR_APIF_WMEM_UNEXPECTED_TEL	Unexpected telegram
4303	ERR_APIF_WMEM_DEFAULT	Default error
4310	ERR_APIF_TT_ENCODE_TEL	Error from coding function
4311	ERR_APIF_TT_DECODE_TEL	Error from decoding function
4312	ERR_APIF_TT_UNEXPECTED_TEL	Unexpected message
4313	ERR_APIF_TT_DEFAULT	Default error
4321	ERR_APIF_DBEV_DECODE_TEL	Error from decoding function
4323	ERR_APIF_DBEV_DEFAULT	Default error
4330	ERR_APIF_SEON_ENCODE_TEL	Error from coding function
4331	ERR_APIF_SEON_DECODE_TEL	Error from decoding function
4332	ERR_APIF_SEON_UNEXPECTED_TEL	Unexpected telegram
4333	ERR_APIF_SEON_DEFAULT	Default error
4334	ERR_APIF_SEON_OVERRUN	Max. number of event masters reached

Error No.:	Error detection	Error text
4341	ERR_APIF_SYEV_DECODE_TEL	Error from decoding function
4343	ERR_APIF_SYEV_DEFAULT	Default error
4600	ERR_IC_INIT_THREAD	No VB thread created
4601	ERR_IC_INIT_EVENT	Error from CreateEvent()
4602	ERR_IC_INIT_TIMER	Error from timeSetEvent()
4603	ERR_IC_INIT_AL2	Error from AL2_Init()
4604	ERR_IC_INIT_SETCONNECTION	Error from AL2_SetNewConnection()
4605	ERR_IC_INIT_FRAME_DEFAULT	INAFrameOpen() – default error
4606	ERR_IC_DEVICE_IN_USE	No more VBs can be opened using the device
4610	ERR_IC_EXIT_EVENT	Error from CloseHandle()
4611	ERR_IC_EXIT_TIMER	Error from timeKillEvent()
4612	ERR_IC_EXIT_FRAME	Error from INAFrameClose()
4620	ERR_IC_AL2_TIMER	Error for cyclic ALI-L2 notification
4621	ERR_IC_AL2_TRANSMITTED	Error from ALI-L2-transmitted notification
4622	ERR_IC_AL2_FRM_RECEIVED	Error from ALI-L2-frame received notification
4630	ERR_IC_TX_CMD_STATE_INVALID	Invalid tasks state
4631	ERR_IC_RX_CMD_STATE_INVALID	Invalid tasks state
4632	ERR_IC_AL2_TX	Error from ALI-L2-TX job
4640	ERR_IC_APIF_SRV_NOT_IMPLEMENTED	APIF service not implemented
4641	ERR_IC_STM_WRSTATE	APIF state machine in invalid state
13000	INAFRM_EC_GenInafrmError	Undefined INAFRM.DLL error
13001	INAFRM_EC_ActiveRecvRequest	Active request still in progress (in receive request)
13002	INAFRM_EC_ActiveSendRequest	Active request still in progress (in send request)
13003	INAFRM_EC_CloseCommPort	Error while closing the interface
13004	INAFRM_EC_CommRead	Error while reading from the interface
13005	INAFRM_EC_CommWrite	Error while writing from the interface
13006	INAFRM_EC_DriverNotFound	Driver not found, cannot be loaded
13007	INAFRM_EC_Ilg_IF_Param	Error in the "/IF" parameter (e.g. wrong COM number)
13008	INAFRM_EC_IlgDataReceived	Illegal data received
13009	INAFRM_EC_IlgDeviceHandle	Illegal device handle
13010	INAFRM_EC_IlgDstAddress	Illegal destination address

Error No.:	Error detection	Error text
13011	INAFRM_EC_IlgParam	Invalid parameter in the initialization string (open)
13012	INAFRM_EC_IlgParamModification	Invalid parameter change (open)
13013	INAFRM_EC_IlgRecvDataLength	Illegal data length while receiving request
13014	INAFRM_EC_IlgSendDataLength	Illegal data length while sending request
13015	INAFRM_EC_IlgScrAddress	Illegal source address
13016	INAFRM_EC_IlgStationHandle	Illegal station handle
13017	INAFRM_EC_No_ID_ListFound	No ID list given
13018	INAFRM_EC_NoFreeRAM	Insufficient RAM memory
13020	INAFRM_EC_NoResponseForRecvRequest	No confirmation data for receive request available
13021	INAFRM_EC_NoResponseForSendRequest	No confirmation data for send request available
13022	INAFRM_EC_ReadCommPortParam	Error while reading interface parameter
13024	INAFRM_EC_UnknownDevice	Unknown device
13025	INAFRM_EC_Unsupported	Function not supported
13026	INAFRM_EC_WriteCommPortParam	Error while writing the interface parameters
13027	INAFRM_EC_WinResource	Error in Window resource (Events....)
13028	INAFRM_EC_StaAlreadyUsed	Connection already being used
13029	INAFRM_EC_StartThread	Transfer thread cannot be started
13030	INAFRM_EC_UnknownDstAddress	Unknown destination address
13035	INAFRM_EC_ReadTimeout	Timeout error when reading (internal timeout)
13036	INAFRM_EC_WriteTimeout	Timeout error when writing (internal timeout)
13040	INAFRM_EC_Pbus_FirmwareDownload	Error when downloading the Profibus firmware.
13041	INAFRM_EC_Pbus_NetwrkConfigDownload	Error when downloading Profibus network configuration module.
13042	INAFRM_EC_Pbus_CreateThread	Error when installing the Pbus read/write threads
13045	INAFRM_EC_TapiInitError	error calling lineInitializeEx --> TAPI not properly installed
13046	INAFRM_EC_TapiNoDevices	there are no line (modem) devices installed
13047	INAFRM_EC_TapiNotModemDevice	the given device does not have the necessary capabilities
13048	INAFRM_EC_TapiModemNotFound	specified modem not found

Error No.:	Error detection	Error text
13049	INAFRM_EC_TapiIncompVersion	the installed TAPI version is not supported
13050	INAFRM_EC_TapiGeneralError	some uncommon TAPI error occurred
13051	INAFRM_EC_TapiDeviceInUse	LineOpen error (LINEERR_ALLOCATED):Line is already in use by a
13052	INAFRM_EC_TapiChannelInUse	channel 0 of the specified device is already in
13053	INAFRM_EC_TapiInvalidPhoneNumber	the specified phone number does not have a valid format -->
13054	INAFRM_EC_TapiMakeCall	in placing the call
13055	INAFRM_EC_TapiNoAnswer	there were no answer from the modem within a
13056	INAFRM_EC_TapiDialingUnsuccess	e.g. if modem not connected to the line, no
13057	INAFRM_EC_TapiModemNotReady	error if modem not ready (switched off)
13058	INAFRM_EC_TapiVersion	wrong TAPI version installed (e.g.1.4) use 2.0 or better
13059	INAFRM_EC_TapiLostConnection	modem connection lost (modem switched off, telephone line unplugged)
13060	INAFRM_EC_TapiCreateThread	can't create thread or mutex
13061	INAFRM_EC_TapiMissingPara	missing parameter, /MO and /TN are mandatory
13070	INAFRM_EC_WIN_AccessDenied	Access denied (5, ERROR_ACCESS_DENIED)
13071	INAFRM_EC_WIN_BadCommand	Device does not recognize command (22, ERROR_BAD_COMMAND)
13072	INAFRM_EC_WIN_BadDevice	The device entered is invalid (1200, ERROR_BAD_DEVICE)
13073	INAFRM_EC_WIN_BadUnit	The system cannot find the specified device (20,
13074	INAFRM_EC_WIN_DiskFull	Insufficient space on the disk drive (112, ERROR_DISK_FULL)
13075	INAFRM_EC_WIN_DriverNotFound	Specified module not found (126, ERROR_MOD_NOT_FOUND)
13076	INAFRM_EC_WIN_FileNotFound	The specified file cannot be found (2, ERROR_FILE_NOT_FOUND)
13077	INAFRM_EC_WIN_GenFailure	A device in the system is not functioning (31, ERROR_GEN_FAILURE)

Error No.:	Error detection	Error text
13078	INAFRM_EC_WIN_InvalidFlags	Invalid flag (1004, ERROR_INVALID_FLAGS)
13079	INAFRM_EC_WIN_InvalidName	Syntax of file name, directory name or disk drive name incorrect (123, ERROR_INVALID_NAME)
13080	INAFRM_EC_WIN_InvalidFunction	Invalid function (1, ERROR_INVALID_FUNCTION)
13081	INAFRM_EC_WIN_InvalidAccess	The access code is invalid (12, ERROR_INVALID_ACCESS)
13082	INAFRM_EC_WIN_InvalidHandle	Invalid handle (6, ERROR_INVALID_HANDLE)
13083	INAFRM_EC_WIN_NotEnoughMemory	Insufficient memory to execute command (8, ERROR_NOT_ENOUGH _MEMORY)
13084	INAFRM_EC_WIN_NotReady	The device is not ready (21, ERROR_NOT_READY)
13085	INAFRM_EC_WIN_OpenFailed	The system cannot open the device/file (110,
13086	INAFRM_EC_WIN_PathNotFound	Path not found (3, ERROR_PATH_NOT_FOUND)
13087	INAFRM_EC_WIN_ReadFault	The system cannot read from the specified device (30,
13088	INAFRM_EC_WIN_StackOverflow	Recursion too deep - stack overflow (1001, ERROR_STACK_OVERFLOW)
13089	INAFRM_EC_WIN_WriteFault	The system cannot write to the specified device (29,
13090	INAFRM_EC_TcpIpVersion	Unsupported winsock version (install winsock 2)
13091	INAFRM_EC_TcpIpInit	Error while initializing winsock
13092	INAFRM_EC_TcpIpSocket	Error creating socket
13093	INAFRM_EC_TcpIpGeneral	General error
13094	INAFRM_EC_TcpIpPort	Port in use
13095	INAFRM_EC_TcpIpBind	Error binding socket
13096	INAFRM_EC_TcpIpCreateThread	Can't create thread
13097	INAFRM_EC_TcpIpNoAddress	Node-resolution not finished yet

6.10 The InaDriver start script

6.10.1 Structure

The InaDriver start script is a shell script with the following structure:

```
#
#   Id: InaDriver.sh,v 1.1.1.1 2000/04/20 11:45:23 hschroeter Exp
#
#   Log: InaDriver.sh,v
#   Revision 1.1.1.1  2000/04/20 11:45:23  hschroeter
#   initial version
#
#

#*****************************************************************
#         COPYRIGHT 1996 – 2000 PCC GmbH. ALL RIGHTS RESERVED
#*****************************************************************
#     AUTHOR:       FELDHAUS
#     USE FOR:      Starting InaDriver
#*****************************************************************
#set –x

VERSION="` echo Revision: 1.1.1.1  | cut -f2 -d " " `"
VDATE="` echo Date: 2000/04/20 11:45:23  | cut -f2-3 -d " " `"

COMP="!= 0"
FILENAME="InaDriver.sh"
DEFFILE="$APROL/etc/globaldefs"
if [ ! -f $DEFFILE ]; then
  Echo "$DEFFILE not found"
  Exit 0
Else
. $DEFFILE
fi

STARTUPFILE="$CNF_USER_PATH/InaDriver/startup.cnf"
LOGFILE=$HOME/tmp/InaDriver.log
TMPFILE=$HOME/tmp/InaDriver.tmp
BIN_DIR="$APROL/bin"

if [ ! -f $STARTUPFILE ]; then
  $ECHO "call -$FILENAME- with -$1- at $DATE"  >> $LOGFILE
  getmsg 0019                          >> $LOGFILE
  getmsg 0020                          >> $LOGFILE
  $ECHO "\n"                           >> $LOGFILE
  exit 0
else
    . $STARTUPFILE
fi

opt_2=""

#SET REDU-OPTION
if [ "$IOSYS_TYPE" = "REDU" ] ; then
  opt_2="-slaveNoConnect $opt_2"
fi

setiosysvar

$ECHO "\t\tIOSYS=$IOSYS"

$ECHO "call -$FILENAME- with -$1- at $DATE" >> $LOGFILE
case "$1" in
  'start')
  $ECHO "\t\t$FILENAME $1"
  startupcheck $startup $prog
```

```
 if [ "$?" = 0 ]; then
 exit 0
 fi
 confcheck $instances
 for instance in $instances; do
  eval startup=\$startup_$instance
 if [ $startup = "NO" -o "$startup" = "no" -o "$startup" = "" ];
 then
  $ECHO "START $prog $instance : NO "
 else
  eval opt=\$opt_$instance
  opt_ps="-self `getoptionvalue -self "$opt"`"
  if [ "$?" = 0 ]; then
  opt_ps=""
  getmsg 0035 -self
 Fi
 # check if program exists and cnf file is readable
 if [ -x "$BIN_DIR/$prog" ]; then
  # check version of module
  CheckVersion $prog $version
  # check if already running ...
  getpid $prog "$opt_ps"
  if [ -z "$PID" ]; then
   LOGFILE=$HOME/tmp/"$prog"_"$instance".log
   TMPFILE=$HOME/tmp/"$prog"_"$instance".tmp
   $ECHO ">$prog $instance<\c"
   getmsg 0002
   # call prog
   OLDIR=`pwd`
   cd $BIN_DIR
   $prog $opt $opt_2 1>> $LOGFILE 2>&1
   RET=$?
   tail -4 $LOGFILE > $TMPFILE
      ERROR=`$CAT $TMPFILE | $GREP "error" 2>/dev/null`
             if [ -n "$ERROR" -o "$RET" $COMP ]; then
                 error $prog
             else
                delay 10
                getpid $prog "$opt_ps"
                  if [ -n "$PID" ]; then
                   $ECHO "start $prog $DATE" >> $LOGFILE
                   $ECHO " ok"
                else
                   starterror $prog
                   fi
              Fi
  else
      beeper 3
      $ECHO ">$prog $instance<\c"
      getmsg 0005
      Getmsg 0006
    fi
   else
    beeper 2
    getmsg 0007
    getmsg 0009
   Fi
  fi
 Done
cd $OLDIR
;;
'stop')
 $ECHO "\t\t$FILENAME $1"
 startupcheck $startup $prog
 if [ "$?" = 0 ]; then
  exit 0
 fi
 confcheck $instances
 instances=`swapstring "$instances"`
 for instance in $instances; do
  eval startup=\$startup_$instance
  eval opt=\$opt_$instance
  opt_ps="-self `getoptionvalue -self "$opt"`"
  if [ "$?" = 0 ]; then
  opt_ps=""
  getmsg 0035 -self
 fi
 getpid $prog "$opt_ps"
 if [ -z "$PID" ]; then
  $ECHO ">$prog $instance<\c"
  getmsg 0004
```

```
  else
   kill $PID
   $ECHO ">$prog $instance<\c"
   getmsg 0003
   LOGFILE=$HOME/tmp/"$prog"_"$instance".log
   $ECHO "stop $PROGS $DATE" >> $LOGFILE
   TMPFILE=$HOME/tmp/"$prog"_"$instance".tmp
   rm -f $TMPFILE
    mv "$LOGFILE" "$LOGFILE"_old
   fi
  Done
 ;;
 'restart')
  ($SH $0 stop; getmsg 0012; delay 5; $SH $0 start)
 ;;
 '-ver')
  printver $VERSION
  exit 0
 ;;
 '-version')
  printversion $0 $VERSION $VDATE
  exit 0
 ;;
 '-help')
  getmsg 0001
  exit 0
 ;;
 *)
  getmsg 0001
  exit 1
 ;;
esac
```

If necessary, it is called from the APROL start script.

A typical APROL start-up file with the following structure:

```
#Enviroment for startup
#Modul : InaDriver
USER=pccrun; LOCALHOST=server1; RUNTIME_USER=pccrun;
IOSYS_HOST='server1'; MASTER=server1; IOSYS_TYPE=LOCAL; IOSYS_PORT=0;
ALARMSERVER=01
#List instance and options for modul
prog="InaDriver"
startup="YES"
version="1.1.0"
instances="01 02 03 04 05 06 07 08"
startup_01="YES"
autostartup_01="NO"

opt_01="-delay 1000 -i SPS1/APROL.cfg -ip 192.168.77.59 -medium
Ethernet -mynode 50 -n DRIVER_SPS1 -route CP -run_task  -self 01
-setRT 1000 -set_id 30010 -socket 0x2b97"
startup_02="YES"
autostartup_02="NO"

opt_02="-delay 1000 -i SPS2/APROL.cfg -ip 192.168.77.52 -medium
Ethernet -mynode 50 -n DRIVER_SPS2 -run_task  -self 02 -setRT 1000
–set_id 30011 -socket 0x2b97"
startup_03="YES"
autostartup_03="NO"

opt_03="-delay 1000 -i SPS3/APROL.cfg -ip 192.168.77.53 -medium
Ethernet -mynode 50 -n DRIVER_SPS3 -run_task  -self 03 -setRT 1000
–set_id 30012 -socket 0x2b97"
startup_04="YES"
autostartup_04="NO"

opt_04="-delay 1000 -i SPS4/APROL.cfg -ip 192.168.77.54 -medium
Ethernet -mynode 50 -n DRIVER_SPS4 -run_task  -self 04 -setRT 1000
–set_id 30013 -socket 0x2b97"
startup_05="YES"
autostartup_05="NO"

opt_05="-delay 1000 -i SPS5/APROL.cfg -ip 192.168.77.55 -medium
Ethernet -mynode 50 -n DRIVER_SPS5 -run_task  -self 05 -setRT 1000
–set_id 30014 -socket 0x2b97"
startup_06="YES"
autostartup_06="NO"

opt_06="-delay 1000 -i SPS6/APROL.cfg -ip 192.168.77.56 -medium
Ethernet -mynode 50 -n DRIVER_SPS6 -run_task  -self 06 -setRT 1000
–set_id 30015 -socket 0x2b97"
startup_07="YES"
autostartup_07="NO"
```

```
opt_07="-delay 1000 -i SPS7/APROL.cfg -ip 192.168.77.57 -medium Ethernet
-mynode 50 -n DRIVER_SPS7 -run_task  -self 07 -setRT 1000
–set_id 30016 -socket 0x2b97"
startup_08="YES"
autostartup_08="NO"

opt_08="-delay 1000 -i SPS8/APROL.cfg -ip 192.168.77.58 -medium
Ethernet -mynode 50 -n DRIVER_SPS8 -run_task  -self 08 -setRT 1000
–set_id 30017 -socket 0x2b97"
```

6.11 Error analysis and handling

6.11.1 Profibus connection

After the installation, the ***Livelist*** cannot be started with ***pb_manager*** (Softing card) or ***br_sys*** (B&R Profibus card).

The following points must be checked:

Is the device driver loaded in the LINUX kernel?

Condition: You are logged in as super user.

Start the ***lsmod*** command. All modules connected to the kernel are displayed.

A PROFIboard or br_driver module must be present.

If one is not present, then go to the ***/boot/modules/PROFIboard*** directory and start ***insmod PROFIboard.o*** or to the ***/boot/modules/pbbr*** directory and start ***insmod br_driver.o***.

Analyze the error message

Is there any kernel output?

Condition: You are logged in as super user.

Enter the ***dmesg*** command.

Search the kernel driver output. After ***dmesg***, you can browse the output using the ***Shift/Page-Up*** and ***Shift/Page-Down*** keys.

The following message should appear after booting. The settings may vary according to your parameters:

PROFIboard card:

```
SOFTING PROFIboard DEVICE DRIVER       V 3.0.3 (C) 1998-2000 by PCC,
register major number 50
isa_board: io_addr = 0x0240, irq =  5, base  = 0x000d0000
found 1 devices
maxWaitLoops = 5
```

B&R Profibus card:

```
B&R PROFIBUS DEVICE DRIVER    V 3.0.2  LINUX   (c) 1998-1999 by pcc,
Created Aug  8 2000 at 08:42:53
poll loop set to 10 ms
register major number 60
found B&R board at 000d8000
device 0, firmware V3.10
status = 0
apb = 0
```

A corresponding error message is given if the Profibus card was not recognized. The device driver has probably not been loaded if the above message does not appear.

Are the device files present?

Make sure that the entry files are located in the ***/dev/PROFIboard*** **directory:**

ls -la /dev/PROFIboard/PROFIboard*

these files should be located in the following for a B&R Profibus card:

ls –la /dev/brpb*

If these files are not present or the displayed major number does not match the device drive message (see inset above: register major number ...) than an error has occurred with ***pb_install*** or ***br_install***. Restart the program and check the error messages.

Is the start script present?

Make sure that the start script is present and contains a link in the start directories.

ls -la /sbin/init.d/PROFIboard

ls -la /sbin/init.d/rc2.d/S51PROFI.sh

ls -la /sbin/init.d/rc3.d/S51PROFI.sh

or

ls -la /sbin/init.d/BuR_PROFIBUS

ls -la /sbin/init.d/rc2.d/S52PROFI.sh

ls -la /sbin/init.d/rc3.d/S52PROFI.sh

If the start files are not present then an error has occurred with ***pb_install*** or ***br_install***. Restart the program and check the error messages.

Are the files for network configuration present?

Do the configuration files for the respective PROFIboard exist?

ls -la /usr/etc/profiboardx.cfg

ls -la /usr/etc/profiboardx.ov with x = board number, starting with 0

If the files are not present, then you have to create them and start ***pb_init***.

Just like with the PROFIboard, the following must be also be checked when using the B&R Profibus card:

ls –la /usr/etc/brpb/profibusx.cfg mit x= board number , starting with 0

ls –la /usr/etc/downloads/profibus.fw

Is the Profibus card on the network?

If the card is already on the network and the bus parameters are not accordingly adjusted, then it is possible that the card is deactivated to prevent a network interruption on this card. Never perform a reinstallation while the Profibus card is connected to the network. Start the Livelist, which will run cyclically, and wait until the card can be seen here before plugging in the cable if necessary. Check the bus parameters again if the Livelist only works when there is no connection to the Profibus network.

Why can't a connection be established?

Although the controller appears in the Livelist and FMS services are working properly (such as an Identify), the programs ***InaDriver*** and ***InaConnect*** cannot establish a connection to the controller and are showing the connection status ***lost connection***. If this is the case, then check if the BR module ***fbpb.br*** has been burned to the controller and if Profibus devices have been enabled for communication in the system setting (***sysconf***).

6.11.2 Ethernet connection

Problem: A connection cannot be established with the controller?

Are the network interfaces correctly configured?

The installation guidelines from the manual are valid for the PC. It is important to check if the system correctly recognized the Ethernet card, which the super user can do by looking at the information output provided by the ***dmesg*** command.

For example:

```
Eth0: Digital DS21143 Tulip rev 65 at 0xa800, 00:00:1C:B5:F4:D1, IRQ 10.
Eth0:  EEPROM default media type Autosense.
Eth0:  Index #0 - Media MII (#11) described by a 21142 MII PHY (3) block.
Eth0:  MII transceiver #5 config 3000 status 7829 advertising 01e1.
```

If the card has been correctly configured, then the ***ifconfig*** program provides output for this interface with the following structure:

```
Eth0      Link encap:Ethernet  HWaddr 00:00:1C:B5:F4:D1
          inet addr:192.168.2.35  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:15041342 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15484661 errors:0 dropped:0 overruns:0 carrier:0
          collisions:1705 txqueuelen:100
          Interrupt:10 Base address:0xa800
```

If these settings are correct, it should be possible to access the controller via the Ethernet network, as long as the controller's Ethernet interface has also been configured correctly. The best way to test this is by using the ***ping*** program. If this test is negative, there could be a problem with the network connection or the hardware being used.

Why can't a connection be established?

Although the controller can be reached via the ***ping*** program, the ***InaDriver*** and ***InaConnect*** programs cannot establish a connection with the controller and are displaying the connection status ***lost connection***. If this is the case, then check if the BR module ***fbtcpip.br*** is present on the controller and if Ethernet devices have been enabled for communication in the system setting (***sysconf***).

A connection is not established when accessing the controller via node number. In this case you should check if the correct broadcast address was used. In rare cases this address is 255.255.255.255, otherwise it results from configuration of the individual devices configured on the PC.

6.12 Notes on literature (InaDriver)

Additional information can be found in the following manuals:

1. Softing documentation for the PROFIboard (free download from Internet at www.softing.com). Detailed information is provided about the bus parameters and connection parameters as well as the values for the object descriptions.
2. The B&R PROFIBUS manual for the 2000 System describes the basics of PROFIBUS such as cable structure, network settings and connection parameters as well as a special section about the PROFIBUS hardware in the 2000 System.
3. PROFIBUS standard, DIN 19245 parts 1/2/3 and the corresponding EU standards

7 Modbus controller driver

7.1 General information about the Modbus controller driver

The Modbus driver is used to serially couple external stations to a B&R controller, using the Modbus RTU, or Modbus ASCII protocol. A mixed RTU/ASCII operation is not possible.

Make sure that the serial interface to be used is correctly configured in the properties of the CPU (sysconf).
The interfaces that are contained as standard in the CPU module do not have to be changed.

When using interface modules, which are run on a separate interface module slot, additional drivers may have to be installed.
***For example**, the device driver ddsfif6.br must be installed when using the interface module 3IF060.*

Details about the *sysconf* settings can be taken from the following screenshots:

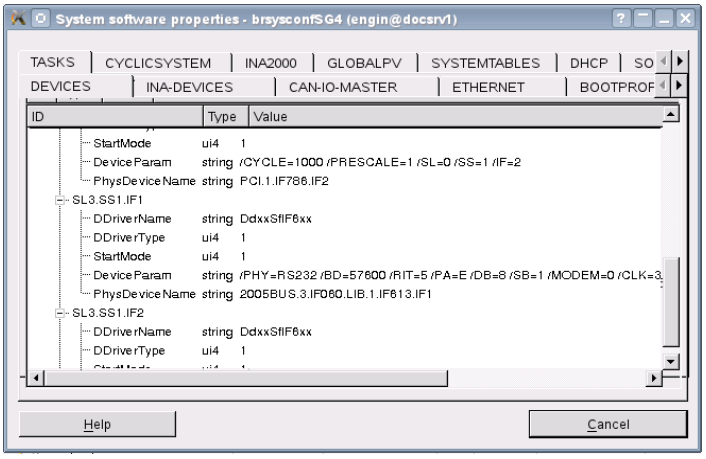


Illustration 17: Tab "DEVICES"

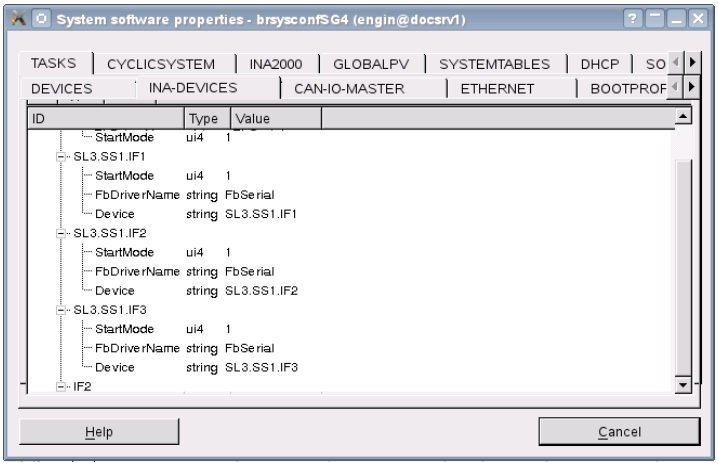


Illustration 18: Tab "INA-DEVICES"

The driver supports up to 4 serial interfaces, and can be configured as master, or slave, separately for each interface. In master mode, the driver processes its jobs cyclically, and in slave mode it only responds to request telegrams from the partner. The configuration of the

driver takes place in the **CaeManager** in the corresponding controller, in the tab '**APROL Couplings**'.

The B&R controller emulates the variable environment of a Modicon controller, meaning that variable ranges are available of the type Coils, Input Status, Input Register und Holding Register. The driver supports both 16 bit register, and 32 bit register access. For the latter, two 16 bit registers have been combined to one item.

*The **corresponding limit values must be observed** in the following description of the data module!*

It is presumed that the user has a basic knowledge about the Modicon system, in order to be able to use the different variable, or register, types.

Function codes supported by the driver *ApDrvMb*:

Function code	Description:	Range:	Limit value (Remote):
1	Read coil status	0xxxx	1-65535
2	Read input status	1xxxx	1-65535
3	Read holding registers	4xxxx	1-65535
4	Read input registers	3xxxx	1-65535
5	Force single coil	0xxxx	1-65535
6	Preset single register	4xxxx	1-65535
15	Force multiple coils	0xxxx	1-65535
16	Preset multiple registers	4xxxx	1-65535

It is necessary to define a relationship between the emulated data range, and the process variables that are used in APROL.

This is the only way to write to or read from these variables using the driver.

Only controller-global variables can be used!
The variables must be created in the configuration editor in the tab 'APROL Coupling'.

7.1.1 Key data of the Modbus controller driver

The following basic data is valid for the driver:

- The *ApDrvMb* driver supports the RTU mode, as well as the ASCII mode.
- The driver supports up to 16 serial interfaces.
- The number of coil, and input, status, holding, and input, register can be configured. However, the total amount cannot exceed 2048 per type.
- The driver supports cyclic communication in master mode, and in the slave mode it is totally passive.

7.2 Data module structure

The Modbus control driver is configured using a data module called ***ApCnfMb***.

The data module is created with a build of any one task belonging to this controller.

This data module contains information about the corresponding interfaces that are used by the Modbus station, as well as variable assignments to the virtual Modbus data areas. Here, one has to pay attention to the number of emulated Modbus stations at one time. The variable list describes how to interpret a variable in a Modbus data module. Downloading a new data module results in a new reconfiguration. The Modbus driver is stopped if the data module is not present.

The data module has the following structure:

```
"ApCnfMb"
"<MODE>:x"
"<COMM>:X:PARA:ADDR:obs:To1:C:I:H:R:Mode1:To2:Mode2"
"<AMPLE>:AMPLE-TIME"
"<COMM_STAT>:VAR"
"<COMM_CNT>:VAR"
"<C>:VAR:OFF"
"<I>:VAR:OFF"
"<H>:VAR:OFF:RemType"
"<R>:VAR:OFF:RemType
"<REQ>:ADR:FCT:OFF:NUM:ROFF:CYCLE"
"<REQ_STAT>:VAR"
"<REQ_CNT>:VAR"
```

The COMM identifier is contained once in the data module, for each emulated Modbus station.
Each entry is composed of an identifier, which can be recognized by the angle brackets, as well as the accompanying settings.

Conventions for text:

:	Colons separate the individual parameters.
;	Semicolons follow comments.
"	Entries in the data module are indicated by quotation marks.

The driver does not run with a misconfiguration. In this case an entry is written into the logbook of the controller with a note about the faulty line in the data module!

7.2.1 Description of the data module entries

"ApCnfMb"

Entry:	Description:	Limit values:
ApCnfMb	Identification for the Modbus driver	constant

"<MODE>:X"

Entry:	Description:	Limit values:
<MODE>	Identifier	-

X	**0=somePVs** (The driver should function even if all of the configured variables have not been found.) **1=allPVs** (The driver should only function if all of the configured variables were able to be assigned.)	0 **or** 1

"<COMM>:X:PARA:ADDR:obs:To1:C:I:H:R:Mode1:To2:Mode2"

Entry:	Description:	Limit values:
<COMM>	Identifier	-
X	Interface name (e.g. IF1)	-
PARA	Interface parameters are composed of: Mode, baud rate, parity, data bits, stop bits (are automatically established by means of the parity, and inserted)	-
ADDR	Own Modbus address	1 to 247
obs	Present due to compatibility reasons with older driver versions. Entry is not evaluated.	-
To1	Timeout (maximum waiting time for an answer telegram in <ms>	-
C	Number of emulated coil status variables from this station	0 to 2048
I	Number of emulated input status variables from this station	0 to 2048
H	Number of emulated holding register variables from this station	0 to 2048
R	Number of emulated input register variables from this station	0 to 2048
Mode1	These settings are only relevant for the slave mode S=16 bit mode (A register item is composed of one data word !) X=32 bit mode (One register item is composed of two data words, meaning in 32 bit mode that double as many offsets are used locally as for the 16 bit mode)	S **or** X
To2	Slave suspension time in seconds This parameter is only relevant for master mode. If the corresponding slave does not respond to demands, then the corresponding jobs are not carried out for this period. Through this, the communication is balanced.	-
Mode2	Modbus protocol	RTU or ASCII

"<AMPLE>: AMPLE -TIME"

Entry:	Description:	Limit values:
<AMPLE>	Identifier	-

AMPLE -TIME	Delay time in [ms] from the reception of an answer until the next request telegram. Only for the master mode.	0 **... 1000**

"<COMM_STAT>:VAR"

Entry:	Description:	Limit values:
<COMM_STAT>	Identifier	-
VAR	Variable name Used to monitor an interface. The variable toggles each second between "0" and "1" if the respective interface has been opened without error.	-

"<COMM_CNT>:VAR"

Entry:	Description:	Limit values:
<COMM_CNT>	Identifier	-
VAR	Variable name **In slave mode:** This status variable is a counter that is incremented after a request has been received without error. **In master mode:** This status variable is a counter that is incremented after a request has been processed without error.	-

"<X>:VAR:OFF:RemType"

Entry:	Description:	Limit values:
<X>	Type: C = Coil status I = Input status H = Holding register R = Input register	-
VAR	Variable name	-
OFF	Offset in the temp. buffer (local)	The number of declared registers in the <COMM> identifier cannot be exceeded

RemType	Valid only for holding and input, registers. BIT0-15 → corresponding BITs of a data word e.g. BIT7 HINT8 → most significant Byte of a data word: signed HUINT8 → most significant Byte of a data word: unsigned LINT8 → least significant Byte of a data word: signed LUINT8 → least significant Byte of a data word: unsigned INT16 → Data word (signed) UINT16 → Data word (unsigned) INT32 → Double Data word (signed) UINT32 → Double Data word (unsigned) IEEE → Floating point number (IEEE format) STRING →64 character inclusive termination = 65 Bytes (is the same as 33 data words)	-

The Modicon addressing starts at the basis address 1.
E.g. Input status 1 (10001) is equivalent to Offset 0; Holding register 2 (40002) is equivalent to Offset 1

"<REQ>:Add:Fct:off:len:rem_off:time:Mode"

Entry:	Description:	Limit values:
<REQ>	Identifier for a task line of a master	-
Add	Address of the slave station from which should be read from, or written to.	1 to 247
Fct	**Function**: 1: Read Coil Status 2. Read Input Status 3: Read Holding Registers 4: Read Input Registers 5: Force Single Coil (Write) 6: Force Single Register (Hold.Reg.) 15: Force Multiple Coils (Coil Status) 16: Force Multiple Registers (Hold.Reg.)	-
off	**Data offset in the local register** There is always a BIT or WORD offset depending ón the mode (16|32 bit). Possible values are [0 ... 2047]. **Example:** Holding register '1' (40001) corresponds to offset '0'.	The number of declared registers in the <COMM> identifier cannot be exceeded

len	Number of items, which can be transferred with this job (maximum 250 Bytes) Dependent on the function, and the mode of the communication partner (16, or 32 bit mode) that are maximum possible. 1: Read Coil Status 2. Read Input Status 3: Read holding registers 4: Read input registers 5: Force Single Coil (Write), 6: Force Single Register (Hold.Reg.) 15: Force Multiple Coils (Coil Status) 16: Force Multiple Registers (Hold.Reg.)	 Range 1 - 2000 Range 1 - 2000 for 16 bit: 1- 125, for 32 bit: 1 - 62 for 16 bit: 1- 125, for 32 bit: 1 - 62 1 1 Range 1 – 2000 for 16 bit: 1- 125, for 32 bit: 1 - 62

When using the 16 bit mode, the sum of <off> and <len> is not allowed to exceed the number that is entered in <COMM> identifier tab.

*When using the 32 bit mode (only with holding and input registers), the sum of <off> and (<len>*2) is not allowed to exceed the number that is entered in <COMM> identifier tab.*

Entry:	Description:	Limit values:
rem_off	**Data offset on the conneceted Modbus station** Possible values are [0 ... 9999] **Example:** Holding register '1' (40001) corresponds to offset '0'.	0 to 65535
time	Cycle in <ms>	-
Mode	S=16 bit mode (a register item is composed **of one data value**) X=32 bit mode (One register item is composed of two data words, i.e. double as many offsets are possible locally in 32 bit mode as for the 16 bit mode)	S **or** X

"<REQ_STAT>:VAR"

Entry:	Description:	Limit values:
<REQ_STAT>	Identifier	-

VAR	Variable name Value of the status variable: 0: OK 2: Timeout error 3: crc error 4: data to short 5: IF error 6: unexpected len 7: response error Fct: 5 ,6, 15, 1	-

"<REQ_CNT>:VAR"

Entry:	Description:	Limit values:
<REQ_CNT>	Identifier	-
VAR	Variable name This status variable represents a counter that is incremented after the respective request has been processed without error.	-

7.3 Creating the data module with the configuration editor

The controller driver *ApDrvMb* is configured in the **CaeManager** in the 'controller' project part, in the '**APROL coupling'** tab.

To create a new configuration of the *ApDrvMb* driver for your controller, select the entry '**MODBUS coupling**', and call up the menu item '**New**' from the short-cut menu with the right mouse button.

*The coupling name **ApCnfMb** is automatically allocated, **and cannot be changed**.*

Finally, create a new Modbus station using the menu item "**New**" in the short-cut menu, and assign a station address to it.

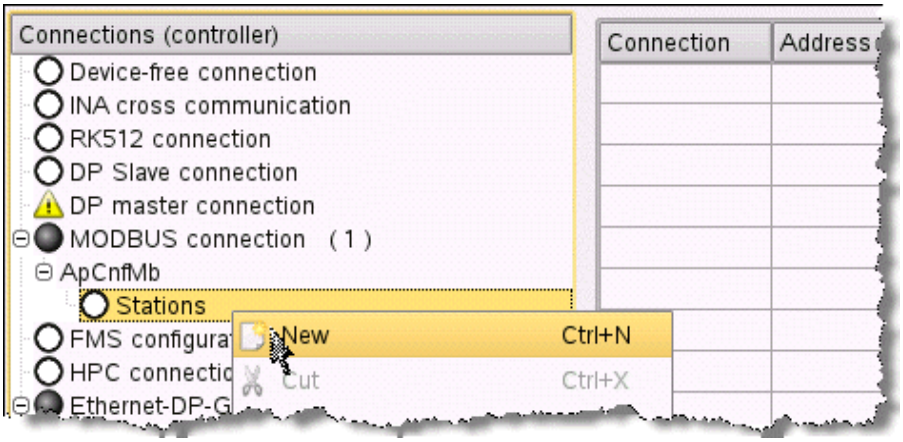


Illustration 19: Creating a new Modbus station

Hereupon, the global settings of the Modbus station must be adjusted (e.g. the interface description, as well as the number of registers to be supported). A short description, as well as possible parameter limit values (e.g. number of holding registers --> 2048) can be found in chapter *Description of data module entries*.

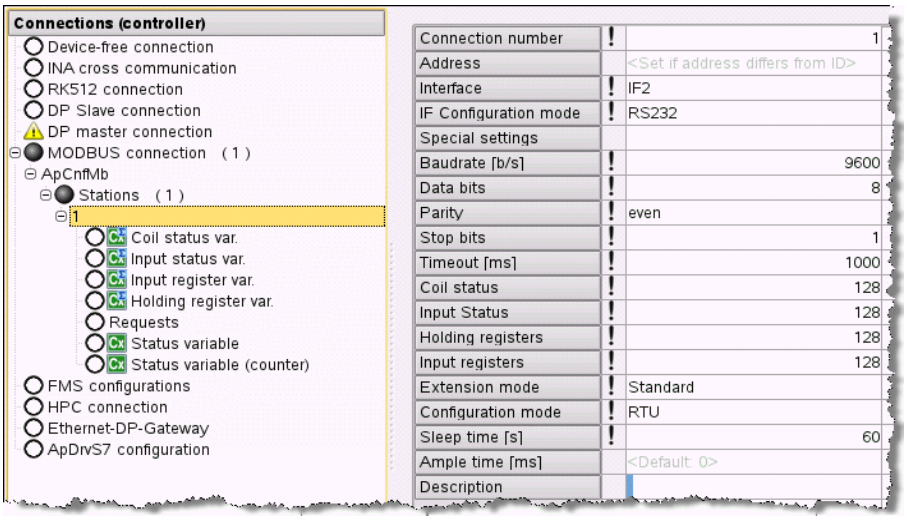


Illustration 20: Entry in the Modbus station global parameters

Finally, carry out the assignment of the variables to the corresponding Modbus registers.

A configured input variable (I/O type) for the Modbus driver is not allowed to be described in the logic diagrams.

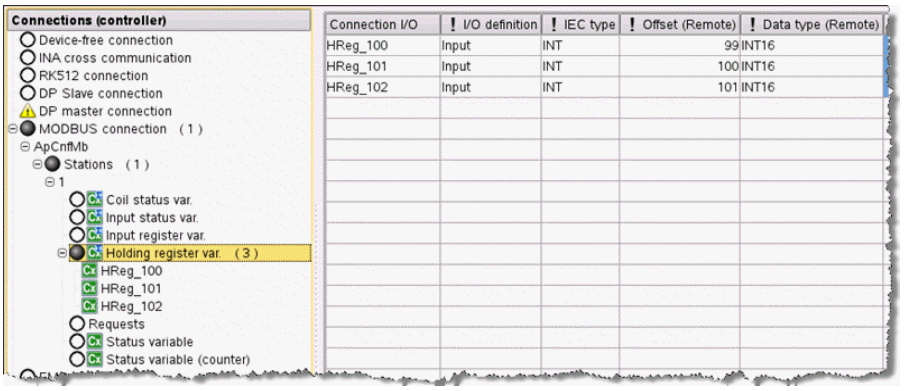


Illustration 21: Assignment of the PVs to the corresponding Modbus data areas

If the created Modbus station should be run as master, then one must create the necessary jobs, otherwise the station will be run as slave.

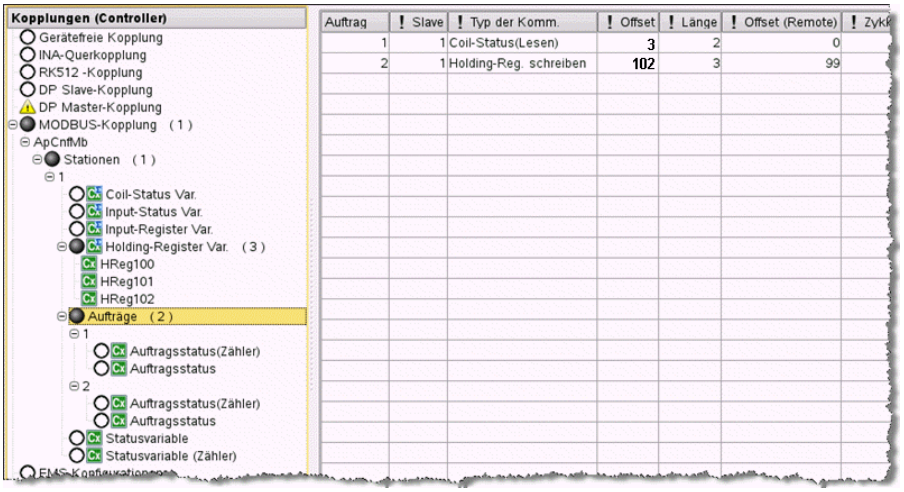


Illustration 22: Create the task:

The before mentioned status variables <*REQ_STAT*> and/or <*REQ_CNT*> can be optionally created for each job.

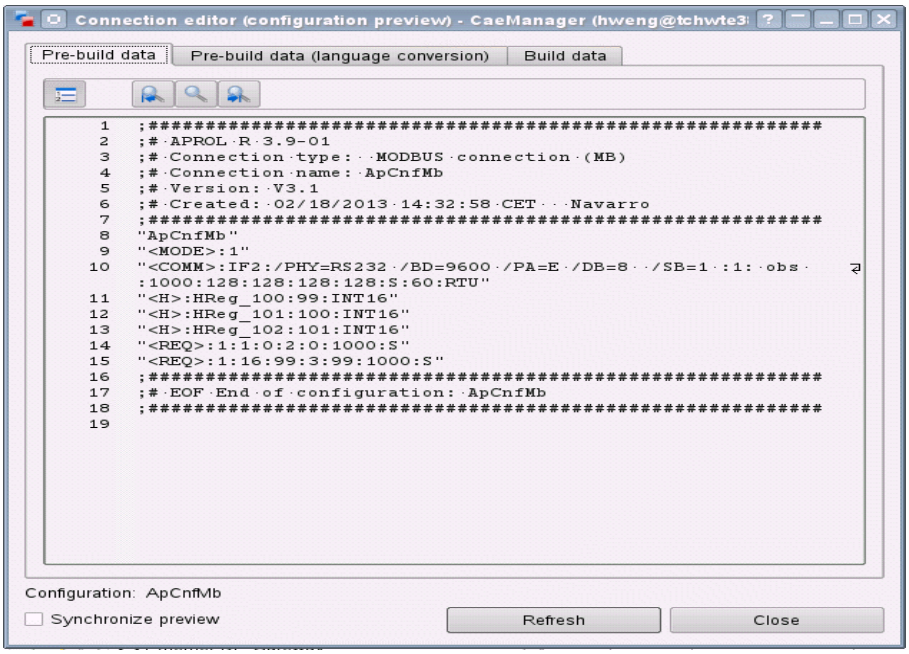


Illustration 23: Preview display of the created Modbus coupling

7.4 Example

Parameters from the configuration example:

A master-capable Modbus station must be created on the interface IF2 of the CPU.

Two inputs (coil status) of the controller to be connected should be read every second from it.
Input 1 (addr 10001), input 2 (addr 10002).

Additionally, a status variable (MB_Cnt) should be incremented for each job that is executed successfully from these inputs.

Three set values are written cyclically:
Holding register 100 (adr 40100), 101 (adr 40101) und 102 (adr 40102)

The Modbus address of the slave to be connected is "2".

The life signal of the driver should be displayed with the help of the MB_Life variables.

The temporary data areas should be configured as follows:
Coils →0
Inputs →10
Holding Register →200
Input Register →0
Protocol: RTU

The settings for the interface being used:
Mode RS232, baud rate 9600 bit/sec, no parity, 8 data bits, 2 stop bits, time out 1 second.

The AMPLE-TIME should be at least 100 ms.

The configuration data module has the following contents:

```
;############################################################
;# APROL R 3.6-0492
;# Coupling type:  MODBUS-Kopplung (MB)
;# Coupling name: ApCnfMb
;# Version: V3.1
;# Created: 15.06.2011 11:14:18 CEST   Schulte
;############################################################
"ApCnfMb"
"<MODE>:1"
"<COMM>:IF2:/PHY=RS232 /BD=9600 /PA=E /DB=8  /SB=1 :1: obs :1000:128:128:128:128:S:60:RTU"
"<AMPLE>:100"
"<C>:ModInput_1:0"
"<C>:ModInput_2:1"
"<H>:HReg100:99:INT16"
"<H>:HReg101:100:INT16"
"<H>:HReg102:101:INT16"
"<REQ>:2:1:0:2:0:1000:S"
"<REQ_CNT>:MB_Cnt"
"<REQ>:2:16:99:3:99:2000:S"
"<COMM_STAT>:MB_Life"
;############################################################
;# EOF End of the configuration: ApCnfMb
;############################################################
```

* These variables can also be read by the master. Whether these variables are read or written depends only on the function code that is received. The *Holding register* and *Coils* can be read and written.

7.5 Modbus controller driver status variables

This description is under construction at present.

*Please inform yourself in regular intervals about the current APROL documentation on our internet page **www.br-automation.com**, in the area Material related downloads.*

8 ModbusPlus driver package

8.1 General information about the ModbusPlus driver

Unlike the single master system *Modbus*, the *Modbus-Plus* network is a multi-master system with token access methods (i.e. only the request telegram contained by the master token can be sent). 32 network nodes are allowed per network segment and the network can consist of up to 5 segments.

The software package APROL *ModbusPlus driver* consists of a kernel driver for the Linux kernel 2.6, a control computer driver and a diagnostics tool (*mbpManager* and *kMbpManager*).

The *ModbusPlus driver* requires an additional *ModbusPlus* card for the PC. Further information about this can be taken from chapter *Supported hardware*.

The driver is configured using the CaeManager.

8.1.1 Contents of delivery ModbusPlus

The software package is delivered as an RPM file. The following files are created when installing the package:

File name	Description
/etc/init.d/SA85-PCI	Start script for installing the kernel driver
/lib/modules/*/kernel/drivers/aprol/SA85-PCI.ko	Kernel driver for different kernel versions (*corresponds to the output from the command "uname –r", which displays the kernel version).
/opt/aprol/bin/mbpInstall	Tool for entering the network address of the card in the ModbusPlus network.
/opt/aprol/bin/mbpDriver	APROL-Driver on the control computer
/opt/aprol/bin/mbpManager	Diagnostics tool (console version)
/opt/aprol/bin/kMbpManager	Diagnostics tool (console version)
/opt/aprol/cnf/mbpDriver/examples/mbp.cnf	Example configuration for the driver on the control computer

8.1.2 Supported hardware

Currently, the kernel driver only supports the ModbusPlus card from *Schneider Electric* with the type ID ***416NHM30030A***. This is a single-channel PCI card for 5 and 3.3 volts.

8.2 Configuration of the APROL driver on the control computer

The configuration of the driver on the control computer takes place in two steps:

The driver's start options must be configured in the CaeManager and a configuration file must be created including all of the information about the process variables.

If the start options and/or the configuration file are changed, then a runtime download and the generation of a control computer task are necessary!

The following table contains a list of the start options and the respective descriptions:

In addition to the options provided in the CaeManager for configuring the driver, the following table also lists options that can be only called from the console for debugging purposes!

A description of the ModbusPlus driver launching options can be found in manual 'X99 CC Modules', chapter ModbusPlus driver launching options.

In addition to configuring the start options in the CC modules, a driver's tasks also have to be configured. This takes place in the CaeManager, 'APROL system' project part, *APROL connections,* in the *MbpDriver* entry.

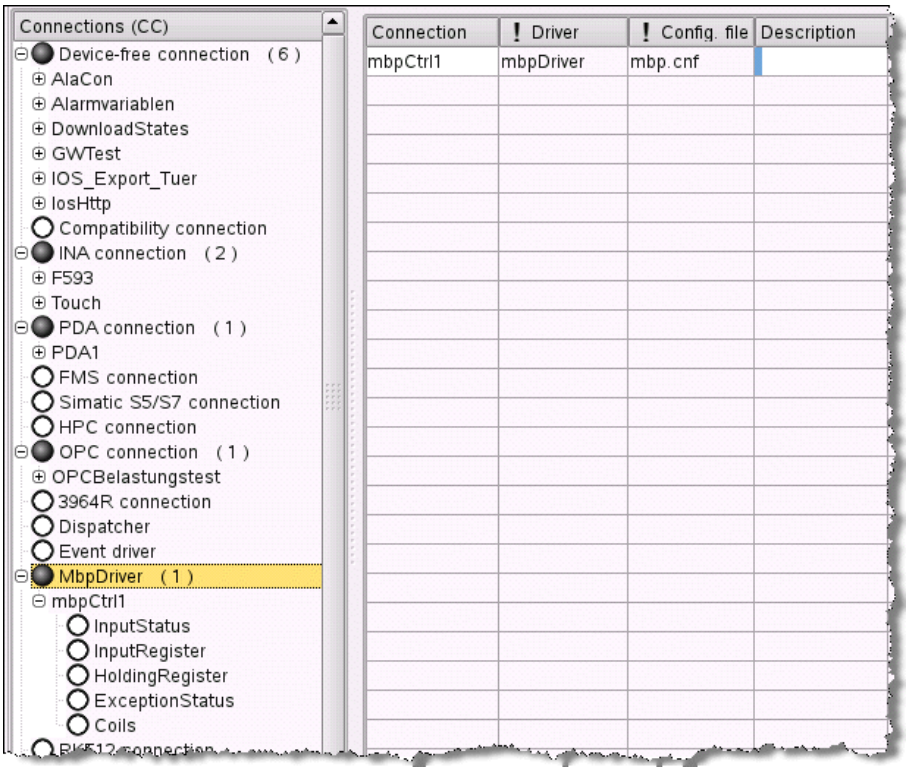


Illustration 24: Configuring the tasks

The different cyclic and non-cyclic task types provided are described here below:

Task type	Description
READ	Cyclic read task.
WRITE	Event-controlled write task, which only writes the most recently changed area to the controller (i.e. each PV is separately transferred to the controller in individual write tasks).

Task type	Description
SYNC	Cyclic read task with event-controlled writing. This is a combination of the two types of tasks mentioned above. *A PV changed in the control computer remains locked for read tasks until all of its value changes have been written to the controller. Write tasks are then only processed once they have been successfully read at least one time.*
IWRITE	**InitWrite**: A task that reads from the controller once after a connection has been established. All controller variables that have different values than the respective control computer variables are overwritten with the control computer's values. A setting of the controller variables on the control computer does not take place!
IBWRITE	**InitBlockWrite**: Comparable with the task type IWRITE. Instead of individual PVs, the entire block is transferred to the controller.
BWRITE	**BlockWrite**: Comparable with the task type WRITE. The entire block is transferred to the controller when value changes are made to individual PVs.
BSYNC	Combination of READ and BWRITE. Reading and writing complete tasks instead of individual PVs.

The task types READ and (I)WRITE should be primarily used to cut down on system load because it is not possible to guarantee that multiple value changes can be transferred in one telegram during block write procedures!

To create a new ModbusPlus connection, select the menu option **"New"** from the *MbpDriver* shortcut menu in the CaeManager.
You are then prompted to enter a directory name.

Enter the name provided in the *-controller* start option here as name for the configuration directory.

Tasks can then be created for the different variable types in the ***Modicon*** controller. The variable types include "*Coils*", "*InputStatus*", "*InputRegister*", "*HoldingRegister*" and "*ExceptionStatus*".

After selecting the desired variable type, you can use the shortcut menu (menu option "**New**") to start creating variables that are automatically given a numeric identifier.

For analysis purposes, names can be assigned to process variables in which the driver enters counter values for tasks that were and were not successfully completed.

These variables are handled internally like "*unsigned short*" variables. That means that their value range is between 0 and 65535. These process variables are handled according to their task and might have to be created individually for each task. If these have not been configured, then statements about tasks that are *successfully completed* or *not successfully completed* are not possible.

Furthermore, a cycle time must be assigned in milliseconds for each task. This cycle time is not important for write tasks because true write tasks are always event-driven (and never cyclic).

The cycle time for the READ section is used for mixed tasks (SYNC). The WRITE section here is also always event-driven.

The process variables must then be created for each task. The name, the address on the controller, the type on the controller and (if necessary) the scaling information must be specified for each PV. The direction of data is determined by the respective task type.

Only PVs with the type BOOL can be assigned for Boolean tasks. Any PV can be assigned for register tasks.

The address on the controller is dependent on the respective task type:
Coils are between 1 and 10000, *InputStati* between 10001 and 20000, *Input-Registers* between 30001 and 40000 and *Holding-Registers* starting at the address 40001. There are different maximum transfer sizes for the different variable types depending on the type of task (see the following table).

Variable type	Address range	Maximum task size (number of items)
Coils	1 – 10000	READ: 2000 BWRITE: 800
Input-Status	10001 – 20000	READ: 2000 BWRITE: -
Input-Register	30001 – 40000	READ: 125 BWRITE: -
Holding-Register	40001 -	READ: 125 BWRITE: 100
Exception-Status	No address	1 byte

The maximum task size for *BSYNC* tasks is determined by the write section.

Furthermore, the limit values for the control computer sizes, and the limits for the controller sizes can be specified for scalable variables. If all four sizes are specified, then the following formulas apply for the value transfer:

Fur read tasks:

$$IosValue = IosMin + \frac{PlcValue - PlcMin}{PlcMax - PlcMin} * (IosMax - IosMin)$$

For write tasks:

$$PlcValue = PlcMin + \frac{IosValue - IosMin}{IosMax - IosMin} * (PlcMax - PlcMin)$$

IosMin and *IosMax* are the limit values in the Iosys (control computer). *controllerMin* and *controllerMax* are the limit values on the controller.

The control computer sizes are interpreted as the minimum and maximum values. Control computer values that are smaller than the minimum value and larger than the maximum value are reset to the respective limit value before being transferred. If no limits are specified, then the limits of whichever variable type is used apply.

You can also use the sample configuration file for reference.

8.3 kMbpManager and mbpManager

The utilities *kMbpManager* and *mbpManager* are applications used for diagnostics purposes. The *kMbpManager* can only be used with a GUI (together with a graphic interface). The *mbpManager* runs as an application on the console.

The following section will only describe the *kMbpManager* because both utilities provide essentially the same functionalities, although with different menus.

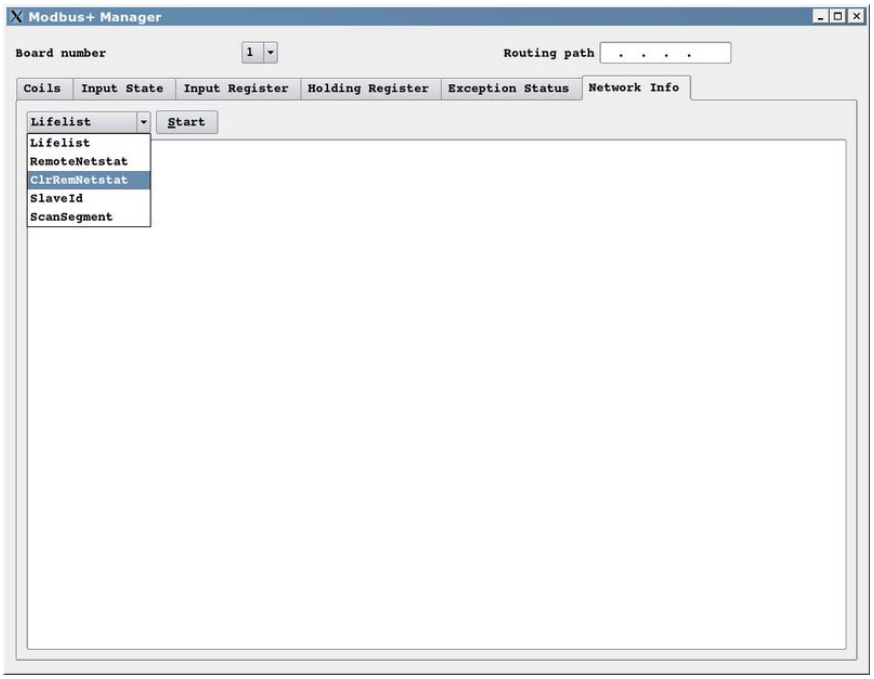


Illustration 25: The mbpManager

The *kMbpManager* requires the number of the ModbusPlus interface in the computer. **Unlike the APROL driver, the first board is addressed with the number 1 here.**

The routing path to the target device must be entered for functions called via the network. The *Lifelist* function always runs locally and does not require specification of a routing path.

Brief description of the functions:

Function	Description
Network Info/Lifelist	Displays all of the station addresses located on the local network.
Network Info/RemoteNetstat	Network statistics on the target address.
Network Info/ClrRemoteNetstat	Deletes the network statistics on the target address (if this is supported at the target address)
Network Info/SlaveID	Current settings / status of the target address (mem protect, run state, etc.)

Network Info/ScanSegment	Checks the next sub-network behind the target station. This is essentially used to display a life list behind a gateway. **Note:** The status of the routing path changes while the scan is running, (i.e. you can see which address is currently being checked). **Example:** Routing path: 10.00.00.00.00 Checks all addresses between 10.01.00.00.00 and 10.64.00.00.00.
Coils/Read	Reads Coils: The number of the first Coil and the amount of Coils to be read must be specified. If any of the entries are incorrect, then the read button is disabled. The read procedure is executed cyclically once every second until the stop button is pressed.
Coils/Write	Writes one or more Coils (with identical value). **Note:** Under certain circumstances, written values are immediately overwritten by the controller's logic. Therefore, they cannot be read during the next read procedure!
InputState/Read	Reads the input states: Behaves the same as Coils/Read.
InputRegister/Read	Reads the *input states*: Behaves the same as Coils/Read.
HoldingRegister/Read	Reads holding registers: Behaves the same as Coils/Read.
ExceptionStatus/Read	Reads the status bytes cyclically from the target station until the stop button is pressed.
HoldingRegister/Write	Writes one or more Holding Registers (with identical value). Compare the note for Coils/Write.

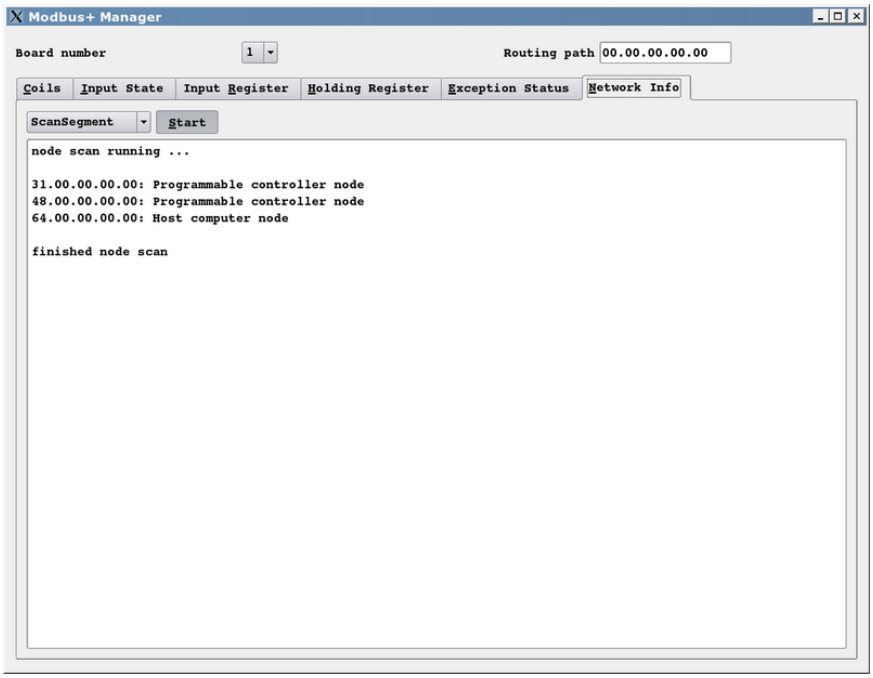


Illustration 26: Completed network scan

8.4 Possible diagnostics when implementing the driver on the control computer

Each instance of a ***ModbusPlus driver*** creates a few PVs in the Iosys for diagnostics purposes:

PV name	Description
AppName_successCtr	Type: unsigned short Increased by 1 with each successfully completed request Value range: 0 .. 65535
AppName_failedCtr	Type: unsigned short Increased by 1 with each request that is **not** successfully completed Value range: 0 .. 65535
AppName_errorText	Type: String Process variable where the driver writes error messages.
AppName_connStatus	Type: Integer Value range: 1 = Driver not connected 2 = Driver connected **Note:** "Not connected" is displayed when the local controller returns the message "Cannot connect to host". "Connected" is displayed as soon as a task was able to be sent successfully.

AppName_pid%d_useLogfile	Type: String *%d* stands for the process ID of the driver output with ps ax. When a valid file name (including path) is set using pio, the driver creates a log file of the name and directs all enabled debug output to this log file. If a log file was already open, then it is first closed and can be deleted if necessary. If an empty string is transferred, then the current log file is closed and can be deleted.
AppName_pid%d_debugFilter	Type: Integer (internal unsigned int) %d stands for the process ID of the driver output with ps ax. Debug output can be enabled or suppressed by setting or clearing individual bits. Debug output is placed (when in debug mode) in the console or in the log file, if open. The following debug bits are available: 0x00000001: Displays error messages 0x00000002: Displays normal messages 0x00000008: Displays the configuration 0x00000080: Displays the variable names in the Iosys (only at startup) 0x00000100: Hex display of incoming telegrams 0x00000200: Hex display of outgoing telegrams 0x00000400: Output of Iosys values **Note:** After being set, the config bit (value 8) is automatically reset once the configuration has been displayed.

If debug output is directed to a log file, then it is absolutely necessary to monitor the available memory space on the hard drive.
Computer failure is a possible result of insufficient memory space!

The driver can be started in debug mode from the console by setting the *-d* option with a debug filter made up of enabled debug bits. As a result, the driver does not go into the background and outputs debug messages directly to the console. This makes it possible to quickly detect errors when starting up. Once all of the errors have been eliminated, then the driver should be started normally using the ***StartManager***!

If errors occur during normal operation, then it is recommended to create a log file and enable debug output. **After a while, the log file should be closed again!**

```
/opt/aprol/bin/pio –pv MbpDriver_mbp1_useLogfile -s "/tmp/myLogfile" -setType 3
```

creates a log file called `myLogfile` in the `/tmp` directory for the driver with the cnf file in *mbp1* (–*controller* option).

```
/opt/aprol/bin/pio –pv MbpDriver_mbp1_debugFilter –s 3 –setType 1
```

enables error and status messages.

```
/opt/aprol/bin/pio –pv MbpDriver_mbp1_debugFilter –s 0 –setType 1
```

ends all debug output.

```
/opt/aprol/bin/pio –pv MbpDriver_mbp1_useLogfile –s "" –setType 3
```

closes the log file. It can now be analyzed and then deleted.

8.5 Driver configuration example (ModbusPlus)

```
# created 2005.08.16 11:46:29 by Admin

^READ, INPUT_STATUS, 1000
 :REQ_POS_CTR, mbpDrv1_IS1_posCtrPv
 :REQ_NEG_CTR, mbpDrv1_IS1_negCtrPv
    IS1, 10001, BOOL
    IS2, 10003, BOOL
    IS3, 10005, BOOL
    IS4, 10006, BOOL

^READ, INPUT_REGISTERS, 1000
 :REQ_POS_CTR, mbpDrv1_IR1_posCtrPv
 :REQ_NEG_CTR, mbpDrv2_IR1_negCtrPv
    IR1, 30001, INT
    IR2, 30002, DINT
    IR3, 30005, UDINT
    IR4, 30010, REAL , -10000 , 10000 , -1000 , 1000

^READ, HOLDING_REGISTERS, 10000
 :REQ_POS_CTR, mbpDrv1_HR1_posCtrPv
 :REQ_NEG_CTR, mbpDrv1_HR1_negCtrPv
    HR1, 40001, BYTE
    HR2, 40002, BOOL
    HR3, 40005, DINT
    HR4, 40020, REAL , -10000 , 0 , 0 , 10000

^READ, EXCEPTION_STATUS, 60000
 :REQ_POS_CTR, mbpDrv1_EX1_posCtrPv
 :REQ_NEG_CTR, mbpDrv1_EX1_negCtrPv
    EX1, 1, SINT

^READ, COILS, 1000
 :REQ_POS_CTR, mbpDrv1_RC1_posCtrPv
 :REQ_NEG_CTR, mbpDrv1_RC1_negCtrPv
    Coil1, 1, BOOL
    Coil2, 2, BOOL
    Coil3, 3, BOOL
    Coil4, 4, BOOL
    Coil5, 5, BOOL
    Coil6, 6, BOOL
    Coil7, 7, BOOL
    Coil8, 8, BOOL
    Coil9, 9, BOOL
```

This example shows cyclic *READ* tasks with different cycle times between 1000 milliseconds and 60000 milliseconds. All tasks have status variables for the positive and the negative counter that can be displayed in the IosEv, for example.

IR4 is stretched by a factor of 10 in the control computer. *HR4* is moved from a negative to a positive value range.

8.6 ModbusPlus driver status variables

This description is under construction at present.

*Please inform yourself regularly about the current APROL documentation on our internet page **www.br-automation.com**, in the Material related downloads section.*

9 OPC server

9.1 Definition of terms for OPC

As a result of the widespread usage of Windows operating systems worldwide and the standardization in the area of PCs, different technologies have emerged for allowing software modules to communicate with each other using standardized interfaces.

One of the first milestones on this path was **DDE** (**D**ynamic **D**ata **E**xchange), which was later replaced by the considerably more powerful **OLE** technology.

The abbreviation **OLE** stands for **O**bject **L**inking and **E**mbedding and describes the dynamic linking of objects in various office applications.

The term **OPC** was coined in the mid 1990s, when a number of well-known companies, mainly software manufacturers, came together to form a task force for the rapidly expanding sector of SCADA systems using HMI interface (e.g. Fisher-Rosemount, Rockwell, Siemens, etc.).

Against the background of the growing number of different communication protocols and bus systems, the OPC task force set themselves the goal of developing a standard for accessing real-time data using Windows operating systems. A basis, **O**LE for **P**rocess **C**ontrols (**OPC**), was hoped to be created.

Since its establishment in 1996, the *OPC Foundation* has had the goal of accommodating the needs of the industry and reflecting them in the form of functional extensions and existing or new *OPC specifications*.

Further information about the *OPC Foundation* can be found on the Internet at:

http://www.opcfoundation.org

9.1.1 General Information about OPC

For manufacturers of *OPC servers* and *OPC clients*, the use of *OPC base technology* allows greater freedom concerning complexity, functionality and implementation of their *OPC components*.

Different types of *OPC components* from various manufacturers can be used together as long as the specifications and guidelines for implementation are adhered to.

No additional programming is required for adapting the interfaces between the individual components.

For example, process data from the field can be displayed in an Excel spreadsheet using *OPC*. This process data can be archived effortlessly in a database using *OPC* or further processed in a production planning system.

The APROL OPC server offers you the following benefits:

- OPC is quicker and more available than DDE
- Each value that is recorded with OPC has a time stamp and a status.
- OPC uses a block format for communicating and can send several requests in one single call, and receive several values.

- OPC allows the clients to specify the scan rates that are needed.
- In contrary to DDE, OPC uses binary data representation (IEEE floating comma numbers, Integer, ...)
- A server's variables can be observed and chosen with a browser.
- The concept of building groups allows different variable blocks with different scan rates.

The APROL OPC server supports the data access specification 2.0.

It concerns the implementation of so called "Custom Interfaces", which as opposed to "Automation Interface", work using function pointers.

The specification describes "Mandatories"; methods that must be implemented, and "Optionals", methods that can be implemented, for this implementation. The APROL-OPC-Server does not support the method IPersistsFile from the "Optionals" IOPCServerPublicGroups, IOPCBrowseServerAddressSpace and IPersistFile, and returns a corresponding error message when calling it. There is a so called AutomationWrapper from the OPC foundation for applications that cannot access using the custom Interface (e.g. Excel-DataAccess-Clients und VisualBasic-Clients, so clients that cannot work with function pointers), which allows access to the custom interface.

A description of the OPC standard is not included in this documentation!

9.1.2 Information about the APROL OPC server

The APROLFehler! Kein Name für eine Eigenschaft übergeben. OPC server allows process variables from process control system created with APROLFehler! Kein Name für eine Eigenschaft übergeben. to be evaluated on a Windows based system using a standardized OPC interface.

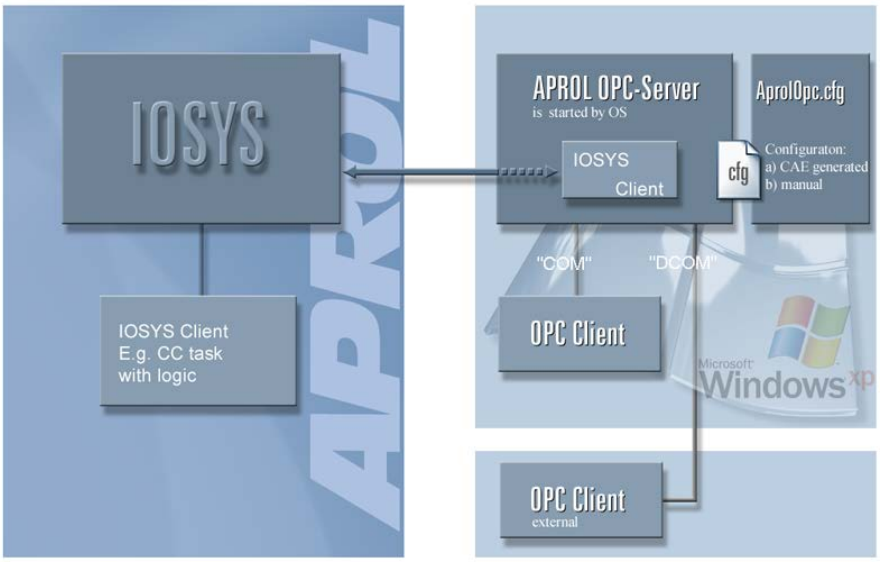


Illustration 27: Overview of the APROL OPC server

The *OPC server* was implemented as an *OPC data access server*. It works as an Iosys client and makes it possible for *OPC clients* (applications) to have **read and write** access to process variables (PVs) on the process control system.

The APROLFehler! Kein Name für eine Eigenschaft übergeben. *OPC server* is an Iosys OPC interface. All variables that are accessed using OPC have to be defined in the respective configuration file (see chapter *Structure of the configuration file*).

The variables that are communicated via OPC receive a time stamp from the APROL OPC server. This time stamp is used in applications such as the TrendViewer or alarms, and should therefore be identical with that of the Iosys (see chapter *APROL OPC server as NTP client*).

The APROLFehler! Kein Name für eine Eigenschaft übergeben. *OPC server* is currently available for **Microsoft Windows XP**, **Windows 2008 server,** and **Windows 7**.

The server is started by starting an OPC client. The OPC server is ended when the last active client closes.

Possible clients are third-party clients, self-made OPC clients with Visual Basic (also Office applications), or the OPC sample client of the PVI Development Setup (can be downloaded from the B&R homepage).

*The OPC server is only called by the operating system after starting a clients and is **not** permitted to be started with a "double-click" on AprolOPC.exe.*

Term	Description
COM	It must be understood how platforms interpret an object in order to make objects that have been implemented on different platforms or computer architectures compatible with each other. For this purpose, a so called object model is necessary. OLE uses the COM model (Component Object Model). It defines a standard for the cooperation of the components. COM allows calls within one process, and to another process.
DCOM	The object model for computer-spanning calls is called DCOM (Distributed Component Object Model) and is integrated as of the operating system Windows 2000.

Performance specifications:

The APROL **Fehler! Kein Name für eine Eigenschaft übergeben.**OPC server guarantees read access to 5000 process variables within one second. The **Fehler! Kein Name für eine Eigenschaft übergeben.** OPC server guarantees read access to 5,000 process variables within a second.

*During the configuration it should be noted that the APROL **Fehler! Kein Name für eine Eigenschaft übergeben.**OPC server functions in a **polling** manner, whereby the Iosys is **event-driven**.*

Important information about the ports to be used:

The **ports which are necessary** for the communication between the Iosys on the runtime system and the OPC server are set up automatically during the installation of the APROL system software.

In APROL versions lower than R 3.6-10, the 'AprolConfigureNewPortForwarding --enable' command must be executed on a terminal with root rights. The script is supplied on demand by B&R.

As of APROL R 3.6-03, only the ports which are necessary for an APROL system are not blocked by the firewall. Ports that are not necessary are locked.

The B&R default configuration must not normally be adjusted.

Detailed information about the firewall configuration can be found in the manual 'A2 Getting Started', chapter Optional Adjustments to the Firewall.

If the port numbers have been adjusted manually, they must be readjusted manually after each new installation. The same applies for an APROL and OPC server installation.

9.2 Installing and registering the OPC server

The APROLFehler! Kein Name für eine Eigenschaft übergeben. OPC server is installed with the help of a Microsoft standard installation package.

The installation file is called '`APROL-OPC-Server-Installer.exe`'.

It is necessary to work with administrator rights for an installation in the Windows 7 operating system. For this, select the installer with the right mouse button and choose 'Run as administrator'.

The installation package consists of the server itself, which is used by libraries, as well as an example configuration file (`AprolOPC.cfg.example`).

The following subdirectories are created in a directory that can be chosen by the user during the installation.

As default, the Setup Wizard uses "C:/APROL" for the installation directory <InstDir>.

The following subdirectories are automatically created:

- **`<InstDir>/bin`** contains the APROLFehler! Kein Name für eine Eigenschaft übergeben. OPC server and its DLLs.
- **`<InstDir>/cnf`** contains an example configuration file called '`AprolOPC.cfg.example`'.
- **`<InstDir>/doc`** contains this documentation in PDF format.
- **<InstDir>/examples**
  Contains an example for connecting to the OPC server from Excel.

The APROLFehler! Kein Name für eine Eigenschaft übergeben. *OPC server* must be registered in Windows. The registration is carried out automatically with the APROL setup. During the uninstallation, APROL-specific entries are removed again from the registry.

For manual registration, the OPC server "`AprolOPC.exe`" must be executed once using the program argument "/*RegServer*" or "/*Install*". The program argument "/*UnregServer*" or "/*Deinstall*" removes the registration.

9.3 Information about the configuration file

All necessary APROL objects are defined in the configuration file.

You basically have **two possibilities** for creating the `AprolOPC.cfg` configuration file, and thus the OPC server configuration:

- The file can be configured in either an integrated way using the user interface
- or manually as a text file using a text editor of your choice. (You can find the structure of a configuration file in chapter *Structure of the configuration file*)

A configuration that is generated by APROLFehler! Kein Name für eine Eigenschaft übergeben. results from the inputs in the *OPC connection* (in the configuration of the APROL system, in APROL connection).

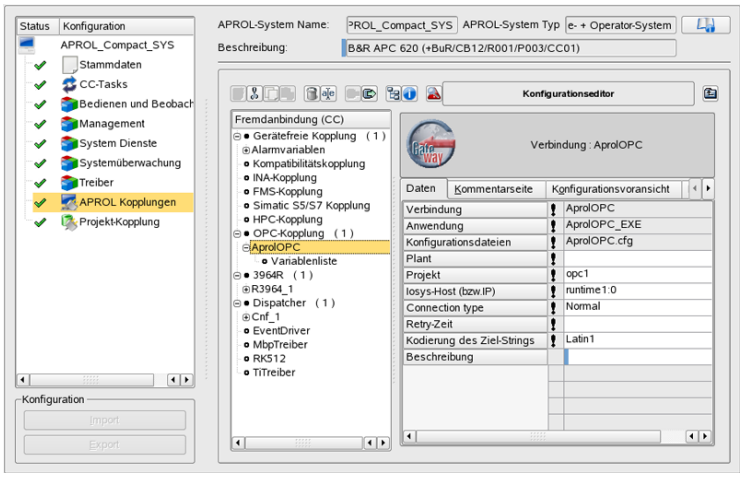


Illustration 28: Creating the OPC configuration

*The name of the configuration is always (**AprolOPC**) and cannot be changed.*

The table below contains the descriptions for the entries that are needed when creating the *OPC configuration* and then when creating the variables:

Short description about the required entries and also tool tips are available.

Description of the entries for defining the configuration:

Entry	Description
Connection	The name of the connection. This is preset (*AprolOPC*) and cannot be changed by the user.
Application	The name of the application is stipulated (AprolOPC_EXE) and cannot be changed by you.
Configuration file	The configuration file name for the APROL OPC server is stipulated (AprolOPC) and cannot be changed by you.
System	The system identifier. At the moment, this is only required for creating and formulating the name of an *OPC item*.

Project	The project identifier. Similar to *System*, this identifier is currently only required for creating and formulating the name of an OPC item.
Iosys Host (or IP)	Computer name or IP address of the runtime system (including the port number of the Iosys). For redundant systems, both computer names and IP addresses must be specified. Entries should be separated by a ",". (e.g. *runtime1:0, 10.49.80.80:1*).
Connection type (ConnType)	Connection type for one or more OPC servers **Normal** (Default setting) for one single OPC server **Redundant** when several OPC servers simultaneously access the Iosys and interact with the same process variable. In this case, these OPC servers cannot be started on the same Windows computer! (please also see the following information text)
TR [s] / Retry Time	Retry time (TR) in <seconds>. Time overrun for failed reconnection attempts (Maximum 3600).
Coding of the target string	**As of** APROLFehler! Kein Name für eine Eigenschaft übergeben. ***OPC Server*** **V3.0**: When coding the target string, you can select between: `Latin1, Chinese (BIG5), Windows (CP1252), Latin2, UTF-8`
Description	A description for OPC connection can be specified here.

The write PVs are handled as bi-directional PVs when using the "Redundant" setting in the "ConnType" option. They thus loose their unique direction of communication. The OPC server still remains as the provider.

Furthermore, process variables are marked with "UNCERTAIN" in OPC clients when the OPC client is connected to an OPC server, which is not the provider of the write PV, and the OPC client tries to set it. This state remains until the providing OPC server accepts either this or another value change for this variable.

The OPC server communicates with the Iosys via a TCP/IP connection. This is monitored with the Watchdog mechanism, i.e. both applications still swap telegrams, even when no actual data exists. The cycle of the Watchdog monitoring is fixed at 60 seconds, i.e. if the amount of time between two telegrams exceeds 60 seconds then the connection to the Iosys is closed. It is thus possible for a redundant OPC server on another machine to take over the communication. A connection loss would be detected much later without this Watchdog mechanism.

*The "**!**" in the individual connection view in the "Data" tab signals that an entry is needed in this field.*

After creating the *OPC configuration*, OPC variables can be created using the shortcut menu [**New**].

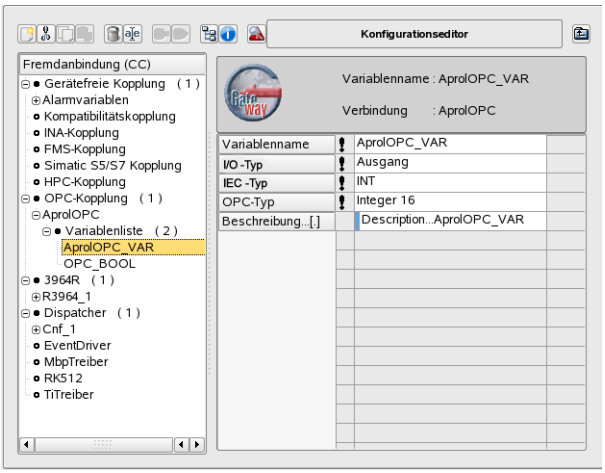


Illustration 29: Creating the OPC variables

After the connection is created, the OPC variables are added. The required entries have the following meaning:

Entry	Description
Variable name	Process variable name. This corresponds to the name of the PV in the APROLFehler! Kein Name für eine Eigenschaft übergeben. Iosys.
I/O type	OPC variables can be of the types input, output, or bidirectional. **Input** = write access **Output** = read access **Bidirectional** = write and read access
IEC type	Data type of the variables in the APROLFehler! Kein Name für eine Eigenschaft übergeben. system.
OPC Type	Data type of the variables in the APROLFehler! Kein Name für eine Eigenschaft übergeben. OPC server.
Description	Entering a description is optional.

The *OPC variables* created here can be used in your function charts as ***Gateway I/O*** depending on the *Type* of input and outputs lines in the function charts.

*A Gateway I/O with type **Output** is only permitted to be placed on the chart output line once. For this reason, the outputs that have already been used are not available when selecting Gateway I/O on the chart output line.*

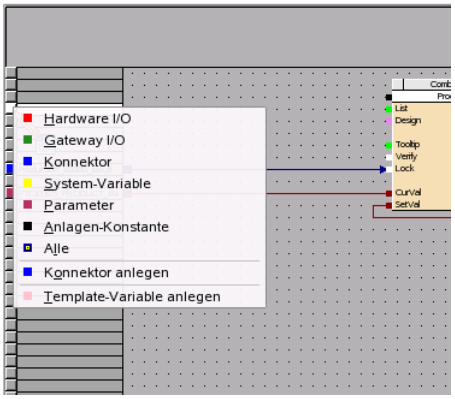


Illustration 30: Storing the OPC PV on the chart output line

*To quickly find variables, use the search algorithm for the **Levenshtein distance**.*

The `AprolOPC.cfg` configuration file is created during compilation of the corresponding control computer and is stored in the directory

/home/<engin>/ENGIN/EXPORT/

The configuration file must then be copied to your Windows system **`<InstDir>/cnf`**.

After an *OPC server* **restart**, the configuration file is read and all defined **APROLFehler! Kein Name für eine Eigenschaft übergeben.** variables are registered in the Iosys.

The integrated configuration of the *OPC server* using the user interface has the following advantages:

- Versions of the hardware and connection are assigned
- Engineering integration
- Advantages when debugging
- Syntax checking (possible errors are intercepted during input)

9.4 Structure of the configuration file

The configuration data is divided into object entries. Each object entry corresponds to an **APROL** process object, including the respective **APROL** link object.

Object entries must be made according to the object hierarchy. Process objects with high priority must be entered in the configuration file first.

Each object entry is divided again into the following sections:

- Name of the process object (**PN, Process Name**)
- Description of the process object (**PD, Process Description**)
- Description of the link object (**LD, Link Description**)

Each object entry begins with a name section. Then the sections for object descriptions can be added.

Section 1: Name of the process object (PN)

Syntax: PN: <*Object name*> OT=<*object type*>

<*Object name*>	Name of the process object. The entry can be made as a single name or with the object path (For syntax, see the APROL User Documentation).
<*Object type*>	Entry for the object type ("line", "conn", etc.).

Section 2: Description of the process object (PD)

Syntax: PD: <*Parameter 1*> <*Parameter 2*> ... <*Parameter n*>

These parameters are identical to those for the object parameter description for the IosysCreate function (see APROLFehler! Kein Name für eine Eigenschaft übergeben. User Documentation).

If the object name 'APROL' is entered as the first object entry in the configuration file (name of the APROLFehler! Kein Name für eine Eigenschaft übergeben. base object), this section is not interpreted as process object description, but instead as an APROLFehler! Kein Name für eine Eigenschaft übergeben. initialization parameter description.

These parameters are identical to those for the initialization parameter description for the IosysInitialize function. Additionally, the parameters TR=<Retry Time> can also be given:

TR parameter: Time used to repeat user messages in seconds.

OpcServer V3.0 and higher:
Additional entry for target string encoding (e.g. EN=Latin1)

All initialization parameters are described in the APROLFehler! Kein Name für eine Eigenschaft übergeben. User Documentation.

Section 3: Description of the link object (LD)

Syntax: LD: <*Parameter 1*> <*Parameter 2*> ... <*Parameter n*>

These parameters are identical to those for the connection parameter description for the function *IosysCreate*.

9.4.1 Structure of an example configuration file

The structure of a configuration file is explained in an example below.

For documentation purposes, the configuration file is divided into 4 sections:

Characters between the "#" and the end of the line are considered comments.

```
#
# AprolOPC-Server configuration file
# generated by Schulte at 2003.10.15  15:20:33
#

PN: @/Aprol
PD: TR=30  EN=Latin1
```

Entry	Description
PN: @/Aprol	This entry is constant!
PD: TR=30 EN=Latin1	Connection value (like with IosysConnect) TR = Retry time in sec. (max. 3600)

```
PN: @/Aprol/Essen                OT=line
PD: CD=iosys
LD: EV=""
```

Entry	Description
PN: @/Aprol/**Essen** OT=line	The entry in the "System" field when creating the configuration.
PD: CD=iosys	This entry is constant!
LD: EV=""	This entry is constant!

```
PN: @/Aprol/Essen/DocProject      OT=conn
PD: IP=runtime:0
LD: EV=""
```

Entry	Description
PN: @/Aprol/Essen/**Docuproject** OT=**conn|reduconn**	Complies with the entries in the "Project" and "ConnType" fields when creating the configuration.
PD: IP=**runtime:0**	Computer name and IP address (including Iosys port) for the runtime system**.**
LD: EV=""	This entry is constant!

Variable List: (**this object entry is repeated for each Iosys-PV**)

```
PN: @/Aprol/Essen/Dokuprojekt/AprolOPC_PV1  OT=pvar
PD: CD=AprolOPC_PV1   AT=r
LD: VT=boolean IO=BOOL

PN: @/Aprol/Essen/Docuproject/AprolOPC_PV2  OT=pvar
PD: CD=AprolOPC_PV2 AT=r
LD: VT=boolean IO=BOOL

PN: @/Aprol/Essen/Dokuprojekt/AprolOPC_PV3  OT=pvar
PD: CD=AprolOPC_PV3 AT=rw
LD: VT=string VL=64 IO=STRING
```

Entry	Description
PN: @/Aprol/Essen/DocProject/ **AprolOPC_PV1** OT=pvar	OPC variable name
PD: CD=**AprolOPC_PV1** AT=r	Iosys variable: AT=r output (CAE) AT=rw input (CAE) AT=s bidirectional (CAE)

Entry	Description
LD: VT=boolean IO=BOOL	Supports OPC types (VT): - boolean, - i8, i16, i32, - u8, u16, u32, - f32, f64, -string: If the data type "string" is selected, then the length must be additionally given in bytes(VL=64) Supported CAE types (IO): - REAL - STRING - INT All other data types in CAE are interpreted as INTEGER variables (Iosys): - e.g. BOOL

Notes about OPC variable names:
With some OPC clients, the symbols "**/**" or "@" are not accepted in OPC variable names. For this reason, it is necessary to define a replacement character.

This replacement character is entered in the registry. In the scope of the installation, and for this purpose, the '**NameSep**' variable is pre-set with '**.**' in the registry.

The variable "NameSep" is found in the Registry under:
HKEY_LOCAL_MACHINE/SOFTWARE/BR_AUTOMATION/APROL/APROLOPC

Example:
From the PN entry: `@/Aprol/Essen/Dokuprojekt/AprolOPC_PV1` the configuration file is created on the OPC page using the replacement character of the OPC item
`Aprol.Essen.Dokuprojekt.AprolOPC_PV1.`

9.4.1.1 Example of a configuration file

```
#
# AprolOPC-Server configuration file
# generated by Schulte at 2003.10.15  15:20:33
#

PN: @/Aprol
PD: TR=30  EN=Latin1

PN: @/Aprol/Essen                OT=line
PD: CD=iosys
LD: EV=""

PN: @/Aprol/Essen/DocProject     OT=conn
PD: IP=runtime
LD: EV=""

PN: @/Aprol/Essen/Dokuprojekt/AprolOPC_PV1  OT=pvar
PD: CD=AprolOPC_PV1   AT=r
LD: VT=boolean

PN: @/Aprol/Essen/Docuproject/AprolOPC_PV2  OT=pvar
PD: CD=AprolOPC_PV2 AT=r
LD: VT=boolean

PN: @/Aprol/Essen/Dokuprojekt/AprolOPC_PV3  OT=pvar
PD: CD=AprolOPC_PV3 AT=r
LD: VT=string VL=64
```

9.5 Event Viewer in Windows for diagnosis

With the help of the ***Event Viewer***, you have the possibility to call up the status of the OPC server and the OPC connection on your Windows-computer for diagnostics purposes, as well as any error states that may exist.

Go to the Windows Explorer and open the "Manage" shortcut menu for the "My Computer" entry.
The select "System Tools/Event Viewer/Application".

The following table contains a short description of all status and error messages according to the ID:

ID	Description
1000	Connection to *Iosys* has been established on <computer name>.
1001	Connection to *Iosys* could not be established on <computer name>.
1002	The *AprolOpc* application has been started.
1003	The *AprolOpc* application has been stopped.
1004	*AprolStringConvert* initialized with <Encoding>.
1005	*AprolStringConvert -iconv_open* failed.
1006	*AprolStringConvert -iconv* failed.

9.6 Debugging the OPC Server

The OPC server offers the possibility to analyze its actions with a console window and an additional log file. In order to do this, it is necessary to activate this option by setting the registry values for 'IosysDebug' and 'IosysDebugFile'.

As of the OPC server package version V3.6.0.0, these registry entries are automatically set during an APROL OPC server installation.

The activation of this feature has an effect both on the performance of the Windows computer, and on the available disk space!

The log file is overwritten during each OPC server restart.

Because the OPC server is automatically started when the **first** client starts, and is stopped when the last client is stopped, the **log file must eventually be backed up**.

This chapter is an internal documentation, which is aimed at experienced users.

The following debug defines are contained at present:

Debug defines	Hexadecimal value
DBG_ERRORS	0x00000001
DBG_NORMAL	0x00000002
DBG_IOS_CONN	0x00000004

Debug defines	Hexadecimal value
DBG_IOS_VAR	0x00000008
DBG_IOS_EVT	0x00000010
DBG_EXTENDED	0x00000100

The individual outputs are to be switched on, or off, by carrying out an OR operation with each bit.

A) The OPC server debug level is activated with the following entry in the registry:

Directory: `HKEY_LOCAL_MACHINE\SOFTWARE\BR_Automation\APROL\AprolOPC\`

Item: IosysDebug

Type: DWORD

Default: 0, no debug output

The registry entry is set back to the value '0' with each new installation of the APROL OPC server package.

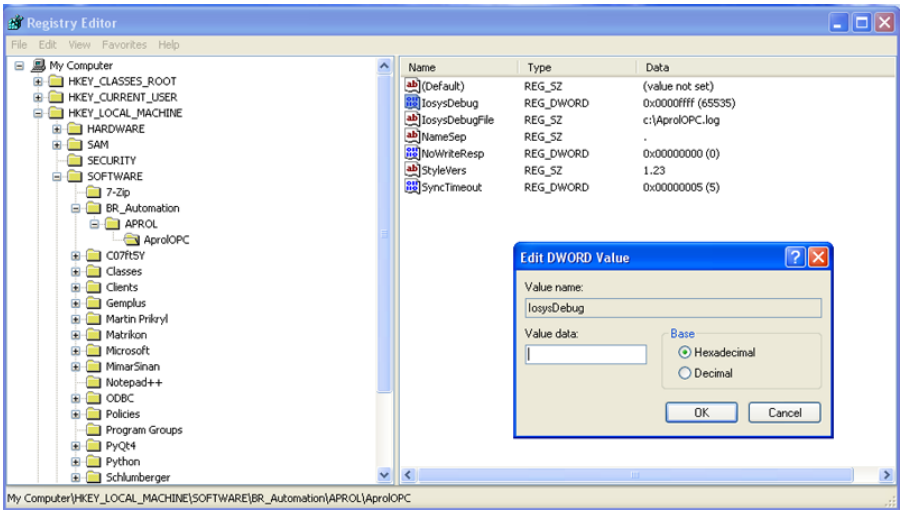


Illustration 31: Activation of the debugging in the registry

If a value that is not equal to null is entered in the *IosysDebug* field then a console window is opened, in which the debug output is shown during an OPC server (re-)start.

The log file is overwritten during each OPC server restart.

Please note that the log file can grow extremely in size, and thus the debug defines are only allowed to be used for debugging purposes.

The following overview contains the possible output of the individual debug bits:

Debug defines	Hex value / possible output
DBG_ERRORS	**0x00000001** - Creation of the iosLib socket failed - bind to socket failed - lists on socket failed - ioctl on socket failed - accept on socket failed - connect to Iosys failed - connection to Iosys lost - select failed - Ios_sync failed - Setting an Iosys variable failed due to a non-supported Iosys type - (Here, 'xxx' is to be replaced with 'int', 'real', 'string')
DBG_NORMAL	**0x00000002** - IosConn_Delete - Shutdown an Iosys connection - Start entry in log file with ReconnectRetryTime output - IosConn_Create - Creates an Iosys connection with IP address, RetryTime - Connection established - IosClient initialize - Output of the start parameter, OK message after initialization - IosClient de-initialize - Status message at end - Message during the creation of an Iosys variable with type description and return code - Message during a de-registration of an Iosys variable - Status message of the Ios_sync thread - Final status message of the Ios_sync thread with number of pending changes, which have not yet been compared via "SYNC" - Iosys debug start message
DBG_IOS_CONN	**0x00000004** - IosConn_namespace_new call with parameter output (Creation of a new Iosys connection) - De-registration (redundant) Iosys connection - Release of an Iosys connection object - Closing of an Iosys connection - Release of an Iosys variable - Detection of an IP address from the host name
DBG_IOS_VAR	**0x00000008** - IosVar_delete with PV names - De-registration of an Iosys variable - Registration of an Iosys variable with output of the callback pointer

Debug defines	Hex value / possible output
DBG_IOS_EVT	**0x00000010** - IosysReadInt function - Reception of an Iosys event, output of the PV name and received value - IosysReadREal function - Reception of an Iosys event, output of the PV name and received value - IosysReadString function - Reception of an Iosys event, output of the PV name and received value - Setting a variable from the OPC client in the Iosys
DBG_EXTENDED	**0x00000100** - Initialization of the select thread - Select error, when the connection to the Iosys is lost

B) The name and the folder of the OPC server's log file is activated with the following entry in the registry:

Directory: `HKEY_LOCAL_MACHINE\SOFTWARE\BR_Automation\APROL\AprolOPC\`

Item: IosysDebugFile

Type: STRING

Default: c:\AprolOPC.log

9.6.1 Changing of the debugging behavior during runtime

APROL allows the OPC server's debugging settings (according to the registry entries) to be changed **whilst the OPC server is running**. The default settings, which are controlled by both above mentioned registry entries, are preserved.

The APROL OPC server creates two new Iosys variables for this purpose:

- The `<Computer name Windows OPC server>.AprolOPC.dbgFileName` Iosys variable is of the type 'String' and states the log file's storage place.
- The `<Computer name Windows OPC server>.AprolOPC.dbgFilter` Iosys variable is of the type 'Integer' and controls the debug level.

The variables are **always** supplied by the APROL OPC server and obtain the respective current value of the debug settings (Registry entries).

It is also possible to change the directory for the log file whilst the APROL OPC server **is running** by changing the value of the `<Computer name Windows OPC server>.AprolOPC.dbgFileName` variable, with the **IosDiagnosticManager** or the '*pio*' Iosys tool.

If a change takes place then the current debug file is closed and a new file, corresponding to the new value is created.

A file that already exists is overwritten by this procedure! The newly created file stays open as long as either the OPC server is closed or another debug file is opened.

Modeled on this, the amount of output during the OPC server runtime can be changed by changing the `<computer name Windows OPC server>.AprolOPC.dbgFilter` variable (via **IosDiagnosticManage**r or '*pio*').

The changes that are made to these Iosys variables during runtime is also recorded in the respective active debugging. A change that has been made to the file name during runtime is recorded as the last entry in the log file.

Both registry entries are overwritten their default values upon each new OPC server installation.

9.6.2 Information about the debug output

`DataCallback - VarName <req = 0,1>`

is a ChangeEvent (req = 0) or a ChangeRequest (req = 1) for a SYNC variable, which is received from the Iosys.

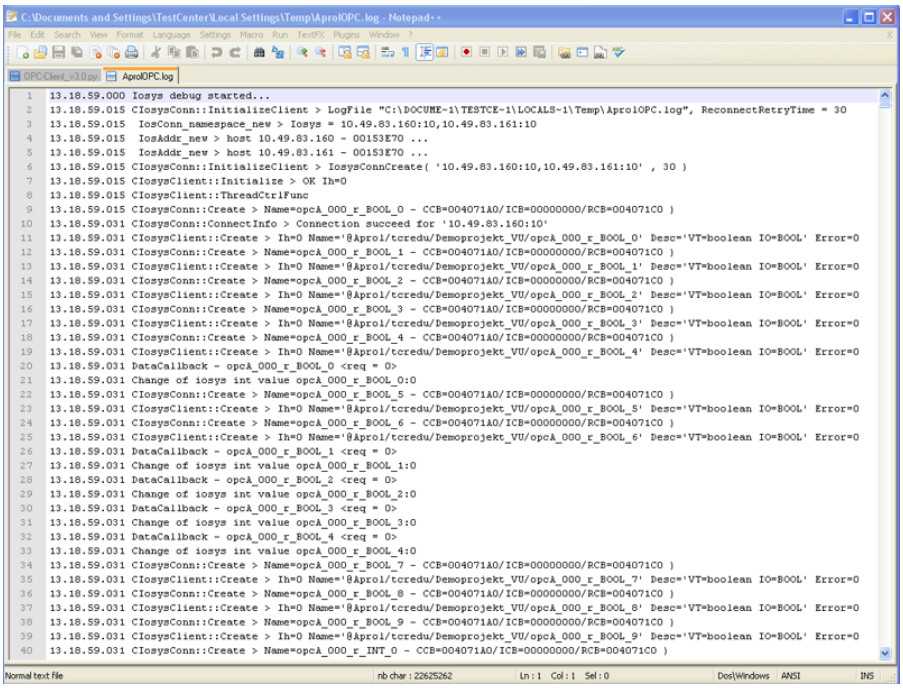
`Change of Iosys [int|real|string] value`

is a change that is made by the Iosys, and changes the value in the OPC client. It is the consequence of a DataCallback output.

`Setting new Iosys [int|real|string] value`

is a change that is made by the OPC client, and changes the value in the Iosys

Exception: A ChangeRequest is already present, then this is based on the value adoption from the controlling OPC server as a result of the request!



```
13.18.59.000 Iosys debug started...
13.18.59.015 CIosysConn::InitializeClient > LogFile "C:\DOCUME~1\TESTCE~1\LOCALS~1\Temp\AprolOPC.log", ReconnectRetryTime = 30
13.18.59.015  IosConn_namespace_new > Iosys = 10.49.83.160:10,10.49.83.161:10
13.18.59.015  IosAddr_new > host 10.49.83.160 - 00153E70 ...
13.18.59.015  IosAddr_new > host 10.49.83.161 - 00153E70 ...
13.18.59.015 CIosysConn::InitializeClient > IosysConnCreate( '10.49.83.160:10,10.49.83.161:10' , 30 )
13.18.59.015 CIosysClient::Initialize > OK Ih=0
13.18.59.015 CIosysClient::ThreadCtrlFunc
13.18.59.015 CIosysConn::Create > Name=opcA_000_r_BOOL_0 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysConn::ConnectInfo > Connection succeed for '10.49.83.160:10'
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_0' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_1 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_1' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_2 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_2' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_3 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_3' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_4 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_4' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 DataCallback - opcA_000_r_BOOL_0 <req = 0>
13.18.59.031 Change of iosys int value opcA_000_r_BOOL_0:0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_5 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_5' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_6 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_6' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 DataCallback - opcA_000_r_BOOL_1 <req = 0>
13.18.59.031 Change of iosys int value opcA_000_r_BOOL_1:0
13.18.59.031 DataCallback - opcA_000_r_BOOL_2 <req = 0>
13.18.59.031 Change of iosys int value opcA_000_r_BOOL_2:0
13.18.59.031 DataCallback - opcA_000_r_BOOL_3 <req = 0>
13.18.59.031 Change of iosys int value opcA_000_r_BOOL_3:0
13.18.59.031 DataCallback - opcA_000_r_BOOL_4 <req = 0>
13.18.59.031 Change of iosys int value opcA_000_r_BOOL_4:0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_7 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_7' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_8 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_8' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_BOOL_9 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
13.18.59.031 CIosysClient::Create > Ih=0 Name='@aprol/tcredu/Demoprojekt_VU/opcA_000_r_BOOL_9' Desc='VT=boolean IO=BOOL' Error=0
13.18.59.031 CIosysConn::Create > Name=opcA_000_r_INT_0 - CCB=004071A0/ICB=00000000/RCB=004071C0 )
```

Illustration 32: Example log file 'AprolPOC.log'

9.7 OPC Server status variables

This description is under construction at present.

9.8 Example for OPC clients with the APROL OPC server

Basically, there are several ways to establish a connection with an OPC server. There are different sorts of clients according to the usage of the variables.

OPC sample client (Development Setup)

Although this client has a limited means of use it can be obtained gratis via download from the B&R homepage, and is very simply constructed, and thus well suited for testing the APROL OPC server. It can only be used on the computer where the APROL OPC server is running. A license for the *Automation Studio* is also necessary in order to be able to run the OPC Sample Client for more than 2 hours at one time; it must be started again afterwards.

Example:

- Start the OPC Sample Client with "Start/Programs/B&R Automation/PVI x.x.x/PVI Developer/Server/OPC Sample Client".
- Choose the OPC server in "OPC/Connect" in the pop-up window and confirm with [**OK**].

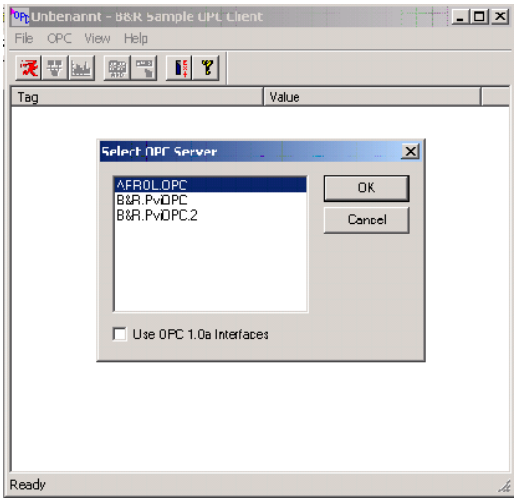


Illustration 33: OPC server choice

- Choose the group settings for the communication in the "OPC/Group Parameters" menu and confirm with [**OK**].

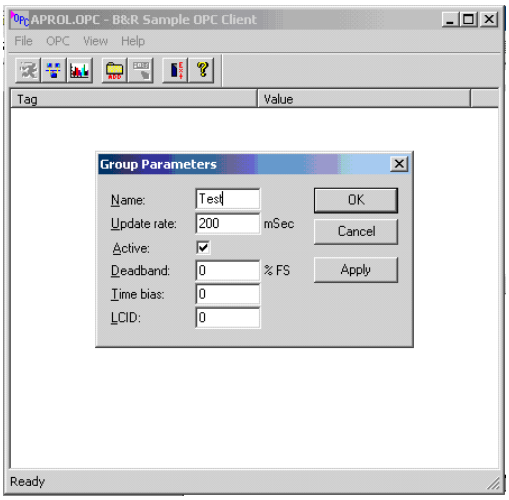


Illustration 34: "Group Parameters" dialog

Add the desired variables that should be displayed with "OPC/Add Item" , and confirm with [**OK**].

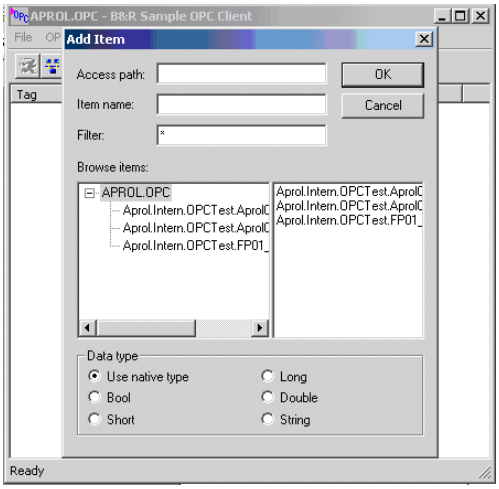


Illustration 35: "Add Item" dialog

If the variable can also be changed then the variable can be written with "OPC/Write Value to Item" and confirmed with [**OK**].

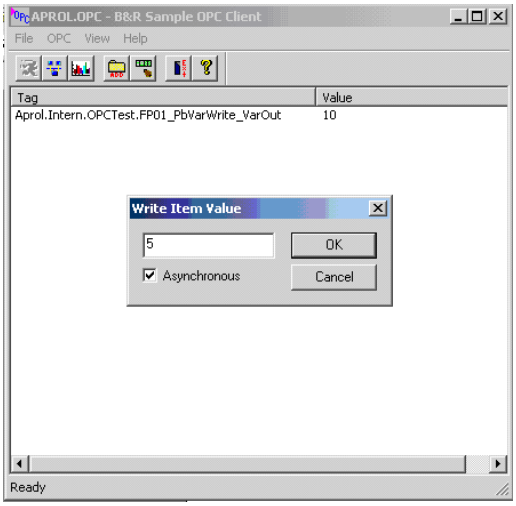


Illustration 36: "Write Item Value" dialog

3rd Party clients

The clients from different companies offer different features, and can be well incorporated in certain areas, according to the application. The 'Matrikon OPC Explorer' can be recommended as a free-ware client that offers a very comfortable operation, and has been tested with the APROL OPC server and has been certified as functioning.

www.matrikonopc.com

Visual Basic applications

OPC applications of any kind can be created with Visual Basic 6.0. Microsoft Office applications such as Excel supply a Visual basic interface. This interface can be used to read and write data from an OPC server directly in an Excel file.

You have the possibility to adjust the Visual Basic code yourself. How to proceed in this case, and which program code is necessary, can be found in point 5 "PVI OPC Programming, in the Training module TM730 - PVI OPC". Although the programming of OPC servers is described here with PVI, it is identical with the programming of the APROL OPC server.

Excel

This directory contains an example for an OPC connection from an MS-Excel sheet.

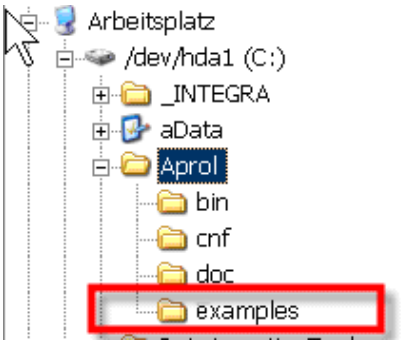


Illustration 37: Example directory in the Windows file system

In order to be able to use the Excel file, you must download and install the necessary 'OPCEx Excel Add-In' Excel add-in from http://www.resolvica.com/p1/download.aspx.

The 'AprolOPC_ExcelClient.xls' Excel file has been adjusted to the B&R start-up project. The Excel file can be used after the B&R start-up project has been installed together with the APROL OPC server.

The 'AprolOPCDemo_ExcelClient.xls' Excel file has been adjusted to the APROL Demo project, and can be used with it.

Although this type of OPC client has a very small scope of functionality, you can read and write process variables in a very simplified manner.

9.9 Additional information about the OPC server

.

9.9.1 Version information

OPC Server version	Description
4.0.0.0	APROL OPC server adjustment to the APROL release 4.0.
3.8.0.0	**Revised version created with Visual Studio 2010**

3.7.0.0	Error correction: The browse method always delivers all items, both for the search for 'Nodes' and for the search for 'leaves'. That leads to problems with the *Matrikon Security Gateway*. All items repeat themselves recursively with unlimited depth. Only 'Leaves' are returned and 'Nodes' do not exist because the APROL OPC server only supports flat structures.
3.6.2.0	Error correction: Cyclic connection loss of the OPC server when - a redundant Iosys connection was configured - and the master is not running when the OPC server is started.
3.6.1.0	Error correction: OPC server crash with non-existent configuration file. Update: Connection monitoring between OPC server and Iosys, with 60 second delay. Update: Delivery of the OPC server with the 'OPC COM Proxy Stub MergeModules' from the OPC foundation. This contains, amongst others, the 'OpcEnum' service that allows the OPC clients to find the installed server.
3.6.0.0	Setting of the debug level and log file during runtime
3.5.0.1	Redundancy configuration, Setting of "UNCERTAIN" status
3.0.5	Various error corrections Safeguard of the Iosys against multi-threading
3.0.0	For connection to control computers (64-bit integer and UTF-8 conversion).
2.4.0	MSI uninstall corrected. Uninstall mechanism was updated!
2.3.0.1	Memory leak corrected.
2.3.0.0	OpcServer with asynchronous access.
2.2.1	OpcServer with write functionality, Synchronous access only.
2.2.0	OpcServer "ReadOnly" Writing values to the Iosys is not supported.

9.9.2 Licensing information about the iconv library

The APROLFehler! Kein Name für eine Eigenschaft übergeben. OPC server uses the *iconv* library, which is subject to the LGPL. B+R Industrie-Elektronik Ges.m.b.H. meets the requirements for recompiling ability that result from this licensing by providing the complete source code for the shared library *AprolStringConvert.dll* including the VisualStudio project files.

This source code is found in compressed form in the installation directory `src` (in the directory `C:\Aprol` as default).

9.9.3 Literature notes on the topic of 'OPC'

Frank Iwanitz, Jürgen Lange
OPC: Grundlagen, Implementierung und Anwendung
Hüthig-Verlag, Heidelberg
ISBN 3-7785-2866-1

10 ProfiboardDriver

10.1 General information about the ProfiboardDriver

This documentation describes the installation and configuration of the software package "FMS-PROFIBUS DRIVER for APROL with SOFTINGs PROFIboard" in LINUX. The package consists of the kernel driver that handles access of the hardware, various utilities for configuration and network analysis and also the APROL driver for communication with the I/O modules. The installation of the software, the configuration of the hardware and the network parameters for a few selected network settings are also described in this documentation. This can be used as an aid when solving problems if errors occur.

The program package requires a LINUX kernel starting with Version 2.0.36 or Version 2.2.0 (not yet available when this documentation was created). The kernel must be compiled and started with the option "enable loadable module support" and without the option "set version information on all symbols for modules". Please check your SuSE LINUX documentation to see how a corresponding LINUX kernel is structured.

10.2 Hardware configuration

The PROFIBUS card used is a ***SOFTING*** product. This is an ISA card that requires a 4 byte IO area, a free interrupt line as well as a 16 KB address area in the first MB on your PC in order to operate. The device driver supports parallel operation of two PROFIBUS cards that can share a common interrupt. The PROFIBUS card is a piece of hardware that supports the following protocols:

- PROFIBUS-FDL (layer 2 communication),
- PROFIBUS-FMS (layer 7 communication),
- PROFIBUS-DP and
- PROFIBUS-DP-V1.

Because of the DP capability, the card has very short reaction times. The card is configured by setting the IO address using DIP switches. They are directly on the slot bracket. The card automatically uses four consecutive addresses. The base address for IO when the card is delivered is address 240h, i.e. the card uses addresses 240h to 243h. If this address is already being used on your computer or if you are using two PROFIBUS cards at the same time, this address must be changed on at least one card. The I/O address is represented using 8 switches, i.e. the address is the sum of the switch positions.

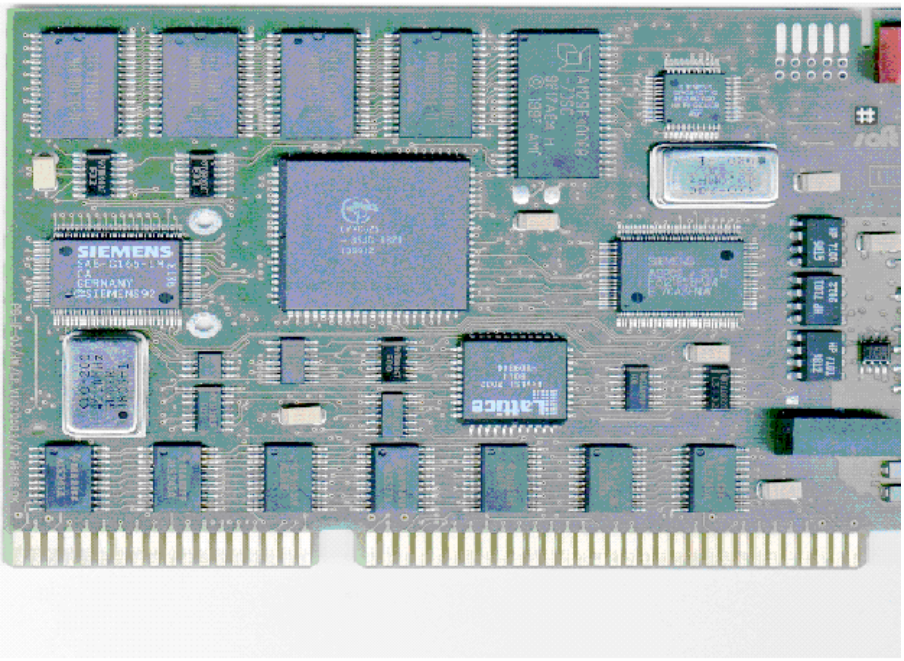


*Illustration **38**: PROFIboard*

The image shows the PROFIboard used with the DIP switch for the I/O addresses (red block on the top right). An LED that is integrated in the slot bracket (seen on the bottom right) blinks during operation when the board is active on the bus.

The following table shows the meaning of the individual switches as the hexadecimal address. The summand is valid if the switch is set to "on".

Switch	DIP1	DIP2	DIP3	DIP4	DIP5	DIP6	DIP7	DIP8
Address	200h	100h	80h	40h	20h	10h	8h	4h

According to this, the addresses between 000h and 3FCh can be set. Some of these addresses are already being used by components on your PC. The following table provides an overview of the most commonly used addresses and the corresponding DIP switch settings. In general, one of these addresses should also be able to be used on your PC.

I/0 address	DIP1	DIP2	DIP3	DIP4	DIP5	DIP6	DIP7	DIP8
240h	on	off	off	on	off	off	off	Off
244h	on	off	off	on	off	off	off	on
300h	on	on	off	off	off	off	off	off
304h	on	on	off	off	off	off	off	on
310h	on	on	off	off	off	on	off	off
314h	on	on	off	off	off	on	off	on

The software settings for shared memory and the interrupt line are made directly by the device driver. Please check the "Software configuration" section for detailed information. Five interrupts are available for the PROFIboard: IRQs 5, 10, 11,12 and 15. These interrupts can be used for multiple PROFIboards at the same time, but not for other hardware. Some of the interrupts are already being used according to the how your PC is designed:

- IRQ 11 for the graphics card,

- IRQ 12 for a PS/2 mouse,
- IRQ 15 for the second IDE controller.

The addresses from 0xC8000 to 0xF0000 are available to insert the card in the first megabyte. The size of the area to be inserted is 16 KBytes (0x04000) for each PROFIboard. These addresses are only permitted to be used once in the system.

10.3 Installing the PROFIboard software

The software is installed using the YaST installation tool for your SuSE Linux version. It is necessary that the installation medium is made available to the system. One needs super-user rights for this.
First, the installation media must be mounted on the existing file system. For installation using YaST, the mounting point must be a root directory (***/suse***). If this directory is not yet available on your computer, it can be created with the following command:

- ***mkdir /suse***

Now the installation media must be mounted there. The device file ID is different for your floppy drive or CDROM depending on the type of installation media used. The following device files are conceivable:

- ***/dev/fd0*** - Floppy drive ID a:,
- ***/dev/fd1*** - Floppy drive ID b:,
- ***/dev/cdrom*** - CDROM drive ID.

The syntax for the mount command is ***mount device file / target directory***
In your case, the command would be as follows:

- for floppy a: ***mount /dev/fd0 /suse*** ,
- or for the CDROM drive ***mount /dev/cdrom /suse***.

If the ***mount*** command was carried out without an error, you can now start the YaST installation tool.

In the YaST main menu, the installation source must be selected. The following menu sequence must be carried out:

- ***"Installation settings"***,
- ***"Selection of installation source"***,
- ***"Installation using a valid directory"***.

In the ***"Installation using a valid directory"*** menu item, enter the directory name ***/suse*** and confirm the entry. Then go back to the main menu. Choose the packets to be installed here. It is important to confirm the select with "***F10"*** and not to go back "***ESC"***! After confirming the selection, you can use the ***"Define/start installation"*** or ***"Change/create configuration"*** menu item to select if you want to start a new installation or change a configuration.
For the installation that is to be carried out here, select the packages pccadp1 and pcckd1, confirm with ***"F10"***, go back to the installation menu and start the installation as described above using the ***"Start installation"*** menu item.

After the installation, YaST makes a few system settings. When the tool is finished doing this, exit YaST by pressing the ***"ESC"*** key several times. After ending YaST, the device driver must be configured. This procedure is started automatically. Exit the ***"Install Softing card driver"*** selection menu with it set to ***"Yes"***. Now start the program ***"pb_install"*** to transfer the settings to the kernel module.

The following queries must be answered:

- ***input major device number (default 50),***
- ***input no of boards to be installed.***

Then for each board that is to be entered:

- ***enter ram_addr for board x***
- ***enter io_addr for board x***
- ***enter irq for board x***

When the settings are made for all boards, they are shown once again in list form. If an entry is incorrect, the procedure can be repeated by pressing the ***"ESC"*** key. Pressing ***"ENTER"*** saves the settings, creates the start script and runs it. If all of the settings are correct, the PROFIboard is started with the standard settings.

Remove the PROFIboard from the network before starting for the first time. Otherwise the network settings can cause your PROFIBUS network to crash.

Use the ***"pb_manager"*** utility to check if the PROFIboard has been started correctly.

Select the ***"Livelist"*** option here. When the PROFIboard is correctly installed, you find exactly one station with one station address 0. If you cannot execute the "***Livelist***" then please go to section "Error".

Here is a list of the files installed by YaST. A detailed description of the utilities can be found in chapter "*Description of the utilities*".

File name	Description
/boot/modules/profibus/PROFIboard.org	Kernel module without hardware settings
/boot/modules/profibus/PROFIboard.smp.org	Multiprocessor kernel module without hardware settings
/opt/aprol/bin/pb_debug	Debugging tool for error analysis
/opt/aprol/bin/pb_init	Utility to load the network settings
/opt/aprol/bin/pb_install	Utility to patch the hardware settings into the kernel modules
/opt/aprol/bin/pb_list	Utility to read the object list for a B&R 2000 controller
/opt/aprol/bin/pb_manager	All-round tool for testing
/opt/aprol/bin/pb_plcreset	Utility for resetting a B&R 2000 controller
/opt/aprol/bin/ProfiboardDriver	APROLdriver for PROFIBUS FMS
/opt/aprol/bin/pb_taskmgr	Tool for downloading controller tasks

File name	Description
/opt/aprol/bin/pb_read	Utility for reading controller variables via PROFIBUS
/opt/aprol/bin/pb_netconfig	Tool for creating the APROL network configuration
/opt/aprol/bin/pb_timesync	Tool for activating controller time synchronization, requires a special controller task
/opt/aprol/bin/pb_settime	Tool for implementing the control computer time on a B&R 2000 controller, requires a controller task
/opt/aprol/bin/pb_history	Tool for reading the logbook entries from a B&R 2000 controller.
/etc/init.d//Profiboard	Start script for the APROL driver
/usr/etc/profibus/br.txt	Error list for B&R 2000 controller error numbers
/usr/etc/profibus/nw_pb_32_0.cfg	B&R standard configuration for Profibus address 0
/usr/etc/profibus/nw_pb_32_1.cfg	B&R standard configuration for Profibus address 1
/usr/etc/profibus/profibus0.cfg	Example network configuration for board 0
/usr/etc/profibus/profibus0.ov	Example object description for board 0
remove.PROFIboard	Uninstall script
setup.PROFIboard	Installation script
/opt/aprol/lib/libPccPROFIboard.so (only with APROL)	Profibus library for PROFIboard
/opt/aprol/lib/libPccPROFIboard.so.2 (only with APROL)	Profibus library for PROFIboard
/opt/aprol/lib/libPccPROFIboard.so.2.1 (only with APROL)	Profibus library for PROFIboard

10.4 Description of the start script

10.4.1 The start script

The following table shows a printout of the start script used to automatically start two PROFIboards when booting the computer.

```
#
#
# Startscript for Softing's PROFIBOARD
#
#
#####################################
BOARD_NO=" 0 1"
echo "Start PROFIboard loadable kernel module ..."
/sbin/rmmod PROFIboard 1/dev/null 2/dev/null
/sbin/insmod -f /boot/modules/PROFIboard/PROFIboard.smp.o -o PROFIboard 1/dev/null 2/dev/null
/sbin/insmod -f /boot/modules/PROFIboard/PROFIboard.o -o PROFIboard 1/dev/null 2/dev/null
for i in $BOARD_NO ; do
 /opt/aprol/bin/pb_init -b $i -f /usr/etc/profibus/profibus$i.cfg -o /usr/etc/profibus/profibus$i.ov -
useFdl
done
```

The start script for automatically starting the PROFIboards can be found in the directory ***/etc/init.d*** and is named ***PROFIboard***. The entries in ***/etc/init.d/rc2.d*** or ***/etc/init.d/rc3.d*** are executed depending on if the computer is operating in runlevel 2 or runlevel 3. The scripts are started in alphabetical order according to their names, which begin with S for start. Symbolic links with the name ***S51PROFI.sh*** are created for ***/etc/init.d/PROFIboard*** in both directories. The start script and the links are automatically generated using the installation tool ***"pb_install"***.

The number of boards to be started is defined with the ***BOARD_NO=...*** line. The command "***rmmod"*** stops a device driver that may already be started. If a device driver is not started, the error message is sent to ***/dev/null***, i.e. it is suppressed. The next two "***insmod"*** lines consecutively start the device drivers for multi-processor and single-processor operation. The mechanism guarantees that the right device driver is started automatically. If the kernel is not multi-processor capable, then the first device driver is rejected by the kernel and the second one is started. If the first one can be started, then the second one is rejected with the error message that a device driver with this name already exists. In principle, a multi-processor capable kernel can also be used if only one processor is operating in your system. The loop that is then run sets the network parameters on the card. The network parameters are found in the files that are named ***/usr/etc/profibus/profibusx.cfg*** (***x*** is the number of the board from ***BOARD_NO= ...***). The object descriptions are in the ***/usr/etc/profibus/profibusx.ov*** files. Both files descriptions are in chapter "Installation of the controller software".

10.5 Software configuration

10.5.1 Description of the network parameter file profibusx.cfg

The file ***profibusx.cfg*** in the directory ***/usr/etc/profibus*** is used to define the network parameters and connection parameters for PROFIboard ***x***. The ***x*** symbol stands for the board number in the device driver, beginning with 0 for the first board.

Following table: Example configuration for station 0 with connections to stations 1, 2, 3, and 4.

```
; PROFIBUS - Parameters
; station 0
; board   0
; ====================
0       ; this station
31      ; highest station
255     ; this segment
1       ; in ring desired
```

```
4      ; baud rate
0      ; medium redundancy
3500   ; slot time
0      ; quiet time
1      ; setup time
500    ; min station delay
1000   ; max station delay
30000  ; target rotation time
1      ; gap update
3      ; retry count
4096   ; ass_abt_ci
128    ; default sap
9      ; symbol length
0      ; vfd_supported
4      ; no of kbl entries

;
; communication reference list
;
; ============================
2     3    1  255   2  MMAC  0  D  3  3  2  2  300 0  240  240  240  240  FF FF FF FF
FF FF 1 [ST1] 0
3     4    2  255   2  MMAC  0  D  3  3  2  2  300 0  240  240  240  240  FF FF FF FF
FF FF 1 [ST2] 0
4     5    3  255   2  MMAC  0  D  3  3  2  2  300 0  240  240  240  240  FF FF FF FF
FF FF 1 [ST3] 0
5     6    4  255   2  MMAC  0  D  3  3  2  2  300 0  240  240  240  240  FF FF FF FF
FF FF 1 [ST4] 0
```

Comment lines in the description file begin with a semicolon and run to the end of the line. The following entries must be in the file:

Position	Info	Value range	Notes
1	Station address of the PROFIboards in the network	0 - 31	In the control system, the control computer is always an active station on the network
2	Highest station address in the network, HSA	0 - 31	The highest station address of an active station in the network
3	Segment size	255	Constant
4	Station active or passive on the bus	1	Always active in the control system
5	Baud rate	0 - 12, not 5	0 - 9.6 kBit/s 1 - 19.2 kBits/s 2 - 93.75 kBit/s 3 - 187.5 kBit/s 4 - 500 kBit/s 6 - 1.5 MBit/s 7 - 3 MBits/s 8 - 6 MBit/s 9 - 12 MBit/s
6	Redundancy setting	0	Constant
7	slot time	37 - 16383	
8	quiet time	0 - 127	

Position	Info	Value range	Notes
9	setup time	1 - 479	
10	min station delay	11 - 1023	Minimum delay between request and response telegram
11	max station delay	35 - 1023	Maximum delay between request and response telegram
12	target rotation time	256 - 16777215	Maximum rotation time for a token cycle.
13	gap update	1 - 255	Number of cycles until a check is made to see if a new station has been added on the network
14	retry count	0 - 7	Number of retries if an error occurs
15	ass_abt_ci	1 - 4294967295	ASS/ABT control interval
16	default sap	128	Constant
17	symbol length	<= 15	Maximum length of the symbolic ID
18	vfd supported	0	Constant
19	Number of connections in the CRL	= 0	Specifies how many entries should be utilized in the following table

The following table describes the structure of a line with the connection parameters, an entry in the communications relations list, CRL:

Position	Info	Value range	Notes
1	Position number, referred to as comm_ref for communication reference in the utilities.	1 – Number of connections in the CRL	In the utilities and in the driver, connections can be entered using the CRL number or the symbolic name. Symbolic names are not permitted to begin with a number (for differentiation)!
2	LSAP	0 - 62 63 128	Local Service Access Point
3	Address of the partner station	0 - HSA for Master-Master connections 0 - 126 for Master-Slave connections	When connecting to B&R 2000 controllers, a Master-Master connection is always used
4	Constant	255	Constant

Position	Info	Value range	Notes
5	DSAP or RSAP	0 - 62 63 128	Destination or Remote Service Access Point
6	Connection type	MMAC MSAC MSAC_SI MSCY MSCY_SI BRCT MULT	Master-Master acyclic Master-Slave acyclic Master-Slave acyclic with slave initiative Master-Slave cyclic Master-Slave cyclic with slave initiative Broadcast Multicast
7	LLI type		
8	Connection attribute	D I O	Defined connection Open connection for initiator Open connection for responder
9	SCC	0 - 20	Maximum number of send requests for confirmed services
10	RCC	0 - 20	Maximum number of telegrams that can be received for confirmed services
11	SAC	0 - 20	Maximum number of send requests for unconfirmed services
12	RAC	0 - 20	Maximum number of telegrams that can be received for unconfirmed services
13	Connection monitoring	0 - (2^32 - 1)	
14	MUL	0 - 255	For cyclic connections
15	SLH	0 - 241	Maximum length for sending high priority telegrams
16	SLL	0 - 241	Maximum length for sending low priority telegrams
17	RLH	0 - 241	Maximum length for receiving high priority telegrams
18	RLL	0 - 241	Maximum length for receiving low priority telegrams
19	C0	See table	supported services client 0
20	C1		supported services client 1
21	C2		supported services client 2
22	S0		supported services server 0
23	S1		supported services server 1
24	S2		supported services server 2
25	VFD	0 - (2^16-1)	VFD number

Position	Info	Value range	Notes
26	Connection name		Symbolic name for the connection
27	Extension		Extension for the name

In general, the network parameters and the connection parameters should be set the same on all stations. If the connection parameters do not match, a connection cannot be established. Primarily, the Livelist option in the ***"pb_manager"*** is used to check if the network parameters are correct. If not all stations are listed there or a response to the ***"Livelist"*** is not sent, this is mostly because of inconsistencies in the network parameters. The symbolic connection name is not permitted to begin with a number. The reason for this is that the utilities are expecting the entry in the communication reference or the connection name. It assumes that a number always describes a communication reference, and an alphanumeric character is the beginning of a symbolic name.

Softing recommends the following bus parameters for FMS operation using their hardware:

Baud rate (Kbit/s)	9,6	19,2	93,75	187,5	500
Slottime	100	200	500	1500	3000
minStationDelay	30	60	125	250	500
maxStationDelay	50	100	250	500	1000
SetupTime	5	10	15	25	50
Quiet Time	22	22	22	22	22
TargetRotationTime	10000	15000	30000	50000	100000
GapUpdate	1	1	1	1	1
HSA	126	126	126	126	126
Retry Count	1	1	1	1	1

Remarks:
The services provided by the individual controllers are variable. Some controllers support all FMS services, other only support a few (but they are enough for control system communication). When establishing a connection, a check is made to determine if the respective partner provides the required services. If this is not the case, the connection is not established. The required services result from carrying out an OR operation on the individual bits in the settings C0 to C2 or S0 to S2. The values C0 to C2 are the services required by the control system on the controller. The values S0 to S2 are the services provided for the controller. The following table describes the value for the OR operation and the respective service connected to it:

Bit value in Byte	Service
0x01 in C0 / S0	Create-/Delete-Program-Invocation
0x02 in C0 / S0	Request-Domain-Upload
0x04 in C0 / S0	Request-Domain-Download
0x08 in C0 / S0	Domain-Upload
0x10 in C0 / S0	Domain-Download

0x20 in C0 / S0	Put-OD
0x40 in C0 / S0	Unsolicited-Status
0x80 in C0 / S0	Get-OD (extensive variant)

Bit value in the byte	Service
0x01 in C1 / S1	Physical-Write
0x02 in C1 / S1	Physical-Read
0x04 in C1 / S1	Write-With-Type
0x08 in C1 / S1	Read-With-Type
0x10 in C1 / S1	Write
0x20 in C1 / S1	Read
0x40 in C1 / S1	Kill-Program-Invocation
0x80 in C1 / S1	Reset-/Resume-/Start-/Stop-Program-Invocation
0x01 in C2 / S2	Addressing-By-Name
0x02 in C2 / S2	Alter-Event-Condition-Monitoring
0x04 in C2 / S2	Acknowledge-Event-Notification
0x08 in C2 / S2	Event-Notification-With-Type
0x10 in C2 / S2	Event-Notification
0x20 in C2 / S2	Define-/Delete-Variable-List
0x40 in C2 / S2	Information-Report-With-Type
0x80 in C2 / S2	Information-Report

Normally, the services FMS-Read and FMS-WRITE (as client services) are enough for communication from the control system to the controller. This corresponds to the following combination (in hexadecimal) for C0 to C2 and S0 to S2: 00 30 00 00 00 00

The other required services are fixed and cannot be switched on explicitly.
In the example configuration shown above, all services are switched on for the client and slave (all bits are set: FF FF FF FF FF FF)

10.5.2 Notes concerning the object file profibusx.ov

The object descriptions are used to provide the network with information about all local variables. The control system normally does not generate its own Profibus variables and the driver does not provide corresponding services, therefore you should work with the default file. The following table shows a printout of the default object description file. A detailed description of the file can be found in the SOFTING documentation.

```
;
1         ; VFD-Number
;
;
;OV-Header Objekt
;Parameter  description
;_____________________________
1         ;OV-Header Object-Code
```

```
0          ;OV-Header Index
255        ;ROM/RAM Flag
14         ;length of symbolic names
0          ;Access Protection
1          ;OV-Version
FFFFFFFF   ;Local Address OV-OB
20         ;ST-OV Length
FFFFFFFF   ;Locale Address ST-OV
0xfe00     ;First Index S-OV
100        ;S-OV Length
FFFFFFFF   ;Local Address S-OV
0xff00     ;First Index DV-OV
2          ;DV-OV Length
FFFFFFFF   ;Local Address DV-OV
0xff64     ;Erster Index DP-OV
5          ;DP-OV Length
FFFFFFFF   ;Local Address DP-OV
;
;
;Basis types
;OC Index description
;_____________________
 5   1     [Bool];
 5   2     [Int8];
 5   3     [Int16];
 5   4     [Int32];
 5   5     [Usign8]
 5   6     [Usign16]
 5   7     [Usign32]
 5   8     [Float]
 5   9     [VString]
 5  10     [OString]
 5  11     [Date]
 5  12     [TofDay]
 5  13     [TDiff]
 5  14     [BString]
;
;
;Static Data types
;OC Index #Elems  Typ/Length ( x #Elems)
;_____________________________________
; 6    15     2 5 / 100  6 / 20
; 6    16     3 3 / 2  4 / 4  2 / 1 ;
; 6    17     4 2 / 1  3 / 2  7 / 4  10 / 3 ;
;
;
;Objekte 'Simple variable'
;OC Index          Typ  Length Pass  AccGrp AccR Adr     SymName   Ext
;_____________________________________________________________________
  7   0xfe00          3     2   17     C0   3303 FFFFFFFF [INFO_RPT]   0
  7   0xfe01          2   200   17     C0   3303 FFFFFFFF [CAE_EVENT]  0
  7   0xfe02          3     1   17     C0   3303 FFFFFFFF [CAE_EVENT]  0
;
;Objekte 'Array'
;OC Index #Elems   Typ  Length Pass  AccGrp AccR Adr    SymName   Ext
;_____________________________________________________________________
;  8   100      20    3     1   01     01    1000 FFFFFFFF [DB100]   0
```

10.6 Description of the utilities (ProfiboardDriver)

10.6.1 pb_install

The "***pb_install"*** tool is in the ***/opt/aprol/bin*** directory. Super-user rights are needed to start it. With ***"pb_install"***, the hardware settings are patched to the device driver, the start script is generated and the PROFIBUS software is restarted. If the hardware settings are changed, it is sufficient to start ***"pb_install"*** and then make the entries like a new installation.

The following options are supported:

Option	Description
-help	Shows help information
-nc	Starts without curses menu

10.6.2 pb_init

The ***"pb_init"*** program is used for loading the network parameters, the connections parameters and the object description on the card. After running ***"pb_init"***, the board is ready for operation and provides the FMS services. The following options are supported here:

Option	Description	Default value
-help	Shows help information	
-b BOARD_NO	Initializes the board BOARD_NO; Valid values are 0 to 2	Board 0
-f FILENAME	Network and connection description in FILENAME	/usr/etc/profibus/profibus.cfg
-o OBJECTFILE	Object description in OBJECTFILE	/usr/etc/profibus/profibus.ov
-d	Starts in debug mode and shows additional information	Switched off
-useFdl	Activates layer 2 communication, which is used by APROL.	Without layer 2 communication

10.6.3 pb_manager

The ***"pb_manager"*** utility combines many of the functions of the individual utilities in a menu driven user interface. This tool provides the most possibilities for analyzing the network and finding errors. Used together with the hexdump for the device driver, nearly any error can be found and corrected. The ***"pb_manager"*** requires the ***"ncurses"*** package from Linux. This usually comes already installed.

The ***"pb_managers"*** is controlled by selecting menu items instead of using start options. The only start option that is supported is the selection of the board using the ***-b BOARD_NO*** option.

There are five basic menus that you can go to directly using the page number even if the menu option is not shown. The following section offers a brief description of the individual services.

Menu item	Function
SPACEBAR	With the spacebar, you can change the board number in the individual selection menus die. This corresponds to the start option -b BOARD_NO. This selection possibility is not shown in any menu!
Page 1	
show device driver version	Shows the version number of the kernel driver.
show device area	Shows the contents of the PROFIboard register. This option is designed for debugging the hardware and has no importance for the normal user!
show device driver common area	Shows the contents of the kernel driver variables. This option is designed for debugging the driver and has no importance for the normal user!
show board info (FM7_IDENT_LOC)	Shows the PROFIboard firmware version (among other things).
Page 2	
show network live list	Shows the list of stations on the network. The program detects this list in two ways. One of the two methods is used by choosing 'a' or 'A'. Method 'a' only shows the active stations; method 'A' also shows the passive station.
show local bus parameters	Shows the bus parameters currently set
show local crl	Shows an individual communication relationship. This must be selected by entering the communication reference or the connection name.
name to commref	Shows the communication reference that corresponds to a symbolic connection name that is to be entered.
Page 3	A connection must be established for all of the services described here. The possibility to establish a connection can be tested with FMS INITIATE.
FMS INITIATE	Establishes an FMS connection. This can be used to test an entry in the communication relations list. After the connection has been established, breaking the connection must be triggered by pressing any key.
FMS IDENTIFY	Shows the ID for the communication partner. This ID is assigned to the hardware by the hardware manufacturer.
VFD STATUS	Shows the status of the communication partner. It is entirely possible that communication via PROFIBUS is functioning when the controller is stopped. This type of status can be requested remotely using this item.
NAME_TO_INDEX	With this function, the index of variables can be requested for variable names on a controller.

Menu item	Function
FMS PHYSICAL READ	Reading a physical memory area on the controller by entering the memory address and block length.
FMS_READ	Reading a variable via PROFIBUS. Reading in this way must be specified precisely. It is possible to read process variables using the name or index and also variable lists.
FMS_WRITE by index	Writing to a variable by entering the index.
Page 4	These services are special functions that only work with a B&R System 2000 controller! A connection must be established for all of these services!
controller RESET	Triggers a reset on the controller. also see pb_controllerreset.
KEY POSITION	Requests the position of the key switch.
MODULE UPLOAD	Uploads a module/task from the controller with the possibility to save it on the control computer.
MODULE DOWNLOAD	Downloads a file to the controller. This can be a library, a module, a task or a data module. As an option, the file can be burned to the controller, i.e. it will still be available after a reset.
MODULE INFO	Requests and displays information about a controller module.
DELETE MODULE	Deletes a module from the controller.
OBJECT LIST	Displays the objects on the controller. also see pb_list.
controller SET TIME	Sets the computer time on the controller. Unlike the tool pb_settime, the time is set here without using a special task on the controller.
DOWNLOAD INFO	Requests information about the last download. This makes it possible to check if the last download was successful and if the downloaded module was able to be started.
Page 5:	This tests the functionality of layer 2 communication (FDL). These are used amongst others by APROL.
ACTIVATE LSAP	Activates an LSAP on the PROFIboard. This LSAP should now no longer be used by the FMS.
DEACTIVATE LSAP	Switches off an activated LSAP.
INA CONNECT	Establishes a connection using APROL communication.

10.6.4 pb_debug

With ***"pb_debug"***, individual debug information items can be activated in the device driver. The individual options can be activated or deactivated by setting or clearing the corresponding bits.

Please note that the debug flags work in the kernel and can affect the performance of the computer considerably

Option	Description
-help	Shows help information
-s DEBUG_FILTER	Sets the debug filter to DEBUG_FILTER
Bit 0: 0x00000001	Outputs error messages
Bit 1: 0x00000002	Outputs warning messages
Bit 2: 0x00000004	Outputs entry messages
Bit 3: 0x00000008	Shows the configuration (only for pb_init)
Bit 4: 0x00000010	Shows sleep messages
Bit 5: 0x00000020	Shows interrupt messages
Bit 8: 0x00000100	Hexdump for the incoming telegram header
Bit 9: 0x00000200	Hexdump for the incoming telegram data
Bit 10: 0x00000400	Hexdump for the outgoing telegram header
Bit 11: 0x00000800	Hexdump for the outgoing telegram data
Bit 28: 0x10000000	Outputs queue information

Debug outputs can be made visible by the super-user using "***dmesg"***.

10.6.5 pb_netconfig

Tool for generating a network configuration file for APROL. This file is created in a format that can be read by ***"pb_init"***. A separate description file ***PROFIBUSa.CFG*** (a = number of the PROFIboard on the computer) in the directory ***TARGET_DIR/STx*** (x = station number in the network) is created for each PROFIboard in the network. The network description file has the following structure:

```
# Comment lines
# One line per station with the following structure
# PROFIBUS_ADDRESS/STATION_TYPE/[REDUNDANT]
#
# Profibus addresses in the range from 0 to 31
# Station types are 2005, 2010, PROFIBOARD, OTHER
# REDU entry 0 = no redundancy, 1 = redundancy (only for PROFIBOARDS)
# Example configuration
0/PROFIBOARD/1
1/PROFIBOARD/0
10/2010
11/2005
12/2003
# Creates ST0/PROFIBUS0.CFG
#         ST0/PROFIBUS1.CFG for redundant PROFIboard
#         ST1/PROFIBUS0.CFG
# The APROL standard configuration is expected on the controller side so that a
# connection can be established!
```

The following options were supported by ***"pb_netconfig"***:

Option	Description	Default value
-help	Shows help information	

Option	Description	Default value
-c CFG_FILE	Reads the network description from CFG_FILE	None, mandatory parameter
-p SERVICES	Number of parallel services that should be supported	7
-hsa HSA	Highest active station in the network	Taken from the network description
-b BAUDRATE	Baud rate to be used. also see above. The network parameters for the respective baud rate correspond to the Softing default settings	4
-t TARGET_DIR	The configuration files for all PROFIboard devices in the network are stored in the directory TARGET_DIR/station number.	Local directory

10.6.6 pb_list

Utility that can be used to read the object directory on the controller. The program requests the object list from the controller and shows the names of the existing variables. The service must be activated in the CRL on the slave side!

Option	Description	Default value
-help	Shows help information	
-b BOARD_NO	Initializes the board BOARD_NO; Valid values are 0 to 2	Board 0
-conn KR	Selects the connection using the communication reference or the connection name	None, mandatory parameter
-area PB_AREA	Lists variables of the type PB_AREA: Valid values for PB_AREA are: 0: Shows all variables 1: Shows the main types 2: Shows all structures 3: Shows the variable lists 4: Shows the program invocations 5: Shows the domain information 6: Display of the process variables	Shows all variables

10.6.7 pb_history

Tool used to read the history buffer on a B&R System 2000 controller. The tool tries to read the text messages from a file that is set up using the environmental variable "***BR_TEXT_FILE"***. Normally, this is the file ***/usr/etc/profibus/br.txt***.

Option	Description	Default value
-help	Shows help information	
-b BOARD_NO	Initializes the board BOARD_NO; Valid values are 0 to 2	Board 0
-conn KR	Selects the connection using the communication reference or the connection name	None, mandatory parameter
-l	Displays additional information (long mode)	Not defined
-s START	Numb of the first entry to be read, START <= 39	0
-n NUMBER	Number of entries to be read. NUMBER <= 40 - START	10

10.6.8 pb_controllerreset

Tool used to trigger a reset command on a B&R System 2000 controller via PROFIBUS. The reset modes ***Normal***, ***Total*** and ***Diagnose*** are supported.

Please note that the debug flags work in the kernel and can affect the performance of the computer considerably

Option	Description	Default value
-help	Shows help information	
-conn KR or NAME	Selects the connection using the communication reference or the connection name	None, mandatory parameter
-mode RESET MODE	Sets the reset mode RESET MODE: 0: NORMAL_INIT 1: TOTAL_INIT 32: DIAGNOSTICS	NORMAL_INIT
-d	Debug mode, shows additional information	No debug mode

10.6.9 pb_read

Tool to cyclically read one or more variables from the controller. The configuration for reading a variable is made in the form of a start parameter. If multiple variables are to be read, a configuration file is required!

Option	Description	Default value
-help	Shows help information	

Option	Description	Default value
-b BOARD_NO	Initializes the board BOARD_NO; Valid values are 0 to 2	Board 0
-conn KR or NAME	Selects the connection using the communication reference or the connection name	None, mandatory parameter
-cycles CNT	Cyclically reads CNT cycles	1 cycle
-delay DELAY	Time between two read cycles in seconds	1 second
-name NAME	Reads the variable NAME or INDEX if the name begins with a number. Ex.: NAME=TEST Reads variable TEST NAME = 0x100 Reads variable with index 0x100	None, mandatory parameter
-format FORMAT	Display format for the variable: I: Integer representation in format signed U: Integer representation in format unsigned F: Float representation	Display in Hex format
-file FILENAME	Configuration file if more than one variable should be read. One line must be used per variable in the format NAME, space, FORMAT.	

10.6.10 pb_timesync

Tool used to distribute the controller time to multiple controllers in order to synchronize the clock times. This utility is only to be used in connection with a standard APROL task and has no other effect.

Option	Description	Default value
-help	Shows help information	
-b BOARD_NO	Initializes the board BOARD_NO; Valid values are 0 to 2	Board 0
-conn KR or NAME	Selects the connection using the communication reference or the connection name	None, mandatory parameter
-start	Starts clock synchronization	

Option	Description	Default value
-stop	Stops clock synchronization	
-sync RATE	Synchronization of all RATE values in seconds	60 seconds

10.6.11 pb_settime

Tool to set the computer time on the controller. This utility is only to be used in connection with a standard APROL task and has no other effect.

Option	Description	Default value
-help	Shows help information	
-b BOARD_NO	Initializes the board BOARD_NO; Valid values are 0 to 2	Board 0
-conn KR or NAME	Selects the connection using the communication reference or the connection name	None, mandatory parameter

10.6.12 pb_taskmgr

Tool for downloading, uploading or clearing, starting or stopping a task on a B&R System 2000 controller.

Option	Description	Default value
-help	Shows help information	
-b BOARD_NO	Initializes the board BOARD_NO; Valid values are 0 to 2	Board 0
-conn KR or NAME	Selects the connection using the communication reference or the connection name	None, mandatory parameter
-download FILE	Loads FILE to the controller	
-burn	Saves the task (FILE) on the controller so that it is still available after a reset Only together with the option -download.	Do not save
-dom_info DOM_NAME	Starts the GET-OV service for the task DOM_NAME	
-start	Starts the task DOM_NAME. Only together with the option dom_info.	
-stop	Stops the task DOM_NAME. Only together with the option dom_info.	

Option	Description	Default value
-delete	Deletes the task DOM_NAME. Only together with the option dom_info.	
-upload MODULE FILENAME	Loads a module/task MODULE from the controller and saves it as FILENAME.	

10.7 Configuration of the APROL driver (ProfiboardDriver)

10.7.1 General information about the driver configuration

The APROL driver can be configured two ways when starting. Part of the configuration is transferred to the driver in the form of start options (also see section *Description of the InaDriver launching options*), the task and the process variables are configured using a configuration file (also see the chapter *APROL driver configuration file*).

10.7.2 Description of the ProfiboardDriver's start options

The following section offers a description of the start options for the driver. The start options can be combined in any way. Some of the options require additional parameters and some can simply be called as they are. When parameters are required, symbolic parameter names are defined that must be replaced by their values. Some of the parameters, such as specification of the connection name, are mandatory parameters and others are optional.

Start option for the Profiboard driver	Description
-D	Displays the date the driver was created and is then exited (optional parameter).
-timeout TIMEOUT_VALUE	Sets the time-out time to TIMEOUT_VALUE milliseconds. When a driver does not respond to its request telegram within this time-out time then the request expires and an error is output. Allowed values are between 1000 and 10000 milliseconds. Limit violations are corrected automatically.° (optional parameter, default value: 2500)
-n DRIVER_NAME	Sets the name of the driver in the Iosys to DRIVER_NAME. The status variables for the connection states and error indicators are generated with this name.° (optional parameter, default value: PROFIBUS)

Start option for the Profiboard driver	Description
-run_task	The driver assumes that the configured task on the controller is in a "running" state and immediately begins communication after starting. This option has no effect on FMS operation, but is only used in APROL. (optional parameter, not for FMS driver)
-ignoreCnfPath	The driver searches for its configuration file relative to the present position instead of in the default directory.° (optional parameter)
-set_id CLIENT_ID	This option sets the client ID in the Iosys to the value CLIENT_ID. Which application has set the value of a variable can be detected with the output of IosEv using the client ID.° (optional parameter, default: 31263)
-l LOGFILE	The driver creates a log file with the name LOGFILE. Start parameters and output status are written here and can be read during operation.° (optional parameter)
-i FMS_CONFIG_FILE	The driver uses the configuration file FMS_CONFIG_FILE for its configuration. If the *-ignoreCnfPath* option isn't used, the location of the configuration file refers to the default configuration directory. Otherwise, it is relative to the start position of the driver.° (mandatory parameter for FMS driver).
-c CAE_CONFIG_FILE	The driver in the APROL operating mode bases its configuration on the file CAE_CONFIG_FILE. Also here, if the *-ignoreCnfPath* option isn't used, the location of the configuration file refers to the default configuration directory, otherwise, it is relative to the start position of the driver.° (not for FMS driver)
-conn0 VBG0	The driver uses the connection VBG0 for communication with the controller. The connection name specified using VBG0 must have a corresponding entry in the CRL for the Profibus parameters. (mandatory parameter)
-conn1 VBG1	The conn1 connection is used as a bus redundant connection for conn0. If the conn0 connection is lost or can't be established, the driver attempts communication via conn1.

Start option for the Profiboard driver	Description
-conn2 VBG2	The conn2 connection is to be used for controller redundancy. The driver establishes the connection to both controllers and decides which one is the master and communicates via the appropriate connection. If the master and slave are switched, then the driver uses the other connection. Controller redundancy only works in APROL since a special task must be present on the controller which lets the driver know whether it (the controller) is the master or slave.
-conn3 VBG3	The conn3 connection is used as a bus redundant connection for conn2. If the conn2 connection is lost or can't be established, the driver attempts communication via conn3.
-d DEBUG_FILTER	The driver is started in the debug mode using the -d option (i.e. it provides permanent status information instead of going into the background). The information is output either to the log file or to the screen via stderr. To avoid having all of the output on the screen, you can tell the driver which information to provide and which to ignore by specifically setting individual bits in the debug filter. The set bits in the debug filter allow the following output: 0x00000001: Output of error messages from the state machine 0x00000002: Output of messages from the state machine 0x00000004: Output of configuration messages at startup 0x00000002: Online configuration messages 0x00000010: Hexdump from answer telegrams 0x00000020: Hexdump from request telegrams 0x00000040: Output of the Invoke ID for the request° 0x00000080: Shows APROL event telegrams 0x00000100: Shows VFD status telegrams° 0x00000200: not used 0x00000400: Shows APROL task info telegrams 0x00000800: Display of connection information 0x10000000: Display of the state machine 0x20000000: Display of the IO telegrams 0x40000000: Display of the Iosys events 0x80000000: Display of the process variables
-f DEBUG_FILTER	The driver enables the debug information output as described above. It starts in the background despite the output.° (optional parameter)

Start option for the Profiboard driver	Description
-delay DELAY_FACTOR	The driver evaluates all time settings in the configuration file as a multiple of the DELAY_FACTOR in milliseconds. The default setting for DELAY_FACTOR is 1000 ms. The lowest possible setting for the DELAY-FACTOR is 100 milliseconds.° (optional parameter, default value: 1000)
-r	The driver is started in redundancy mode.° (optional parameter)
-reduCheckTime <time [s]>	The driver starts an Iosys timer with the time "reduCheckTime" in seconds. The corresponding callback routine **checks whether at least one functioning connection exists**. If there is, the timer is restarted with the time configured here. **If there is no functioning connection**, the driver returns the master status and restarts itself after the "restartTime" in seconds. Value range: [10 – 300]
-restartTime <time [s]>	Using an Iosys timer and the 'reduCheckTime' time, this parameter checks whether at least one functioning connection exists. If there is, the timer is restarted with the 'reduCheckTime' time configured here. **If there is no functioning connection**, the driver returns the master status and restarts itself after the "restartTime" in seconds. **Value range: [5 – 300]**
-a	The driver is started in the so called access mode.° (optional parameter)

Start option for the Profiboard driver	Description
-t6 TIMER6	The automatically generated VFD task for the driver executed and updated cyclically every TIMER6 * DELAY_FACTOR milliseconds. The VFD task checks the status of the controller at regular intervals. Two state types are defined under FMS: the logical state and physical state. The automatically generated VDF variables (integers) contain the logical status in the second byte and the physical status in the third byte (beginning with zero byte). A value of 0x00000000 means that the controller is 'running'. A value of 0x0000FFFF is shown if the actual VFD status cannot be determined because, for example, a connection cannot be established. All other values mean that an error has occurred and the controller must be serviced. (optional parameter, default value: 10) The following status values are used Logical status Meaning 0 Ready for communication, all services allowed 2 Limited services available 4 Loads the object description 5 All connections closed because an object description is being loaded Physical Meaning 0 Ready for use 1 Partially ready for use 2 Not ready for use 3 Service required
-board BOARD_NUMBER	The driver communicates using the PROFIboard BOARD_NUMBER. Allowed values are 0 or 1.° (optional parameter, default value: 0)
-maxPar SERVICE_COUNT	Allowed values are between 0 and the *scc* setting of the parallel confirmed services for this connection.° Here, you stipulate how many read requests the driver is allowed to send in parallel until it must wait for an answer.° (optional parameter, default value: Connection settings)
-readTimer READ_TIMER_VALUE	Allowed values are between 10 ms and 500 ms.° (optional parameter, default value: 50)° Specifies how often the APROL driver attempts to receive telegrams from the device driver. 50)° Specifies how often the **Fehler! Kein Name für eine Eigenschaft übergeben.** driver attempts to receive telegrams from the device driver. At the same time, you are changing the system load, so use this option with care.

10.7.3 Configuration file of the APROL driver (ProfiboardDriver)

A configuration file is used to configure the driver's tasks and the process variables that will be used as well as their positions. The name and position of the configuration file are specified in the driver at startup using the ***"-i FILENAME"*** option. The position is relative to the default setting for driver configurations in APROL. More information can be found in the respective documentation. This default setting can be ignored by using the ***"-ignoreCnfPath"*** option and FILENAME describes the absolute position of the configuration file.

The driver configuration consists of a list of tasks. Each task consists of a task line and a list of the process variables used in this task. Each task line begins with the character **^** as code for a new task. Comment lines begin with the # character and run to the end of the line.

A driver configuration generally looks something like this:

```
# Comment line
#
^AUFTRAGSART_1/PB_NAME1 oder PB_INDEX1/Task/PB_LÄNGE1/AUFTRAGSZYKLUS1
PV_NAME11 OFFSET11 PV_TYP11 IOSYS_TYP11 LS_LOW11 LS_HIGH11 SPS_LOW11 SPS_HIGH11
PV_NAME12 OFFSET12 PV_TYP12 IOSYS_TYP12 LS_LOW12 LS_HIGH12 SPS_LOW12 SPS_HIGH12
PV_NAME1x OFFSET1x PV_TYP1x IOSYS_TYP1x LS_LOW1x LS_HIGH1x SPS_LOW1x SPS_HIGH1x
^AUFTRAGSART_2/PB_NAME2 oder PB_INDEX2/Task/PB_LÄNGE2/AUFTRAGSZYKLUS2
PV_NAME21 OFFSET21 PV_TYP21 IOSYS_TYP21 LS_LOW21 LS_HIGH21 SPS_LOW21 SPS_HIGH21
PV_NAME22 OFFSET22 PV_TYP22 IOSYS_TYP22 LS_LOW22 LS_HIGH22 SPS_LOW22 SPS_HIGH22
PV_NAME2x OFFSET2x PV_TYP1x IOSYS_TYP2x LS_LOW2x LS_HIGH2x SPS_LOW2x SPS_HIGH2x
```

The number of tasks in a configuration file is theoretically unlimited, and so is the number of process variables within a task. The only limitation to the configuration results from the amount of memory used by the driver.

The meanings of the individual identifiers are described in the following sections.

TASK TYPES:
The driver differentiates between four different types of tasks during FMS communication:

- ***"FMS_READ"***,
- ***"FMS_WRITE"***,
- ***"FMS_IWRITE"***,
- ***"FMS_READ"***,

The differences between the types of operation depend on the communication direction.

"***FMS_READ"*** tasks read cyclic variables from the controller and update the process variables in the Iosys. If the variables are changed (written) by another application, then the values are overwritten the next time the driver reads the values.

"***FMS_WRITE"*** tasks write data to the controller as soon as the process variable is changed within the task. Variables that for whatever reason could not be written properly are reset to the previous value.

"***FMS_IWRITE"*** tasks behave mostly like ***"FMS_WRITE"***, but the task must be read just once to align the control computer with the controller. This read process generally occurs once after each time the connection is established. If process variables are changed before the alignment has been made, then the variables are reset to the previous value.

FMS_SYNC tasks allow communication in both directions. This is a combination of cyclic READ procedures with event-controlled WRITE tasks. Variables can only be written to the controller if they have been read at least once. If value changes are made before they can be transferred to the controller, then driver undoes the changes and the previous value is set on the controller.

Task:
To use the driver as an FMS driver, this field must contain a default value (e.g. "Dummy_Task").

When using the driver in APROL, the PVs are organized in tasks; individual tasks can then be activated and deactivated.

PB_NAME:
This is used to specify the name of the Profibus variables to be read/written on the controller. The index of a variable can be used as an alternative to the name. The use of names is not supported by all controllers.

PB_INDEX:
This is used to specify the index of the Profibus variables to be read/written on the controller. The name of a variable can be used as an alternative to the index if the controller supports using names.

PB_LENGTH:
This is used to specify the expected size of the variables. If the size received does not match the size configured, then an error message is output and the telegram is thrown out.
In general, not more than 200 bytes should be read or written!

TASK CYCLE:
The task cycle specifies how often read tasks are to be repeated i.e. the time between them. The task cycle refers to the time between receiving the last response telegram and sending the next request telegram. The task cycle is the product of the DELAY_FACTOR and the TASK CYCLE in milliseconds. For WRITE tasks and IWRITE tasks, TASK CYCLE has no effect.

PV_NAME:
Name of the process variables in Iosys. Using this name, the process variables in the Iosys can be accessed. Process variable names cannot have more than 64 characters.

OFFSET:
The offset specifies the position of the process variables relative to the beginning of the task block. It is entered in bytes.

PV_TYPE:
The PV type determines what kind of variable must be found at the position for the offset. Among other things, the setting for the PV type affects the default measuring range of the variables.

The following PV types are recognized:

PV type	PV length	Default measurement range
BIN0 to BIN7	1 bit	0 to 1
INT8	1 byte	-128 to 127

PV type	PV length	Default measurement range
UINT8	1 byte	0 to 255
INT16	2 Bytes	-32768 to 32767
UINT16	2 Bytes	0 to 65535
INT32	4 Bytes	-2147483648 to 2147483647
UINT32	4 Bytes	0 to 4294967295
FLOAT	4 Bytes	-3E38 to 3E38
S5FLOAT	4 Bytes	-3E38 to 3E38
STRINGxx	XX bytes	None, maximum 64 characters long

The driver carries out a plausibility check during the configuration. If offset and variable length go beyond the task block, a corresponding error message is output and the driver is ended.

IOSYS_TYPE:
The Iosys type specifies which type the variable has in the Iosys. Three types are known to the Iosys:

- Integer (whole numbers),
- Real (floating point numbers),
- String type with a maximum of 64 characters.

The Iosys type is entered in the form of a single character: I for integer, R for real and S for string. The type of the variables affects the maximum measurement range and the representation precision. For precision reasons, it doesn't make much sense e.g. to normalize an integer variable.

SPS_LOW and SPS_HIGH:
Measurement range for the variables on the controller. The measurement range is used for scaling to determine the multiplication factor and is also used to prevent operating errors. It only works in the direction from the control computer to the controller. If a variable value is entered in the control computer that is outside of the permissible measurement range (directly or as a result of scaling), then the driver resets the variable to the limit value and the limit value is sent to the controller. A value from the controller that is outside the limits is transferred to the control computer as is. SPS_LOW and SPS_HIGH are optional. If not otherwise specified, the default values from the PV type are used. Both of these values must be explicit if scaling is required!

LS_LOW and LS_HIGH:
LS_LOW and LS_HIGH are the control computer's limit values. Together with SPS_LOW and SPS_HIGH, they are used to compress or stretch (scaling) a variable. They are optional. When these values are not specified then there is a 1:1 scale. They can only be specified when SPS_LOW and SPS_HIGH have also been set.

The following formula is used to calculate a variable value:

```
# Configuration for normal FMS services
#
# ^TASK/NAME or INDEX/Task_Name/LENGTH in Bytes/DELAY in milliseconds
#   PV_NAME  OFFSET in Bytes  PV_TYPE  IOS_TYPE [LS_LOW  LS_HIGH]  [SPS_LOW
SPS_HIGH]
#
# TASK: FMS_READ
#          FMS_WRITE
#          FMS_IWRITE
#          FMS_SYNC
#
# PV_TYP:  BIN0 - BIN7
#          INT8 / UINT8
#          INT11
#          INT12
#          INT16 / UINT16
#          INT32 / UINT32
#          FLOAT
#          S5_FLOAT
#          STRINGxx ( String der Laenge xx Bytes )
#
# IOS_TYP: I fuer Integer
#          R fuer Real
#          S fuer String
#
# Optionale Parameter:
# SPS_LOW:  Messbereichsanfang
# SPS_HIGH: Messbereichsende
# LS_LOW:   Wert im Leitsystem bei SPS_LOW auf SPS
# LS_HIGH:  Wert im Leitsystem bei SPS_HIGH auf SPS
#
#########################################################################
# Bits in Byte
^FMS_SYNC/BTEST1/Dummy/01/1/1/
  BIN00  0 BIN0
  BIN01  0 BIN1
  BIN02  0 BIN2
  BIN03  0 BIN3
  BIN04  0 BIN4
  BIN05  0 BIN5
  BIN06  0 BIN6
  BIN07  0 BIN7
# Bits in Wort
^FMS_SYNC/WTEST1/Dummy/01/2/1/  /* auf Words, Byte-Offsets vertauschen */
  BIN00  1 BIN0
  BIN01  1 BIN1
  BIN02  1 BIN2
  BIN03  1 BIN3
  BIN04  1 BIN4
  BIN05  1 BIN5
  BIN06  1 BIN6
  BIN07  1 BIN7
  BIN10  0 BIN0
  BIN11  0 BIN1
  BIN12  0 BIN2
  BIN13  0 BIN3
  BIN14  0 BIN4
  BIN15  0 BIN5
  BIN16  0 BIN6
  BIN17  0 BIN7
```

```
# Bits in Long ( 4 Bytes )
^FMS_SYNC/LTEST1[0]/Dummy/01/4/1/  /* auf Long, Drehen zur Variablenmitte */
  BIN00  3 BIN0
  BIN01  3 BIN1
  BIN02  3 BIN2
  BIN03  3 BIN3
  BIN04  3 BIN4
  BIN05  3 BIN5
  BIN06  3 BIN6
  BIN07  3 BIN7
  BIN10  2 BIN0
  BIN11  2 BIN1
  BIN12  2 BIN2
  BIN13  2 BIN3
  BIN14  2 BIN4
  BIN15  2 BIN5
  BIN16  2 BIN6
  BIN17  2 BIN7
  BIN20  1 BIN0
  BIN21  1 BIN1
  BIN22  1 BIN2
  BIN23  1 BIN3
  BIN24  1 BIN4
  BIN25  1 BIN5
  BIN26  1 BIN6
  BIN27  1 BIN7
  BIN30  0 BIN0
  BIN31  0 BIN1
  BIN32  0 BIN2
  BIN33  0 BIN3
  BIN34  0 BIN4
  BIN35  0 BIN5
  BIN36  0 BIN6
  BIN37  0 BIN7
# Position von Bytes und Words in Langwort
^FMS_SYNC/LTEST1[0]/Dummy/01/4/1/
  BYTE00  3 UINT8
  BYTE01  2 UINT8
  BYTE02  1 UINT8
  BYTE03  0 UINT8
  WORD00  2 UINT16      0   1000    -1000   +1000
  WORD01  0 UINT16      0   1000    0       10000
  LONG00  0 UINT32
# Sync mit Float-Array
^FMS_SYNC/FLOAT1/Dummy/01/16/1/
  FLOAT00  0 REAL
  FLOAT01  4 REAL
  FLOAT02  8 REAL
  FLOAT03 12 REAL
# Lesen einer Strukturvariablen
^FMS_READ/sys_data/Dummy/01/46/1/
  pv.sys.aws_name    0   STRING6
  pv.sys.aws_typ     6   STRING2
  pv.sys.cpu_info    8   UINT32
  pv.sys.ma_global  12   UINT16
  pv.sys.md_global  14   UINT16
  pv.sys.os_len     16   UINT32
  pv.sys.user_len   20   UINT32
  pv.sys.tmp_len    24   UINT32
  pv.sys.eprom      28   UINT32
  pv.sys.fixram     32   UINT32
  pv.sys.battery    36   UINT8
  pv.sys.run        37   UINT8
  pv.sys.init_mode  38   UINT32
  pv.sys.os_version 42   STRING4
```

10.8 Profiboard driver status variables

The driver creates a few status variables in the Iosys for displaying and analyzing errors. These are generated and supplied automatically. The driver name is the main part of the status variable name. The following variables are generated:

Variable name	Meaning	Type and direction	Values
DRIVER_NAME.CONN_NAME.connected	Displays the connection status	Integer, from the driver	0 = not connected 1 = establishing connection 2 = connection established
DRIVER_NAME.channel	Displays the currently active connection in redundancy mode	Integer, from the driver	0 = conn0 1 = conn1 2 = conn2 3 = conn3
DRIVER_NAME.error	Displays the last error number	Integer, from the driver	See error messages
DRIVER_NAME.neg_counter	Counter that is increased by one with each negative response	Integer, from the driver	
DRIVER_NAME.pos_counter	Counter that is increased by one with each positive response	Integer, from the driver	
DRIVER_NAME.act_counter	Number of active tasks in the driver	Integer, from the driver	
DRIVER_NAME.config_req	Variable for online driver configuration	Message box, to the driver	Only for APROL
DRIVER_NAME.debug_filter	Variable for reading and setting the driver's current debugging filter	Integer, from and to the driver	See above
DRIVER_NAME.error_txt	Error text for the most recent error	String64, from the driver	See error messages
DRIVER_NAME.vfd_status	Variable for evaluating the controller status.	Integer, from the driver	

10.9 APROL driver error numbers and error messages

10.9.1 Error numbers and messages (ProfiboardDriver)

The utilities and the APROL driver use different error numbers and handling mechanisms.

The APROL driver works with error numbers written to the error variable and partially also with plain text messages written to the error text variable.

For communication, the utilities use a library whose error numbers are the same for all utilities. Library errors start at the offset 400. The error numbers from the APROL driver all start at the offset 1000.

The library error numbers:

Error number	Cause of the error
401	Error writing telegram
402	Timeout waiting for the response telegram
403	Negative response telegram received (structure T_ERROR)
404	Error reading telegram
405	Abort indication received from the board with the result: Connection lost
406	Reject telegram received Possible reason: service not allowed° Too many parallel requests
407	Download aborted/ended by error (task not present on controller)
408	Get-OV error, Object not present
409	Cannot open device driver Possible cause: Board not recognized/present Too many applications at the same time
410	Communication reference already occupied by another application
411	Parameter error, Value outside valid range

APROL driver error numbers:

Error number	Cause of the error
1001	Task started, but driver is not connected
1002	Incorrect access type for Read task (only APROL)
1003	An unpredictable event has occurred (e.g. timeout too short, response arrived after timeout)
1004	Timeout, response did not arrive in time
1005	Negative response to request received (e.g. object not present, service not allowed)
1006	Driver not yet initialized, WRITE task is sent before reading once with SYNC or IWRITE

Error number	Cause of the error
1007	System error (only APROL)
1008	Task not present on the controller (only APROL)
1009	Driver in simulation mode, write not possible
1010	Access to invalid controller memory location (only APROL)

In addition to error numbers, the driver also outputs plain text messages. The following messages are possible:

Error text	Cause of the error
got unexpected length: got %d, exp %d, %s	The response to a read telegram does not contain as many bytes as expected. The first selection shows the number of received bytes, the second shows the expected number and the third shows the task name
TASK_INFO failed, %s	Negative response to receive for task info telegram. The first parameter shows the task name. The task is likely not present on the target system. Only APROL
SEARCH_TASK failed, %s	Negative response received to search telegram for controller task. The first parameter shows the task name. The task is likely not present on the target system. Only APROL
out of range: %s	The specified variable delivers a value outside of the defined measurement range.
packet error: %s	A newly defined variable does not fit in the defined task. Either the beginning or the end or the complete variable lies outside the defined task block. Only possible with online configurations.

10.10 Integrating the APROL Profiboard start script

10.10.1 The APROL start script

The APROL start script has the following structure:

```
#    $HOME/etc/init.d/driverProfiboard.sh
#*****************************************************************
#         COPYRIGHT 1996 - 1999 PCC GmbH . ALL RIGHTS RESERVED
#*****************************************************************
#     AUTHOR:       DRIEFMEIER
#     USE FOR:      Starting Driver profibus
#*****************************************************************
#
# 1.0.0  10.06.1999 LU  driverProfiboard.sh erzeugt
VERSION="1.0.0"
VDATE="10.06.1999"
COMP="!= 0"
PROGNAME="ProfiboardDriver"
FILENAME="driverProfiboard.sh"
DEFFILE="$APROL/etc/init.d/globaldefs"
```

```
LOGFILE=$APROL/tmp/$PROGNAME.log
TMPFILE=$APROL/tmp/$PROGNAME.tmp
BIN_DIR="$APROL/bin"
PROGPARAM1=" -n"
PROGPARAM2=" -i"
REDUPROGPARAM1=" -n"
REDUPROGPARAM2=" -r -i"
PROGCNF="$CNF_PATH/$PROGNAME"
LIST="\
ST10  \
"

if [ ! -f $DEFFILE ]; then
 echo "$DEFFILE not found"
 exit 0
else
 . $DEFFILE
fi
$ECHO "call -$FILENAME- with -$1- at $DATE" $LOGFILE
case "$1" in
 'start')
  $ECHO "\t\t$FILENAME $1"
  confcheck $LIST
  for PROGS in $LIST; do
   # check if program exists and cnf file is readable
   if [ -x "$BIN_DIR/$PROGNAME" -a -r "$PROGCNF/$PROGS" ]; then
    # check if already running ...
    getpidcnf $PROGNAME $PROGS
    if [ -z "$PID" ]; then
                    LOGFILE=$APROL/tmp/$PROGS.L
                    TMPFILE=$APROL/tmp/$PROGS.T
     $ECHO "$PROGS<\c"
     getmsg 0002
     # call PROGS
     OLDIR=`pwd`
     cd $BIN_DIR
     # check if redundancy
                    redumode
                    if [ -z "$REDUMODE" ]; then
                     $PROGNAME -conn0 $PROGS $PROGPARAM1 $PROGS $PROGPARAM2
$PROGCNF/$PROGS/profi.cnf 1 $LOGFILE 2&1
                        RET=$?
                    else
                        getmsg 0030
                        getmaster
                        showmachinestate
                        $PROGNAME -conn0 $PROGS $REDUPROGPARAM1 $PROGS $REDUPROGPARAM2
$PROGCNF/$PROGS/profi.cnf 1 $LOGFILE 2&1
                        RET=$?
                    fi
                    tail -4 $LOGFILE $TMPFILE
                   ERROR=`$CAT $TMPFILE |$GREP "error" 2/dev/null`
                    if [ -n "$ERROR" -o "$RET" $COMP ]; then
                        error $PROGS
                    else
                        delay 5
                        getpidcnf $PROGNAME $PROGS
                        if [ -n "$PID" ]; then
                            $ECHO "start $PROGS $DATE" $LOGFILE
                            $ECHO " ok"
                        else
                            starterror $PROGS
                        fi
                    fi
    else
     beeper 3
     $ECHO "$PROGS<\c"
     getmsg 0005
     getmsg 0006
    fi
   else
    beeper 2
    getmsg 0007
    getmsg 0009
   fi
  done
  cd $OLDIR
 ;;
 'stop')
  $ECHO "\t\t$FILENAME $1"
  confcheck $LIST
  for PROGS in $LIST; do
```

```
   getpidcnf $PROGNAME $PROGS
   if [ -z "$PID" ]; then
    $ECHO "$PROGS<\c"
    getmsg 0004
   else
    kill $PID
    $ECHO "$PROGS<\c"
    getmsg 0003
                LOGFILE=$APROL/tmp/$PROGS.L
                $ECHO "stop $PROGS $DATE" $LOGFILE
                TMPFILE=$APROL/tmp/$PROGS.T
                rm $TMPFILE
   fi
  done
 ;;
 'restart')
  ($0 stop; getmsg 0012; delay 5; $0 start)
 ;;
 '-ver')
  printver $VERSION
  exit 0
 ;;
 '-version')
  printversion $0 $VERSION $VDATE
  exit 0
 ;;
 '-help')
  getmsg 0001
  exit 0
 ;;
 *)
  getmsg 0001
  exit 1
 ;;
esac
```

10.11 Error analysis

Error description:
After the installation, the "***Livelist***" cannot be started with ***pb_manager***.

The following points must be checked:

Is the device driver loaded in the LINUX kernel?

Condition: You are logged in as super user.

Start the "***lsmod***" command. All modules connected to the kernel are displayed.

A PROFIboard module must be present.

If one is not present, then go to the ***/boot/modules/PROFIboard*** directory and start ***insmod PROFIboard.o***.

Analyze the error message.

Is there any kernel output?

Condition: You are logged in as super user.

Enter the "***dmesg***" command.

Search the kernel driver output. After "***dmesg***", you can browse the output using the "***Shift/Page-Up***" and "***Shift/Page-Down***" keys.

The following message should appear after booting. The settings may vary according to your parameters:

```
SOFTING PROFIboard                                    V1.0.x
(c) 1999 by PCC,
   register major number 50
   request region from 0x240-0x243
   request irq 5
   using dpr_addr 0x000d8000
```

A corresponding error message is displayed if the PROFIboard card was not recognized. The device driver has probably not been loaded if the output from the first line is not present (see above).

Are the device files present?

Make sure that the entry files are located in the ***/dev/PROFIboard*** directory:

ls -la /dev/PROFIboard/PROFIboard*

If these files are not present or the displayed major number does not match the device drive message (see inset above: register major number ...) than an error has occurred with "***pb_install***". Restart "***pb_install***" and check the error messages.

Is the start script present?

Make sure that the start script is present and contains a link in the start directories.

ls -la /sbin/init.d/PROFIboard

ls -la /sbin/init.d/rc2.d/S51PROFI.sh

ls -la /sbin/init.d/rc3.d/S51PROFI.sh

If the start files are not present then an error has occurred with "***pb_install***". Restart "***pb_install***" and check the error messages.

Are the files for network configuration present?

Do the configuration files for the respective PROFIboard exist?

ls -la /usr/etc/profiboardx.cfg

ls -la /usr/etc/profiboardx.ov with x = board number, starting with 0

If the files are not present, then you have to create them and start "***pb_init***".

Is the PROFIboard on the network?

If the PROFIboard card is already on the network and the bus parameters are not accordingly adjusted, then it is possible that the card is deactivated to prevent a network interruption. Never perform a reinstallation with the card is connected directly to the network. Start the Livelist, which will run cyclically, and wait until the PROFIboard can be seen in the Livelist before plugging in the cable if necessary. Check the bus parameters again if the Livelist only works until the cable is plugged in.

Error description:

"pb_init" registers ***"pb_init failed: ret = 402, errno = 11"*** after starting

Cause:

It seems as though an interrupt is set, which is already occupied by another component. This component does not necessarily have to be used in order to block the PROFIboard. In the past, the onboard Ethernet connections and onboard sound cards were typical sources of errors. This error can also occur when the interrupt is enabled for PCI in a PnP BIOS.

Switch off the potential sources of error, reboot and check again. Change the interrupt settings if necessary.

Error description:
"pb_init" registers ***"devOpen failed, errno = 19"*** after starting

Cause:
The device drive is not loaded or the major number being used does not match that of the entry files in ***/dev/PROFIboard***. Furthermore, it is also possible that the device driver did not recognize the PROFIboard. Perform a test using "***dmesg"*** as described above.

Error description:
"ABORT_INDICATION" appears after attempting to establish a connection.

Cause:
The connection is either already occupied by another application or the settings in the KBL do not match those of the partner. In this case, the services might be set incorrectly or the bus parameters are not compatible with each other. For further reading, refer to the Softing documentation, which also describes the contents of the error telegrams.

10.12 Notes on literature (ProfiboardDriver)

- Softing documentation for the PROFIboard available as free download from the Internet at ***www.softing.com***. Detailed information is provided about the bus parameters and connection parameters as well as the values for the object descriptions.
- The B&R PROFIBUS manual for the 2000 System describes the basics of PROFIBUS such as cable structure, network settings and connection parameters as well as a special section about the PROFIBUS hardware in the 2000 System.
- PROFIBUS standard, DIN 19245 parts 1/2/3 and the corresponding EU standards

11 Process bus redundancy for Ethernet connections

11.1 General information about process bus redundancy for Ethernet connections

Process bus redundancy for Ethernet makes it possible to assign TCP/IP addresses twice for CPUs running on an Intel platform.

Process bus redundancy is implemented when using type *CP360, CP380, CP382* CPUs in combination with an IF781.9 interface module - parallel to the onboard interface. The configuration **and the controller's OS version V0271** is made with help of the Ethernet settings.

Note the description about 'Configuration of the Ethernet interface for controllers' in the APROL documentation, 'E1 Controller & I/O'*!*

To do this, call the menu option "*Ethernet settings*" from the shortcut menu in the *CPU view* of the *CaeManager*.

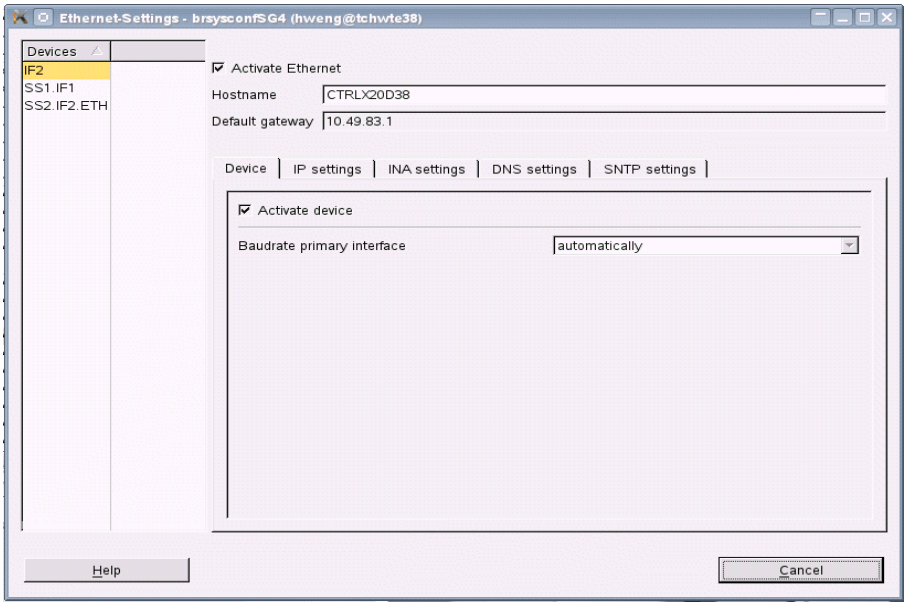


Illustration 39: Configuring the Ethernet settings

Make sure that both network cards are located in different sub-networks!

Process bus redundancy in relation to the ***InaDriver***, the ***EventDriver,*** and the ***controller cross communication*** is described in the following sections.

11.1.1 Configuring process bus redundancy with the InaDriver

Process bus redundancy is configured for the *InaDriver* using the following options:

Option	Description
-medium [EthernetRedundancy]	Activates the redundant Ethernet configuration
-ip_r IP_ADDRESS	Entry for the TCP/IP address **or the host name** of the connected controller **(redundancy connection)**.
-socket_r SOCKET_NR	This option is used to set the socket number which the driver uses when attempting to connect to the controller **(redundant connection)**. If this option is not specified, then the setting for the *-socket* option is used. For INA, the default socket is 11159 (0x2b97).
-node_r NODE	Setting for the redundant node address of the controller.
-bcast_r <*BCAST_ADDR*	Sets the redundant broadcast address.

Please note that the "socket" number and the "socket_r" number must be different (z. B. 11159 und 11160).

These numbers must be the same as those in the controller's Ethernet interface configuration (INA port number).

After starting, the *InaDriver* attempts to establish the initial connection. If the communication level reports "Lost connection" with a redundant configuration, then an attempt is made to reach the controller via the second connection. If this also fails, then the driver is restarted to enable process redundancy. This enables a second driver on another computer to address the controller. This is repeated until a driver is able to reach the controller.

Switching between the first and the second connection **occurs exactly one time** if it is not possible to establish a connection. If a connection is established, then the alternative route to the controller is used a maximum of one time the next time the connection is lost. The driver is restarted if this does not work.

Therefore, an InaDriver requires exactly one connection resource from the controller because two connections are never opened at the same time.

11.1.2 Configuring process bus redundancy with the EventDriver

Process bus redundancy is configured for the *EventDriver* using the following option:

Option	Description
-reduPlcIpAddr IP_ADDRESS	The -reduPlcIpAddress option is used to specify an alternative route to the controller.

First the *EventDriver* attempts to reach the partner station on the controller via the default connection (- *controllerIpAddr* option). If the IP stack from the control computer reports an error, then an attempt is made to establish a connection via the alternative route. If this fails, the first connection is tried again. This is repeated until the process redundancy timeout is reached (- *restartTime* SECONDS) triggering the control computer driver to start again. This enables a redundant driver that might be present to access the controller. The presence of the control computer driver is monitored using a communication timeout because an *EventDriver* server on the controller **runs exactly once and supports exactly one connection**. The socket on the controller is closed and a new connection is waited for if no telegram is received from the control

computer within the *-timeout* MILLISECONDS time range. However, events will still be recorded and buffered so that they can be sent after reconnecting.

11.1.3 Configuring process bus redundancy with controller cross-communication

Process bus redundancy affects the drivers *ApDrvCross* and *ApDrvIna*.
The redundancy is configured in the *CaeManager* under the communication tab (*ApDrvCross*) or under the APROL connections tab (*ApDrvIna*).
After creating a connection, the necessary settings for the alternative route and for the connection timeout <in seconds> can be made under "*Redundancy*" using the menu option "*New*" in the shortcut menu.

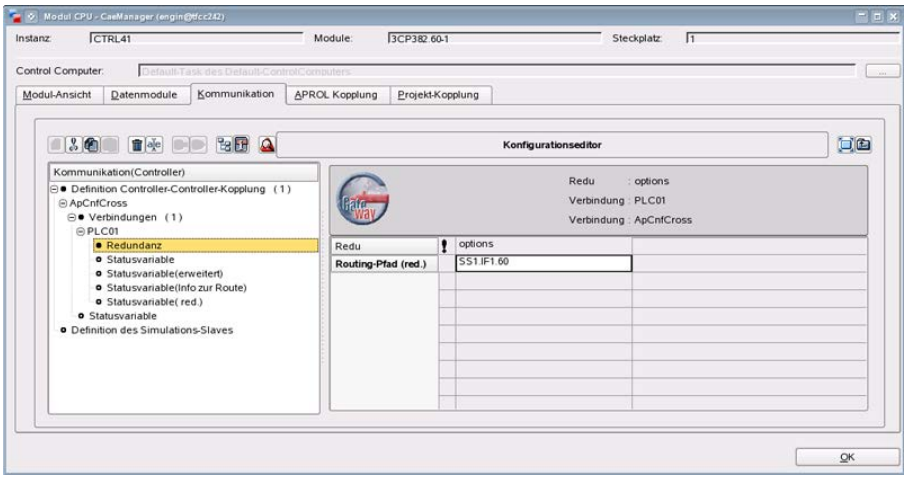


*Illustration **40**: Configuring process bus redundancy with controller-controller connection*

Both drivers attempt to reach the partner via the first connection. The alternative connection is activated if an error is reported or the connection cannot be made within the connection timeout time. The program switches between both connections repeatedly as long as the connection cannot be established.

In principle, an attempt is always made to establish a connection via the alternative route. That means that two INA connections are always active in the ideal situation.

As a result, it might be necessary to make adjustments according to the number of connections in the sysconf!

12 RK512 driver

12.1 General information about the RK512 driver

In this manual, the installation of the ***RK512Driver*** driver package, the configuration of the driver for the control computer, and the B&R controller are described.

Once the driver has been installed and configured, it can be used in the following ways:

Via 3964R	CC	Controller	Third-party controller
CC	X	-	X
Controller	-	-	-

Via RK512	CC	Controller	Third-party controller
CC	X	x	X
Controller	x	x	x

Knowledge of how communication protocols work is required to understand the information contained in this manual. This manual does not contain descriptions of the RK512 and 3964R communication protocols.

12.2 Information about the *RK512Driver* driver package

Drivers for the various process control system components are supplied in an *RPM package* (normally on a diskette). Once this *RPM package* has been installed on the corresponding computers, these drives are ready to be used. The drivers for the controllers are downloaded from the Engineering system with the *DownloadManager*.

The RPM package must be installed on a computer that has both the Engineering system and Runtime system installed.

12.3 Delivery contents of the RK512 driver package

- Data medium with the RPM package *APROL_RK512Driver-1.0.0-0.noarch.rpm*
- This documentation -- *RK512 / 3964 R drivers* -- in PDF format (also on the data medium)

After the installation, the following files will be located on the computer selected for the installation:

File with path specification	Description
/opt/aprol/bin/RK512Driver	The driver for the control computer (Runtime system).

File with path specification	Description
opt/aprol/br/aprol/V*/ i386/module/ApDrvRK512.br	The driver as module for the controller
/opt/aprol/cnf/RK512Driver/ R3964/rk512.cnf	3964 R example configuration.
/opt/aprol/cnf/RK512Driver/ RK512/rk512.cnf	RK512 example configuration.
/opt/aprol/doc/packages/ RK512Driver/ver.txt	Version information.
/opt/aprol/doc/packages/ RK512Driver/version.txt	Version information.
/opt/aprol/lib/libRK512.so*	Communication library.

12.4 Installing the *RK512Driver* RPM package

Installation should be carried out on a PC with a Runtime system and Engineering system installed since this is where the drivers and utilities will be needed.
*The installation can only be carried out by the **super-user** (**root**).*

The installation procedure:

Step	Description
1	Insert the diskette and mount the disk drive to the `/media/floppy` directory.
2	Then go to the `/media/floppy` directory.
3	Install the required package(s) with the rpm command. Example: *rpm –i <RPM-FILENAME> --nodeps --force*

With the command ***rpm –e <PACKAGE_NAME>***, you can uninstall a package if necessary.

Please note the difference between RPM-FILENAME and PACKAGE_NAME!

Examples:
Install:
```
rpm -i APROL_RK512Driver-1.0.0-0.noarch.rpm--nodeps --force
```

Uninstall:
```
rpm -e APROL_RK512Driver
```

After the installation, use ***umount*** to remove the disk drive from the file system and take the diskette out of the drive.

12.5 *RK512Driver* for the control computer

You can use the RK512Driver, which runs on a runtime system, to connect controllers or other field devices to the control computer and exchange data. The connected device must be able to understand the *RK512* or *3964R* communication protocol.

In order to use the driver, the RPM package must be installed!

The driver is configured in the CC modules, in the 'APROL system' project part, in the 'Driver' section.

A maximum of 64 RK512Driver instances can be configured.

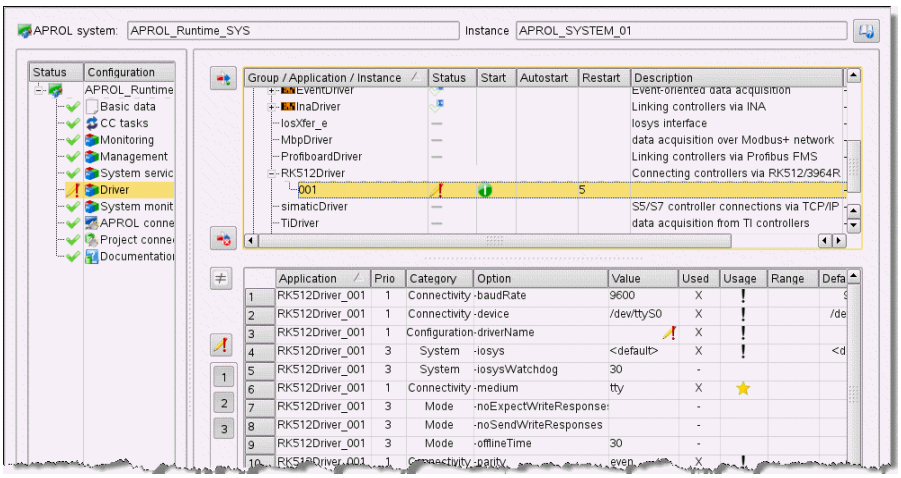


*Illustration **41**: Configuration of the RK512Driver*

After the driver has been configured, you will need to create PVs over APROL connections so that they are available during the engineering phase.

See chapter *Creating the rk512.cnf configuration file*

12.5.1 Launching options RK512Driver

Individual options are also explained in the online help.

Option	Value range	Description
-medium <tty | server>		This option selects between a physical interface [**tty**] and an Ethernet terminal server [**server**]. It determines whether a *serial interface* (**tty**) on the computer or a *terminal server* (**server**) should be used for communication. The IP connection goes from the computer to an external interface converter with a serial interface.
-addr <name | IP>		Use only for *-medium server*. Specifies the IP address or host name of the terminal server.

Option	Value range	Description
-iosys <server1:port [, server2:port]>	port=[0 - 15]	Specifies the IP address / name and *port number* of the computer with *Iosys*. For a redundant Runtime computer configuration, the redundant Iosys connection is configured as well. The Iosys port must always correspond to the Iosys port specified when the control computer master or slave is configured. **Examples:** -iosys TEST1:0 Iosys is running on the TEST1 computer at port 0. -iosys TEST1:0,10.49.80.93:1 Iosys is running on the TEST1 computer at port 0 and redundantly on a computer with the IP address 10.49.80.93 at port 1.
-self <id>		The *Self ID* is a unique identifier for the program instance and is assigned by the system, which increments this two-digit number for each instance. A maximum of 64 RK512Driver instances can be configured. If several instances of this application run on the same computer, these instances can be differentiated by the different Self IDs. The *Self ID* can also be overwritten by a named string to make it more descriptive.
-type <WuT | Lantronic>		**Only used with *-medium server*.** Sets the communication method to one of the supported terminal servers. **Options:** WuT or Lantronic (name of the manufacturer of the terminal server).
-port <value>	value=[0.0.3]	**Only used with *-medium server* and *-type Lantronic.*** This setting specifies the **IP port** on the terminal server. This selection depends on the selected type. **WuT** Port fixed at 8000 **Lantronic** Port = 3000 + <value> Terminal servers usually have several connections being used simultaneously, so this option can be used to specify which one is the partner. *WuT stands for Wiesemann and Theiss.*
-device </dev/ttyS**x**>	x=[0..3]	**Only used with *-medium tty.*** Device name and number of the serial interface. **Example:** /dev/ttyS0 (corresponds to COM1 in the Windows world). and /dev/ttyUSB0 (for USB/serial adapters)

Option	Value range	Description
-baudrate <value>	value = [1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200]	Only used with *-medium tty.* Specifies the transfer rate [baud] of the serial interface.
-parity <even | odd | none>		**Only with *-medium tty*.** This option sets the type of parity bits for the serial interface.
-priority		*Specifies the priority of the communication. If a collision occurs (driver and partner want to transmit at the same time), the partner with the lower priority yields and answers the one with the higher priority.* *Default = high*
-retry <value>		Number of retries before an error is reported. Default value: 3
-offlineTime <value>		Time until the driver automatically restarts (and sets up the option of a redundant partner) if constant connection errors occur. *Configuration errors like data blocks, etc. do not cause the driver to restart.* Unit: Seconds Default value: 30
-restart <value>	value=[0..20]	This option makes it possible to restart the driver automatically if the application was ended externally. It is **disabled by default**. If the specified value is exceeded, automatic restarts no longer take place. This mechanism is only switched back to active after manually resetting it with the StartManager or carrying out a new download.
-plc <PLC_NAME>		Specifies the subdirectory where the configuration file **rk512.cnf** is looked for. PLC_NAME thus results in: $HOME/RUNTIME/cnf/RK512Driver/controller_NAME/rk512.cnf, with $HOME as the home directory of the Runtime system.
-suspendTime <value>		Timeout for a task if it couldn't be executed successfully. If a task returns an error (e.g. data block not found), then it will be suspended for the length of **value** and only then re-launched. **Unit**: Seconds **Default value**: 30

Option	Value range	Description
-timeout <value>		Maximum time that the driver waits for a response (send permission) from the partner when sending a send request. If this time is exceeded, the driver returns an **error**. **Unit**: Milliseconds **Default value**: 1000
-time_ext <value>		Maximum time that the driver waits for a data telegram from the partner if a send request was sent successfully. **Unit:** Milliseconds **Default value:** 5000

Options for debugging output

Option	Value range	Description
-nofork		The driver doesn't **fork** (doesn't go on to a background process), but remains in the foreground (this option makes sense for debugging output).
-d_errors 1		The driver outputs all error messages.
-d_normal <LEVEL>		The driver outputs all status messages in levels <= LEVEL.
-d_protocol <LEVEL>		The driver outputs all communication messages in levels <= LEVEL.
-d_redu <LEVEL>		The driver outputs all status messages in levels <= LEVEL.
-help		The driver outputs the list of all start options and then quits immediately.

12.5.2 Creating the rk512.cnf configuration file

The driver reads the *rk512.cnf* configuration file in the Runtime system when the process control system runtime begins. It contains all of the information needed by the driver to exchange the engineered data between peripherals and the control computer.

If this configuration file has been created, the APROL system must be recompiled and generated. After this procedure, the PVs you've created for exchanging data during the engineering phase are available in the project.

The configuration of the *rk512.cnf* file takes place in the APROL system, in the APROL *connections* tab. Select *RK512 connection* via the shortcut menu to create a new entry. A structure is automatically created for this connection.

Please note that the APROL driver currently only supports the variable type 'Data block'.

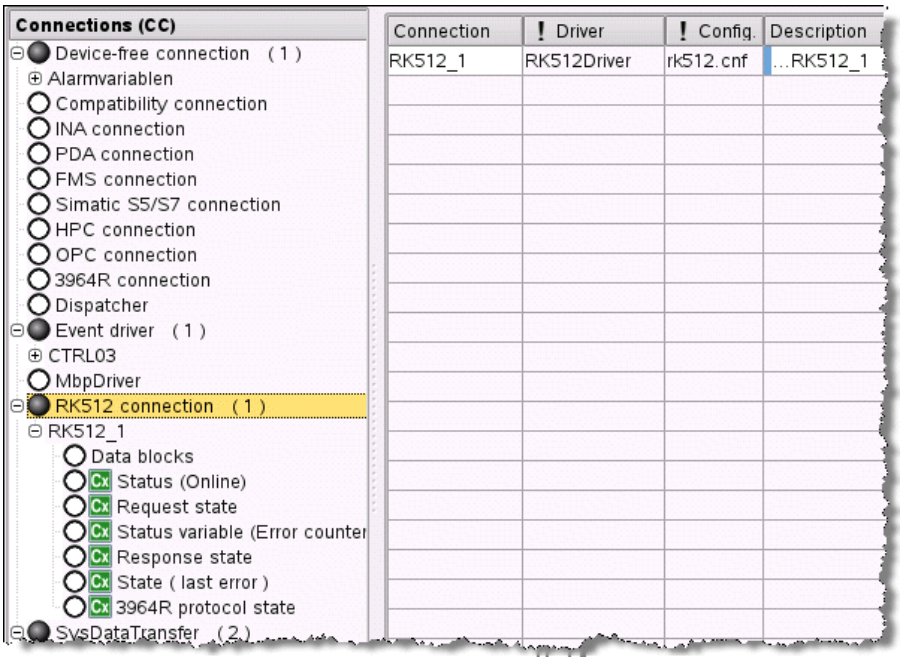


*Illustration **42**: Illustration Fehler! Es wurde keine Folge festgelegt.:*

The list of all data blocks used by the driver must be created under *Data blocks*. A new entry with the respective DB number must be created for each data block.

The following value range is valid for the DB number [1 ... 255].

The following fields must be filled for each data block (see image): *Len* with the number of data words, *Init* with the initialization method, *Block* with the transfer method, and *Access* with the type of access (*read, write, or read/write*).

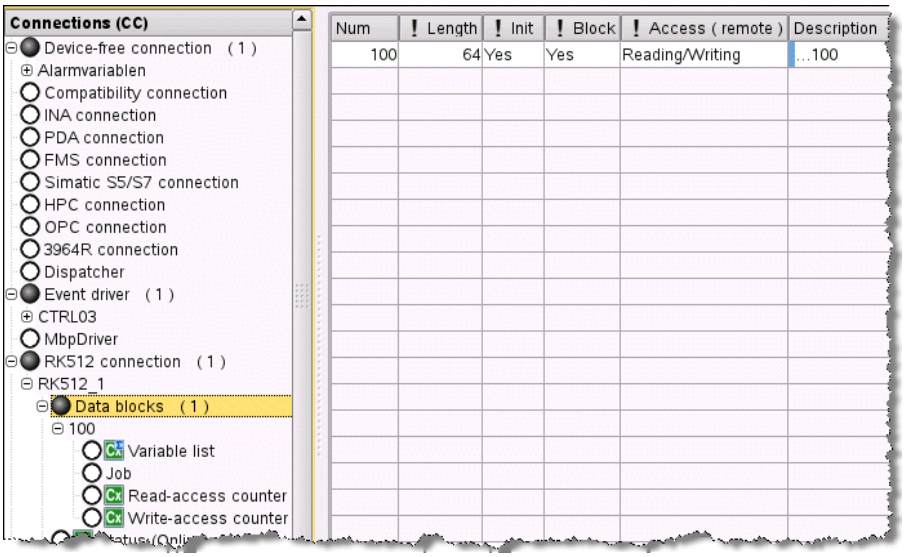


*Illustration **43**: Illustration Fehler! Es wurde keine Folge festgelegt.:*

If *Init* is set, the data block may only be written once it has been read successfully once. This is important for the first synchronization in order that the actual states of the data block are not destroyed. If *Block* is set, the complete block is transferred to the controller when a value is written. Otherwise, only the data words being used by the PV are written.

Next, the PVs to be used should be declared for each data block. When doing so, the name of the variable during the engineering phase (and therefore the one to be used on the system later during runtime) should be entered first.

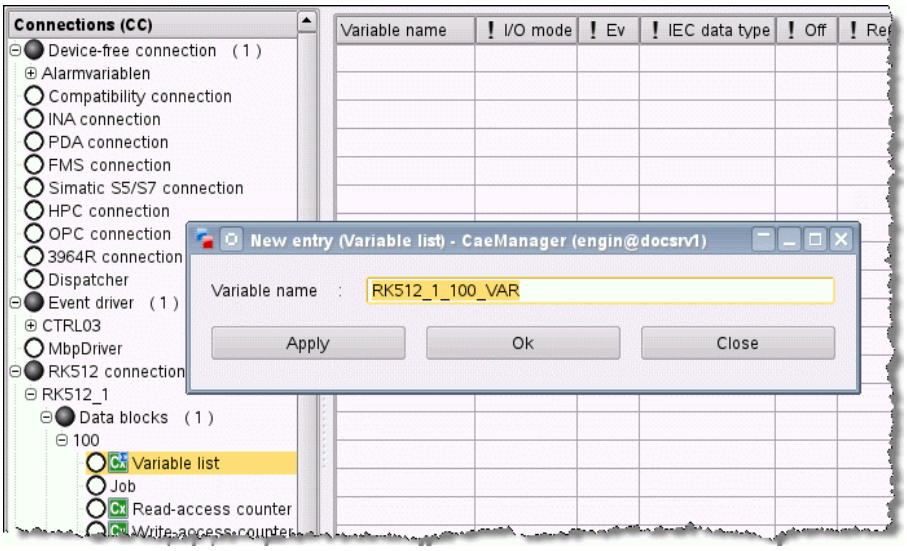


*Illustration **44**: Definition of the variables*

After all names have been issued, the mandatory fields *Mode*, *Ev*, *Type*, *Off* and *Remote type* must be filled in. The *plsMba*, *plsMbe*, *controllerMba*, and *controllerMbe* fields can be filled in if needed. *Mode* describes the data direction for this PV. It can be used as an input, an output, or bi-directionally.

Ev specifies whether this PV should be used as an event PV, i.e. written as soon as a change in value is detected. All PVs that have not been declared as event PVs are simply copied to the local data area only and sent with the complete block during the next write procedure. However, it's also possible that they will be overwritten with a read block before the next write takes place.

This explains why bidirectional PVs should generally be event PVs!

The *Type* column is used to determine the type in the system. This type must be connected in CFCs. *Off* describes which data word of the block this PV begins in. For multi-word PVs, the smallest data word being used should be specified. *Remote Type* is used to specify the type of variable being used by the communication partner. Both of the types declared here don't have to be the same; the driver converts them according to the configuration.

Variables that were of data type "REAL" within the framework of the R3964 connection are assigned data type "S5_REAL" within the framework of the automatic conversion of the RK512 connection.

plsMba and *plsMbe* determine the measurement limits for the process control system. If a PV is written to the controller with a value outside of the measurement limits, it will be reset to the respective limit value and transferred. If a value outside of these limits is read from the controller, it will not be limited in order for it to be used in alarming!

controllerMba and *controllerMbe* make it possible to scale PVs. *controllerMba* and *controllerMbe* make it possible to scale PVs. Details about this can be found in chapter *Scaling values*

The data block tasks are created next. Assign a symbolic name for this task (e.g. *READ*) and fill in the fields *Type* (read or write), *Cycle time* in milliseconds, *Offset* for the first data word to be transferred, *Length in DW* for the number of data words to be transferred, and the optional *Description* (not really necessary but very helpful for documentation purposes).

The value range for the offset is between [0 ... 255] data words. A maximum of 2048 data word can be transferred (maximum value for "Length").

As a telegram can contain a maximum of 128 bytes of data, longer data ranges are transferred with subsequent telegrams.

Finally, status variables for the driver and for each task can be created. Simply configure the name of the PV to be created under the respective function. At runtime, the driver creates and supplies a corresponding PV of type *Integer* or *String* for error texts.

Status variables per driver:

identifier	Description
Online variable	Variable for monitoring the online state. **Value 1**: Connected to the partner. **Value 0**: No connection. Note: A value of 1 doesn't mean that the tasks are executed without errors. It only shows that the communication (which might also contain errors) with the partner is working.
Status variable (error counter)	Error counter: This counter is incremented each time an error is found. **Note:** This PV is reset to 0 each time the driver is restarted.
Status variable (requests)	Variable that outputs status values for each request telegram. typedef enum { RK_IDLE = 5000, RK_READY_FETCH, RK_READY_FETCH_FOLLOW, RK_READY_SEND, RK_READY_SEND_FOLLOW, RK_READY_WRITE, RK_READY_WRITE_FOLLOW, RK_READY_RECEIVE, RK_READY_RECEIVE_FOLLOW, RK_READY, RK_REC_HEADER = 5090, RK_WRITTING = 5100, RK_WRITE_MAIN, RK_WRITE_MAIN_REACT, RK_WRITE_FOLLOW, RK_WRITE_FOLLOW_REACT, RK_RECEIVING = 5200, RK_RECEIVE_MAIN, RK_RECEIVE_MAIN_REACT, RK_RECEIVE_FOLLOW, RK_RECEIVE_FOLLOW_REACT, RK_FETCHING = 5300, RK_FETCH_MAIN, RK_FETCH_MAIN_REACT, RK_FETCH_FOLLOW, RK_FETCH_FOLLOW_REACT,

identifier	Description
	RK_SENDING = 5400, RK_SEND_MAIN, RK_SEND_MAIN_REACT, RK_SEND_FOLLOW, RK_SEND_FOLLOW_REACT, RK_SEND_ERROR = 5500, RK_TEMPORARY_STATES = 5900, RK_OKAY, RK_BUSY, RK_ERRORS = 6000, RK_ERROR_WRITE_LEN_ZERO, RK_ERROR_WRITE_MAIN, RK_ERROR_3964R, RK_ERROR_UNKNOWN = 6999 } STATE_RK512;
Status variable (responses)	Variable that outputs status values with each response telegram. Values used: See "Status variable (requests)".
Status variable (last error message)	Creates a PV of type String for holding plain text error messages. Note: This PV always indicates the last detected error. For this reason, you must always check the time stamp for when the error was detected. It is very possible that the error actually occurred days ago.
Status variable (3964R protocol)	Specifies the current state of the 3964R state machine. The following states are defined: typedef enum { R_IDLE = 1000, R_READY_SND, R_READY_RCV, R_READY, R_SENDING = 1100, R_SEND_BEGIN, R_SND_STX, R_REC_DLE_STX, R_SND_DATA, R_REC_DLE_DATA, R_SEND_END, R_RECEIVING = 1200, R_SND_DLE_STX, R_REC_DATA, R_RECEIVE_END, R_TRASH = 1300, R_TEMPORARY_STATES = 1900, R_OKAY, R_BUSY,

identifier	Description
	R_ERRORS = 2000, R_ERROR_TRASHED, R_ERROR_ABBORTED_STX, R_ERROR_ABBORTED_NAK, R_ERROR_ABBORTED, R_ERROR_LEN, R_ERROR_BCC, R_ERROR_RETRIES_REACHED, R_ERROR_WRONG_LEN_DATA, R_ERROR_WRONG_LEN_HEADER, R_ERROR_TIMEOUT, R_ERROR_TIMEOUT_STX, R_ERROR_TIMEOUT_DATA, R_ERROR_UNEXPECTED, R_ERROR_TEL, /* General errors */ R_ERROR_UNKNOWN = 2999 } STATE_3964R;

Status variables per task:

identifier	Description
Trigger variable	Variable that can be set externally to activate a cyclic task once non-cyclically. If the driver detects this setting, it resets the PV and enables the task as quickly as possible (after the last active task is finished).
Status variable	This variable assumes various status values with regard to the task (such as started, waiting for response, idle, etc.). The following values are used: 0 – Task not currently being processed 1 – Task being processed 1000 – Task completed with errors 1001 – Task temporarily cancelled 1002 – Task temporarily not being processed due to errors In the case of 1002, the task continues operating after the time in *SUSP_TIME* (see the driver's start options).
Pos. receive counter	Variable that gets incremented with each successful access. Note: The value of this variable is reset to 0 at each restart.
Neg. receive counter	Variable that gets incremented with each unsuccessful access. Note: The value of this variable is reset to 0 at each restart.

identifier	Description
Read access counter	This is a status variable **from the point of view of the data block**. This counter is incremented whenever the data block is read (e.g. reading the data block as a response to a read task from the partner, or reading to prepare write task data). Note: The value of this variable is reset to 0 at each restart.
Write access counter	This is a status variable **from the point of view of the data block**. This counter is incremented whenever the data block is written (e.g. writing the data block as a response to a read task from the driver, or receiving a send task from the partner). Note: The value of this variable is reset to 0 at each restart.

12.6 RK512 driver status variables

This description is under construction at present.

*Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.*

12.7 The *ApDrvRK512* driver for controllers

The *ApDrvRK512* driver is used if the B&R controller should exchange data with 3rd-party controllers or other field devices using the *RK512* or *3964R* protocol.

12.7.1 General information about ApDrvRK512

This driver and the additionally required data modules are automatically taken into account when downloading to the controller in the CPU's *Software configuration*. If the driver is finished being configured and the CPU has been regenerated, the PVs are ready for the engineering phase. The *ApDrvRK512* driver is configured in the CaeManager after selecting the CPU under the APROL *connection* tab. Configuring the *ApDrvRK512*

The configuration results in the *ApCnfRK512* module, which is loaded when it is downloaded to the controller.

A maximum of 8 interfaces are supported.

Create a new configuration for each interface. Correct the default values in the *Interface*, *Mode*, *Baud rate*, *Parity*, *Stop bits*, *Priority*, and *Timeout* fields (as well as the description if necessary).

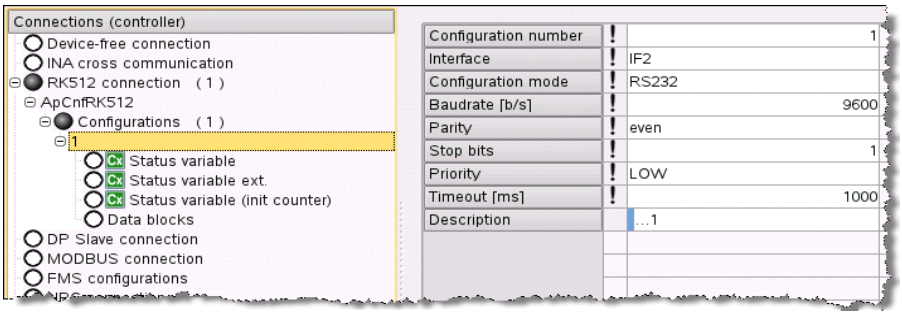


*Illustration **45**: Starting the configuration for the ApDrvRK512*

The *interface* must be specified in the controller notation and consist of s*lot*, *subslot*, and *interface number*.

CP360 example for a serial interface:
SS1.IF1

The serial interface mode must be specified under *Mode* (*RS232, RS422, RS485, TTY*). The respective hardware manual contains information about modes supported by the interface.

The *Baud rate*, *Parity*, and *Stop bits* fields are self-explanatory.

Priority determines who gets to continue (*HI*=high) and who needs to relent (*LOW*=low). Also see chapter Creation of the configuration file *rk512.cnf* for the driver in the runtime system.

Timeout determines the time it takes for the task to be cancelled with an error if a response isn't received from the partner.

Now create the data blocks to be simulated by the controller. Specify the data block number and the number of data words. The *Init* field doesn't currently have a function.

The list of PVs to be used is created next. Specify the name, the *kind* (input or output), the *type* on the controller (according to IEC standards), the offset of the smallest respective data word, and the partner type (also according to IEC standards). A description can also be added if desired. The PVs created here can be used after the configuration has been enabled in the charts.

After all PVs have been specified, the tasks for this data block must be configured. Cyclic (*tasks*) and non-cyclic (*event tasks*) read and write tasks can be configured. Specify the task type (*read* or *write*), the offset, and the number of data words per task. The cycle or interval time must be specified for cyclic tasks. Non-cyclic tasks are started only when needed using a trigger PV (event variable). If the trigger PV is set, then the driver sets it immediately back to 0 before starting the accompanying task.

Limit values of the task:

DB number	[1 ... 255]
Offset	[0 ... 255] Data words
Length	[1 ... 2048] Data words

Status variables can then be specified for the driver, each data block, and each task.

These status variables must already be present on the controller. The driver determines their memory addresses and writes their status values to them.

The following status variables need to be created for each driver:

identifier	Description
Status variable	Specifies the current state of the communication state machine. See also the process control system driver status variable (3964R protocol).
Status variable adv.	32-bit variable that displays the DB number, the offset, and the length of the current block when an error occurs: *DB number DB offset Length high Length low* **Bit 31** **Bit 0**

The following status variables need to be created for each data block:

identifier	Description
Status variable (read)	This is a status variable **from the point of view of the data block**. This counter is incremented whenever the data block is read (e.g. reading the data block as a response to a READ task from the partner, or reading to prepare write task data). Type: unsigned short, overflow at 65535 +1 = 0
Status variable (write)	This is a status variable **from the point of view of the data block**. This counter is incremented whenever the data block is written (e.g. writing the data block as a response to a read task from the driver, or receiving a SEND task from the partner). Type: unsigned short, overflow at 65535 +1 = 0

The following status variables need to be created for each task:

identifier	Description
Status variable (neg)	Counter that increases by 1 for each unsuccessful task. Type: unsigned short, overflow at 65535 +1 = 0
Status variable (pos)	Counter that increases by 1 for each successful task. Type: unsigned short, overflow at 65535 +1 = 0
Event variable (for non-cyclic tasks only)	Triggers a non-cyclic task. If a value change is detected by the driver, it resets the PV to 0 and starts the task.

12.8 Commissioning and Debugging

The RK512Driver process control driver can be started from the console for testing purposes. The following call should be adapted to check the configuration to see if it meets your needs:

```
/opt/aprol/bin/RK512Driver -medium tty -device /dev/ttyS0 -nofork
-d_normal 5 –d_protocol 5 –d_redu 5 -self 01 -priority high
-offlineTime 50000 –controller SPS1
```

Examples for adaptation:

- **-plc <controller name>**
  The configuration file *rk512.cnf* is expected in the $HOME/RUNTIME/cnf/RK512Driver/<controller name> directory.
- **-d_normal, -d_protocol, -d_redu**
  Unnecessary messages can be hidden by reducing the output level (here, all 5).
- **-device /dev/ttyS0**
  If an interface other than COM1 (used here) should be used, this parameter must be changed.

The driver outputs status messages on the console so that you can check whether communication is taking place successfully.

With the command:

IosEv –pv PV1 PV2 ... PVn

you can monitor the specified PVs. All read PVs (and SYNC PVs) must have valid values when the driver is running; all write PVs remain in their previous states. If write PVs are being set with *pio*, the values can be seen in advance in IosEv. If the write fails, the driver sets the PV to invalid; otherwise, nothing happens. The error text PV should also be monitored with IosEv in case error messages need to be read.

12.9 Scaling values

When scaling controller values to the Runtime system and vice versa, the value range of the one value is mapped linearly to the value range of the other. This is done according to the following formula:

Data direction: controller → runtime system

```
plsValue = plsMba + (plsMbe – plsMba) * (controllerValue – controllerMba) /
                    (controllerMbe – controllerMba)
```

plsValue = Value in the process control system

controllerValue = Value on the controller

Note:
When scaling in this direction, the system **doesn't** check whether the value range on the controller was adhered to. This allows process control system values that fall outside of the configured value range.

Data direction: runtime system → controller

```
controllerValue = controllerMba + (controllerMbe – controllerMba) * (plsValue – plsMba) /
                (plsMbe – plsMba)
```

controllerValue = Value on the controller

plsValue = Value in the process control system

Note:
In this direction, the value in the process control system is checked to see whether it falls within the valid value range. If not, the driver resets it to the respective limit value and uses this limit value in the formula!

for plcMba etc. also see chapter Creation of the configuration file *rk512.cnf*.

12.10 Scaling values

Scaling is used to convert Iosys variables to suitable values on the controller (or opposite side) or vice versa. Raw sensor values often fall in an unusable range to be displayed in APROL. They can only be displayed correctly after being adapted (scaled).

MRB, MRE	Measurement range start and end. If a measurement value falls outside of the limits, it is automatically reverted to the limit value.
NA, NE	Scaling start and scaling end for manual scaling. If NA and NE are not configured, the values from MRB and MRE are used!

The **scaling range NR** results from:
NR = NE - NA

NA and NE may not be the same value!

Controller value:
Controller_VALUE = NA + (NR) * (IOSYS_VALUE - MRB) / (MRE - MRB)

Iosys value:
IOSYS_VALUE = MRB + (MRE - MRB) * (Controller_WERT - NA) / (NR)

Example:
A variable should be supplied with a value between 4 - 20 mA, which corresponds to the REAL range of 0.004 to 0.020

Therefore:

MRB= 0.004
MRE = 0.020
NA = 4096
NE 16384

13 SimaticDriver

13.1 General information about the SimaticDriver

The driver package containing the Simatic driver for APROL is used to connect Siemens controller types S5 and S7 to APROL using the TCP/IP protocol. Cyclic read tasks and event-driven write tasks can be configured.

A combination of both task types build the so called SYNC tasks.

The **control computer driver** supports both a redundant communication bus and a redundant control system. Ethernet CPs that support communication via TCP connections (not ISO-on-TCP connections) must be installed as communication partners. The driver is configured in the **CaeManager** using the configuration editor for APROL coupling.

With the **controller driver**, data can be exchanged between B&R controllers and S5/S7 series Simatic controllers over the TCP/IP protocol. Corresponding communication processors must be present on the Simatic controller. **Communication over the Simatic HI protocol is not supported.**

13.1.1 SimaticDriver delivery contents

The driver package is contained on the APROL system software CD. It contains the actual communication driver, an example configuration, and this documentation.

The following files are installed:

- `/opt/aprol/bin/simaticDriver`
  The communication driver for coupling the Simatic controllers.
- `/opt/aprol/cnf/simaticDriver/examples/simaticDriver.cnf`
  An example configuration file.
- `/opt/aprol/cnf/simaticDriver/importFiles/drvPlssimaticDriver.imp`
  `/opt/aprol/cnf/simaticDriver/importFiles/drvPlssimaticDriver.imp`
  Import files for installation of the driver on older APROL releases.
- /`opt/aprol/doc/packages/SimaticDriver/simaticDriver.pdf`
  This documentation as online help for the *Acrobat Reader*

Controller driver (For all supported AR versions Vxxx)

- `/opt/aprol/br/aprol/Vxxx/i386/module/ApDrvS7.br`

13.2 Simatic driver for the control computer

13.2.1 Reference values of the Simatic driver for the control computer

- up to 512 data words per task

no support for follow-up telegrams

Because of the use of the RK512 protocol that demands block numbers with a single byte, blocks can be addressed to a maximum of 255!

13.2.2 Driver start options

The following table explains the driver's start options.

Option	Description
-bitsInBytes	The *-bitsInBytes* option detects the value of a binary variable from the bit position within a Byte (allowed types are BIN0 to BIN7). Without this option, the bit position within a Word is detected (allowed types are BIN0 to BIN15).
-cfg CONFIGFILE	Specifies the name of the configuration file, which must be located in the directory mentioned above. This parameter must be set!
-ipMasterAddr ADDR1	Sets the IP address for the master connection to the controller. This is a start parameter that must be transferred! e.g.: -ipMasterAddr 192.168.1.1
-ipSlaveAddr ADDR2	Sets the IP address for the slave connection if the driver is operated over a redundant bus connection. This start parameter is optional.
-controller DRIVERNAME	This parameter defines the environment where the driver should be started. The driver searches for its configuration file in the directory $HOME/RUNTIME/cnf/simaticDriver/DRIVERNAME. At the same time, the driver is registered in the system with this name for the Iosys and forms the name of its status variables (see below) via DRIVERNAME. This parameter must be set!
-setConnTimeoutTime T1	Specifies the maximum time the driver waits for a response telegram from the controller before it reports an error that it hasn't received a response to a request. T1 is a millisecond value, the default value is 1000. This parameter must be set!
-setRecvBufferSize SIZE1	Determines the maximum size for communication. The default value is 8 KBytes. The parameter should not be changed!
-setNumRetries NUM1	Determines the maximum number of read attempts the driver carries out to receive a complete telegram. If all of the telegram data cannot be received in this time, then an error message is given and the connection is reinitialized. The default value is 3, and normally does not need to be changed.

Option	Description
-writeWholeBlockFirst	After a write connection is established, all WRITE tasks are automatically sent to the controller as a whole if the option is set. Otherwise, only data in the control system that has changed is sent to the controller. Additional information shown below.
-setTcpKeepIdle IDLE	The driver monitors its connections using TCP routines. If no communication takes place in the IDLE time specified (in seconds), the system itself sends a query to the partner, asking whether it is still active. The time can be influenced in this way. The default setting of the driver for this value is 5 seconds.
setTcpKeepIntvl INTERVALL	The driver sends the monitoring telegram described above in cycles of INTERVAL. The default setting is 1 second.
-setTcpKeepCnt COUNT	The driver resends the monitoring telegram a maximum of COUNT times. If it still doesn't get a response, it reinitializes the connection and outputs an error message. If necessary, it activates a switch to redundancy. The default setting for COUNT is 5.
-reduMode REDU_COUNT	The driver is started in redundancy mode. It cyclically checks all configured connections. If it doesn't get the CONNECT status from any of its connections within REDO_CPUNT number of times, it goes into slave mode and leaves the master status to the slave driver. The cyclic check of the connection takes place every 200 milliseconds and begins two seconds after one of its own connections is lost at the latest. This results in redundancy switching after 2 + REDU_COUNT * 0.2 seconds at the latest. However, the partner driver can only establish a connection if the controller provides the resources for the connection. Additional information concerning this topic can be found in the redundancy section.
-s5Mode	The task line in the configuration file is specified in words, not in bytes!
-s5Offsets	The PV offsets in the configuration file are specified in words, not in bytes!

13.2.3 Description of the configuration file

The driver is configured using the start options together with the configuration file. The configuration consists of connection, task and process variable settings.

A connection includes the IP address of the partner station as well as the port number of the socket on the controller used for communication. A separate connection is required for each data direction (reading from the controller, writing to the controller). Only one connection is permitted to be selected for writing, multiple connections are permitted for reading and may, in some circumstances, improve performance.

A task consists of the task type (read, write or both), description, data to be transferred and, if applicable, the communication cycle that is to be used. Any number of tasks can be configured for a connection, as long as they are the same type.

The process variable declaration consists of the PV name, the offset in the telegram, the variable type on the controller as well as optional scaling information. Process variables are assigned directly to a task which automatically determines how they can be used in the diagram (input, output or bidirectional variable).

In the following section, the structure of the configuration file is described in detail using examples:

```
# This is a comment

^READ/2000/DB100/0/100/500                          # Read task, every 500 milliseconds
     READ_PV1_NAME_1      0    BIN0
     READ_PV1_NAME_2      0    BIN14
     READ_PV1_NAME_3      2    INT16
     READ_PV1_NAME_4      4    FLOAT
     READ_PV1_NAME_5      8    INT32
     READ_PV1_NAME_6A   12  INT16      0     1000     0    100
     READ_PV1_NAME_6B   12  INT16      0     1000     0    10000

^READ/2000/DB101/10/100/700                        # read task, every 700 milliseconds
     READ_PV2_NAME_1      0    BIN0
     READ_PV2_NAME_2      0    BIN14
     READ_PV2_NAME_3      2    INT16
     READ_PV2_NAME_4      4    FLOAT
     READ_PV2_NAME_5      8    INT32
     READ_PV2_NAME_6A   12  INT16      0     1000     0    100
     READ_PV2_NAME_6B   12  INT16      0     1000     0    10000

^SYNC/2002/DB200/0/50/200                        # Read task, every 200 milliseconds

# with the write possibilities
# using the WRITE connection
     SYNC_PV1_NAME_1      0    BIN0
     SYNC_PV1_NAME_2      0    BIN14
     SYNC_PV1_NAME_3      2    INT16
     SYNC_PV1_NAME_4      4    FLOAT
     SYNC_PV1_NAME_5      8    INT32
     SYNC_PV1_NAME_6A   12  INT16      0     1000     0    100
     SYNC_PV1_NAME_6B   12  INT16      0     1000     0    10000

^WRITE/2001/DB10/0/100/0                         # Write channel for pure WRITE tasks

# and for writing SYNC variables
     WRITE_PV1_NAME_1      0    BIN0
     WRITE_PV1_NAME_2      0    BIN14
     WRITE_PV1_NAME_3      2    INT16
     WRITE_PV1_NAME_4      4    FLOAT
     WRITE_PV1_NAME_5      8    INT32
     WRITE_PV1_NAME_6A   12  INT16      0     1000     0    100
     WRITE_PV1_NAME_6B   12  INT16      0     1000     0    10000
```

The previous example contains a configuration for a driver that uses read, write and bidirectional communication.

The "#" character indicates the start of a comment; all characters in the line including this character are ignored. Empty lines are also ignored.

Task declarations are introduced with the "^" character. Task line fields are separated from each other with the "/" character.

Lines that are not interpreted as comment lines or task lines are PV declarations. Here, the individual fields are separated by spaces or tabs. PV lines always refer to the last configured task, which is why the lines are never allowed to come before the task lines.

A task line contains the following information:

Task type	Determines the communication direction as well as the behavior of the PVs declared for this purpose. Three key words are defined for the task types: READ cyclic read task WRITE event-driven write task SYNC cyclic read with event-driven write
Port number for the connection	Represents the connection, together with the IP address of the target node, and must be configured on the controller accordingly.
Type and block number	Specifies the block type and block number on the controller. If a type code is not listed, then the module type is Data block (DB). Type and number must be entered one after the other without spaces. The following codes can be used: DB, MB, EB, AB, PB, ZB, TB, BS, AS, DX, DE, QB
Number of the first data word used	Specifies the offset on the module where the data can be found. Important: The offset is listed in words, 1 word consists of two bytes.
Number of data words used	This specifies how many data words are to be read or written. Attention: Unused areas in a block are overwritten with zeros one time during a block write (option –writeWholeBlocks is set).

Cycle for cyclic communication	Cycle for cyclic read tasks in milliseconds. If a cycle time that is shorter than possible is set, communication takes place as quickly as possible. Write tasks are not carried out cyclically, so this value is ignored.

13.2.4 Mode of operation of the different task types

READ:
Read tasks are called cyclically. The control computer driver sends a read request with address and number of items to the controller. From there, a response telegram is sent back that supplies the desired data.

If the response data is not received within a response timeout, then a timeout error is generated that closes and re-initializes the connection. Afterwards, the read cycle starts from the beginning.

Because the response data do not contain any relation to the request telegram, it is not possible to send several READ requests simultaneously over the same connection.

I.e. All read tasks for a connection are carried out sequentially so that the minimum cycle time results from the sum of the read tasks required. That means if 4 read tasks are to be executed cyclically every 500 milliseconds and each task has a runtime of 300 milliseconds, the real cycle time is 1200 milliseconds. For this reason, READ tasks can be distributed using two or more connections at the same time. In this case, the running time for the tasks is limited by the controller's backplane communication, so more than two connections should not result in an improvement.

WRITE:
Write tasks are sent "on event". If an Iosys value change is recognized then the driver creates a write request telegram and sends it to the controller. A receipt telegram that contains the information regarding error or "no error" is sent from there. Because the response telegram also contains no relation to the request telegram, it is only possible to send one write request.

The next telegram can only be sent when this has been acknowledged. Incoming events are hung onto a write queue in order to be written, and are sent according to the FIFO principle (**F**irst **I**n **F**irst **O**ut).

If more events are created than can be sent in the long run then the queue grows longer. This is why the driver's memory usage grows, and the speed with which changes arrive at the target system gets smaller. Amongst others, the number of change events can be reduced by this as the control computer task's cycle time has been raised. This also naturally has the effect on the update time of the entire system.

The smallest possible unit according to the protocol is transferred whilst writing, 1 to 2 data words with S5, and at least 1 Byte with S7.

Bits that are contained within a data word are therefore always transferred as a whole word, non-defined Bits within the word are always set to 'Null'. If the non-defined bits in the target system are being used in another way then they will eventually be overwritten! A clear separation of read and write variables in separate areas is guaranteed in this way!

SYNC:
SYNC, or also "bidirectional" tasks, are a combination of READ and WRITE tasks, whereby different connections are used for this. A connection is configured for the read task, another connection for the write task. Several SYNC tasks are allowed to use the same write connection,

whereby the delays that have been described for the write tasks also manifest here for each write event.

If SYNC tasks are used and no pure WRITE variables are declared, then the WRITE connection still must be configured without variables so that the write connection is recognized by the driver.

```
      ^SYNC/2001/DB100/0/100/1000         Task for reading DB100
                                          from offset 0 the length 100
                                          Data words every 1000
                                          Milliseconds
            PV1 ...
            PV2 ...
            PVx ...
      ^WRITE/2002/
```

The write connection to port umber '2002' is defined here. All changes regarding a SYNC task's PV are sent over this channel.

13.2.5 Additional notes about the mode of operation

When using an S7 controller, data is transferred in partial telegrams with ca. 220 bytes of reference data each. For example, 100 real variables of 4 bytes each need two individual telegrams.

*Depending on the size of the data area to be read, the '-setNumRetries' and '-setConnTimeout' parameters must be adjusted so that the telegram is received completely **without a timeout**. The time, which is necessary for the complete reception of the data, depends on the load of the S7 and the amount of the individual telegrams.*

***As a partially received packet blocks the driver** until it is completely received, **it is recommended to dissect large data areas into partial tasks**. There are more request telegrams because of this, but the reaction time of the driver is shortened due to the non-blocking and the possibility to process several connections in parallel.*

When using an S7 controller, response telegrams are always dissected into partial telegrams with a size of about 220 bytes.

For example, 4000 bytes are necessary for reading 1000 REAL variables (Float = 4 byte).

*Accordingly, the'-setNumRetries' and '-setConnTimeout' parameters must be raised. Apart from that, at least 18 (partial telegrams)*50 ms. = 900 ms. communication time is needed. Almost the entire communication time is used in this example with a reading cycle of 1 second!*

*This means that telegrams with a larger size than about 220 bytes are dissected into partial telegrams **and thus the communication time is substantially raised**.*

The waiting time between a request and acknowledgement telegram is set via '**-setConnTimeout**' [in milliseconds]. 1000 ms is used per default. If no response telegram arrives within this time then this leads to an error with disconnection and subsequent re-establishment.

The timeout time that should be set here depends on the number of words that are to be transferred, on the controller's load, and on the network. A higher data volume, and thus more communication between CPU and CP leads to an increased response time.

Details can be found in the documentation of the respective CPs (Communication processors).

With large data volumes, the data are transferred in part-telegrams.

In this case the data do not arrive simultaneously at the APROL driver, but after one another as data 'fragments'. The driver recognizes the arrival of the first fragment and waits for the completion of the data. A maximum of the retries that are configured via the "**-setNumRetries**" option are used for this purpose. A retry is about 50 milliseconds. **If all of the data could not be transferred within the time then the parameter must eventually be increased.**

You can compare the driver's output in debug mode for this purpose:

"sleeping for data" = Driver is waiting for a further data fragment

"read successfully done" = all data are available

"timeout waiting for data" = Data have not been received within the waiting time

The messages are output when the debug filter is set to 0x3. (See chapter *Notes for starting up the driver*).

13.2.6 PV declaration

A PV declaration consists of the following fields:

Name of the PV in Iosys	The name of the PV in the Iosys is defined here. These names must be used to connect the variables in the project plan.
Offset of the PV in the telegram	The position of the PV in the telegram in **BYTES** is defined here, not on the controller. The first PV of a block is always at offset 0, even if the first data word has offset 10!
PV type on the controller	The type, size and measuring range of the PV as well as the PV type in the Iosys are defined here. The following types are possible: BIN0 to BIN15 — Bit 0 to bit 15 in data Word or BIN0 to BIN7 — Bit 0 to bit 7 data Byte (with the option *-bitsInBytes*) **Both cannot be used with** the ***-s5Offsets*** option UINT8, INT8 — Byte in data word, signed or unsigned. Range: 0 – 255 or -128 – 127 UINT8L, INT8L / UINT8R, INT8R — left or right byte (only with option -s5 offsets can be used!

	UINT16, INT16	Data byte, signed or unsigned. Range: 0 – 65535 -32768 - 32767
	INT32	Signed double word. Range: -2147483648 up 2147483647
	FLOAT	Variable of type IEEE Floating Point
	S5FLOAT	Variable in format S5 Floating Point
	STRINGxx	String with length xx
	The BIN and INT types are stored in the Iosys as integer variables, the FLOAT types as double variables and the STRING types as string variables. If INT types are scaled, they are also stored in the Iosys as double variables!	
	The string variables are transferred in C notation from the APROL point of view, i.e. they must be closed with a null byte. Therefore, there is 1 character available for the string and 1 character for the end marking in a STRINGxx with xx being between 1 and 255 (Transfer length in bytes) xx. At any rate, the target area on the controller must be able to handle xx characters. Unused string characters are written with null bytes, i.e. a STRING20 with '12345' is comprised of '12345' and 15 null bytes.	
PLS_MBA, PLS_MBE, SPS_MBA, SPS_MBE	Measurement range in the control system and on the controller (MBA= start of measurement range, MBE= end of measurement range). These entries are optional and must be available as pairs if they are to be used. When writing variables, they are only sent within the measurement range. When reading, also values outside the measurement range are accepted. When scaling, the range on the control system is represented in the controller range, which allows stretching or compressing. If the measurement range is not specified, the measurement range for the variable types (see above) are used, stretching or compressing is not carried out.	

13.2.7 Scaling formulas

Variables are scaled using the following formulas:

Abbreviations:

PLS_MBA, PLS_MBE:	Start and end of measurement range in the process control system (PCS)
controller_MBA, controller_MBE:	Start and end of measurement range on the controller

IOS_VALUE:	Variable value in the Iosys
controller_VALUE:	Value of the variables on the controller
PLS_RANGE:	PLS_MBE – PLS_MBA
controller_RANGE:	controller_MBE – controller_MBA

1. Iosys variables for READ/SYNC tasks

```
IOS_VALUE = PLS_MBA + (controller_VALUE – controller_MBA) / controller_RANGE * PLS_RANGE
```

2. controller variables for WRITE/SYNC tasks

```
controller_VALUE = controller_MBA + (IOS_VALUE – PLS_MBA) / PLS_RANGE * controller_RANGE
```

13.2.8 SimaticDriver's status variables

After starting, the driver automatically creates some status variables in the Iosys. The naming for these PVs results from the parameter *–controller*. The following section contains a list of PVs and a description of their meanings. The parts of the name written in italics are variable types, which are then described, and the parts written normally are fixed texts.

<table>
<tr><th>PV name</th><th>Description</th></tr>
<tr><td>PlcName_Pid_debugFilter</td><td>controllerName: Name of the driver, corresponds to the option –controller
Pid: Process ID for the driver, determined using ps ax

With these variables, driver debugging outputs can be activated. Depending on the bits within the set value, messages can be shown or hidden.
<table>
<tr><td>The following list shows the bits and the message types activated by them:
0x00000001</td><td>Activates error outputs</td></tr>
<tr><td>0x00000002</td><td>Activates process messages</td></tr>
<tr><td>0x00000004</td><td>Messages for connection monitoring</td></tr>
<tr><td>0x00000010</td><td>Debugging the configuration according to cnf file,
one time, deactivates itself</td></tr>
<tr><td>0x00000020</td><td>Output of socket handle and socket settings</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>PV name</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>after the connection has been established, not if connection is already active!</td></tr>
<tr><td>0x00000100</td><td>Hex dump for received telegram header</td></tr>
<tr><td>0x00000200</td><td>Hex dump for received telegram data</td></tr>
<tr><td>0x00000400</td><td>Hex dump for sent telegram header</td></tr>
<tr><td>0x00000800</td><td>Hex dump for sent telegram data</td></tr>
<tr><td>0x10000000</td><td>Status output for tasks lists, number of read or write tasks waiting to be carried out</td></tr>
</table>
Important: After setting the desired bits, the outputs are placed in the corresponding log file. The outputs must be reset to zero after the analysis so that the hard drive is not filled up!!!!
</td></tr>
<tr><td>controllerName_lastErrorText</td><td>ControllerName: Name of the driver, corresponds to the option –controller:
In this string variable, the driver writes the error texts which also contain the time stamp for the message.</td></tr>
<tr><td>controllerName_IpAddr_PortNo_connState</td><td>controllerName: Name of the driver, corresponds to the option –controller
IpAddr: IP address for the connection to the controller
PortNo: Port number on the controller

This integer variable shows the current state of this connection.

<table>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>0</td><td>This connection is not configured</td></tr>
<tr><td>1</td><td>The connection is now inactive</td></tr>
<tr><td>2</td><td>The connection is being established</td></tr>
<tr><td>4</td><td>The connection has been established</td></tr>
</table>
The selection of the states has been made so that bit-oriented processing is possible in the plan.</td></tr>
</table>

This description is under construction at present.

*Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.*

13.2.9 Workflow description for the control computer driver

After starting the driver, it reads its configuration file and registers its process variables in the Iosys. Then it remains inactive in slave mode until it is informed by the redundancy mechanism that it should become active. Now the driver registers its status variables in the Iosys, establishes the connections to the controller and starts its cyclic tasks, i.e. READ and SYNC tasks. WRITE tasks or SYNC tasks that write information are only executed after change events.

The following notes must be taken into consideration when carrying out a driver analysis:

- SYNC variables can only be written if they have been read at least once. After losing the connection, it is necessary to read them again.
- READ and SYNC variables are set to invalid in the Iosys when reading fails (e.g. connection is lost, or read block on the controller is not available).
- WRITE variables are set to invalid in the IOSYS when writing to the controller fails. They may remain invalid until a required write tasks is carried out.
- Write tasks that have failed are not automatically repeated by the driver, this is normally done using control system logic.
- SYNC variables that are written are internally blocked from being read for the duration of the write task. If a successful read task takes place during an active write task, then the value read is ignored. If faulty logic allows many changes to take place in a short period of time, then it may be impossible to read this PV.
- A driver in slave mode may register its PVs in the Iosys, but only receives events when it becomes master. Events set to the slave are thrown out.
- If a driver is started with the option ***-writeWholeBlockFirst***, it writes all current control system values to the controller after a connection is established. Without this option, only events sent after the connection was made are taken into consideration.

13.2.10 Driver redundancy

The driver makes it possible to implement bus redundancy and computer redundancy.

By configuring two different IP addresses for the controller, it is possible for the driver to reach the controller using two network cards (and therefore using two network lines). In this case, the controller must also use two communication processors so that the different network masks can be linked. When set up accordingly, controller redundancy can also be achieved here, as long as it is possible to remove the passive controller from the network (or to prevent the CP from accepting a connection). The driver checks the connections in the order of the configuration and always attempts to communicate using the first connection and only attempts to communicate using the second connection if an error has occurred.

The second type of redundancy is process redundancy, which can also be used to support bus and controller redundancy. The disadvantage of this type of operation is a relative long switching time because several monitoring cycles are needed before the active driver gives up its status as master and the passive driver can take over.

This type of operation is configured using the '–reduMode' option together with the entry for the number of checks that should be made before the driver becomes passive. An active driver cyclically monitors all configured connections. If communication is not possible on ALL connections, it starts a countdown. If it is not possible for the driver to reestablish a connection using one of these connections before the countdown is finished, then it gives up the status as master, waits for a second to give the partner the chance to become master and then operates as passive driver once again. Note for calculating the switching time for process redundancy:

The driver checks its connections cyclically every 200 ms until all connections are made. Once all connections are made, the monitoring cycle is changed by a factor of 10 to 2000 ms. If it then detects the loss of at least one connection, the cycle is reduced again to 200 ms. The countdown begins as soon as all connections have been lost (detected after 2000 ms in the worst case) and lasts for a period of COUNT times 200 ms (Whereby COUNT must be set with the '–reduMode' option).

If the COUNT is set to 1, the switching time could still be up to 2 seconds, while bus redundancy using a driver is switched immediately as long as the parallel connection is established.

13.2.11 Configuration of the driver in CaeManager

The start options for the *SimaticDriver* are set in the **CaeManager** using the CC modules in the 'APROL system' project part.

If more than one driver should be started then another driver instance must be created for each additional driver using the "**Create new instance**" shortcut menu.

This menu item is only available when the first driver instance is selected!

Then each driver instance must be configured individually. Make sure that the start option is set (***Start*** column). The start option is set via double-click.

Then the individual options are configured. There are default options that do not have to be configured, and options that do not have default values and therefore must be configured.

It is absolutely necessary to configure the options for the master controller IP address, driver name, which corresponds to the directory with the configuration file, and the Iosys option if the automatically assigned default value should be changed. After successful configuration and download to the runtime computer, the corresponding *SimaticDrivers* are listed in the **StartManager**.

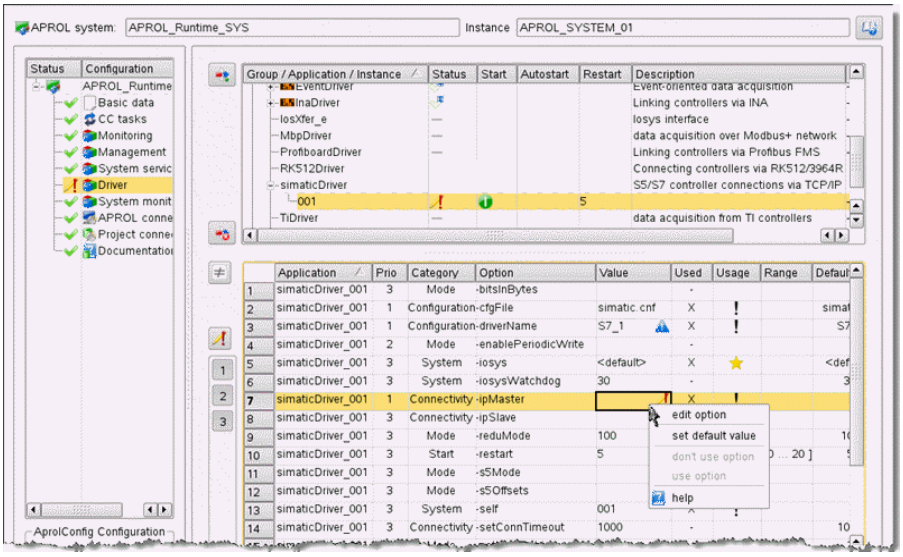


*Illustration **46**: Configuration of the SimaticDrivers*

13.2.12 Creating a configuration file with the configuration editor

In order for the *simaticDriver* instances to be able to communicate, a configuration file must be created for each driver in the **CaeManager**, in the 'APROL system' project part (APROL connections tab).

A valid configuration consists of **at least one task and the corresponding process variables**. The data direction (read or write) must be specified as well as the access type used by the driver for the variables. A separate task must be created for each data block that should be transferred. If the data to be transferred does not fit in one telegram, several tasks can also be created for a data block using offsets.

The procedure will be clarified using an example:
A data block DB100 with data values 0 to 99 should be read once per second from an S7 controller. As an example, 5 PVs will be created which are stored at various positions within this data block.

Please note that this is an example used to describe the procedure. It doesn't make sense to transfer all data words when no PVs exist.

In addition to this read task, a write task will be generated that also consists of 5 PVs stored in various words in DB101.

Now create a new configuration. For this purpose, choose the "APROL connections" entry in the configuration part of the "APROL system" project part, in the **CaeManager**.

In "**3rd-party connections CC**" then choose the "**Simatic S5/S7 connection**" entry and create a new connection with the "**New**" menu item in the shortcut menu.

Enter the name here (**must correspond to the parameter -controller in the start options** and therefore the directory name used when saving the configuration file).

In the Explorer view for the coupling list, a corresponding entry will be created with this name. Under this entry, a field called ***Task*** is also automatically created.

Then go to the "**New**" menu item in the shortcut menu and create the tasks numbers 1 and 2.

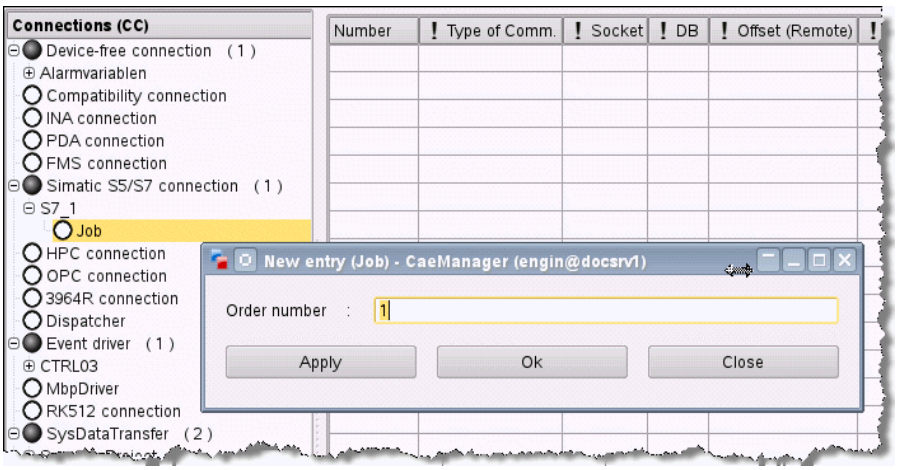


*Illustration **47**: Create the task:*

After selecting the task number, you can select the task type (*READ, SYNC, WRITE*), the port number of the socket for this connection (this also must be configured in the Simatics software accordingly!), the number of the data block, the offset, the number of data words to be transferred and the cycle time in milliseconds.

Enter the following for task number **1**:
READ, <Port number Read>, 100, 0, 100, 1000.

Enter the following for task number **2**:
WRITE, <Port number Write>, 101, 0, 100, 0.

The write task does not require a cycle time (we only send data when it changes in the control system, i.e. "on event"). Optionally, a short description can be added for each task.

The PVs can be created for each task using the ***New*** menu item in the shortcut menu and the ***Variables*** entry.

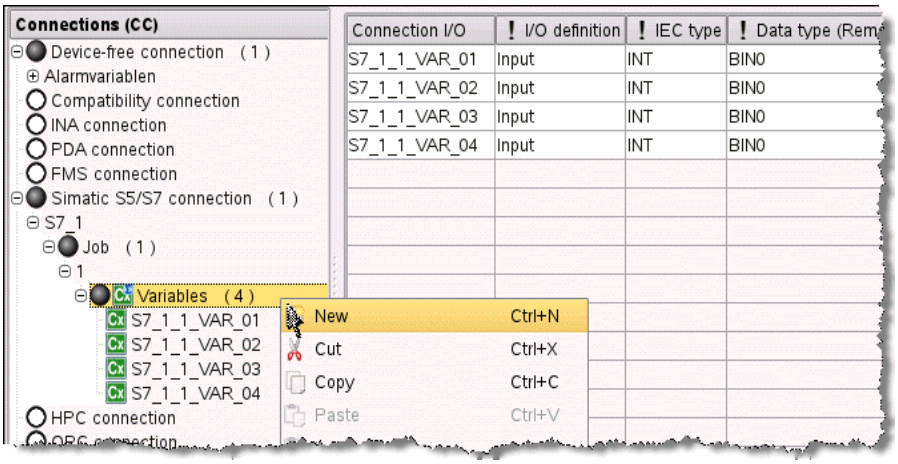


*Illustration **48**: Creating the variables*

If all variable names have been entered, each PV can be defined by going to the ***Variables*** field in the list view of the configuration: Depending on the task type, **Input** must be selected in the ***I/O type*** column for READ tasks, **Output** is to be set as ***I/O type*** for WRITE tasks and ***bidirectional*** for bidirectional tasks.

The ***I/O type*** field specifies the variable type in the logic diagram. This is necessary for correct wiring. The ***remote type*** defines the variable type saved on the controller in this location. You can select bits within a word, whole words, double words or float variables.

The ***I/O type*** has no meaning for the driver, but it is **necessary** to make sure that the ***I/O types*** match the ***remote types***. The float remote type creates a float variable in the Iosys, therefore the I/O type must also match the float group. Integer remote types converted using scaling information are also float types. Integer variables that are not scaled and bits are integer types and can be represented using all I/O integer types. The ***Offset*** field specifies where the variables are located with reference to the star of the telegram.

Please note that offsets are stipulated in bytes.

Scaling information can also be entered if necessary.

After all the required entries have been made, the resulting configuration file is shown after selecting the coupling name below the ***Configuration preview*** tab.

A plausibility check is carried out and you will be informed of any configuration errors.

After saving the control computer configuration and then activating, compiling and generating it, the configuration can be loaded onto the runtime computer and the driver can be started for the first time.

Using the tool ***IosEv***, you can check if all PVs are read correctly, and using the tool **pio**, Write PVs can be written to the controller.

13.2.13 Notes for starting up the driver

When starting up the driver, we recommend starting it manually at first using a suitable configuration. Set Debug mode using the option ***–d 0x27*** and start from the directory containing the configuration file with the option ***–ignoreCnfPath***. Now you can analyze the Debug messages and relatively quickly determine if the connection has been established and if the telegrams are being answered. If everything is OK, communication can be started using the *AprolLoader*.

13.3 Simatic driver for the controller

13.3.1 Reference values of the Simatic driver for the controller

The following briefly summarizes the features of the driver.

- 16 connections are allowed for the driver.
- 64 tasks are allowed to be configured on the 16 connections.
- Up to 500 data Words can be transferred per task.
- The maximum cyclic time is 1000 milliseconds for **active tasks**.

Because of the use of the RK512 protocol that demands block numbers with a single byte, blocks can be addressed to a maximum of 255!

13.3.2 General information about the configuration data module

The driver looks for a data module called *ApCnfS7* on the controller. This data module configures the tasks, external status variables, and shovel tasks for distributing or preparing data.

It differentiates between task lines, their suitable status lines, and the accompanying variable lists. This information must be configured in succession. Status lines are optional; theoretically, variable lists can be any length.

13.3.2.1 Structure of the configuration data module

A task line between with the "^" character. A comment line is set off by a "#" character.

The following task types exist:

READ	Active cyclic reading of data blocks of the Simatic controller
READP	**Passive** reading of data blocks of the Simatic controller. The controller sends telegrams, which contain a header, which then again contains information about the source data. Telegrams are acknowledged by ourselves, not only positive, but also negative when the data range is not configured.
WRITE	Active cyclic writing of data blocks of the Simatic controller This acknowledges the receipt of the telegram.
RECV	Receipt of raw data (without telegram header) from the Simatic controller. A receipt is not sent.
SEND	Cyclic transmission of raw data (without telegram header) to the Simatic controller. Receipts are not sent.

Thereby, the following tasks can be configured:

```
^READ/IP_ADDRESS/PORT/DBNO/OFFS/NUM_DWS/POLL_CYCLE
^READP/IP_ADDRESS/PORT/DBNO/OFFS/NUM_DWS/RECV_TIMEOUT
^WRITE/IP_ADDRESS/PORT/DBNO/OFFS/NUM_DWS/WRITE_CYCLE
^RECV/IP_ADDRESS/PORT/0/0/NUM_DWS/RECV_TIMEOUT
```

```
^SEND/IP_ADDRESS/PORT/0/0/NUM_DWS/WRITE_CYCLE
```

Note about the syntax of a task line:

IP_ADDRESS	IP address of the Simatic controller.
PORT	Socket number of the Simatic controller
DBNO	Number of the data block on the Simatic controller (Only data blocks are allowed, no other type of block)
OFFS	Number of the first data word to be processed on the Simatic controller
NUM_DWS	Number of data words to be transmitted, each is composed of 2 Bytes
POLL_CYCLE	Query cycle in milliseconds, relative to the last receipt telegram, or time out with missing receipt.
RECV_TIMEOUT	Maximum time in ms between two telegrams from the Simatic, before the connection is "reconnected" - 0 means no reconnect
WRITE_CYCLE	Send cycle in milliseconds, respective to the last receipt telegram. In the case of a time out (no receipt), or loss of connection, Retry is set permanently to 1 second.

The following rules are to be taken into account with respect to the telegram structure:

- **A *SEND*** and **a *RECV* task can be configured over the same connection**, but only exactly one per type. If more data (or less with receive retries) is received with *RECV* than is configured, then these are discarded and the connection is "reconnected". The mutual SEND task on the same connection is also affected by this.
- Several *READP*s can be configured over the same connection.
- Several *READ*s are allowed to be configured over the same connection.
- Several *WRITE*s are allowed to be configured over the same connection.
- A connection can always only be simultaneously used for exactly one task, before the next task is processed. An active task (*READ, WRITE*) is composed of a request telegram, and a response from the partner. A response is waited for with a time out of 2 seconds. A time out triggers a reconnect to the partner. With several tasks over the same connection, the minimum cycle time results from the sum of the individual cycles of the corresponding tasks. Meaning that taking 3 tasks of each of 300 milliseconds between request and response into account, the minimum cycle time of each individual task should be taken as 1 second, independent of the set cycle time. If receipts are missing, further tasks are delayed respectively.
- Mixing task types over a mutual connection is **not allowed, apart for** *SEND/RECV*.

*Please note that **with a faulty configuration, the load on the controller can be so great that an INA connection cannot be processed correctly.***
The reason for this could be permanent reconnects due to wrong answers, or false telegram sizes.
At the moment the reconnect loop is 1 second, when not all connections have been established, and a monitoring time of 5 seconds, as soon as all connections are established.

Up to 4 status variables can optionally be declared after each task line. The driver can then write information about its operating state to them. **These status variables must be created using**

logic in an APROL task (placed on the input border), and are **not made available by the driver itself**.

Status variables begin with the ":" character.
They have the following structure:

```
:TYPE:    EXT_NAME
```

TYPE	Type of status variable. Permitted types: REQ_STAT, REQ_POS_CTR, REQ_NEG_CTR, REQ_NEG_DATE Further information about this can be found in chapter *Description and value range of the status variables*.
EXT_NAME	Name of the external global variables where the status info should be written. If these variables aren't present when the driver is started, a cyclic check takes place every 60 seconds to see if the variables have been created by downloading the APROL task. Until this is done, the status information is considered lost. The driver checks the length of the PVs on the controller and interprets it as a USINT, UINT, or UDINT depending on the length.

After the desired status variables are configured, the shovel table is created. This table consists of a list of entries that specify the source/target variables, the offset in the telegram, and the length in bytes.

Individual elements are separated by a "," (comma):

```
VAR_NAME, BYTE_OFFSET, BYTE_LEN, CVT_INFO
```

VAR_NAME	Name of the external global variable being used as a source or destination for the communication. If the variable isn't available by the time the driver is starter, then every 60 seconds the system attempts to determine its address.
BYTE_OFFSET	Position of the variable in the telegram (regardless of the DB start for the task). This is specified in bytes!
BYTE_LEN	Number of bytes that should be copied. It is mandatory that the same variable types are present on the controller and the Simatic controller. The driver only copies the bytes; it doesn't interpret the values contained in the telegram.

CVT_INFO	Depending on the byte number BYTE_LEN, the settings **0-15**, **1**, or "**-**" are possible. **0 .. 15:** Gateway rule shows BIN0 .. BIN15 Transfers an individual bit (0..15) of a data word to a BOOL variable. This only affects variables with a BYTE_LEN == 1. **1:** Gateway-Regel shows S5FLOAT Converts two data words from or into the S5 FLOAT format. This only affects variables with a BYTE_LEN == 4. **-:** Gateway rule shows DEFAULT No type conversion, i.e. source bytes are simply copied to destination bytes, with swap routines handling the platform-dependent byte ordering. *Unallowable settings such as S5FLOAT with BYTE_LEN==1 are handled like DEFAULT!*

13.3.2.2 Configuration data module example

The following is an example of a configuration data module:

```
# created 2006.04.10 11:39:37 by Hartmann
""
"^WRITE/10.49.80.80/2000/100/0/100/1000
""
"READ/10.49.80.79/2001/101/10/100/1000
" :REQ_STAT:      Read_Db_100_reqStat
 ":REQ_POS_CTR:   Read_Db_100_posCtr
" :REQ_NEG_CTR:   Read_Db_100_negCtr
" :REQ_NEG_DATE:  Read_Db_100_negDate
"  DB100_PV_0,   0,  1, 0 "
"  DB100_PV_1,   1,  1, - "
"  DB100_PV_10, 148,  1, 1"
"  DB100_PV_2,   2,  1, -"
"  DB100_PV_3,   4,  4, -"
"  DB100_PV_4,   8,  4, -"
"  DB100_PV_5,   12,  4, 1"
"  DB100_PV_6,   16,  4, -"
"  DB100_PV_7,  20,  64, 0"
"  DB100_PV_8,  84,  64, 0"
"  DB100_PV_9, 148,   1, 0"
```

13.3.3 Configuring the driver for the controller

The controller driver *ApDrvS7* is configured in the **CaeManager** in the view of a CPU in the "APROL connections**"** tab.

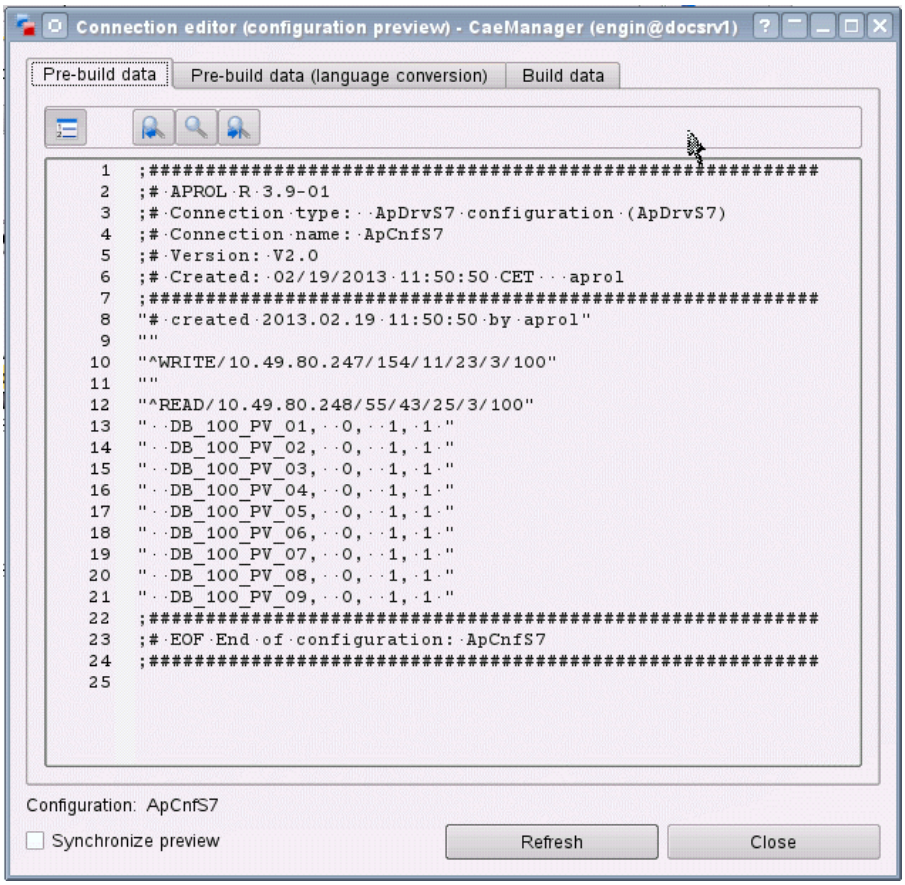


*Illustration **49**: Configuration preview in the CaeManager*

To create a new configuration for the *ApDrvS7* driver for your controller, select the field *ApDrvS7 configuration* and then select **New** from the shortcut menu.

The name 'ApCnfS7' is permanent and cannot be modified!

Then select the *Requests* field and create a new task from the shortcut menu (**New** menu item).

Carry out the necessary entries. Then the name of the status variables can be assigned.

If not names are entered, then the fields are not exported and the driver cannot output any status information.

Then the variable list for this task must be created.

Up to 64 tasks can be created, but make sure that no more than 16 connections are being used. In other words, the IP address and socket number values may be different only up to 16 times.

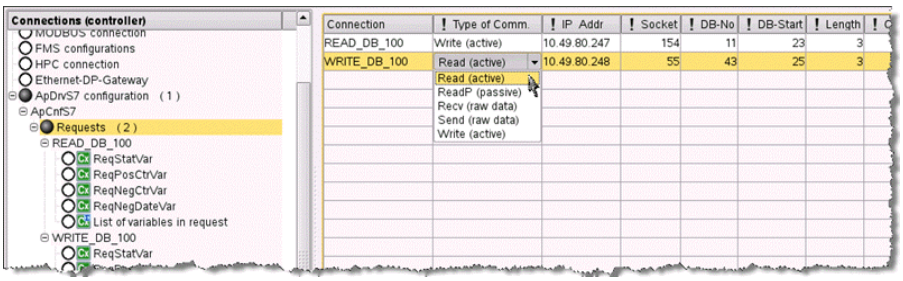


*Illustration **50**: Task list with two tasks using two connections*

READ- and WRITE tasks generally have to be handled over two different connections. The necessary settings need to be made on the Simatic controller.

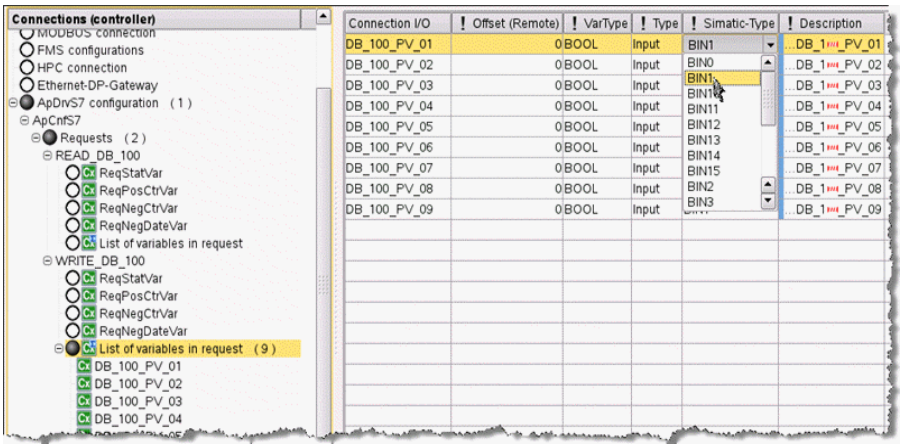


*Illustration **51**: Configuring a variable*

Note about VarTyp:
STRING types are interpreted as having 64 bytes only; this corresponds to 32 data words on the Simatic controller.

After the data has been entered, at least one compilation procedure must be executed for this controller in order for the data module to be exported!

13.3.4 Workflow description for the controller driver

After the download, the driver creates a "non-cyclic" task, which handles data traffic and is not tied so tightly to cycle times. The cyclic section remains and subsequently executes a watchdog function. If no signs of life occur from the non-cyclic task for approx. 30 seconds, it is stopped by the cyclic section, removed, and restarted.

The non-cyclic task first checks whether the data module is present. If it's found, all tasks are created internally. Cyclic timers are used to monitor the data module, the connection, each task, and to determine variable addresses. The latter is repeated until all PV addresses have been successfully retrieved.

The following defined timer settings are available:

Connection monitoring:	1000 ms if not all connections are established 5000 ms if all connections are established
Data module monitoring:	2000 ms
Detection of the variable addresses:	60000 ms
Timeout for acknowledgements:	2000 ms

Within the framework of the connection monitoring, the socket connection is established as unblocked and monitored cyclically. If an error occurs, the connection is closed and reopened.

Monitoring the data module checks whether the data module is (still) present and whether it has the same time stamp as the last time it was checked. If necessary, the driver is stopped and restarted with the new configuration. If the data module is removed, then the driver also stops and waits for a new data module. Determining the variable addresses cyclically checks whether all communication and status variable addresses are known. Determining the variable addresses cyclically checks whether all communication and status variable addresses are known. Since only global variables are accessed, it's not dangerous to remove and download APROL tasks. Global addresses are no longer valid only after recompiling, and the driver must be stopped before the download. Otherwise, the controller may crash.

Timeouts for acknowledgements are always carried out after a send telegram is issued. If the acknowledgement comes before the timer expires, then *POS_CTR* is incremented and the task is considered finished. If the timeout expires, then *NEG_CTR* is incremented and the connection is closed as a precaution. It generally takes a few seconds after this before the connection can be reestablished and everything runs normally.

The cyclic task timer is always reverted to when a task has ended. This happens regardless of the connection status or whether the task was successful. If the connection is not established when the timer expires, then *NEG_CTR* is incremented and the task is ended. If a negative acknowledgement is received, the same thing happens.

If a *WRITE* task is initiated, then all output data is copied to the telegram buffer, which is then sent. Copying in the non-cyclic task doesn't guarantee that all data in the telegram has been taken in at exactly the same point in time of the task class. Due to the cyclic and higher-priority system, interruptions by the operating system are possible. Only the consistency of 4-byte variables and smaller are guaranteed; strings are not guaranteed from the same cycle.

For a *READ* task, the request telegram is sent, and the data is copied to the destination with the response telegram. The same applies as for *WRITE* variables here. Variables with a length of 4 bytes or less are shoveled without interruption; strings can be put together from different cycles.

13.3.5 Description and value ranges for status variables

The following tables contain a brief description of the status variables and their value ranges:

REQ_STAT	Documents the current state of a request. The following values can be used: 0: Last task finished without an existing connection 1: Task inactive, waiting for request timer to expire 2: Task sent, waiting for acknowledgement **Passive jobs (READP and RECV) are always set to "1" with an existing connection.**
REQ_POS_CTR	Incremented **with each task that finishes successfully**. The variable is interpreted as a USINT, UINT, or UDINT and has the corresponding overflows.
REQ_NEG_CTR	Incremented **with each task that doesn't finish successfully**. The variable is interpreted as a USINT, UINT, or UDINT and has the corresponding overflows.
REQ_NEG_DATE	Must be of string type at least 32 bytes long. It receives the time stamp of the last error in text form.

13.4 Configuration using the Simatic software

Here is a short description of the steps required to configure a connection on the S7 page using the Simatic software.

Our test environment was created with the Step7 software version 5.2 (release V5.2.0.0).

First, a station is created where the corresponding communication modules will be installed. On the Ethernet modules, ***TCP connections*** must be supported. ***ISO-on-TCP*** does not work! Using the ***Configure Network*** menu item, go to the NetPro window and click on the CPU (not the respective CP). A window is shown listing the configured connections.

Right-click ***Add New Connection*** to select a connection with an ***unspecified station*** and ***TCP connection***. The warning that follows can be ignored and you will then see the properties window for the connection. The name for the connection can be freely selected here. ***Active connections*** must be switched **off**, the port address must be set locally under the address tab, the fields for the partner remain empty. Under options, set the ***operating mode*** to ***Fetch passive*** for ***READ*** tasks for the driver and ***Write passive*** for ***WRITE*** tasks for the driver. Additional settings are not necessary, not even in the cyclic program on the CPU.

13.4.1 Configuration of the jobs with step 7 -NCM or INAT for S5

Job on the B&R controller	configuration with step7-NCM	Configuration with INAT f. S5
READ	- No active connection establishment - Local address and port are to be configured - Partner address can remain empty when any IP addresses are allowed - Partner port is always unspecified - Operation mode "**Fetch passive**"	- Protocol type S5 - Task type "**Fetch passive**" - Source/target unused - Connection establishment passive - Protocol "TCP (safe)" - No control header - Life telegram - Target address 0.0.0.0 - Port address is to be configured (S5 port)
WRITE	- No active connection establishment - Local address and port are to be configured - Partner address can remain empty when any IP addresses are allowed - Partner port is always unspecified - Operation mode "**Write passive**"	- Protocol type S5 - Task type "**Write passive**" - Source/target unused - Connection establishment passive - Protocol "TCP (safe)" - No control header - Life telegram - Target address 0.0.0.0 - Port address is to be configured (S5 port)
READP		- Protocol type S5 - Task type "**Send direct**" - Source/target unused - Connection establishment passive - Protocol "TCP (safe)" - No control header - Life telegram - Target address 0.0.0.0 - Port address is to be configured (S5 port)
SEND	- No active connection establishment - Local address and port are to be configured - Partner address can remain empty when any IP addresses	- Protocol type "no protocol" - Task type "**Receive direct**" - Source/target unused - Connection establishment passive - Protocol "TCP (safe)"

Job on the B&R controller	configuration with step7-NCM	Configuration with INAT f. S5
	are allowed - Partner port is always unspecified - Operation mode "**Send/Receive**" - Handling of send data via the controller program	- No control header - Life telegram - Target address 0.0.0.0 - Port address is to be configured (S5 port)
RECV	- No active connection establishment - Local address and port are to be configured - Partner address can remain empty when any IP addresses are allowed - Partner port is always unspecified - Operation mode "Send/Receive" - Handling of received data via the controller program	- Protocol type "no protocol" - Task type "Send direct" - Source/target unused - Connection establishment passive - Protocol "TCP (safe)" - No control header - Life telegram - Target address 0.0.0.0 - Port address is to be configured (S5 port)

14 TI-Driver

14.1 General information about the TI driver

.

14.1.1 Important information about the TI driver

The driver package described here is used to connect TI controllers (Texas Instruments) from the Siemens company to APROL over a serial connection.

The user must be familiar with the TI system documentation in order to connect TI controllers. The user must know how to connect stations over a serial connection and also be able to define routing parameters and be familiar with the structure of data points on the controllers!

This driver has been added to an existing APROL release. After updating to a newer release (if necessary), the TI driver updates are ready in your system.

14.1.2 Description of driver behavior

The TI driver differentiates between cyclic read, cyclic write, and event-driven write tasks. The various variable ranges on the controller can be addressed as needed. The individual read access operations are executed consecutively. When doing so, the driver tries to keep to the configured cycle time. If all of the tasks cannot be executed within the cycle times, then the tasks are automatically executed as quickly as possible. Write tasks where process control system variables change are given precedence and then read as quickly as possible.

The following information should be taken into account during engineering!

Since the driver carries out read access operations for both task types (*read* and *write*), it's the supply for all of its variables. This makes it impossible to connect the pins of display driver function blocks directly with driver variables to supply the dynamics of the display driver function block. At least one block must always be placed and connected between the pins of the display driver function block and the I/O border with the driver variables so that the process control system task can set the driver variables.

If an error occurs, the driver sets all read PVs from the respective task to invalid.

14.2 Installation of the TI driver software

The installation of the driver is made when the APROL system software is installed, and must be selected here, in the dialog for driver selection.

14.2.1 Delivery contents of the driver packet TI driver

The following files are present on computers with the engineering and runtime system after the installation (path specified as well):

File with path	Description
/opt/aprol/bin/**TiDriver**	The driver program.
/opt/aprol/cnf/TiDriver/ example/**TiDriver.cnf**	Example of a configuration file for the TI driver.
/opt/aprol/cnf/TiDriver/ example/**TI_Treiber.pdf**	This documentation about the TI driver.

14.3 Start options and configuration

14.3.1 Description of TI driver start options

The driver must be selected under the CC modules for the Runtime computer and then identified for starting. It's possible to enable process redundancy with automatic switching when an error occurs. To influence driver behavior, a number of start options must be configured. The following table lists values that can be configured in the CaeManager as well as their description:

Option	Description
-baudRate	Baud rate at which the serial interface is operated. The following baud rates can be selected: 9600, 19200, 38400, 57600,115200
-databits	Number of data bits for communication. Valid values range from 5 to 8 bits.
-device	Name of the serial interface to be used. This can be /dev/ttyS0 to /dev/ttyS3 for COM1 to COM4 or /dev/ttyUSB0 to /dev/ttyUSB3 for serial interface modules on the USB port. Be aware that the serial interfaces requires the rights "**RW-RW-RW-**" (**666**) so that they can be opened by the driver. If this setting has not been made, the super user must execute *chmod 666 DEVICE_NAME*.

Option	Description
-d	Activates the debug mode and ensures that the driver remains in the foreground when opened via the console, and shows the debug output. Bit pattern (DEBUG_FILTER) for debug outputs: 0x00000001: Outputs error messages 0x00000002: Outputs normal messages 0x00000004: Outputs loop messages 0x00000008: Outputs the configuration 0x00000010: Output of Iosys events 0x00010000: Hex input telegram header 0x00020000: Hex input telegram data 0x00040000: Hex output telegram header 0x00080000: Hex output telegram data Example: The individual bit patterns are added to output the normal and error messages. -d 0x3
-ignoreString	A chain of letters that are not permitted to be present in variable names and are automatically replaced with the character "_" by the driver. The following characters are the default for this option: &.[]+-*/<>@ This option is useful if old configuration files are to be used.
-iosys	Almost every APROL module establishes the connection to Iosys in the Runtime system via the port (no. 0 to no. 15). The number of the port and the name of the computer are given when configuring the default control computer.
-l LOGFILE_NAME	All output is routed to a log file named LOGFILE_NAME. Normally, the driver does not create output. However, it can use the debug filter to write targeted status messages.
-model	Defining the controller type. This definition is necessary because the various CPU types support different telegram lengths. A differentiation is made between types 545, 555, and 565.
-n APP_NAME	Changes the default application name for the driver. The default name is **'TiDriver'**. This name is also found in the names of status process variables.
-noIgnoreString	This parameter can be used to disable the replacement of invalid characters by the character "_" (see option -ignoreString).
-parity	Parity setting for serial communication. The following values can be set: none, even and odd

Option	Description
-driverName PLC_NAME	Name of the controller and therefore also the directory name where the driver's configuration file can be found: This means: $HOME_RUNTIME/RUNTIME/cnf/TiDriver/ PLC_NAME/**TiDriver.cnf**
-restart	The AprolLoader activates the automatic restart function if the driver is stopped for an unknown reason. The number of automatic restarts must be specified.
-restartTime T	Enables automatic process redundancy switching if successful communication is not possible for T seconds. This function is switched off by setting **T to NULL**. If the driver is not able to successfully exchange data with the controller for T seconds, it closes itself and starts again automatically after one second. In this way, a slave driver has the possibility to take over handling of communication. Please note that T cannot be less than shortest cycle time for all tasks!
-self	If this option is not changed, a two digit number (instance beginning with 01 for the computer) is automatically attached to the module name for each module in the process list that has been started. If the module is started more than once on a computer, this number is incremented.
-stopbits	Number of stop bits for a data byte. The values 1 and 2 are used.

14.3.2 Creating the configuration file

Select *Project connection* in the control computer. Select *TI-Driver* and create a new entry with the shortcut menu.

Specify the CPU to be connected. Since this CPU is only available as a directory name, it doesn't have to be engineered in the project.

Now create a task for writing and/or a task for writing for each data type to be acquired. For each task, specify the type of data and the desired cycle time for reading in seconds.

Then the list of PVs contained in each task needs to be entered for each task. In the Variables sub-item for the respective task, enter the name of all PVs to be used. The *Address info* and *Variable type* fields must be filled in.

Address info is either the numeric address of the variable or the structure component and structure number for certain structure types.

Example:

For a variable word of 300, enter *300* and *Word*.

Enter *ALA.6* and *Word* for an alarm, which means that the *ALA* field must be transferred by *Alarm 6*.

To wire the variables in the chart, an IEC type that fits to the measurement range must be entered under *type*. This entry is not significant for the driver and is also not written to the configuration file.

However, the variable type definitely matters to the Iosys and the engineering.

Controller variables of type *Bit* are in Iosys integer variables; variables of type *String* are in Iosys string variables. All others are variables of type *Real* (Double in Iosys).

The IEC types must be compatible to the Iosys type so that the events can be analyzed correctly! If needed, scaling parameter and measurement range limits can be specified for types that aren't *Bit* or *String*.

The number of characters (without null termination) must be configured for string variables.

The TiSoft controller programming software can be used to configure the number of words.

After the configuration, you can view the structure of the configuration file under the controller entry in the configuration preview.

14.3.3 TI driver status variables

Each driver instance of the TI driver creates a pool of status variables in the Iosys that can be used for diagnostic purposes. The name of these PVs is formed from the name of the application <A> and the name of the controller <P>.

See also the start options -n and -controller.

Status variable	Description
A_P_debugFilter	PV that can be used to influence log output. Messages can be enabled or disabled by setting or deleting individual bits. All messages are usually disabled. However, if this PV already has a value when the driver is restarted, then it is used. The messages are output via **stderr**. This channel is redirected to a log file if desired. See also the start options -l and A_P_useLogFile. The following bit patterns can be combined with each other: 0x00000001: Outputs error messages 0x00000002: Outputs normal messages 0x00000004: Outputs loop messages 0x00000008: Outputs the configuration 0x00000010: Output of Iosys events 0x00010000: Hex input telegram header 0x00020000: Hex input telegram data 0x00040000: Hex output telegram header 0x00080000: Hex output telegram data
A_P_errorText	Name of a PV of type String where the driver writes error texts. Error texts always remain until the next error occurs. For this reason, always check the data of the occurrence when analyzing errors (IosEv indicates the date). **Note**: If this PV should be placed in a CFC, it needs to be created in the configuration editor during configuration!

Status variable	Description
A_P_useLogFile	PV of type String. Specifying a valid filename (this can be done e.g. with pio) tells the driver to create a new log file under this name and redirect its output to it. If the PV is filled with an empty string, the output is redirected to the null device and deleted. If this PV has a value when the driver is started and the -I option is not set explicitly, then the file is used as a log file by the driver. If the -l option was used when started, then this takes priority and the driver overwrites the value of this PV with the name specified with *-l.* **Attention!** A log file is generally cut and refilled when opened!
A_P_error	Name of a PV of type Integer where the driver writes error numbers. The value 0 means "No error". A value other than zero corresponds to a TI communication error. The error number applies only for the time until the next task; at that point, it will change. **Note**: If this PV should be placed in a CFC, it needs to be created in the configuration editor during configuration!
A_P_status	Status PV that can take on two values: 0 - "Everything OK", 1 - "An error number has occurred". **Note**: If this PV should be placed in a CFC, it needs to be created in the configuration editor during configuration!

14.3.4 Diagnosis of the driver

Diagnosis of the driver can take place for start-up using the console.

When starting the driver with the option '*–d DEBUG_FILTER'*, various debug output texts can be read allowing simple analysis.

During normal operation, an analysis of the driver should be carried out using the IosEv status variables and, if necessary, records in a log file. When using a log file, it is important, for reasons of space, not to forget to deactivate outputting debug information after the analysis and to divert the output to */dev/null*. Only then can the log file be deleted from the computer and the space on the hard drive freed up again.

Example call of a driver in debug mode on the console

```
/opt/aprol/bin/TiDriver -driverName RPSTi1 -device /dev/ttyS0 -baudRate 19200 -parity none -
stopbits 1 -databits 8 -model 555 -d 0x3 -iosys tfredu9:4,tfredu10:4
```

The driver connects with a redundant runtime system in this example.

14.3.5 Example configuration file TiDriver.cnf

After the APROL installation, the example `TiDriver.cnf` configuration file can be found in the directory `/opt/aprol/cnf/TiDriver/example/`

```
# System test configuration
# Created: 2005.01.11  14:37:02 by aprol
# Status variable name: TiDriver_RPSTi1_status
# Error variable name: TiDriver_RPSTi1_error
# ErrorText variable name: TiDriver_RPSTi1_errorText
^DB 1                          ALARM   A   10
VAR_RPSTi1_01AHA1              AHA.1   16  5   0   100
VAR_RPSTi1_01AHHA1             AHHA.1  16  5   0   100
VAR_RPSTi1_01ALA1              ALA.1   16  5   0   100
VAR_RPSTi1_01ALLA1             ALLA.1  16  5   0   100
^DB 2                          X       A   10
VAR_RPSTi1_02X1                1   15
VAR_RPSTi1_02X2                2   15
VAR_RPSTi1_02X3                3   15
VAR_RPSTi1_02X4                4   15
^DB 3                          Y       A   10
VAR_RPSTi1_03Y1                1   15
VAR_RPSTi1_03Y2                2   15
VAR_RPSTi1_03Y3                3   15
VAR_RPSTi1_03Y4                4   15
^DB 4                          WX      A   10
VAR_RPSTi1_04WX1               1   16  0   0   100
VAR_RPSTi1_04WX2               2   16  0   0   100
VAR_RPSTi1_04WX3               3   16  0   0   100
VAR_RPSTi1_04WX4               4   16  0   0   100
^DB 5                          WY      A   10
VAR_RPSTi1_05WY1               1   16  0   0   100
VAR_RPSTi1_05WY2               2   16  0   0   100
VAR_RPSTi1_05WY3               3   16  0   0   100
```

```
VAR_RPSTi1_05WY4               4   16  0   0   100
^DB 6                          V       A   10
VAR_RPSTi1_06V1                1   16  3   0   100
VAR_RPSTi1_06V2                2   16  3   0   100
VAR_RPSTi1_06V3                3   16  3   0   100
VAR_RPSTi1_06V4                4   16  3   0   100
^DB 7                          V       A   10
VAR_RPSTi1_07V5                5   18  2
^DB 8                          STW     A   10
VAR_RPSTi1_08STW1              1   16
VAR_RPSTi1_08STW2              2   16
VAR_RPSTi1_08STW3              3   16
VAR_RPSTi1_08STW4              4   16
^DB 9                          K       A   10
VAR_RPSTi1_09K1                1   16
VAR_RPSTi1_09K2                2   16
VAR_RPSTi1_09K3                3   16
VAR_RPSTi1_09K4                4   16
^DB 10                         C   A   10
VAR_RPSTi1_10C1                1   15
VAR_RPSTi1_10C2                2   15
VAR_RPSTi1_10C3                3   15
VAR_RPSTi1_10C4                4   15
^DB 11                         LOOP    A   1
VAR_RPSTi1_11LHA6              LHA.6   16  5   0   2.2
VAR_RPSTi1_11LHHA6             LHHA.6  16  5   0   2.2
VAR_RPSTi1_11LLA6              LLA.6   16  5   0   2.2
VAR_RPSTi1_11LLLA6             LLLA.6  16  5   0   2.2
^DB 12                         TMR A   10
VAR_RPSTi1_12V13               13  16  5   0   3200
VAR_RPSTi1_12V14               14  16  5   0   3200
VAR_RPSTi1_12V15               15  16  5   0   32
VAR_RPSTi1_12V16               16  16  5   0   32
```

15 wdpfDriver

15.1 General information about the wdpfDriver

The APROL *wdpfDriver* is used to cyclically exchange process data with other applications using the file system.
The driver supports a READ file, a WRITE file and a SYNC file for bidirectional variables.
Multiple drivers can be operated at the same time so that more than one file can be used. Using the APROL redundancy concept, the driver can also be started in a redundant manner.
The driver supports variable types BIT, UINT and FLOAT as input types and can represent these as desired on integer, float, and string Iosys types. It is also possible to rescale the UINT and FLOAT types.

15.2 wdpfDriver start options

The following table shows a list of the start options supported by the driver with the corresponding descriptions.

Option	Description
-n NAME	Sets the internal driver name to NAME. This name will be used, for example, for the automatic creation of status variables in Iosys (see status variables in Iosys)
-ignoreCnfPath	The driver searches its configuration files relative to the current directory and not relative to the APROL environment (see configuration of the driver).
-l LOGFILE	The driver creates a log file named LOGFILE and writes its STDOUT and STDERR output texts to this file. Debug output texts (which are activated as a supplement) are also written to this file and can be output using "tail -f".
-ignoreString STRING	All STRING characters are replaced in the Iosys variable names with the character '_'
-setBufferSize SIZE	Limits the file size to a maximum of SIZE bytes, the default is 510 bytes
-setConnTimeout COMM_TIMEOUT	If an input file is not found after COMM_TIMEOUT milliseconds, then the Conn Status PV is set to Error, the default is 3000
-setCheckTime CHECK_TIME	The input file is searched for cyclically every CHECK_TIME milliseconds, the default is 500 milliseconds
-setWriteTime WRITE_TIME	The write file is recreated cyclically every WRITE_TIME milliseconds, the default is 1000 milliseconds

Option	Description
-d DEBUG_FILTER	Starts the driver in debug mode with the debug filter DEBUG_FILTER. With this filter, bit masking can be used to output various content (output via stderr).
-f DEBUG_FILTER	Starts the driver without debug mode with the debug filter DEBUG_FILTER. With this filter, bit masking can be used to output various content (output via stderr).
	Bit mask for debug filter 0x00000001: Output of error messages 0x00000002: Output of process messages 0x00000004: Output of extended error messages 0x00000008: Output of extended process messages 0x00000010: Output of the task configuration, can also be activated online 0x00000020: Output of connection status messages 0x00000100: Hexdump of the input data 0x00000200: Hexdump of the output data
-readCfg READ_CFG_FILE -syncCfg SYNC_CFG_FILE -writeCfg WRITE_CFG_FILE	Specifies the name of the configuration file for the various task types. At least one of these options must be set, and all three task types are permitted to be set at the same time
-readLen RLEN -syncLen SLEN -writeLen WLEN	Specifies the size of the respective communication file if it is different than the automatically generated size for the respective configuration files. Example: If 10 words are defined in READ_CFG_FILE, but 40 bytes (20 words) are written, then RLEN must be set to 40 here. Example: If 10 words are written in the write task, but the partner station expects 80 bytes, then WLEN is to be set to 80. The excessive bytes are filled with zeros.
-readFile RNAME -writeFile WNAME -syncInFile SINAME -syncOutFile SONAME	Specifies the name of the respective data file. When doing this, take note that an input and an output file must be entered for the bidirectional SYNC data! The absolute file name must be entered, and is independent of the APROL environment.

15.3 Configuration of the wdpfDriver

The following section explains the driver's configuration.

```
Variable name,       data type,     optional bit number  #Comment
Variable name,       data type,     optional bit number  #Comment
Variable name,       data type,     optional bit number  #Comment
Variable name,       data type,     optional bit number  #Commentar
...
...
```

Variable name:

The variable name is provided as single variable in the WDPF and in APROL. The variable

name always starts with a letter and does not contain any special characters except for underline. The length is variable (and limited to 8 characters).

Data type:
Only 3 different data types are needed for the connection to be implemented.
REAL, 32 bit float (in accordance with IEC), shown in APROL using a double/float PV
UINT, 16 bit unsigned integer, shown in APROL using an integer PV
BIT, individual bit, coded using the bit number as part of a 16 bit variable
word, shown in APROL using an integer PV.

Optional bit number:
The optional bit number is only provided with the BIT data type. All other types have lines with only two columns.
The BIT data type is transferred compressed in a 16 bit variable word. The bit number defines which bit in the variable word is occupied by the variable.
The bit number can only be defined from 0 - 15. A number larger than 15 is considered a configuration error.
The individual bit variables are stored in the same variable word as long as the bit number is ascending (i.e. a new variable word is started if a smaller or identical number follows a bit number). A new variable word is also started if a different data type follows a bit.

Comment:
A comment is started with the character "#"; all characters after the comment character are cut out.

15.4 The wdpfDriver status variables

The driver automatically creates a PV in the Iosys for each configured task.
Depending on the type of task, the PV name uses the following the convention:

*DRIVERNAME*_READ_connState
*DRIVERNAME*_WRITE_connState
*DRIVERNAME*_SYNC_connState

DRIVERNAME is equal to the value of the start option "**-n**".
Status variables are not created for tasks types that are not configured.

The value of the PV in the Iosys provides an overview of the connection status:
0x0000 means that the connection is created, but not yet operated
0x0001 means that this connection currently has an error
(the file for reading is not present after the timeout time, cannot be opened, cannot be written, etc.)

0x0002 means that there are currently no errors

The driver also creates a PV with the name:

*DRIVERNAME*_debugFilter

This PV is used to activate debugging online. Please refer to the start options for the meaning of the bits in this PV.

This description is under construction at present.

*Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.*

15.5 Debugging

If the status variable for a connection is set to error (value equal to 1) and the cause of error is unknown, then the online debugging can be activated using *pio*:

For this

pio -pv *DRIVER-NAME*_debugFilter -set 15

entered into the console, the driver's text output can now be viewed in the driver's output file for stderr and fix the cause of error by determining the error text.

It is important to deactivate debugging after the error has been corrected, so that the messages do not fill up the hard drive:

pio -pv DRIVER-NAME_*debugFilter -set 0*

15.6 Additional notes

Please take the following information into consideration:

- Start files are not sent to the driver.
- The driver must be integrated in the project via "customer applications" (see corresponding documentation).

16 HPC

This description is under construction at present.

Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.

17 Dflt

.

17.1 Dflt in CC

This description is under construction at present.

Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.

17.2 Dflt in Plc

This description is under construction at present.

Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.

18 DrvEthDp

This description is under construction at present.

Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.

19 Et200

This description is under construction at present.

Please inform yourself in regular intervals about the current APROL documentation on our internet side **www.br-automation.com**, in the area Material related downloads.

20 Appendix

20.1 Typical reported problems / solutions (FAQ)

The following overview contains **frequently asked questions** (FAQ) which concern the **interfaces between control computers / controllers** and the **connection of foreign systems** via the APROL standard drivers.

The list is not sorted by themes (drivers) and will be extended on the basis of support queries.

1	**Problem report:**
	A serial APROL driver which accesses an external serial interface that is on a POWERLINK bus coupler via POWERLINK bus, seems to be slow.
	Solution approach:
	As comparison, two almost identical CAE projects are used where the POWERLINK bus cycle times are configured differently (with the factor 5). The increase in the POWERLINK bus cycle time had an effect in the same magnitude of the response times on the serial (MODBUS) driver. The problem described here surely affects other non-serial drivers which do not have their communication interface on the same hardware where the driver is running. In order to restore the previous response times of the slave station, the POWERLINK bus cycle time must be set back to the original value.

20.2 Revision history

Manual version	Date	Change	Author / *checked by*
4.06	25.08.2014	New chapter *ApDrvAnsl diagnosis (ANSL cross-communication)*	MHa
4.05	12.08.2014	Launching options ModbusPlus driver: Link to manual 'X99 CC Modules'	KSc
4.04	12.05.2014	Revision of chapter *Modbus Controller Driver*.	KSc *ALu*
4.03	04.03.2014	Hyperlinks for connections	RN
4.02	28.01.2014	Revision of the chapter *EventDriver*	KSc *MHa*
4.01	13.12.2013	Update of chapter '*OPC server*'	KSc
	10.12.2013	Format to DIN A4	RN
4.00	21.10.2013	Revision of chapter *AnslDriver*	KSc *MHa, MM*
	03.09.2013	New chapter *AnslDriver*	KSc
	13.02.2013	New chapter *Typical reported problems / solutions (FAQ)*	KSc
	25.01.2013	New names for the system variables chapter *Status variables of the EventDriver*	RN *EM*
	14.12.2012	Revision of chapter *TI driver*	KSc *MM*
	06.09.2012	Revision of terms (APROL system / CC-Account)	KSc
3.07	05.12.2011	Simatic driver for the control computer: Extension of the chapter *PV declaration*	KSc *MHa*
3.06	25.11.2011	Actualization of the chapter *EventDriver's status variables*	KSc *MHa*
3.05	15.06.2011	Update of chapter *Modbus Controller Driver*. New field 'Address' and 'Ample time'	KSc / ALu
	14.06.2011	Actualizing of chapter *InaDriver status variables*	KSc
	24.05.2011	Expansion of the chapter *Event variables in external tasks:* Use of the 'EventLink mode' per C function block	EM

Manual version	Date	Change	Author / *checked by*
3.04	11.03.2011	Update of chapter *OPC server* (V3.7.0.0)	KSc *MHa*
	04.10.2010	Update of chapter *OPC server* (V3.6.1.0)	KSc *MHa*
3.03	12.08.2010	Revision of terms: ST, SFC, CFC	RN
3.02	17.05.2010	Chapter *Simatic driver for the control computer* and chapter *Simatic driver for the controller* Addressing up to 255 blocks	KSc *MHa*
3.01	17.12.2009	Extension of the chapter *EventDriver*	KSc *MHa*
3.00	23.11.2009	Note about telegram size in chapter *Additional notes about mode of operation*	RN *MHa*
	09.10.2009	Revision of the chapter *EventDriver*	KSc *MHa*
	05.10.2009	Expansion of the chapter *Debugging the OPC server*	KSc *MHa, JRu, BS*
	14.09.2009	Change: "Module view" tab to "Software configuration" tab	KSc
2.07	30.07.2009	Revision of the chapter *Simatic driver for the control computer*	KSc *MHa*
2.06	17.07.2009	Actualization of the chapter *InaDriver's status variables*	RN *JF*
	17.07.2009	New chapter *Debugging the OPC Server*	KSc *MHa, JR*
	09.06.2009	Update of chapter *OPC server*	RN *JR, BS, MSc*
2.05	06.04.2009	Correction of the chapter *Dispatcher start options*	KSc
2.04	06.02.2009	Chapter *Description of the InaDriver's start options* Default value for "-delay" option changed.	KSc
	20.01.2009	Chapter *FMS coupling* removed	KSc
2.03	10.10.2008	SimaticDriver: Chapter *Driver start options* revised	KSc *MHa*
	08.10.2008	Revision of chapter *RK512Driver launching options* (maximum possible number of instances).	KSc
2.02	24.09.2008	Chapter *Reference values of the Simatic driver for the control computer* extended	MSa *MHa*

Manual version	Date	Change	Author / *checked by*
2.01	14.07.2008	Chapter *RK512Driver* (driver reference values)	KSc MHa
2.00	25.06.2008	Chapter *InaDriver status variables* revised	KSc *RP, JF*

File name of this documentation: *APROL_R40_F1_DriversBRCouplings_001.pdf*

20.3 Document information

Version: 4.06
Date: 25.08.2014

21 Glossary

Module

Has several meanings.

When referring to **B&R controller software**, a module is a file that can be loaded to the controller.

The following **software modules** are used:

System module, data module, configuration module, task, driver module, and when referring to **B&R controller hardware** in the area hardware, a module is an insert card that can be mounted to the backplane or plugged into another module. See also "Mixed module".

Event

Event. Any change to a process variable (status or value) is considered an event.

PV

Abbreviation for **p**rocess **v**ariable

RPM package

A special memory format for files. Files with similar functions are saved in a packaged for installation by *YaST*.

runtime system

This system contains all APROL programs required for a process control system. These programs are used to acquire, distribute, archive, and control the data in a process control system. A Linux login and an APROL login must exist for this system. The data for the Runtime system is determined by and downloaded from the Engineering system.

Drivers

A program that enables communication (data exchange) via a communication level between two units in the process control system.

CORPORATE HEADQUARTERS

Bernecker + Rainer Industrie-Elektronik Ges.m.b.H.
B&R Straße 1
5142 Eggelsberg
Austria
Tel.: +43 (0) 77 48 / 65 86 - 0
Fax: +43 (0) 77 48 / 65 86 - 26
info@br-automation.com
www.br-automation.com

Always near to you - 120 offices in over 50 countries - **www.br-automation.com/contact**

Argentina · Australia · Austria · Belarus · Belgium · Brazil · Bulgaria · Canada · Chile · China · Croatia · Cyprus · Czech Republic
Denmark · Egypt · Emirates · Finland · France · Germany · Greece · Hungary · India · Indonesia · Ireland · Israel · Italy · Japan · Korea
Kyrgyzstan · Malaysia · Mexico · New Zealand · The Netherlands · Norway · Pakistan · Poland · Portugal · Romania · Russia · Singapore
Slovakia · Slovenia · South Africa · Spain · Sweden · Switzerland · Taiwan · Thailand · Turkey · Ukraine · United Kingdom · USA