rho 3

DDE Server
Software manual

Edition 102

rho 3

Software manual

1070 072 161-102 (97.08) GB

© 1997

by Robert Bosch GmbH, Erbach / Germany
All rights reserved, including applications for protective rights.
Reproduction or distribution by any means subject to our prior written permission.

Discretionary charge 10,- DM

1 Safety Instructions

Before you start working with the DDE Server, we recommend that you thoroughly familiarize yourself with the contents of this manual. Keep this manual in a place where it is always accessible to all users.

1.1 Proper use

This instruction manual presents a comprehensive set of instructions and information required for the standard operation of the described products.

The products described hereunder

- were developed, manufactured, tested and documented in accordance with the relevant safety standards. In standard operation, and provided that the specifications and safety instructions relating to the project phase, installation and correct operation of the product are followed, there should arise no risk of danger to personnel or property.
- are certified to be in full compliance with the requirements of the
  - COUNCIL DIRECTIVE 89/336/EEC of May 3rd 1989 on the approximation of the laws of the Member States relating to electromagnetic compatibility, 93/68/EEC (amendments of Directives), and 93/44/EEC (relating to machinery)
  - COUNCIL DIRECTIVE 73/23/EEC (electrical equipment designed for use within certain voltage limits)
  - Harmonized standards EN 50081–2 and EN 50082–2
- are designed for operation in an industrial environment (Class A emissions). The following restrictions apply:
  - No direct connection to the public low–voltage power supply is permitted.
  - Connection to the medium and/or high–voltage system must be provided via transformer.

  The following applies for application within a personal residence, in business areas, on retail premises or in a small–industry setting:
  - Installation in a control cabinet or housing with high shield attenuation.
  - Cables that exit the screened area must be provided with filtering or screening measures.
  - The user will be required to obtain a single operating license issued by the appropriate national authority or approval body. In Germany, this is the Federal Institute for Posts and Telecommunications, and/or its local branch offices.

⇒ **This is a Class A device. In a residential area, this device may cause radio interference. In such case, the user may be required to introduce suitable countermeasures, and to bear the cost of the same.**

Proper transport, handling and storage, placement and installation of the product are indispensable prerequisites for its subsequent flawless service and safe operation.

1.2 Qualified personnel

This instruction manual is designed for specially trained personnel. The relevant requirements are based on the job specifications as outlined by the ZVEI and VDMA professional associations in Germany. Please refer to the following German–Language publication:
Weiterbildung in der Automatisierungstechnik
Publishers: ZVEI and VDMA Maschinenbau Verlag
Postfach 71 08 64
60498 Frankfurt/Germany

Interventions in the hardware and software of our products not described in this instruction manual may only be performed by our skilled personnel.

Unqualified interventions in the hardware or software or non–compliance with the warnings listed in this instruction manual or indicated on the product may result in serious personal injury or damage to property.

Installation and maintenance of the products described hereunder is the exclusive domain of trained electricians as per IEV 826–09–01 (modified) who are familiar with the contents of this manual.

Trained electricians are persons of whom the following is true:

- They are capable, due to their professional training, skills and expertise, and based upon their knowledge of and familiarity with applicable technical standards, of assessing the work to be carried out, and of recognizing possible dangers.
- They possess, subsequent to several years' experience in a comparable field of endeavour, a level of knowledge and skills that may be deemed commensurate with that attainable in the course of a formal professional education.

With regard to the foregoing, please read the information about our comprehensive training program. The professional staff at our training centre will be pleased to provide detailed information. You may contact the centre by telephone at (+49) 6062 78–258.

1.3 Safety markings on components

DANGER! High voltage!

DANGER! Corrosive battery acid!

CAUTION! Electrostatically sensitive components!

Disconnect mains power before opening!

Lug for connecting PE conductor only!

Functional earthing or low–noise earth only!

Screened conductor only!

1.4 Safety instructions in this manual

DANGEROUS ELECTRICAL VOLTAGE
This symbol warns of the presence of a **dangerous electrical voltage**. Insufficient of lacking compliance with this warning can result in **personal injury.**

DANGER
This symbol is used wherever insufficient or lacking observance of this instruction can result in **personal injury.**

CAUTION
This symbol is used wherever insufficient or lacking observance of instructions can result in **damage to equipment or data files.**

⟹ This symbol is used to alert the user to an item of special interest.

1.5 Safety instructions for the described product

DANGER
Fatal injury hazard through ineffective Emergency–OFF devices! Emergency–OFF safety devices must remain effective and accessible during all operating modes of the system. The release of functional locks imposed by Emergency–OFF devices must never be allowed to cause an uncontrolled system restart! Before restoring power to the system, test the Emergency–OFF sequence!

DANGER
Danger to persons and equipment!
Test every new program before operating the system!

DANGER
Retrofits or modifications may interfere with the safety of the products described hereunder!

The consequences may be severe personal injury or damage to equipment or the environment. Therefore, any system retrofitting or modification utilizing equipment components from other manufacturers will require express approval by Bosch.

DANGEROUS ELECTRICAL VOLTAGE
Unless described otherwise, maintenance procedures must always be carried out only while the system is isolated from the power supply. During this process, the system must be blocked to prevent an unauthorized or inadvertent restart.

If measuring or testing procedures must be carried out on the active system, these must be carried out by trained electricians.

CAUTION
Danger to the module!
Do not insert or remove the module while the controller is switched ON! This may destroy the module. Prior to inserting or removing the module, switch OFF or remove the power supply module of the controller, external power supply and signal voltage!

CAUTION
Only Bosch–approved spare parts may be used!

CAUTION
Danger to the module!
All ESD protection measures must be observed when using the module! Prevent electrostatic discharges!

Observe the following protective measures for electrostatically endangered modules (EEM)!

- The Employees responsible for storage, transport and handling must be trained in ESD protection.
- EEMs must be stored and transported in the protective packaging specified.
- Out of principle, EEMs may be handled only at special ESD work stations equipped for this particular purpose.
- Employees, work surfaces and all devices and tools that could come into contact with EEMs must be on the same potential (e.g. earthed).
- An approved earthing wrist strap must be worn. It must be connected to the work surface via a cable with integrated 1 MW resistor.
- EEMs may under no circumstances come into contact with objects susceptible to accumulating an electrostatic charge. Most items made of plastic belong to this category.
- When installing EEMs in or removing them from an electronic device, the power supply of the device must be switched OFF.

1.6 Trademarks

All trademarks referring to software that is installed on Bosch products when shipped from the factory represent the property of their respective owners.

At the time of shipment from the factory, all installed software is protected by copyright. Software may therefore be duplicated only with the prior permission of the respective manufacturer or copyright owner.

MS–DOSr and Windows™ are registered trademarks of Microsoft Corporation.

PROFIBUS® is a registered trademark of the PROFIBUS Nutzerorganisation e.V. (user organization).

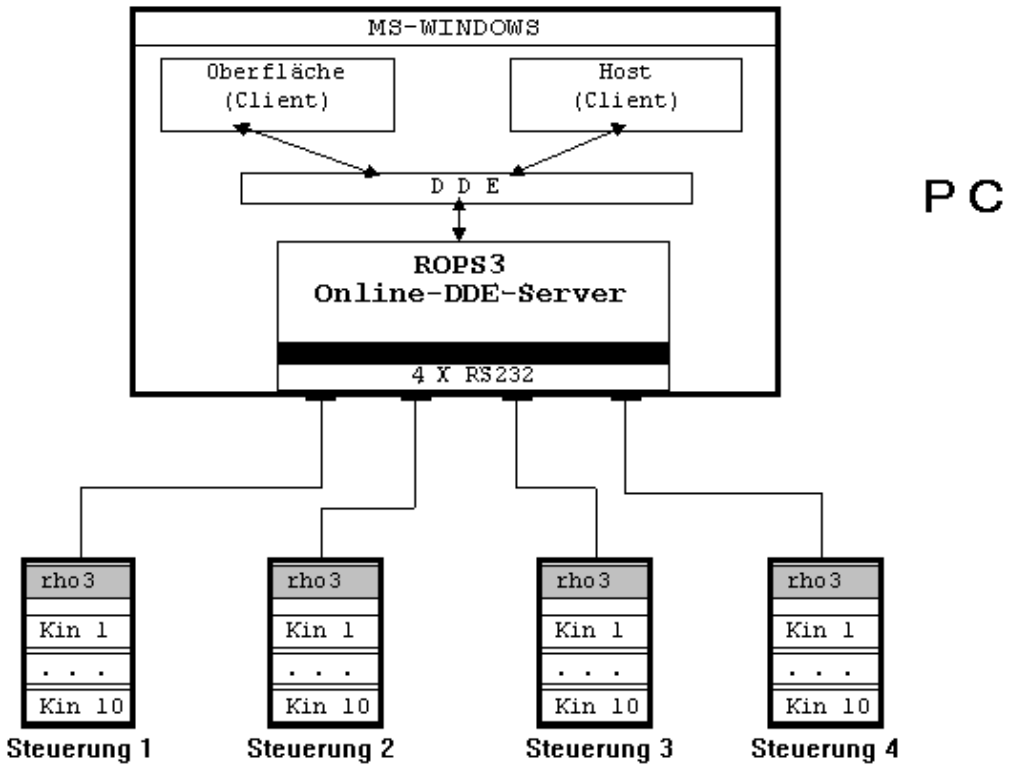
2 Introduction

Beginning with version W3B, the ROPS3 software package provides a variety of functions for communicating with the rho 3.0 Robot Control.

The complement of features encompasses file transfer functions, processing and status functions, the latter of which are also referred to as *online functions*. These services are integrated in a program package that is provided with a comfortable graphical user interface (GUI) for operation with the Microsoft® Windows® operating system. The ROPS3 software package also contains several DOS tools which shall not be specifically discussed. The extent of these communications options is limited to file transfer functions.

In order to enable the user to integrate the online functions into his own GUI, or to "remote-control" the rho 3.0 control by means of Windows commands, as opposed to direct manual operation, a function library in conjunction with a standardized interface is required. For this reason, the DDE inter-process communications interface for Windows is provided. It is supported by all Windows operating system variants, such as Microsoft Windows 3.1 and Windows for Workgroups 3.11, as well as stand-alone operating systems, such as Windows 95 and Windows NT.

The descriptions in this manual apply to software version 3.0 of the Online DDE Server.



Overview of DDE Server

2.1 DDE and DDEML

The Dynamic Data Exchange (DDE) comprises a means of inter-process communications within the Windows environment. It uses the principle of shared memory to effect the data exchange between two Windows applications. For this purpose, one application must act as the *client* (i.e., the GUI, or Windows *desktop*), while the other acts as the *server* (i.e., the ROPS3 DDE server).

In this context, an application is designated as a server if it offers services to other applications. (Throughout the following descriptions, these services will also be referred to as *items*.) The application utilizing the services or items provided by a server is deemed to be the client.

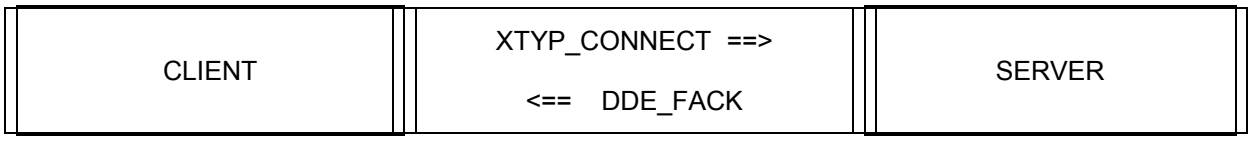
To enable the required communications, the On-line DDE Server provides several services that can be utilized by the client. The referred services facilitate the setting up of connections, performing data exchange, monitoring, execute and other functions. In the case of data exchange, a differentiation can be made between a one-time data transfer (i.e., process start) and a dynamic data exchange providing a continuous update (i.e., axis display). All functions governing the process communications between client and server are located in the DDEML, or Dynamic Data Exchange Management Library. It can safely be said that the functions stored in the DDEML are an indispensable prerequisite for all interactions between client and server

A DDE Server can support several data exchange formats. The default format is the CF_TEXT clipboard format which, at the same time, constitutes the minimum requirement.

The following discussion explains the operational principles governing the interactions between client and server. All message exchange or command transfer functions, as well as the message types themselves that effectively make up the commands (e.g. **XTYP_CONNECT**), are defined in the DDEML.

2.1.1 Setting up a Connection

Before a client can request data from a server, it must establish a connection with the same.



Connection Setup

The client sends the **XTYP_CONNECT** message to the server (via DDEML). The server initializes the connection and confirms the fault-free completion of the functional connection by returning the **DDE_FACK** signal.

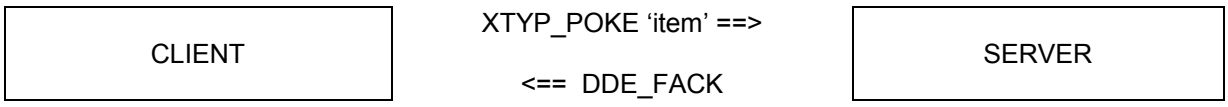
2.1.2 Static Data Exchange

The one-time data exchange between client and server is also known as a *cold link*.

There are two options for exchanging static data:

Option 1

The client transmits data to the server (e.g. interface parameters).

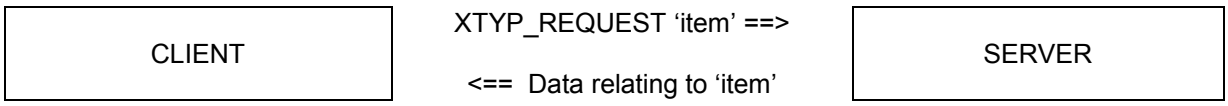


Static Data Exchange — Option 1

The client transmits, via the DDEML, the **XTYP_POKE** message, along with an identifier (the *item*) and the corresponding data, to the server. The item identifies the data type to the server. The server then sends the **DDE_FACK** message to acknowledge that it has received the data.

Option 2

The client requests from the server specific data on a one-time basis (e.g. kinematics information).

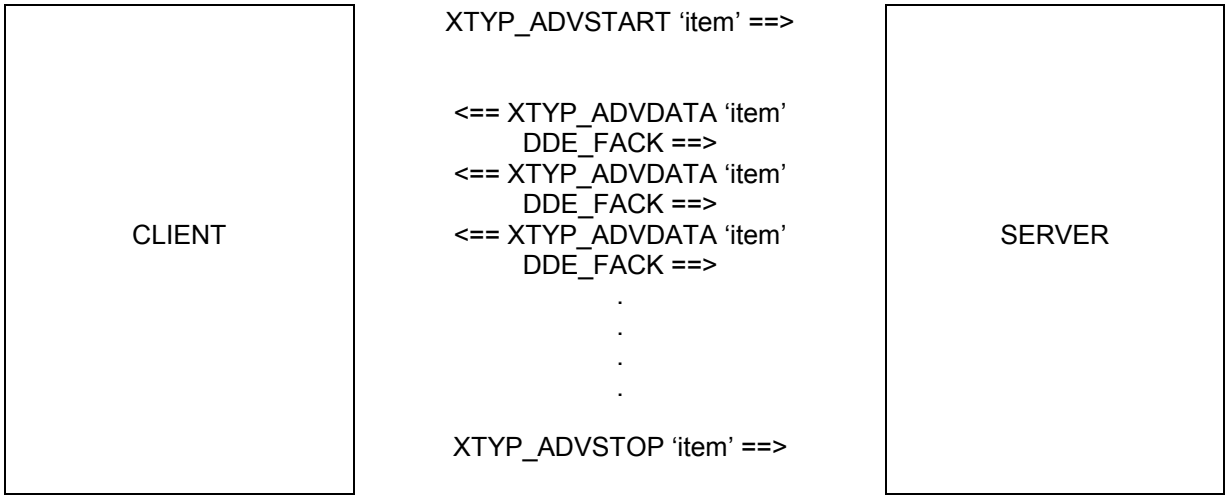


Static Data Exchange — Option 2

The client transmits, via the DDEML, the **XTYP_REQUEST** message, along with an identifier (the *item*), to the server. The item that is included in the transmission informs the server which data it is requested to send to the client.

2.1.3 Requesting Data Dynamically

For data that is subject to constant change, the client is able to establish a dynamic connection that is known as a *hot link*. The server will subsequently send its data in cyclical intervals. This process will continue until the client cancels the dynamic connection. To prevent unnecessary system loads, certain data is transmitted only if a change in data contents has occurred. An example of this transmission mode is the request for axis positions in ASCII code.



Dynamic Data Exchange

The client transmits, via the DDEML, the **XTYP_ADVSTART** message, along with an identifier (the *item*), to the server. The item that is included in the transmission informs the server which dynamic data it is requested to send to the client. The data is then transmitted to the client along with the **XTYP_ADVDATA** message. The client is then required to acknowledge the receipt by sending the **DDE_FACK** message. The **XTYP_ADVSTOP** message is used to terminate the dynamic data exchange.

2.1.4 Terminating a Connection

If a client no longer requires data from a server, it must again terminate the connection. Only in this manner will the interface initially occupied by a **CONNECT** command again be released.

The client transmits, via the DDEML, the **XTYP_DISCONNECT** message to the server. The server terminates the connection and releases the interface.

2.1.5 Conventions Used in this Manual

To simplify the visual interpretation of the bi-directional data exchange between client and server, the interactions and their attendant commands and/or messages are arranged in the form of tables.

Example of a DDE table:

Client	Message "Item"	Data	<=>	Server
Start cyclical status query	XTYP_ADVSTART "StFehler"	---	=>	
	TRUE	---	<=	Acknowledge command
continue until	XTYP_ADVDATA "StFehler"	StFehler	<=	Transmit data cyclically
Stop	DDE_FACK "StFehler"	---	=>	
Stop status	XTYP_ADVSTOP "StFehler"	---	=>	

Description of Client / Server Data Exchange

Description of table:

Column 1 (Client): Brief explanation of the DDE command from the client's viewpoint.

Column 2 (Command): DDE commands and possible "items."

Column 3 (Data): The names of structures which are used to facilitate the data exchange. An explanation of structures appears subsequent to the respective table.

(The associated "structs" and/or "defines" are located in the file named **Client.h** which is provided as part of the software complement. The enclosing quotation marks, "---", indicate no data is exchanged by means of the associated message.

Column 4 (<=>): Depicts the direction of data transfer:

=> Indicates client-to-server transfer.

<= Indicates server-to-client transfer.

Column 5 (Server): Brief explanation of the DDE command from the server's viewpoint.

3 Hardware and Software Requirements

The minimum requirements are listed below:

- BOSCH PG5 programming device or similar IBM AT-compatible PC.
- **386SX33 Mhz processor or better**
- 2 MByte RAM (4 MByte recommended)
- Hard disk
- 1 serial interface (16-byte FIFO recommended)
- Microsoft Windows v. 3.1, Windows 95 or Windows NT (version 3.5 or higher)

With a view to developing a client application, the user should possess solid skills with regard to programming Windows applications and the DDE interface. The creation of a client application will be greatly facilitated by the availability of suitable tools (i.e., InTouch, Visual Basic, Visual C, etc.). The compiler must be set to **ALIGNMENT2**. The timeout parameter required by several DDE functions must be set to 5 seconds.

Though the following bibliography listing is by no means exhaustive, the following reference works will, provide useful assistance with Windows and DDE programming:

For Microsoft Windows 3.1 / Windows for Workgroups 3.11:

Charles Petzold, Programming Windows Third Edition. Microsoft Press. ISBN number 1-55615-395-3.

For Windows 95:

Charles Petzold, Programming for Windows 95. Microsoft Press. ISBN number 1-55615-676-6.

It is instructive to note that the DDE Server supports in its services only filenames that are up to 8 characters in length.

4 Software Package Contents

The software for the DDE Server is provided on 2 diskettes.

Diskette 1 contains:

ERROR.H Possible server error messages

ERROR.TXT User-specific error messages (English).

FEHLER.TXT User-specific error messages (German).

ROPS3SVR.INI Initialization datafile.

ROPS3SVR.EXE. Executable server file.

CLIENT.H Include file containing all data structures and Defines utilized by the server.

README.WRI MS-WRITE document containing general information and description of licensing procedure.

FAX.WRI MS-WRITE document; blank order form.

CRYPSERV.EXE

CKLDRV.SYS

CKCONFIG.EXE and

SETUP_CK.EXE Files required by Windows NT

Diskette 2 contains:

SERVER_V.DOC Microsoft Word 6.0 document; detailed description of the server.

Sample programs for access to BAPS variables:

DDE_AC2.MDB Microsoft Access 2.0 sample file

DDE_EX5.XLS Microsoft Excel 5.0 sample file

DDE_WW6.DOC, and

DDE_WW6.DOT Microsoft WORD 6.0 files

DDEDEMO.QLL,

DDEDEMO.IRD,

DDEDEMO.PKT, and

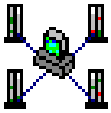
DDEDEMO.SYM BAPS programs for accessing BAPS variables.

The Online DDE Server is available in a German-language and English-language version. The desired language is determined by an entry in the file named ROPS3SVR.INI.

5 Software Protection

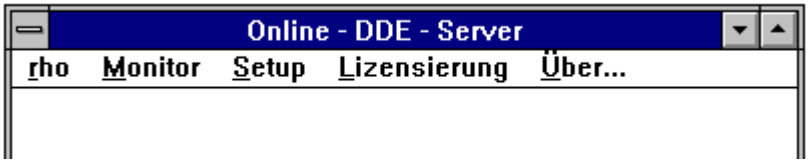
The DDE Server is protected by a software dongle. Subsequent to its installation, the Server must be enabled by entering a software key number (specified by Bosch). The procedure required for license application and actual licensing is described in the **README.WRI** file. An application form for the software key is provided in the **FAX.WRI** file.

6 DDE Server Operation



The Online DDE Server comprises a stand-alone Windows application. The Server does not feature an active user interface but is represented by an icon while running in the background.

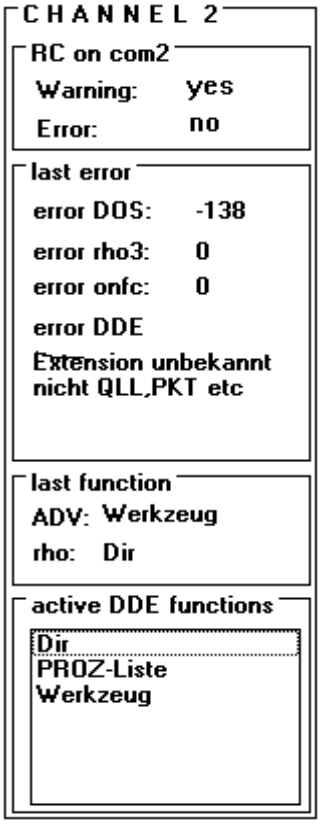
The Server menu is opened by double-clicking the Server icon. The menu contains all Server configuration and monitoring functions.



The menu provides the following functions:

rho This command displays the various versions of the control unit. This command is used for communications testing. Any errors that may occur will be displayed in the monitor. Prior to selecting the **rho** command for the first time, the interface parameters must be set up (see **Setup**).

Monitor This command is used to visualize the internal Server statuses. It displays a variety of information for each channel (see explanation).



Channel 2 is connected to Com2
Warnings in the rho 3.0
No errors in the rho 3.0

Last error (see also GStatus)
DOS Error number

Error text message, (unknown extension not QLL, PKT, etc.)

Last function:
Client / Server

Server rho 3.0

List of all active DDE functions of this channel

Description of Client / Server Data Communications

Setup This command is used for setting the communications and refresh rate parameters. The interface data entered here possess relevance only for the rho control version selected by means of the **rho** menu command. The interface parameters for server operation are set by means of the **InitUART** service (see below).

The refresh rate (expressed in ms) comprises the timing rate for all cyclical services provided by the server. This value is hardware-dependent. A fast refresh rate will translate into high system loads. The standard value is 500 ms (386-generation processor running at 66 MHz).

The data generated under the Setup menu command are saved in the .INI file.

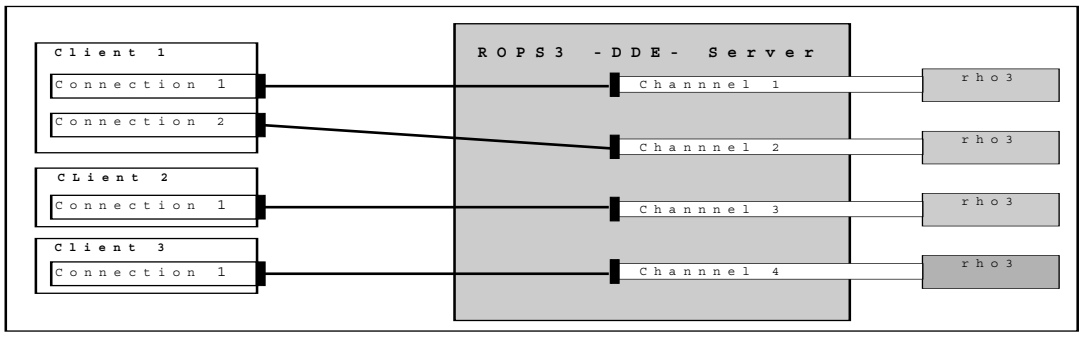
Lizensierung (Licensing) Licensing will be required subsequent to the successful installation of the server. As a consequence of the completed licensing procedure, the applicant becomes a Registered User who is deemed to have obtained the Online DDE Server by lawful means, authorizing him to work with the software. A detailed description of the installation and licensing procedures is provided in the **README.WRI** file.

Über (About ...) This command displays the software version of the server being used.

7 Setting Up a Connection

7.1 Connection between Client and Server

The Online DDE Server supports four serial interfaces (COM1- COM4). A connection between client and server is established by means of a **DDE Connect**. The parameters for the referred DDE Connect are comprised by the **ROPS3_SERVER** server name and the respective Topic name. As each interface is assigned one Topic, i.e., **Channel1** through **Channel4**, the server is able to maintain connections with four clients simultaneously. A client requiring connections to several controls must therefore execute several Connects.



DDE Server Channel Structure

Subsequent to **DDE Connect**, only 4 services per channel are available:

GStatus	global status
InitUART	interface initialization
Formats	List of formats (CF_TEXT only)
TopicItemList	List of all items **currently** available

Subsequent to **InitUART** (see Section 6.2, "Connection between PC and Control Unit"), all services are enabled for the selected channel:

Del	Deletes a file on the rho 3.0 Robot Control
Dir	Display rho 3.0 directory
UpLoad	Copies file/files from the rho 3.0 to the PC
DownLoad	Copies file/files from the PC to the rho3.0
Ren	Renames a file on the rho 3.0
ADVKinAchsen	Axis information, kinematics, **cyclical**
ADVGlobAchsen	Axis information, all axes, **cyclical**
Werkzeug	Tool, cyclical
RK_SYS	Space coordinate (SC) system, cyclical
ProzListe	List of all processes, **cyclical**
ProzStatus	Status of a single process, **cyclical**
Signale	Signal display, **cyclical**
FehlerFlag	rho3 error has occurred
Fehler_A	rho3 error / warnings
Control_Client	Client / Server control functions

Control_Server	Server / Client control functions
ServerFehler	Server fault / error
A1_POS	
A20_POS	Axis positions
A1_ENDPOS	
A20_ENDPOS	End positions of axes
A1_INPOS	
A20_INPOS	IN-position signals from axes
B1_POS	
B8_POS	Tape positions
TopicItemList	List of all items **currently** available
CloseUART	Closes the interface
GRDStellung	Home position, Robot Control
KinInfo	Kinematiks information, rho 3.0
KinAchsen	Axis information, kinematics
GlobAchsen	Axis information, all axes
Fehler	rho 3.0 error
ProzAnw	Selects a process
ProzStopp	Stops a process
SetRCA	Sets RCA signals 10.1 through 10.8
GStatus	Global status
VarRead1	
VarRead32	Reading user variables
VarWrite1	
Varwrite32	Writing user variables
VarRead1_A	
VarRead32_A	Reading user variables (ASCII protocol)
VarWrite1_A	
VarWrite32_A	Writing user variables (ASCII protocol)
Heartbeat	Control unit / PC connection monitoring

7.2 Connection between PC and Control Unit

In order to effect the exchange of data between control unit and server, initialization of the interface connecting the PC with the control unit is required. This can be accomplished in two ways:

- By invoking the **InitUART** server service, along with the associated parameters, OR
- in ROPS3SVR.INI file, by setting AUTOINIT=1 (see also Section 8, "ROPS3SVR.INI" and Section 9.1.9, "Automatic Initialization"). This initialization methods utilizes the parameters from the .INI file.

All server services will be available only subsequent to proper initialization.

Recommendation: Inadvertent interruptions of the connection between control unit and PC, e.g. through removal of the plug connection or through RC start-up during the data exchange, will disrupt the entire data exchange with the server. To facilitate the restoration of communications on the part of the server subsequent to correcting the malfunction, the **HeartBeat** monitoring function should always remain active (see Section 9.1.10).

8 Server Services

The server services are divided into four categories. These are discussed in the following sections.

8.1 File Management Functions

Del, Dir, UpLoad, DownLoad, Upload_A, Download_A, Ren

Only one of these 7 Items can be active (on each channel). As one function is initialized, the other four are deleted from the TopicItemList. Once the function has been completed, all Items are again added to the list.

In the event that cyclical services are found to be active, they will be halted for the time interval required by the file transfer function.

8.2 Cyclical Services

ADVKinAchsen, **ADVGlobAchsen**, **Werkzeug**, **RK_SYS**, **ProzListe**, **ProzStatus**, **Signale**, **FehlerFlag**, **Fehler_A**, **Control_Client, Control_Server, ServerFehler, A1_POS -A20_POS**, **A1_ENDPOS - A20_ENDPOS**, **A1_INPOS - A20_INPOS**, and **B1_POS - B8_POS**, **VarRead1 - VarRead32**, **VarWrite1 - VarWrite32**, **VarRead1_A - VarRead32_A**, **VarWrite1_A - VarWrite32_A**, and **Heartbeat**

The Server maintains an instruction list for each Channel. At the time of initialization, the cyclical services are inserted into this queue, and started by means of a timer. Each tick of the timer causes one instruction from the queue to be processed. The active functions alternate (*round robin* sequence). The referred timer can be set by means of the **Setup** menu command, using the **Taktrate** option (see also Section 5, "DDE Server Operation").

8.3 Non-cyclical Services

TopicItemList, **CloseUART**, **GRDStellung**, **KinInfo**, **KinAchsen**, **GlobAchsen**, **Fehler**, **ProzAnw**, **ProzStopp**, **SetRCA**, **FehlerFlag**, **Fehler_A**, **Control_Client**, **Control_Server**, **ServerFehler**, **A1_POS - A20_POS**, **A1_ENDPOS - A20_ENDPOS**, **A1_INPOS - A20_INPOS**, and **B1_POS - B8_POS**, **VarRead1 - VarRead32**, **VarWrite1 - VarWrite32**, **VarRead1_A - VarRead32_A**, and **VarWrite1_A - VarWrite32_A**

The above named functions can be invoked anytime while the server is ready to process a function, i.e., also in parallel to cyclical services.

8.4 Services with ASCII Protocol

Upload_A, **Download_A**, **FehlerFlag**, **Fehler_A, Control_Client**, **Control_Server**, **ServerFehler**, **A1_POS - A20_POS**, **A1_ENDPOS - A20_ENDPOS**, **A1_INPOS - A20_INPOS**, **B1_POS - B8_POS**, **VarRead1_A - VarRead32_A**, **VarWrite1_A -VarWrite32_A**, and **Heart-Beat**

These services communicate with the client via ASCII string.

8.5 GStatus Special Function

Each error that occurs, including any rho 3.0 error/warning, is entered into the GStatus of the respective channel. The internal errors (but not the rho 3.0 error/warnings) are subsequently reset.

Activating the **GStatus** function will now cause the record to be transferred to the client (see Section 9.1.1, "Global Status").

This service should always be active to enable error responses.

9 ROPS3SVR.INI File

The Online DDE Server utilizes an .INI file featuring the following contents:

[CHANNEL1]
COM=COM1
BAUD=9600
DATA=8
STOP=1
HANDSHAKE=1
TIMEOUT=2
ERRTIMEOUT=300
PARITY=N
[DEFEXTENSION]
EXT=.QLL,.IRD,.PKT,.SYM,.ERR,.ERB,.QLS,.TXT,.INC,.DAT
[SERVERINIT]
REFRESH=500
AUTOINIT=0
KOORDINATEN=1
Language=DEUTSCH
[ITEMLIMITS]
ASCIIITEMS=1
BINAERITEMS=1
READITEMS=32
WRITEITEMS=32
ASCIIACHSEN=20

Section	**Explanation**
CHANNELx	Interface data; one section per channel These entries are set up via the **Setup** menu command (see above). **Exception:** The entries COM, INIT, and ERRTIMEOUT must be edited directly in the ROPS3SVR.INI file.
Entry	**Explanation**
INIT	This entry is of importance only with the AUTOINIT=1 setting. If AUTOINIT is set to 1, all interfaces on which INIT is set to 1 are initialized automatically. In case of INIT=0, the respective channel will be skipped during automatic interface initialization.
COM	Assignment of physical interface to logical channel.
BAUD	Baudrate / transmission speed
DATA	Number of data bits
STOP	Number of stop bits
HANDSHAKE	0 = No hardware handshake 1 = Hardware handshake
TIMEOUT	Timeout interval in seconds with functioning connection

ERRTIMEOUT	Timeout interval (ms) in case of interrupted connection. A setting of 300 ms or higher is recommended (see also Section 9.1.10).
PARITY	Parity checking: N = No parity check E = Even parity O = Odd parity

[DEFEXTENSION] In the case of a file transfer using wildcard characters, only files corresponding to the "Ext=" setting of the .INI file will be selected. Files with the **.P2X** (PIC250 pgm.) and **.BIN** extensions (machine parameters) are never transferred when using wildcard characters for loading files.
This entry is missing in the factory-supplied version of the ROPS3SRV.INI file, and must be inserted manually if required.
If this entry is not contained in the ROPS3SRV.INI file, the filename extensions .QLL, .IRD, .PKT, .SYM, .ERR, :ERB, .QLS, .TXT, .INC, and .DAT will be used as defaults.

[SERVERINIT] Initialization data for the server.

Entry	**Explanation**
REFRESH	The parameters for this entry are set by means of the **Setup** menu command, using the **Taktrate** option (see above). The parameter value determines the transfer rate for cyclical data. (The parameter setting should exceed 200 ms.)
AUTOINIT	= 1 At the time the Connect command is invoked, the interface of this channel is initialized automatically. = 0 Auto-initialization OFF
KOORDINATEN	Selection of coordinates for axis or tape data to be transferred in ASCII form. = 0 Positions in the coordinate system that is currently enabled. = 1 Positions in machine coordinates = 2 Positions in solid coordinates = 3 Positions in datum coordinates,TO06x and up.
LANGUAGE	Selection of language version (German or English

[ITEMLIMITS] Limitation of server items in use. A limitation to the server items actually required can result in shorter server response times.

Entry	**Explanation**
ASCIIITEMS	0 = Items with ASCII protocol are disabled. 1 = Items with ASCII protocol are enabled.
BINAERITEMS	0 = Items with binary protocol are disabled. 1 = Items with binary protocol are enabled.
READITEMS	Number of items for reading user variables.
WRITEITEMS	Number of items for wrting user variables.
ASCIIACHSEN	Number of items for axis positions, In positions and End positions, transferred yia ASCII protocol.

10 List of Functions

The ROPS3 DDE Server provides three groups of functions. These comprise status, file transfer and online functions.

10.1 Status and Initialization Functions

The functions described below can be used to monitor the server and the connected control units, and to initialize the respective interfaces.

10.1.1 Global Status

The **GStatus** function is used to monitor the server as well as the connected control unit. The status may be subject to a one-time request or a cyclical request. This status record is also automatically included in each returned response record.

GStatus — One-time request:

Client	Message "Item"	Data	<=>	Server
Request status	XTYP_REQUEST "GStatus"	---	=>	
		TGSTATUS	<=	Send GStatus

GStatus — Cyclical request:

Client	Message "Item"	Data	<=>	Server
Start cyclical status query	XTYP_ADVSTART "GStatus"	---	=>	
	TRUE	---	<=	Acknowledgement
Continue until Stop	XTYP_ADVDATA "GStatus"	TGSTATUS	<=	Send GStatus
	DDE_FACK "GStatus	---	=>	
Stop Status	XTYP_ADVSTOP "GStatus	---	=>	

Start parameters
none

Return parameters

```
struct TGSTATUS
{
  int       nStWarnungen;
  int       nStFehler;
  int       nFehler;
  UINT      nLastDDEError;
  /*-------------------------*/
  UINT      f3Frei          :3;
  UINT      fDOSFehler      :1;
  UINT      frhoFehler      :1;
  UINT      fOnFktFehler    :1;
  UINT      f9Frei          :9;
  UINT      fServerStatus   :1;
```

```
  int       nFc;
  int       nState;
  char      szItem[50];
  WORD      wTransaction;
  WORD      wState;
}
```

Parameter	Description
nStWarnungen, nStFehler	Control status, read from the control unit with each online function; no update in case of basic functions.

Value	Explanation
-1	Undefined, control unit status is unknown
0	No warnings and/or errors
1	Warnings and/or errors have occurred in the control unit

nFehler	Error code; see **Error.h** error code file
nLastDDEError	Last DDE error; see **Error.h** error code file

Bit	Explanation
0-2	Not yet assigned
3	DOS error; see **nFehler**
4	rho3 error (during data transfer) see **nFehler**
5	Error of last online function
5-14	Not yet assigned
15	Server status = ready

nFc Indicates the online function last executed.

Value	Explanation
-1	Undefined
1	Dir (list directory)
2	Copy PC-> RC
3	Copy RC-> PC
4	Rename
5	Delete
1003	Search for process
1005	Search for next process
1007	Process selection
1010	KinX position
1011	Kinematics information
1013	Error
1016	Version
1022	Process stop
1023	Set RCA
1030	Signals
1031	rho3 position
1034	RC home position
1037	List processes
1042	Tool

nState Transaction status of item named "szItem"

Value	Explanation
0	Ready
1	Initialization
2	Running
3	Stop
4	Waiting for stop
5	Cancel

szItem Name of last item
wTransaction Last DDE command

The flags labelled *f3Frei* through *wState* are of significance only for diagnostic purposes; they will not be interpreted during standard operation.

Each error that occurs, including a rho3 error/warning, is entered in the GStatus of the respective channel. Once this is accomplished, the internal error is reset (but not the rho3 error/warnings).

With the **GStatus** function enabled, the server will now send the TGSTATUS record to the client.

10.1.2 Control and Monitoring Option for ASCII Services

These functions are used to control and monitor services that exchange their data via ASCII strings.

10.1.2.1 Control Server Service

The server can utilize the **Control_Server** service to report the status of other services to the client.

Control_Server — One-time request

Client	Message "Item"	Data	<=>	Server
Request Control_Server	XTYP_REQUEST "Control_Server"	---	=>	
		szServer-Control	<=	send Control_Server

Control_Server — Cyclical request

Client	Message "Item"	Data	<=>	Server
Start cyclical server control	XTYP_ADVSTART "Control_Server"	---	=>	
	TRUE	---	<=	Acknowledgement
continue until status stop	XTYP_ADVDATA "Control_Server"	szServer-Control		Send Control_Server
	DDE_FACK "Control_Server"	---	=>	
Stop status	XTYP_ADVSTOP "Control_Server"	---	<=	

Start parameters
no data

Return parameters
char szServerControl [_MAX_STRING];

Parameter	Description	
szServerControl	**Byte 1**	
	Bit 0	1 = Error/warning in rho control
	Bit 1	1 = Server error has occurred
	Bit 2	1 = UpLoad_A concluded
	Bit 3	1 = Download_A concluded
	Bits 4-7	Spare
	Bytes 2-4	Spare

The server provides control data only if changes occur. Bit 0 and 1 are preset with 0, and bits 2 and 3 with 1. An interface timeout will be recognized also if no service remains active.

10.1.2.2 Control Client Service

The client can utilize the **Control_Client** service for indirect manipulation of services that are active on the server.

Client	Message "Item"	Data	<=>	Server
Start cyclical client control	XTYP_ADVSTART "Control_Client"	---	=>	
	TRUE	---	<=	Acknowledgement
Control server services	XTYP_POKE "Control_Client"	szClient-Control	=>	
	DDE_FACK "Control_Client"	---	<=	Acknowledgement
Stop status	XTYP_ADVSTOP *) "Control_Client"	---	=>	

*) The use of XTYP_ADVSTART and/or XTYP_ADVSTOP is not mandatory.

Start parameters
char szClientControl [_MAX_STRING];

Parameter	**Description**
szClientControl	**Byte 1** Bit 0 1 = Abort Upload_A Bit 1 1 = Abort Download_A Bit 2 1 = Halt no. of axes/tape display (ASCII) 0 = Start no. of axes/tape display (ASCII) Bit 3 1 = Stop "Server Error" function 0 = Restart "Server Error" function Bit 4 1 = Halt "Fehler_A" function 0 = Restart "Fehler_A" function Bit 5 1 = Halt "FehlerFlag" function 0 = Restart "FehlerFlag" function Bit 6 1 = Halt cyclical reading of user data 0 = Restart cyclical reading, user data Bit 7 Spare **Bytes 2-4** Spare

All functions that can be disabled are initialized in their respective active states (bit = 0).

Note: The statuses of all bits are interpreted with each transmission to the server. The client itself is responsible for administering the statuses of disabled functions.

Return parameters
none

10.1.3 Server Error Function

This function is used for monitoring the server, as well as the DOS and online functions.

ServerFehler — One-time request

Client	Message "Item"	Data	<=>	Server
Request ServerFehler	XTYP_REQUEST "ServerFehler"	---	=>	
		szServerFehler	<=	send ServerFehler

ServerFehler — Cyclical request

Client	Message "Item"	Data	<=>	Server
Start cyclical status query	XTYP_ADVSTART "ServerFehler"	---	=>	
	TRUE	---	<=	Acknowledgement
continue until status stop	XTYP_ADVDATA "ServerFehler"	szServerFehler	<=	Send ServerFehler
	DDE_FACK "ServerFehler"	---	=>	
Stop status	XTYP_ADVSTOP "ServerFehler"	---	=>	

Start parameters
none

Return parameters
char szServerFehler [_MAX_STRING];

Parameter	Description
szServerFehler	Error code; ASCII string with concluding "\0"; The internal error is subsequently reset. The error code is listed in **Error.h** error code file. If no server error is present, the service will return "0\0".

The server provides control data only if changes occur.
The transmission of server errors can be temporarily halted by setting a control bit in the **Control_Client** function.

Note: The **ServerFehler** service neither requires data from the control unit, nor does it have access to the interface connecting the PC and control unit. For this reason, in the event that no service is active that requires this connection, a timeout of the interface cannot be recognized (however, refer also to Section 9.1.3, "*Control_Server* Service").

10.1.4 Signalling Control Unit Errors / Warnings

This function is used to monitor the connected control unit.

FehlerFlag — One-time request

Client	Message "Item"	Data	<=>	Server
Request FehlerFlag	XTYP_REQUEST "FehlerFlag"	---	=>	
		szFehlerFlag	<=	send FehlerFlag

FehlerFlag — Cyclical request

Client	Message "Item"	Data	<=>	Server
Start cyclical status query	XTYP_ADVSTART "FehlerFlag"	---	=>	
	TRUE	---	<=	Acknowledgement
continue until status stop	XTYP_ADVDATA "FehlerFlag"	szFehlerFlag	<=	Send FehlerFlag
	DDE_FACK "FehlerFlag"	---	=>	
Stop status	XTYP_ADVSTOP "FehlerFlag"	---	=>	

Start parameters
none

Return parameters
char szFehlerFlag [60];

Parameter	Description
szFehlerFlag	Control unit status; ASCII string with concluding "\0"; update occurs only if changes are detected.

Value	Explanation
0	No error and no warning has occurred
1	Errors and/or warnings are present

The server provides the **FehlerFlag** error flag signal only if changes occur.

The transmission of the control unit status can be temporarily halted by setting a control bit in the **Control_Client** function.

10.1.5 List of Control Errors / Warnings

This function returns errors and warnings relative to rho3.0 operations.

These functions encompass:

- The number of active (current) warnings.
- The number of active (current) errors.
- The associated error code.
- Error message text in ASCII format, including kinematics information and/or axis reference.

The client can determine whether to obtain the error message texts from the control unit or from an ASCII file. In the referred file, each error code is accompanied by an explanatory text. The file vcan be edited by the user. This provides the user with the option to generate his own error messages and supplementary information. In the English-language program version, the file is named **ERROR.TXT**.

The unaltered standard file contains the texts obtained from the signal description.

10.1.5.1 Syntax of ERROR.TXT File

The file is structured as follows:

No. = *Text* ; Text is copied to szFehMsg (TDDEFEHLER)
PHG display: '*Text*' ; Text is not copied
Ursache: Cause ; Text is copied to szUrsache (TDDEFEHLER)
Hinweis: Remedy ; Text is copied to szHinweis (TDDEFEHLER)

Example of entry in **ERROR.TXT** file:

1 = In Automatic: Programmed Kinematics in SETUP MODE
PHG display: 'In Handbetr. unzul.'
Cause: The referred kinematics are in SETUP MODE.
Hinweis: Switch to AUTOMATIC MODE.

Client	Message "Item"	Data	<=>	Server
Initialize error query	XTYP_POKE "Fehler"	nModus	=>	
	DDE_FACK "Fehler"	---	<=	Acknowledgement
Request error	XTYP_REQUEST "Fehler"	---	=>	
		TDDEFEHLER	<=	Send error

Start parameters

int nModus;

Parameter	Description
nModus	Display mode

Value	Explanation
0	Error texts from control unit
1	Error texts from the FEHLER.TXT file
2	Error texts from the ERROR.TXT file

Return parameters

```
struct TFEHLER
{
  TGSTATUS  GStatus;
  int       nAnzWarnungen;
  int       nAnzLaufzeitFehler;
  int       nAnzSonstigeFehler;
  int       nFehKode[_MAX_FEHLER];
  char      szFehMsg[_MAX_FEHLER][_MAX_FEH_LEN];
  char      szUrsache[_MAX_FEHLER][_MAX_FEH_LEN];
  char      szHinweis[_MAX_FEHLER][_MAX_FEH_LEN];
};
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
nAnzWarnungen	Number of warnings that have occurred in the control unit
nAnzLaufzeit	Number of runtime errors that have occurrred in the control unit
nAnzSonstige	Number of miscellaneous errors that have occurred in the control unit
nFehKode	Error codes and warning codes, identical to the signal description
*szFehMsg***[]**	Associated error message texts
*szUrsache***[]**	Associated texts from the error file; Mode 2/3 only
*szHinweis***[]**	Associated texts from the error file; Mode 2/3 only

10.1.6 List of Control Errors / Warnings in ASCII

This function returns the codes of all errors and warnings concerning the rho3 in the form of an ASCII string.

Fehler_A — One-time request

Client	Message "Item"	Data	<=>	Server
Request Fehler_A	XTYP_REQUEST "Fehler_A"	---	=>	
		szFehler	<=	Send Fehler_A

Fehler_A — Cyclical request

Client	Message "Item"	Data	<=>	Server
Start cyclical status query	XTYP_ADVSTART "Fehler_A"	---	=>	
	TRUE	---	<=	Acknowledgement
continue until status stop	XTYP_ADVDATA "Fehler_A"	szFehler	<=	Send Fehler_A
	DDE_FACK "Fehler_A"	---	=>	
Stop status	XTYP_ADVSTOP "Fehler_A"	---	=>	

Start parameters
none

Return parameters
char szFehler_A [_MAX_STRING]; "WarnKode,FehKode,...\0"

Parameter	Description
szFehler	Codes of warnings and errors, similar to signal description. If no errors are present, only "\0" will be transmitted.

The server provides the error codes only if changes occur.

The transmission of the error codes can be temporarily halted by setting a control bit in the **Control_Client** function.

10.1.7 Initializing an Interface

This function is used to initialize the interface, and to enable all items for this channel. The UART remains assigned until it is again closed, and cannot be used by any other application.

The standard interface parameters are as follows:

9600, N, 8, 1, hardware handshake, Timeout=8 sec.

In order to detect any errors that may have occurred during initialization, the actual interface status should be determined immediately following the initialization.

Client	Message "Item"	Data	<=>	Server
Initialize interface	XTYP_POKE "InitUART"	TUART	=>	
	DDE_FACK "InitUART"	---	<=	Acknowledgement
Request status	XTYP_REQUEST "GStatus"	---	=>	
		TGSTATUS	<=	Send GStatus

Start parameters

```
struct TUART
{
 int     nConNo;
 int     nBaud;
 char    cParity;
 int     nDatenBits;
 int     nStopBits;
 int     nHandShake;
 int     nTimeOut;
}
```

Parameter	Description
nComNo	Indicates the number of the interface (1-4).
nBaud	Baud rate (110, 300, 1200, 4800, 9600, 19200)
cParity	Parity (N, E, O)
nDatenBits	Data bits (7,8)
nStopBits	Stop bits (1,2)
nHandShake	Handshake 0= no handshake, 1= hardware handshake
nTimeOut	Timeout in seconds (1-99)

Return parameters
none

10.1.8 Closing an Interface

This function is used to close an interface, and to release the UART. At the same time, all cyclical functions of this topic, or channel, are deleted.

Subsequent to closing the interface, only four items remain that are available for this channel. They are **GStatus**, **InitUART**, **Formats**, and **TopicItemList**.

Client	Message "Item"	Data	<=>	Server
Reset Server	XTYP_POKE "CloseUART"	TCOMNO	=>	
	DDE_FACK "CloseUART"	---	<=	Acknowledge
Request status	XTYP_REQUEST "GStatus"	---	<=	
		TGSTATUS	<=	Send GStatus

Start parameters
int *nComNo*

Parameter	**Description**
nComNo	Indicates the number of the interface (1-4)

Return parameters
none

10.1.9 Automatic Initialization

Automatic initialization of the server interfaces can be preset by making specific changes to the ROPS3SVR.INI file.

Once the entry AUTOINIT=1 has been added to the [SERVERINIT] group, a CONNECT will cause the automatic initialization of the respective interface with the values belonging to the associated group ([CHANNEL1] .. [CHANNEL4]).

See also Section 9 ROPS3SVR.INI File

10.1.10 Monitoring the Control Unit <-> Server Connection

This function is used to monitor the connection between control unit and server. In the event that a data transmission error, such as SW Timeout, Overrun error or similar error, occurs in a cyclical service (with the exception of **HeartBeat** itself), the monitoring service returns a constantly incrementing counter value. Once the malfunction has been corrected, the service will return the one-time counter value of zero.

In order to facilitate the restoration of communications on the part of the server subsequent to correcting a malfunction (such as control start-up or disruption of the connection between robot control and PC), the **HeartBeat** monitoring function should always remain active.

As the HeartBeat function is enabled only in case of a malfunction, the service does not normally impose any load upon the runtime behaviour of the server.

To activate the HeartBeat function, proceed as follows:

Client	Message "Item"	Data	<=>	Server
Start cyclical monitoring	XTYP_ADVSTART "HeartBeat"	---	=>	
	TRUE	---	<=	Acknowledgement
Continue until status stop, or until malfunction remedied	XTYP_ADVDATA "HeartBeat"	szHeartBeat	<=	Send counter value
	DDE_FACK "HeartBeat"	---	=>	
Stop status	XTYP_ADVSTOP "HeartBeat"	---	=>	

Start parameters
none

Return parameters
char szHeartbeat [_MAX_STRING]; "Counter value*0*"

Parameter	Description
szHeartbeat	Counter value

Value	Explanation
0	The connection is functional
1	The connection is interrupted.

Note: In normal circumstances, the **HeartBeat** service neither requires data from the control unit, nor does it have access to the interface connecting the PC and control unit. For this reason, a timeout of the interface can only be recognized if another active cyclical service requires this connection. In the event of a malfunction, all services of the affected channel (with the exception of HeartBeat itself) will be terminated. To keep the system load of the PC at a minimum until the malfunction can be remedied, a special timeout value for malfunctions (ERRTIMEOUT) can be set in the ROPS3SVR.INI file.

See also Section 8, "ROPS3SVR.INI File."

10.2 File Transfer Functions

The DDE Server provides five commands for file handling functions:

Initializing one of these functions causes the simultaneous disablement of all file transfer functions for this topic, or channel. To an attempted initialization, the server will respond by returning the **DDE_FNOTPROCESSED** message.

In the event of a file transfer with the use of wildcard characters, only files that correspond to the "WildcardExt=" setting in the .INI file are selected (see also Section 7, "ROPS3SRV.INI File"). Filenames with the filename extensions .P2X (PIC250 programs) and .BIN (machine parameters) are automatically excluded from file transfers with wildcard characters.

10.2.1 Download Command

The client can utilize the **Download** command to load files into the control unit. To effect the initialization, the client transfers the filename to the server. The filename may contain wildcard characters. The file transfer is initiated by starting the cyclical query.

During the file transfer, the server reports after each packet of 200 bytes the total number of transferred bytes to the client. The completion of a transfer is signalled by sending "nStatus=2" to the client. If the transfer job encompasses several files, the next transfer is then started. The number of files remaining to be transferred is indicated in *dwCounter*.

At any time, the client has the option to abort the file transfer by sending the **XTYP_ADVSTOP** "**Download**" command.

If an error occurs during the downloading phase, this is indicated by means of "nStatus=-1", and the transfer job is interrupted.

Client		Message "Item"	Data	<=>	Server
Initialize transfer		XTYP_POKE "Download"	TCALLDOWNLOAD	=>	
		DDE_FACK "Download"	---	<=	Acknowledge
Start cyclical query		XTYP_ADVSTART "Down-Load"	---	=>	
		TRUE	---	<=	Acknowledge
	Continue until end of file, error or stop	XTYP_ADVDATA "Down-Load"	TUPLOADRET	<=	Transfer file(s)
		DDE_FACK "Download"	---	=>	
until	all files have been transferred				
Stop Transfer		XTYP_ADVSTOP "Download"	---	=>	

Start parameters

```
struct TCALLDOWNLOAD
{
char        szSRCName[_MAX_DOSNAME];
char        szDSTName[_MAX_RHONAME]
int         nUeberschreiben
}
```

Parameter	Description
szSRCName	Complete filename (hard disk, path, name, extension) of the file to be transferred. Name and extension may be substituted by wildcard characters ("*").
szDSTName	Control unit file subsequent to a download. Name and extension may be substituted by wildcard characters ("*"). Although the filename must not be the same as that in *szSRCName*, the filename extension must be identical.
nUeberschreiben	Overwrite rho file; this parameter may have one of two values:

Value	Explanation
0	The file is not overwritten. If the file is found to exist, the task is aborted.
1	The file is overwritten.

Return parameters

```
struct TUPLOADRET
{
 TGSTATUS   GStatus;
 char       szActName[_MAX_PATH];
 int        nStatus;
 DWORD      dwCounter;
 DWORD      dwAnzDat;
}
```

Parameter	Description
GStatus	Global status, see status functions
szActName	Name of rho3 control
nStatus	Transfer status; this parameter may have one of the following values:

Value	Explanation
0	File transfer is initialized; counter = file length
1	File transfer in progress; counter = number of transferred bytes
2	File transfer concluded; counter = file length
-1	Errors, see *GStatus*

dwCounter	Returns the number of transferred databytes
dwAnzDat	Returns the number of files remaining to be transferred, which in turn is derived from the wildcard characters. This counter is decremented after each file transfer.

See also Section 7, "ROPS3SVR.INI File."

10.2.2 ASCII Download Command

The **ASCII Download** function behaves exactly like the standard **Download** command described in the previous section, with the exception that the transfer parameters take the form of ASCII strings.

Download_A with download status message upon request:

Client	Message "Item"	Data	<=>	Server
Start download	XTYP_POKE "DownLoad_A"	szDownLoad	=>	
	DDE_FACK "DownLoad_A"	---	<=	Acknowledge
Request download status	XTYP_REQUEST "Down-Load_A"	---	=>	
		szDownLoadRet	<=	Send down-load status

DownLoad_A with cyclical download status message:

Client		Message "Item"	Data	<=>	Server
Initialize transfer		XTYP_ADVSTART "DownLoad_A"	---	=>	
		TRUE	---	<=	Acknowledge
Start cyclical query		XTYP_POKE "DownLoad_A"	szDownLoad	=>	
		DDE_FACK "DownLoad_A"	---	<=	Acknowledge
until	Continue until end of file, error or stop	XTYP_ADVDATA "Down-Load_A"	szDownLoadRet	<=	Transfer file(s)
			---	=>	
	all files have been transferred				
Stop Transfer		XTYP_ADVSTOP "Down-Load_A"	---	=>	

Start parameters
char szDownLoad [_MAX_STRING]; *"SourceName, DestName, ü\0"*

Component	Description
SourceName	Complete filename (hard disk, path, name, extension) of the file to be transferred. Name and extension may be substituted by wildcard characters ("*").
DestName	Control unit file subsequent to a download. Name and extension may be substituted by wildcard characters ("*"). Although SourceName and DestName can be different, the filename extension must be identical.
ü	Overwrite rho file; this parameter may have one of two values:

Value	Explanation
0	The file is not overwritten. If the file is found to exist, the task is aborted.
1	The file is overwritten.

The three components are separated by commas.

Return parameters

char szDownLoadRet[_MAX_STRING];*"DestName,Status, Counter,AnzDat\0"*

Component	Description
DestName	Control filename during download process.
Status	Transfer status; this parameter may have one of the following values:

Value	Explanation
0	File transfer is initialized; counter = file length
1	File transfer in progress; counter = number of transferred bytes
2	File transfer concluded; counter = file length
-1	Errors, see *ServerFehler*

Counter	Returns the number of transferred databytes (see status).
AnzDat	Returns the number of files remaining to be transferred, which in turn is derived from the wildcard characters. This counter is decremented after each file transfer.

See also Section 7, "ROPS3SVR.INI File."

Monitoring or termination of the function can be accomplished by means of the **Control_Client** function. Errors that have occurred are returned by the **ServerFehler** function.

10.2.3 Upload Command

The client can utilize the **Upload** command to load files from the control unit into the PC.

To effect the initialization, the client transfers the filename to the server. The filename may contain wildcard characters.

The file transfer is initiated by starting the cyclical query.

During the file transfer, the server reports after each packet of 200 bytes the total number of transferred bytes to the client. The completion of a transfer is signalled by sending "nStatus=2" to the client. If the transfer job encompasses several files, the next transfer is then started. The number of files remaining to be transferred is indicated in *dwCounter*.

At any time, the client has the option to abort the file transfer by sending the XTYP_ADVSTOP "Upload" command.

If an error occurs during the downloading phase, this is indicated by means of "nStatus=-1".

Client		Message "Item"	Data	<=>	Server
Initialize transfer		XTYP_POKE "Upload"	TCALLUPLOAD	=>	
		DDE_FACK "Upload"	---	<=	Acknowledge
Start cyclical query		XTYP_ADVSTART "UpLoad"	---	=>	
		TRUE	---	<=	Acknowledge
	Continue until end of file, error or stop	XTYP_ADVDATA "UpLoad"	TUPLOADRET	<=	Transfer file(s)
		DDE_FACK "Upload"	---	=>	
until	all files have been transferred				
Stop Transfer		XTYP_ADVSTOP "Upload"	---	=>	

Start parameters

```
TCALLUPLOAD
{
 char       szSRCName[_MAX_PATH];
 char       szDSTName[_MAX_RHONAME];
 int        nUeberschreiben
}
```

Parameter	Description
szSRCName	Control unit filename, name and extension can be substituted by wildchard characters ("*"). While the filename must not be identical to that in *szSRCName*, the extension must be identical.
szDSTName	Complete filename (drive, path, name, extension) of the PC file. Name and extension may be substituted by wildcard characters ("*").
nUeberschreiben	Overwrite PC file. This parameter can have one of the following values:

Value	Explanation
0	File is not overwritten. If the file is found to exist, the process is aborted.
1	File is overwritten.

Return parameters

```
struct TUPLOADRET
{
 TGSTATUS   GStatus;
 char       szActName[_MAX_PATH];
 int        nStatus;
 DWORD      dwCounter;
 DWORD      dwAnzDat;
}
```

Parameter	Description
GStatus	Global status, see "Status Functions"
szActName	der rho3-Name
nStatus	Transfer status; This parameter can have one of the following values:

Value	Explanation
0	File transfer initalized and running. Counter = length of file
1	File transfer in progress. Counter = Number of bytes transferred
2	File transfer completed. Counter = file length
-1	Errors, see *GStatus*

dwCounter	Returns the number of bytes transferred, see nStatus.
dwAnzDat	Returns the number of files remaining to be transferred, derived from wildcard characters. This counter is decremented with each file transfer.

See also Section 9 ROPS3SVR.INI File

10.2.4 ASCII Upload Command

The **ASCII Upload** function behaves exactly like the standard **Upload** command described in the previous section, with the exception that the transfer parameters take the form of ASCII strings.

UpLoad_A with download status message upon request:

Client	Message "Item"	Data	<=>	Server
Start upload	XTYP_POKE "UpLoad_A"	szUpLoad	=>	
	DDE_FACK "UpLoad_A"	---	<=	Acknowledge
Request upload status	XTYP_REQUEST "UpLoad_A"	---	=>	
		szUpLoadRet	<=	Send upload status

UpLoad_A with cyclical download status message:

Client		Message "Item"	Data	<=>	Server
Initialize transfer		XTYP_ADVSTART "UpLoad_A"	---	=>	
		TRUE	---	<=	Acknowledge
Start cyclical query		XTYP_POKE "UpLoad_A"	szUpLoad	=>	
		DDE_FACK "UpLoad_A"	---	<=	Acknowledge
	Continue until end of file, error or stop	XTYP_ADVDATA "UpLoad_A"	szUpLoadRet	<=	Transfer file(s)
			---	=>	
until	all files have been transferred				
Stop Transfer		XTYP_ADVSTOP "UpLoad_A"	---	=>	

Start parameters
char szUpLoad [_MAX_STRING]; *"SourceName, DestName, ü\0"*

Component	Description
SourceName	Control unit filename for upload. Name and extension may be substituted by wildcard characters ("*").
DestName	Complete filename (drive, path, name, extension) odf file to be transferred. Name and extension may be substituted by wildcard characters ("*").
ü	Overwrite PC file; this parameter may have one of two values:

Value	Explanation
0	The file is not overwritten. If the file is found to exist, the task is aborted.
1	The file is overwritten.

The three components are separated by commas.

Return parameters
char szDownLoadRet[_MAX_STRING];*"DestName,Status, Counter,AnzDat\0"*

Component	Description
DestName	Control filename during upload process.
Status	Transfer status; this parameter may have one of the following values:

Value	Explanation
0	File transfer is initialized; counter = file length
1	File transfer in progress; counter = number of bytes transferred
2	File transfer concluded; counter = file length
-1	Errors, see *ServerFehler*

Counter	Returns the number of transferred databytes (see *Status*).
AnzDat	Returns the number of files remaining to be transferred, which in turn is derived from the wildcard characters. This counter is decremented after each file transfer.

See also Section 9, "ROPS3SVR.INI File."

Monitoring or termination of the function can be accomplished by means of the **Control_Client** function. Errors that have occurred are returned by the **ServerFehler** function.

10.2.5 Directory Command

The **Directory** command returns a listing of the control unit files.

To start initialization, the filename is transferred. Wildcards are supported. The client then starts the directory transfer.

The server first sends the control software version identifier, followed by the filenames, including file length and date of last modification. The list is followed by the number of files and the storage capacity they occupy. The last item returned is the size of both available and occupied storage capacity. The client can cancel the function at any time.

Client	Message "Item"	Data	<=>	Server
Initialize Dir query	XTYP_POKE "Dir"	szDirName	=>	
	DDE_FACK "Dir"	---	<=	Acknowledge
Start cyclical query	XTYP_ADVSTART "Dir"	---	=>	
	TRUE	---	<=	Acknowledge
continue until Dir transferred, or Stop	XTYP_ADVDATA "Dir"	TRHO3DIR	<=	Send Dir
	DDE_FACK "Dir"	---	=>	
Stop Dir	XTYP_ADVSTOP "Dir"	---	=>	

Start parameters

char *szDirName*[_MAX_RHONAME];

Parameter	Description
szDirName	Control unit filename, name and extension can be substitued by wildcard characters ("*").

Return parameters

```
struct TDIR
{
 GSTATUS  GStatus;
 int      nStatus;
 char     szData[_MAX_RHO3_DIR];
}
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
nStatus	Dir status; this parameter can have one of the following values:

Value	Explanation
1	szData contains the sofware version ID and the control unit date
2	szData contains a filename
3	szData contains the .P2X filename; length in words
4	szData contains the number of files
5	szData contains the memory contents; end of transfer
-1	Error, see *GStatus*

szData Zero-terminated ASCII string. This parameter can have one of the following contents:

Contents	Format (with example)
	"12345678901234567890123456789012345678901234567890123456789012345678"
SW-Version	"rho3 : TO06F 03.04.1995"
File	" WERKZ .IRD 1012 29.03.95 08:44"
P2X File	" PIC200 .P2X 688 03.04.95 13:10"
No. bytes	" 1 file occupied 1012 Byte."
Memory capy.	" 122880 bytes of 124160 available."

10.2.6 Rename Command

The **Rename** command can be used to rename a file in the rho3. The function **does not** support wildcard characters. The .P2X file **cannot** be renamed with the use of this function.

To ensure the detection of errors that may have occurred as a result of a **Rename** action, the current status should be determined subsequent to executing the command (using **GStatus** or **ServerFehler** functions).

Client	Message "Item"	Data	<=>	Server
Initialize Ren query	XTYP_POKE "Ren"	TREN	=>	
	DDE_FACK "Ren"	---	<=	Acknowledge
Request status	XTYP_REQUEST "GStatus"	---	=>	
		TGSTATUS	<=	Send GStatus

Start parameters

```
struct TREN
{
 char      szOldNAme[_MAX_RHONAME];
 char      szNewNAme[_MAX_RHONAME];
 int       nUeberschreiben;
}
```

Parameter	Description
szOldName	*Old* name of control unit file
szNewName	*New* name of control unit file
nUeberschreiben	Overwrite rho3 file. This parameter may have one of the following values:

Value	Explanation
0	The file is not overwritten. If the file is found to exist, the process is aborted.
1	File is overwritten.

Return parameters

none

10.2.7 Delete Command

The **Delete** command is used to delete a control unit file. Wildcard characters are supported. Subsequent to initialization and the start of the cyclical query, the server reports all deleted files to the client. The task can be aborted at any time. The **Delete** command **cannot** be used to delete the .P2X file.

Client	Message "Item"	Data	<=>	Server
Initialize Del command	XTYP_POKE "Del"	szDelName	=>	
	DDE_FACK "Del"	---	<=	Acknowledge
Start cyclical query	XTYP_ADVSTART "Del"	---	=>	
	TRUE	---	<=	Acknowledge
continue until all files have been deleted	XTYP_ADVDATA "Del"	TDEL	<=	Send Delete response
	DDE_FACK "Del"	---	=>	
Stop Delete	XTYP_ADVSTOP "Del"	---	=>	

Start parameters

char *szDelName[*_MAX_RHONAME*];*

Parameter	Description
szDelName	Name of control unit file; name and extension can be substituted by wildcard characters("*").

Return parameters

```
struct TDEL
{
 TGSTATUS GStatus;
 int          nAnzDateienl
 char         szActName[_MAX_RHONAME];
}
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
nAnzDateien	Number of files remaining to be deleted.
szActName	Name name of last deleted control unit file

10.3 Online Functions

Online functions are used for visualizing control unit statuses, and for remote control purposes. The online function commands are available, effective with control unit version TO04x.

10.3.1 Kinematics Information

The **KinInfo** kinematics information command returns information about all kinematics that are applied in the control unit.

Client	Message "Item"	Data	<=>	Server
Request kinematics information	XTYP_REQUEST"KinInfo"	---	=>	send KinInfo
		TDDEKININFO	<=	

Start parameters
none

Return parameters

```
struct TDDEKINDATA
{
  char      szKinName[_MAX_KINNAME];
  int       nReferenz;
  int       nAchsAnzahl;
  int       nBandAnzahl;
};

struct TDDEKININFO
{
 TGSTATUS   GStatus;
 int        nKinAnzahl;
 TDDEKINDATA KinArray[_MAX_KIN];
};
```

Parameter	Description
TGStatus	Global status, see Section 9.1, "Status and Initialization Functions."
nKinAnzahl	Number of applied kinematics
TDDEKINDATA:	
szKinName	Name of individual kinematics
nReferenz	Indicates whether this kinematic has referenced (TRUE/FALSE)
nAchsAnzahl	Number of axis, this kinematic
nBandAnzahl	Number of tapes, this kinematic

10.3.2 Axis Positions

The **KinAchsen** axis position command can be used to request the axis and tape data from the control unit. The data can be requested per individual kinematics (KinAchsen) or globally across all kinematics (GlobAchsen).

There are two methods of execution for the Client.

10.3.2.1 Client Requires Data Only Once

The client requests the server to provide the data. For intilaization purposes, it transfers the TACHSINFO record. The record describes which axes and which tapes are to be sent, and in what sequence they are to be sent.
The axis data can then be requested.

Axis information, per individual kinematics:

Client	Message "Item"	Data	<=>	Server
Request axis information for individual kinematics	XTYP_POKE "KinAchsen"	TACHSINFO	=>	
	DDE_FACK "KinAchsen"	---	<=	Acknowledge
	XTYP_REQUEST"AchsData"	---	=>	Send KinAchsen
		TACHSDATEN	<=	

Cross-kinematics (global) axis information:

Client	Message "Item"	Data	<=>	Server
Request cross-kinematics (global) axis information	XTYP_POKE "GlobAchsen"	TACHSINFO	=>	
	DDE_FACK "GlobAchsen"	---	<=	Acknowledge
	XTYP_REQUEST"AchsData"	---	<=	
		TACHSDATEN	=>	Send GlobAchsen

10.3.2.2 Polling Axis Data

The client initializes the cycle by sending the TACHSINFO record. The cycle is then started. The server will now continue to supply axis data until the client terminates the request by sending **Stop**. The client has the option to stop the polling cycle in order to start a file transfer, for example.

Client	Message "Item"	Data	<=>	Server
Initialize	XTYP_POKE "ADVKinAchsen"	TACHSINFO	=>	
	DDE_FACK "ADVKinAchsen"	---	<=	Acknowledge
Start cycle	XTYP_ADVSTART "ADVKinAchsen"	---	=>	
	TRUE	---	<=	Acknowledge
contuinue until Stop	XTYP_ADVDATA "ADVKinAchsen"	TACHSDATEN	<=	ADVKinAchsen senden
	DDE_FACK "ADVKinAchsen"	---	=>	
Stop	XTYP_ADVSTOP"ADVKinAchsen"	---	=>	

The application of the cross-kinematics command is identical.

Start parameters

```
struct TACHSINFO
{
  int       nFc;
  int       nKinNr;
  int       nKoord;
  int       nAchsAnfang;
  int       nAchsAnz;
  int       nBandAnfang;
  int       nBandAnz;
};
```

Parameter	Description
nFc	Determines the subfunction. This parameter can have one of the following values:

Value	Explanation
OM_BAND	Returns the tape position
OM_NAME	Returns SC or MC names
OM_STAPOSBND	Returns axis positions, tracking, end point, in-pos flag, RK, referenced, auto.
OM_STAPOSBND	Returns OM_STAPOS + tape position

nKinNr	Number of kinematics with **KinAchsen** item, otherwise not defined.
nKoord	Defines the desired coordinate system. This parameter can have one of the following values:

Value	Explanation
AUTO_SYS	Returns the axis positions in the currently active coordinate system.
MK_SYS	Returns the axis positions in machine coordinates (MC).
RK_SYS	Returns the axis positions in space coordinates (SC).
UK_SYS	Returns the base coordinates, TO06x & up
nAchsAnf	Defines the first axis.
nAchsAnz	Defines the number of desired axes.
nBandAnf	Defines the first tape.
nBandAnz	Defines the number of desired tapes.

Return parameters

```
struct TACHSDATEN
{
 TGASTAUS  GStatus;
 int       nAchsAnz;
 char      aszName [_MAX_ACHS][_MAX_ACHSNAME];
 int       nKoord;
 int       nInPos [_MAX_ACHS];
 int       nReferiert [_MAX_ACHS];
 int       nAutoHand [_MAX_ACHS];
 float     AchsPos [_MAX_ACHS];
 float     EndPos [_MAX_ACHS];
 float     NachPos [_MAX_ACHS];
 int       nBandAnz;
 char      szBandName [_MAX_BAND][_MAX_BNDNAME];
 float     BandPos [_MAX_BAND];
};
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
aszName	Coordinate names and/or axis names
nKoord	Axis position coordinate system (SC, MK, UK)
nInPos	In-pos flag, indicates whether the axis is IN POSITION.
nReferiert	Indicates if this axis has referenced.
nAutoHand	Indicates whether the kinematics associated with this axis are in automatic or manual mode.
AchsPos	Indicates the current axis position in SC or MC. This value is invalid for non-referenced SC axes.
EndPos	Returns the programmed end position. This value is invalid in manual mode.
NachPos	Returns the nominal tracking value
nBandAnz	Number of tapes
szBandName	Tape name set up similar to MPP
BandPos	Current tape position

10.3.3 Axis Data in ASCII

This function can be used to request the axis and tape data from the control unit. The axis position, end point and InPos flag can be determined for up to 20 axes.

One-time request for axis and/or tape data:

Client	Message "Item"	Data	<=>	Server
Request axis / tape data	XTYP_REQUEST "item"	---	=>	
		szAchsDaten	<=	Send axis/tape data

Cyclical request for axis and/or tape data:

Client	Message "Item"	Data	<=>	Server
Start cycle	XTYP_ADVSTART "item"	---	=>	
	TRUE	---	<=	Acknowledge
contuinue until Stop	XTYP_ADVDATA "item"	szAchsDaten	<=	Send axis data
	DDE_FACK "item"	---	=>	
Stop	XTYP_ADVSTOP"item"	---	=>	

Items:

A1_POS	.. A20_POS	Requesting axis positions
A1_ENDPOS	.. A20_ENDPOS	Requesting axis end positions (useful only in Automatic mode)
A1_INPOS	.. A20_INPOS	InPos flag; indicates whether the axis is in position.
B1_POS	.. B8_POS	Requesting tape positions

(The number of items can be limited in the ROPS3SVR.INI file.)

Start parameters
none

Return parameters
char szAchsDaten[60] "+123456.78\0"
The server supplies axis data only if the data has been changed. The cyclical display of data is interrupted by file transfer functions (Upload, Download, etc.) The axis data is provided in the "6.2" format used by the control unit.

The transmission of axis and/or tape data can be temporarily halted by setting a control bit in the **Control_Client** function.

Notes: In the event that the server recognizes an error (e.g. invalid number of axes), all items of the channel, carrying axis or tape information will be closed.

The coordinate system for axis data can be selected in the ROPS3SVER.INI file ([SERVERINIT] - KOORDINATEN).

10.3.4 Tool

The **Tool** function provides a cyclical return of tool name and tool coordinates (TO06x and up).

Client	Message "Item"	Data	<=>	Server
Initialize tool query	XTYP_POKE "Werkzeug"	nKinNr	=>	
	DDE_FACK "Werkzeug"	---	<=	Acknowledge
Start cyclical query	XTYP_ADVSTART "Werkzeug"	---	=>	
	TRUE	---	<=	Acknowledge
continue until Stop	XTYP_ADVDATA "Werkzeug"	TWERKZEUG	<=	Send tool
	DDE_FACK "Werkzeug"	---	=>	
Stop tool function	XTYP_ADVSTOP "Werkzeug"	---	=>	

Start parameters
int nKinNr;

Parameter	Description
nKinNr	Number of kinematics, the tool of which is to be determined.

Return parameters

```
struct TWERKZEUG
{
 TGSTATUS GStatus;
 char     szWerkName[_MAX_WERKNAME];
 float    Value[_MAX_VAL];
};
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
szWerkName	Name of the currently selected tool for these kinematics
Value[]	Gripper X, Gripper-Y, Gripper Z, Gripper orientation1, Gripper orientation2, Gripper orientation3

10.3.5 SC System

The SC System function (**RK_Sys** command) provides a cyclical return of the SC system (TO06x and up).

Client	Message "Item"	Data	<=>	Server
Initialize SC system	XTYP_POKE "RK_Sys"	nKinNr	=>	
	DDE_FACK "Rk_Sys"	---	<=	Acknowledge
Start cyclical query	XTYP_ADVSTART "RK_Sys"	---	=>	
	TRUE	---	<=	Acknowledge
Continue until Stop	XTYP_ADVDATA "RK_Sys"	TRK_SYSTEM	<=	Senden RK_Sys
	DDE_FACK "RK_Sys"	---	=>	
Stop SC system	XTYP_ADVSTOP "RK_Sys"	---	=>	

Start parameters
int *nKinNr*;

Parameter	**Description**
nKinNr	Number of kinematics, the SC system of which is to be determined.

Return parameters

```
struct TRK_SYSTEM
{
 TGSTATUS GStatus;
 float    Value [_MAX_VALUE];
}
```

Parameter	**Description**
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
Value[]	Shifting the SC in X-direction, Shifting the SC in Y-direction, Shifting the SC in Z-direction, Rotating a about X, Rotating b about Y, Rotating c about Z.

10.3.6 Process Selection

The Process Selection function (**ProzAnw** command) is used to select a process within the control unit.

Client	Message "Item"	Data	<=>	Server
Initialize process selection	XTYP_POKE "ProzAnw"	TPROZANW	=>	
	DDE_FACK "ProzAnw"	---	<=	Acknowledge
Select process	XTYP_REQUEST "ProzAnw"	---	=>	
		TPROZSTATUS	<=	Send ProzAnw

Start parameters

```
struct TPROZANW
{
  char szrhoName[_MAX_RHONAME];
  int          nPrio;
};
```

Parameter	Description
szrhoName	Steuerungsdateiname
nPrio	Priorität des Prozesses

Return parameters

```
struct TPROZSTATUS
{
 TGSTATUS GStatus;
 int          nProzFound;
 char         szProzName[_MAX_RHONAME];
 int          nProzArt;
 int          nAnzSubProz;
 int          nProzPrio;
 int          nProzZustand;
 long         ProzFehler;
 char         szFehlerText[_MAX_FEHLEN];
 int          nProzZeile;
 int          nProzSubZeile;
 int          nProzKin;
 int          nProzEbene;
 char         szHPName[_MAX_RHONAME];
};
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
nProzFound	Indicates whether the requested process is available (TRUE/FALSE)
szProzName	Name of process
nProzArt	Indicates the process type. This parameter can have one of the following values:

Value	Explanation
0	Standard process
1	Permanent process
2	Subprocess

nAnzSubProz	returns the number of subprocesses within this main process.
nProzPrio	Indicates the process priority.
nProzZustand	Indicates the process status. This parameter can have one of the following values:

Value	Explanation
0	Process in standby
1	Process ready
2	Process haltet
3	Process running
6	Process jogging
7	Process haltet by error

ProzFehler	Process error, see error list
szFehlerText	Error message in ASCII text
nProzZeile	Indicates currently active QLL line
nProzSubZeile	Indicates QLL line of insertion file
nProzKin	Active kinematics of this process
nProzEbene	Main program level
szHPName	External main program

10.3.7 Process Stop

This function (**ProzStop** command) can be used to stop a process in the control unit.

In order to detect any errors that may have occurred during the execution of this command, the actual interface status should be determined immediately following the initialization.

Client	Message "Item"	Data	<=>	Server
Process Stop	XTYP_POKE "ProzStopp"	szProzName	=>	
	DDE_FACK "ProzStopp"	---	<=	Acknowledge
Request status	XTYP_REQUEST "GStatus"	---	<=	
		TGSTATUS	<=	Send ProzStopp

Start parameters
char szProzName[_MAX_RHONAME];

Parameter	**Description**
szProzName	Name of a main process

Return parameters
none

10.3.8 Process List

This function (**ProzListe** command) dynamically supplies the list of all processes.

Client	Message "Item"	Data	<=>	Server
Start cyclical query	XTYP_ADVSTART "ProzListe"	---	=>	
	TRUE	---	<=	Acknowledge
Contunie until Stop	XTYP_ADVDATA "ProzListe"	TDDEPROZLISTE	<=	Send ProzListe
	DDE_FACK "ProzListe"	---	=>	
Stop Process List	XTYP_ADVSTOP "ProzListe"	---	=>	

Start parameters
keine

Return parameters

```
struct TPARRAY
{
 char            szProzName[_MAX_RHONAME];
 unsigned char   ProzZustand;
 int             nQLLZeile;
};

struct TPROZLISTE
{
 TGSTATUS GStatus;
 int             nAnzPerm;
 int             nAnzNorm;
 int             nAnzSub;
 int             nAnzErr;
 TPARRAY         ProzArray[_MAX_PROZ];
};
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
nAnzPerm	Number of permanent processes
nAnzNorm	Number of standard processes
nAnzSub	Number of subprocesses
nAnzErr	Number of errored processes
szProzName	Process name; main processes are identified by .IRD extension. Associated subprocesses have the same name, and .Sxx extension, where xx is the number of the subprocess.
ProzZustand	Indicates the process status. This parameter can have one of the following values:

Value	Explanation
0	Process in standby
1	Process ready
2	Process haltet
3	Process running
6	Process jogging
7	Process haltet by error

nQLLZeile Indicates the QLL line that is currently active.

10.3.9 Process Status

This function (**ProzStatus** command) cyclically supplies the status of a process.

Client	Message "Item"	Data	<=>	Server
Initialize request	XTYP_POKE "ProzStatus"	szProzName	=>	
	DDE_FACK "ProzStatus"	---	<=	Acknowledge
Start cyclical request	XTYP_ADVSTART "ProzStatus"	---	=>	
	TRUE	---	<=	Acknowledge
Continue until Stop	XTYP_ADVDATA "ProzStatus"	TPROZSTATUS	<=	Send ProzStatus
	DDE_FACK "ProzStatus"	---	=>	
Stop Status	XTYP_ADVSTOP "ProzStatus"	---	=>	

Start parameters

```
char   szProzName[_MAX_RHONAME];
```

Parameter Description

szProzName Name of process

Return parameters

```
struct TPROZSTATUS
{
 TGSTATUS GStatus;
 int          nProzFound;
 char         szProzName[_MAX_RHONAME];
 int          nProzArt;
 int          nAnzSubProz;
 int          nProzPrio;
 int          nProzZustand;
 long         ProzFehler;
 char         szFehlerText[_MAX_FEHLEN];
 int          nProzZeile;
 int          nProzSubZeile;
 int          nProzKin;
 int          nProzEbene;
 char         szHPName[_MAX_RHONAME];
};
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
nProzFound	Indicates whether the requested process is available (TRUE/FALSE)
szProzName	Name of process
nProzArt	Indicates the process type. This parameter can have one of the following values:

Value	Explanation
0	Standard process
1	Permanent process
2	Subprocess

nAnzSubProz	Returns the number of subprocesses of this main process.
nProzPrio	Indicates the process priority.
nProzZustand	Indicates the process status. This parameter can have one of the following values:

Value	Explanation
0	Process in standby
1	Process ready
2	Process haltet
3	Process running
6	Process jogging
7	Process haltet by error

ProzFehler	Process error, see error list
szFehlerText	Error message in ASCII text
nProzZeile	Indicates currently active QLL line
nProzSubZeile	Indicates QLL line of insertion file
nProzKin	Active kinematics of this process
nProzEbene	Main program level
szHPName	External main program

10.3.10 Control Reset Command

This function can be used to issue the **Control Reset** command.

Client	Message "Item"	Data	<=>	Server
Execute Control Reset	XTYP_REQUEST "GRDStellung"	--- TGSTATUS	=> <=	Send GStatus

Start parameters
none

Return parameters

```
struct TGSTATUS
{
  int       nStWarnungen;
  int       nStFehler;
  int       nFehler;
  UINT      nLastDDEError;
  /*-------------------------*/
  UINT      f3Frei          :3;
  UINT      fDOSFehler      :1;
  UINT      frhoFehler      :1;
  UINT      fOnFktFehler    :1;
  UINT      f9Frei          :9;
  UINT      fServerStatus   :1;
  int       nFc;
  int       nState;
  char      szItem[50];
  WORD      wTransaction;
  WORD      wState;
}
```

Parameter	Description
nStWarnungen, nStFehler	Control unit status; read from the control unit with each online function, no update for basic functions.
nFehler	Error code; see **Error.h** error code file
nLastDDEError	Last DDE error; see **Error.h** error code file

Value	Explanation
-1	Not defined; control unit status is unknown
0	No warnings and/or errors
1	Warning and/or errors have occurred in the control unit

Bit	Explanation
0-2	Not yet assigned
3	DOS error; see **nFehler**
4	rho3 error (during data transfer) see **nFehler**
5	Error of last online function
5-14	Not yet assigned
15	Server status = ready

nFc Indicates the online function last executed.

Value	Explanation
-1	Undefined
1	Dir (list directory)
2	Copy PC-> RC
3	Copy RC-> PC
4	Rename
5	Delete
1003	Search for process
1005	Search for next process
1007	Process selection
1010	KinX position
1011	Kinematics information
1013	Error
1016	Version
1022	Process stop
1023	Set RCA
1030	Signals
1031	rho3 position
1034	RC home position
1037	List processes
1042	Tool

Value	Explanation
0	ready
1	init
2	running
3	stop
4	waiting for stop
5	Abort

szItem Name of last item
wTransaction Last DDEcommand

As the flags labelled *f3Frei* through *wState* are used for diagnostic purposes only, their interpretation is not required in standard operation.

10.3.11 Set RCA

This function can be used to set the RCA signals 10.1 through 10.8.

In order to detect any errors that may have occurred during the execution of this command, the actual interface status should be determined immediately following this function.

Client	Message "Item"	Data	<=>	Server
Set RCA signals	XTYP_POKE "SetRCA"	SigArray	=>	
	DDE_FACK "SetRCA"	---	<=	Acknowledge
Request status	XTYP_REQUEST "GStatus"	---	=>	
		TGSTATUS	<=	Send GStatus

Start parameters
int *SigArray*[8];

Parameter	Description
SigArray	Defines the nominal status of the signals. This parameter can have one of the following values:

Wert	Explanation
0	low
1	high
127	don’t care

Return parameters
none

10.3.12 Signal Status

This function is used for cyclical signal status queries. The signals can be requested only in byte mode.

Client	Message "Item"	Data	<=>	Server
Initialize request	XTYP_POKE "Signale"	TMIXEDARRAY	=>	
	DDE_FACK "Signale"	---	<=	Acknowledge
Start cyclical quesry	XTYP_ADVSTART "Signale"	---	=>	
	TRUE	---	<=	Acknowledge
Continue until Stop	XTYP_ADVDATA "Signale"	TDDESIGNALE	<=	Send signals
	DDE_FAck "Signale"	---	=>	
Stop signals	XTYP_ADVSTOP "Signale"	---	=>	

Start parameters

```
struct TMIXED
{
 int        nSigTyp;
 int        nSigAdr;
};

struct TMIXEDARRAY
{
 int        nAnzSignale;
 TMIXED     MixedArray[_MAX_SIGNALE];
}
```

Parameter	Description
nAnzSignale	Number of signal bytes;
SigTyp	Defines the signal type. This parameter can have one of the following values:

Value	Explanation
0	RC outputs
2	RC inputs
5	Digital inputs
4	Digital outputs

Return parameters

```
struct DDESIGNALE
{
 TGSTATUS       GStatus;
 int            nAnzSignale;
 unsigned char  SigArray[_MAX_STATUS_SIGNALE];
};
```

Parameter	Description
GStatus	Global status, see Section 9.1, "Status and Initialization Functions."
SigArray[]	Status of requested signal byte.

10.4 Access to User Variables

The ROPS3 server is capable of monitoring as well as modifying the contents of the user variables of any desired BAPS program. This is possible regardless of whether the file containing the referred variables is located on the PC or in the control unit, or whether a process is active in the RC or processing has just been concluded.

10.4.1 General Information

10.4.1.1 Prerequisites

In order to enable symbolic access to the variables, the server requires information from the .SYM file. This file must be available on the PC, and the server must be informed where it is located (path description).

The .IRD file in which the contents of most variables have been stored, can be located in the control unit as well as on the PC. Which file the server is to access will be specified in the corresponding DDE messages. In the event that an .IRD file on the PC is to be accessed, this file must be located on the same path as the .SYM file.

In the event of access to the point variables that are stored in the :PKT file, the .PKT file will also be required. In this case, too, the user determines via his DDE message where the file he wishes to access is located.

The DDE server has simultaneous access to the user variables in up to 20 different user files.

Note: As regards its services, the Online DDE Server supports only filenames up to 8 characters in length.

10.4.1.2 Permitted Variables

The server is basically capable of accessing all user variables, the contents of which are stored in the .IRD or .PKT file, i.e., variables that are defined in the main program.

User variables that the control unit has not written to the .IRD and/or .PKT file, but that are only present on the internal IRD stack during runtime, cannot accessed by the server, and can therefore neither be read not written to. This type of variables includes, for example, the transfer parameters for subprograms, or variables that are defined in the subprograms themselves.

The server has no access to so-called *system variables*. These are variables that are always present in all processes, and that do not have to be explicitly declared by the programmer.

The group of *system variables* includes the following:

IPOS, @IPOS, @MPOS, GRENZE_MIN, GRENZE_MAX, V, VFEST, T, TFEST, A, AFEST, V_PTP, VFEST_PTP, VFAKTOR, AFAKTOR, SKALA, RK_SYSTEM, DFAKTOR, R_PTP, and R.

The current values of these variables are not stored in the :IRD file but are handled separately by the operating system of the control unit.

10.4.1.3 Entering Names of Variables

The server must be given the name of the variable in the same way in which it was defined in the BAPS program. Upper case and lower case characters are interpreted in compiler fashion, e.g. with equal value.

Name extensions, such as kinematics names or components of point variables, are separated by a decimal point from the actual variable name. The use of wildcard characters is not permitted.

Example:
Accessing a point component ("name.Komponente")

```
'pl.a_1'        ; This entry returns the component
                ; 'a_1' of the point labelled 'pl'
```

Example:
Accessing a point with kinematics data ('kinematic.name'

```
'sr800.pl'      ; This entry returns the value of
                ; point 'pl' which belongs to kine-
                ; matics 'sr800'
```

Example:
Accessing a point component with kinematics data
('kinematic.name.Komponente')

```
'sr800.pl.a_1'  ; This entry returns the value of
                ; the component 'a_1' of 'point 'pl'
                ; which belongs to kinematics
                ; 'sr800'
```

When requesting fields, the indexes of the individual field dimensions are enclosed in square brackets. In the case of multidimensional fields, not all dimensions must be specified. Ranges of a given dimension are separated by a hyphen.

It should be noted that a range specification may be specified only once per request, and this applies only to the last dimension specified.

Example:

Definition of a two-dimensional field in BAPS:

```
"FELD [1..30] FELD [1..10] GANZ :
INT_ARRAY
```

Access to a field variable

```
'int_array[1][1]'    ; Returns a total value of
                     ; field named 'int_array'
```

Access to a complete field dimension

```
'int_array[1]'       ; Returns 10 total values of
                     ; field named 'int_array'
                     ; ('int_array' [1][1] through
                     ; 'int_array' [1][10])
```

Access to a range of a field dimension

```
'int_array[1][2-5]' ; Returns 4 total values of
                     ; field named 'int_array'
                     ; ('int_array' [1][2],
                     ; 'int_array'[1][3],
                     ; 'int_array'[1][4] and
                     ; 'int_array' [1][5])
```

OR

```
'int_array[1-2]'     ; Returns 20 total values of
                     ; field named 'int_array'
```

Not permitted are the following entries:

```
'int_array[1-5][2-5]'
```

OR

```
'int_array[1-5][2]'
```

The specification of variables requires similar definitions.

10.4.1.4 Security Query (Common ID)

In order to guarantee the correctness of the contents of variables it supplies, the server must receive all information it requires for this purpose from files that were created during the same compilation session. To safeguard this prerequisite, a so-called *Common ID* monitoring function is used. (The *Common ID* comprises an identification number that is written into each file during compilation, with the system time stamp indicating at which point in time this file was created or last modified.)

In the event that the Common ID of the :IRD, .PKT and .SYM files does not match, an error message will be returned by the **GStatus** or Server-Control (**Control_Server**) services, and the service terminated.

The Common ID monitoring can also be disabled upon request by the client. It is instructive to note, however, that the user must be fully aware of the consequences. In the worst-case scenario, the disablement of the function can also cause the destruction of a .PKT or .IRD file.

10.4.2 Reading Variables

This service provided by the server returns the contents of any desired user variables. It is possible to observe a maximum of 32 variables with a total of 200 bytes of information at the same time.

Possible errors are signalled by the **GStatus** or **ServerFehler** service.

One-time request for variables:

Client	Message "Item"	Data	<=>	Server
Initialize request	XTYP_POKE "item"	TINITREADWRITE	=>	
	DDE_FACK "item"	---	<=	Acknowledge
Request contents (random number of requests)	XTYP_REQUST "item"	---	=>	
		TREADVARDATA	<=	Send contents of variables
Stop request	XTYP_ADVDATA "item"	TEXITREADWRITE "EXIT_POKE"	<=	
	DDE_FACK "item"	---	=>	Acknowledge

Note: When using **INIT_POKE**, the files required by the server for an access to the variable are opened. Because the server needs the **EXIT_POKE** instruction to close all associated files and to release the internal memory capacity it has been using for this function, the user must ascertain that an initialized request is properly ended with the **EXIT_POKE** command.

For each item, a maximum of 200 bytes can be read.

Cyclical request for variables:

Client	Message "Item"	Data	<=>	Server
Initialize	XTYP_POKE "item"	TINITREADWRITE "INIT_POKE"	=>	
request	DDE_FACK "item"	---	<=	Acknowledge
Start cyclical	XTYP_ADVSTART "item"	---	=>	
request	TRUE		<=	
Continue	XTYP_ADVDATA "item"	TREADVARDATA	<=	Send contents
until				of variables
stop	DDE_FACK "item"	---	=>	
Stop reading	XTYP_ADVSTOP "item"	---	=>	

Note: At the point of starting the cyclical query, the files required by the server for access to the variable are opened. The user is advised to make certain that a cyclically initializing request is properly terminated with the **XTYP_ADVSTOP** instruction. Only in this case will the server be able to close all files it has opened, and release the internal memory range it has occupied for this function. Also, the DDE management will then register the cyclical service as concluded.

In the course of cyclical read accesses, all active items are grouped, and their contents are simultaneously requested by the RC (or by the PC). In this manner, a synchronized image of the contents of the desired variables is obtained. For this reason, a maximum total of 200 bytes per channel can be cyclically read-accessed.

Items:
VarRead1 .. VarRead32
(The number of items can be limited in the ROPS3SVR.INI file.)

Start Parameters for initialization
typedef enum { INIT_POKE, EXIT_POKE, DATA_POKE } TPOKESTATUS:

```
struct TINITREADWRITE
{
  TPOKESTATUS  PokeStatus
  char         szPath        [MAX_DIR];
..char         szVarName  [_MAX_STRING];
  BOOL         bCommonID;
  BOOL         bPCRC;
};
```

Parameter	Description
PokeStatus	This datum has 3 states (INIT_POKE, EXIT_POKE, and DATA_POKE), and is used by the server to differentiate which type of message is represented by a particular poke. When initializing, this datum must be set to "INIT_POKE."
szPath	Complete .SYM file pathname and filename (without extension).
szVarName	Name of variable (including possible field indexes)
bCommonID	Common ID monitoring

Value	Explanation
0	Monitoring is disabled
1	Monitoring is enabled

bPCRC Reads variable from file in RC or on PC

Value	Explanation
0	Reads variable from file in RC
1	Reads variable from file on PC

Start Parameters for termination

typedef enum { INIT_POKE, EXIT_POKE, DATA_POKE } TPOKESTATUS:

```
struct TEXITREADWRITE
{
  TPOKESTATUS  PokeStatus
};
```

Parameter	Description
PokeStatus	This datum has 3 states (INIT_POKE, EXIT_POKE, and DATA_POKE), and is used by the server to differentiate which type of message is represented by a particular poke. When initializing, this datum must be set to "INIT_POKE."

Return parameters

```
struct TBINEA
{
  long        1BinEA;
  long        1Kanal;
}

struct TDEZA
{
  float       fdezEA;
  long        1Kanal;

struct TGANZEA
{
  long        1GanzEA
..long        1Kanal;
}
```

```
struct TREADVARDATA
{
  TGSTATUS      GStatus
  int           nGroesse
  union{
      float     fDez          [50];
      long      1Ganz         [50];
      long      1Binaer       [50];
      char      cZeichen      [200];
      char      szText        [200];
      float     fPunkt        [50];
      float     fMKPunkt      [50];
      float     fRKRahmen     [50];
      TBINEA    1BinEingang   [25];
      TBINEA    1BinAusgang   [25];
      TDEZEA    fDezEingang   [25];
      TDEZEA    fDezAusgang   [25];
      TGANZEA   1GanzEingang  [25];
      TGANZEA   1GanzAusgang  [25];
      }Var;
};
```

Parameter	**Description**
bBinEA	Status of binary channel
fdezEA	Status of DEZ channel
1GanzEA	Status of GANZ channel
1Kanal	Channel number of inputs and outputs
GStatus	Global status (see Section 9.1, "Status and Initialization Functions")
nGroesse	Number of bytes transferred
fDez	Contents of a variable of DEZ type
1Ganz	Contents of a variable of GANZ type
1Binaer	Contents of a variable of BINAER type
cZeichen	Contents of a variable of ZEICHEN type
szText	Contents of a variable of TEXT type
fPunkt	Contents of a variable of PUNKT type
FMKPunkt	Contents of a variable of MK_PUNKT type
fRKRahmen	Contents of a variable of RK_PUNKT type
1BinEingang	Contents & channel number of binary input
1BinAusgang	Contents & channel number of binary output
1DezEingang	Contents & channel number of DEZ input
1DezAusgang	Contents & channel number of DEZ output
1GanzEingang	Contents & channel number of GANZ input
1GanzAusgang	Contents & channel number of GANZ output

Note: In the case of undefined points, the server will return "ffffffff". With IPOS and @IPOS, the channel number (long) is included as the last datum in the transfer.

With cyclical requests, the server returns the contents of variables only once a change has occurred in the variable.

The transmission of the contents of variables can be temporarily halted by setting a control bit in the **Control_Client** function.

10.4.3 Reading Variables via ASCII Protocol

This service provided by the server returns the contents of any user variables. The communication between client and server is effected by means of ASCII characters. It is possible to observe a maximum of 32 variables with a total of 200 bytes of information at the same time.

Possible errors are signalled by the **GStatus** or **ServerFehler** service.

One-time request for variables:

Client	Message "Item"	Data	<=>	Server
Initialize request	XTYP_POKE "item"	szReadVar "INIT"	=>	
	DDE_FACK "item"	---	<=	Acknowledge
Request contents (random number of requests)	XTYP_REQUST "item"	---	=>	
		szReadVarData	<=	Send contents of variables
Stop request	XTYP_POKE "item"	szReadVar "EXIT"	<=	
	DDE_FACK "item"	---	=>	Acknowledge

Note: While initializing, the files required by the server for an access to the variable are opened. Because the server needs the **EXIT** instruction to close all associated files and to release the internal memory capacity it has been using for this function, the user must ascertain that an initialized request is properly ended with the **EXIT** command.
For each item, a maximum of 200 bytes can be read.

Cyclical request for variables:

Client	Message "Item"	Data	<=>	Server
Initialize request	XTYP_POKE "item"	szReadVar "INIT"	=>	
	DDE_FACK "item"	---	<=	Acknowledge
Start cyclical request	XTYP_ADVSTART "item"	---	=>	
	TRUE	---	<=	
Continue until stop	XTYP_ADVDATA "item"	szReadVarData	<=	Send contents of variables
	DDE_FACK "item"	---	=>	
Stop request	XTYP_ADVSTOP "item"	---	=>	

Note: At the point of starting the cyclical query, the files required by the server for access to the variable are opened. The user is advised to make certain that a cyclically initializing request is properly terminated with the **XTYP_ADVSTOP** instruction. Only in this case will the server be able to close all files it has opened, and release the internal memory range it has occupied for this function. Also, the DDE management will then register the cyclical service as concluded.

In the course of cyclical read accesses, all active items are grouped, and their contents are simultaneously requested by the RC (or by the PC). In this manner, a synchronized image of the contents of the desired variables is obtained. For this reason, a maximum total of 200 bytes per channel can be cyclically read-accessed.

Items:
VarRead1_A .. VarRead32_A
(The number of items can be limited in the ROPS3SVR.INI file.)

Start Parameters for initialization
char szReadVar[_MAX_STRING]; "INIT, szPath, szVarName[,cCommonId,cPCRC]\0"

The "cCommonId" and "cPCRC" parameters can be omitted, in which case the default values will apply.

Parameter	Description
INIT	Keyword for initializing a task.
szPath	Complete .SYM file pathname and filename (without extension).
szVarName	Name of variable (including possible field indexes)
bCommonID	Common ID monitoring

Value	Explanation
0	Monitoring is disabled
1	Monitoring is enabled (default)

cPCRC Reads variable from file in RC or on PC

Value	Explanation
0	Reads variable from file in RC (default)
1	Reads variable from file on PC

Start Parameters for termination
char szReadVar[_MAX_STRING]; "EXIT\0"

Parameter	Description
EXIT	Keyword for terminating a request.

Return parameters
struct TBINEA
char *szReadVarData*[_MAX_ASCII_ANSWER];
"szWert1[,szWert2,szWert3..]\0"

Parameter	Description
szWert1,szWert2..	Contents of variable(s) in ASCII. If there are more than value, (e.g. with points), the individual values are separated by commas.

Examples: ASCII string structure

Type of variable	String structure
DEZ	"1.0,-32.66,0,177\0"
GANZ	"10,20,-33,1235\0"
BINAER	"1,1,1,0,0,0\0"
ZEICHEN	"x\0" Special feature with character fields: Here, the individual character fields are **not** separated by commas!
TEXT	"ABCDEfghljk01\0" Special features with texts and text fields: In BAPS, a text can have a maximum length of 80 characters. In the event is shorter than 80 characters, the remaining characters of the text (up the maximum size) must be filled up with zeroes. If a text is 80 characters long, the 0 at the end of the text is omitted. The server always transfers 80 characters for each text, and/or for each field element of a text field.
PUNKT, MKPUNKT RKRAHMEN	"333.444,-777.44,0.98\0" Special feature with points: In the case of undefined points, the server will return the contents "--.--\0"
EINGANG BINAER, AUSGANG BINAER	"1,1,0,2\0"
EINGANG DEZ, AUSGANG DEZ	"11.22,201,-44.55,402\0"
EINGANG GANZ, AUSGANG GANZ	"11.22,201,-44.55,402\0" Special feature with channels: Transfers for channels always aleays include 2 values, with the first value representing the channel status and/or channel value, and the second being the channel number.

With cyclical requests, the server returns the contents of variables only once a change has occurred in the variable.

The transmission of the contents of variables can be temporarily halted by setting a control bit in the **Control_Client** function.

10.4.4 Writing Variables

The user can avail himself of these services for the purposes of changing variables. To ensure the detection of errors that may have occurred as a result of write-accesses to variables, the current status should be determined subsequent to executing the command (using **GStatus** or **Server-Fehler** functions).

Because it is possible that both the BAPS process of the control unit and the server may access the same variable at the same time, the application programmer must safely exclude any possible addressing conflict. Accordingly, the responsibility for precluding unwanted control unit responses while writing variables with the use of this server function rests with the application programmer.

As regards validity or value range, the server does not perform any type of verification of the new values sent by the client, but writes these values directly into the file indicated to the server.

One-time write-access to variables:

Client	Message "Item"	Data	<=>	Server
Initialize	XTYP_POKE "item"	TINITREADWRITE "INIT_POKE"	=>	
request	DDE_FACK "item"	---	<=	Acknowledge
Write to contents (random number of requests)	XTYP_POKE "item"	TWRITEVAR "DATA_POKE"	=>	Send contents of variables
	DDE_FACK "item"	---	<=	
Stop	XTYP_POKE "item"	TEXITREADWRITE "EXIT_POKE"	<=	
request	DDE_FACK "item"	---	=>	Acknowledge

Note: When using **INIT_POKE**, the files required by the server for an access to the variable are opened. Because the server needs the **EXIT_POKE** instruction to close all associated files and to release the internal memory capacity it has been using for this function, the user must ascertain that an initialized request is properly ended with the **EXIT_POKE** command.
For each item, a maximum of 200 bytes can be written.

Cyclical write-access to variables:

Client	Message "Item"	Data	<=>	Server
Initialize	XTYP_POKE "item"	TINITREADWRITE "INIT_POKE"	=>	
request	DDE_FACK "item"	---	<=	Acknowledge
Start cyclical write-access	XTYP_ADVSTART "item"	---	=>	
	TRUE	---	<=	
Continue writing (random number of accesses)	XTYP_POKE "item"	TWRITEVAR "DATA_POKE"	<=	Send contents of variables
	DDE_FACK "item"	---	=>	Acknowledge
Stop reading	XTYP_ADVSTOP "item"	---	=>	

Note: At the point of starting the cyclical query, the files required by the server for access to the variable are opened. The user is advised to make certain that a cyclically initializing request is properly terminated with the **XTYP_ADVSTOP** instruction. Only in this case will the server be able to close all files it has opened, and release the internal memory range it has occupied for this function. Also, the DDE management will then register the cyclical service as concluded. For each item, a maximum total of 200 bytes per channel can be written.

Items:
VarWrite1 .. VarWrite32
(The number of items can be limited in the ROPS3SVR.INI file.)

Start Parameters for initialization
typedef enum { INIT_POKE, EXIT_POKE, DATA_POKE } TPOKESTATUS;

```
struct TINITREADWRITE
{
  TPOKESTATUS  PokeStatus
  char         szPath       [MAX_DIR];
..char         szVarName    [_MAX_STRING];
  BOOL         bCommonID;
  BOOL         bPCRC;
};
```

Parameter	Description
PokeStatus	This datum has 3 states (INIT_POKE, EXIT_POKE, and DATA_POKE), and is used by the server to differentiate which type of message is represented by a particular poke. When initializing, this datum must be set to "INIT_POKE."
szPath	Complete .SYM file pathname and filename (without extension).
szVarName	Name of variable (including possible field indexes)
bCommonID	Common ID monitoring

Value	Explanation
0	Monitoring is disabled
1	Monitoring is enabled

bPCRC Reads variable from file in RC or on PC

Value	Explanation
0	Reads variable from file in RC
1	Reads variable from file on PC

Start Parameters for termination

typedef enum { INIT_POKE, EXIT_POKE, DATA_POKE } TPOKESTATUS;

```
struct TEXITREADWRITE
{
 TPOKESTATUS  PokeStatus
};
```

Parameter	Description
PokeStatus	This datum has 3 states (INIT_POKE, EXIT_POKE, and DATA_POKE), and is used by the server to differentiate which type of message is represented by a particular poke. When initializing, this datum must be set to "INIT_POKE."

Start Parameters for write-access

typedef enum { INIT_POKE, EXIT_POKE, DATA_POKE } TPOKESTATUS;

```
struct TWRITEVAR
{
 TPOKESTATUS  GStatus
 int          nGroesse
 union{
     float    fDez          [50];
     long     1Ganz         [50];
     long     1Binaer       [50];
     char     cZeichen      [200];
     char     szText        [200];
     float    fPunkt        [50];
     float    fMKPunkt      [50];
     float    fRKRahmen     [50];
     long     1BinEingang   [50];
     long     1BinAusgang   [50];
     float    fDezEingang   [50];
     float    fDezAusgang   [50];
     long     1GanzEingang  [50];
     long     1GanzAusgang  [50];
     }Var;
};
```

Parameter	Description
PokeStatus	This datum has 3 states (INIT_POKE, EXIT_POKE, and DATA_POKE), and is used by the server to differentiate which type of message is represented by a particular poke. When initializing, this datum must be set to "INIT_POKE."
nGroesse	Number of bytes to be written
fDez	New contents of a variable, DEZ type
1Ganz	New contents of a variable, GANZ type
1Binaer	New contents of a variable, BINAER type
cZeichen	New contents of a variable, ZEICHEN type
szText	New contents of a variable, TEXT type
fPunkt	New contents of a variable, PUNKT type
FMKPunkt	New contents of a variable, MK_PUNKT type
fRKRahmen	New contents of a variable, RK_PUNKT type
1BinEingang	New contents of a variable, binary input type
1BinAusgang	New contents of a variable, binary output type
1DezEingang	New contents of a variable, DEZ input type
1DezAusgang	New contents of a variable, DEZ output type
1GanzEingang	New contents of a variable, GANZ input type
1GanzAusgang	New contents of a variable, GANZ output type

Return parameters

none

10.4.5 Writing Variables via ASCII Protocol

For the purpose of changing variables, these services are also available to the user. The communication between client and server is effected by means of ASCII characters. To ensure the detection of errors that may have occurred as a result of write-accesses, the current status should be determined subsequent to executing the command (using **GStatus** or **ServerFehler** functions).

Because it is possible that both the BAPS process of the control unit and the server may access the same variable at the same time, the application programmer must safely exclude any possible addressing conflict. Accordingly, the responsibility for precluding unwanted control unit responses while writing variables with the use of this server function rests with the application programmer.

As regards validity or value range, the server does not perform any type of verification of the new values sent by the client, but writes these values directly into the file indicated to the server.

One-time write-access to variables:

Client	Message "Item"	Data	<=>	Server
Initialize	XTYP_POKE "item"	szWriteVar "INIT"	=>	
request	DDE_FACK "item"	---	<=	Acknowledge
Write to contents (random number of requests)	XTYP_POKE "item"	szWriteVar "DATA"	=>	Send contents of variables
	DDE_FACK "item"	---	<=	Acknowledge
Stop	XTYP_POKE "item"	szWriteVar "EXIT"	<=	
request	DDE_FACK "item"	---	=>	Acknowledge

Note: During initialization, the files required by the server for an access to the variable are opened. Because the server needs the **EXIT** instruction to close all associated files and to release the internal memory capacity it has been using for this function, the user must ascertain that an initialized request is properly ended with the **EXIT** command.
For each item, a maximum of 200 bytes can be written.

Cyclical write-access to variables:

Client	Message "Item"	Data	<=>	Server
Initialize	XTYP_POKE "item"	szWriteVar "INIT"	=>	
request	DDE_FACK "item"	---	<=	Acknowledge
Start cyclical query	XTYP_ADVSTART "item"	---	=>	
	TRUE	---	<=	
Continue writing (random number of accesses)	XTYP_POKE "item"	szWriteVar "DATA"	<=	Send contents of variables
	DDE_FACK "item"	---	=>	Acknowledge
Stop reading	XTYP_ADVSTOP "item"	---	=>	

Note: At the point of starting the cyclical query, the files required by the server for access to the variable are opened. The user is advised to make certain that a cyclically initializing request is properly terminated with the **XTYP_ADVSTOP** instruction. Only in this case will the server be able to close all files it has opened, and release the internal memory range it has occupied for this function. Also, the DDE management will then register the cyclical service as concluded.
For each item, a maximum total of 200 bytes can be written.

Items:
VarWrite1_A .. VarWrite32_A
(The number of items can be limited in the ROPS3SVR.INI file.)

Start Parameters for initialization
char szWriteVar[_MAX_STRING]; "INIT, szPath, szVarName[,cCommonId,cPCRC]\0"

The "cCommonId" and "cPCRC" parameters can be omitted, in which case the default values will apply.

Parameter	Description
INIT	Keyword for initializing a task.
szPath	Complete .SYM file pathname and filename (without extension).
szVarName	Name of variable (including possible field indexes)
cCommonID	Common ID monitoring (optional)

Value	Explanation
0	Monitoring is disabled
1	Monitoring is enabled (default)

cPCRC Reads variable from file in RC or on PC

Value	Explanation
0	Reads variable from file in RC (default)
1	Reads variable from file on PC

Start Parameters for termination
char *szWriteVar*[_MAX_ASCII_ANSWER]; "EXIT\0"

Parameter	Description
EXIT	Keyword for terminating a request.

Start Parameters for write-access
char szWriteVarData[_MAX_ASCII_ANSWER];
"DATA,szWert1[,szWert2,szWert3..]\0"

Parameter	Description
DATA	Keyword for sending new values
szWert1,szWert2...	New contents of variable(s) in ASCII form. In the case of more than one value (e.g. with points) the individual values are separated by commas.

Examples: ASCII string structure when sending new values

Type of variable	String structure
DEZ	"DATA,1.0,-32.66,0,177\0"
GANZ	"DATA,10,20,-33,1235\0"
BINAER	"DATA,1,1,1,0,0,0\0"
ZEICHEN	"DATA,x\0" Special feature with character fields: Here, the individual character fields are **not** separated by commas!
TEXT	"DATA,ABCDEfghIjk01\0" Special features with texts and text fields: In BAPS, a text can have a maximum length of 80 characters. In the event is shorter than 80 characters, the remaining characters of the text (up the maximum size) must be filled up with zeroes. If a text is 80 characters long, the 0 at the end of the text is omitted. The server always transfers 80 characters for each text, and/or for each field element of a text field.
PUNKT, MKPUNKT RKRAHMEN	"DATA,333.444,-777.44,0.98\0" Special feature with points: Defining a point with the use of "--.--" (undefined) is not possible.
EINGANG BINAER, AUSGANG BINAER	"DATA,1,0\0"
EINGANG DEZ, AUSGANG DEZ	"DATA,11.22,-44.55\0"
EINGANG GANZ, AUSGANG GANZ	"DATA,11.22,-44.55\0"

Return parameters
none

10.4.6 Example

A system that is provided with a controller can manufacture a product in four different versions. The number of items to be manufactured and the product version is entered via a PC (any user interface), and this data is transmitted to the On-line Server via DDE. The sequential program in the controller receives this data from the server and arranges the production of the required parts.

Note:

Additional application examples for user programming in ACCESS, EXCEL and WORD are located on the server diskettes.

The main sequential program:

```
;;CONTROLLER = RHO3

;;KINEMATICS: (1=SR800)

PROGRAMM PROD

;************************************************
;  Variables specified by client
;************************************************

GANZ:    AUFTRAG       ;Product version
GANZ:    ANZAHL        ;Required number of products
BINAER:  1 = STARTSIG  ;Start signal => Assemble
                       ;specified product(s)


;************************************************
;  Variables read by client
;************************************************
BINAER:    FPRODUKT    ;Unknown version
BINAER:    FANZAHL     ;Incorrect number
BINAER:    1 = ENDSIG  ;End signal => Products
                       ;assembled
GANZ:      SUMME1,     ;Sum of product version 1
GANZ:      SUMME2,     ;Sum of product version 2
           SUMME3,     ;Sum of product version 3
           SUMME4,     ;Sum of product version 4


ANFANG

SCHLEIFE:


;************************************************
;  Wait for start signal from client
;************************************************
WARTE BIS STARTSIG=1
```

```
;************************************************
;  Initialization and review of specifications from client
;************************************************
FPRODUKT = 0
ENDSIG   = 0
WENN ANZAHL <  0  DANN ANFANG    ;Check number
                       FANZAHL = 1
                       SPRUNG SCHLEIFE
                       ENDE
                  SONST FANZAHL  =  0



;************************************************
;  Branch according to job
;************************************************
FALLS   AUFTRAG
        GLEICH  1:          ;COMPLETED VERSION 1
                ANFANG
                   WDH ANZAHL MAL
                    PROD1;  ;SUB-PROGRAM ASSEMBLED
                            ;PRODUCT 1
                    SUMME1  =  SUMME1  +  1;
                  WDH_Ende
                ENDE

        GLEICH  2:          ;COMPLETED VERSION 2
                ANFANG
                   WDH ANZAHL MAL
                    PROD1;  ;SUB-PROGRAM ASSEMBLED
                            ;PRODUCT 2
                    SUMME2  =  SUMME2  +  1;
                  WDH_Ende
                ENDE

        GLEICH  3:          ;COMPLETED VERSION 3
                ANFANG
                   WDH ANZAHL MAL
                    PROD1;  ;SUB-PROGRAM ASSEMBLED
                            ;PRODUCT 3
                    SUMME3  =  SUMME3  +  1;
                  WDH_Ende
                ENDE

        GLEICH  4:          ;COMPLETED VERSION 4
                ANFANG
                   WDH ANZAHL MAL
                    PROD1;  ;SUB-PROGRAM ASSEMBLED
                            ;PRODUCT 4
                    SUMME4  =  SUMME4  +  1;
                  WDH_Ende
                ENDE
        ANSONSTEN  PRODUKT  =  1;INCORRECT VERSION
FALLS_ENDE
```

```
;************************************************
;  Machine action completed; Message to client
;************************************************

      ENDESIG = 1;

      SPRUNG SCHLEIFE          ;WAITING FOR NEW
                               ;JOB

PROGRAMM_ENDE

;************************************************
;  Subprograms for product assembly
;************************************************
;PRODUCTION SEQUENCE VERSION 1
UP PROD1
      ANFANG
;       .
;       .
;       .
UP_ENDE

;PRODUCTION SEQUENCE VERSION 2
UP PROD2
      ANFANG
;       .
;       .
;       .
UP_ENDE

;PRODUCTION SEQUENCE VERSION 3
UP PROD3
      ANFANG
;       .
;       .
;       .
UP_ENDE

;PRODUCTION SEQUENCE VERSION 4
UP PROD4
      ANFANG
;       .
;       .
;       .
UP_ENDE
```

Sequential progress of client/Server operation:

Starting position:

- The server is started and running.
- The .SYM file named **Prod.SYM** is located on the PC, with path-name **c:\projekt**.
- The client has already established connection with the server.
- In the RC, the **PROD** process has already been selected and started.

Initialization (starting cyclical serv.) for reading variables:

- Reading error query for wrong execution number.
  Server service: **VarRead1**
  Variable in BAPS program: **FPRODUKT**

  Data struct for transfer to server: **TINITREADWRITE**
  Contents of struct elements:
  PokeStatus : INIT_POKE
  szPath: : "c:\projekt\prod\0"
  szVarName : "FPRODUKT\0"
  bCommonID : 1
  bPCRC : 0

- Reading error query for wrong product number.
  Server service: **VarRead2**
  Variable in BAPS program: **FANZAHL**

  Data struct for transfer to server: **TINITREADWRITE**
  Contents of struct elements:
  PokeStatus : INIT_POKE
  szPath: : "c:\projekt\prod\0"
  szVarName : "FANZAHL\0"
  bCommonID : 1
  bPCRC : 0

- Reading sum of versions already machined.
  Server service: **VarRead3**
  Variable in BAPS program: e.g. **Summe1**

  Data struct for transfer to server: **TINITREADWRITE**
  Contents of struct elements:
  PokeStatus : INIT_POKE
  szPath: : "c:\projekt\prod\0"
  szVarName : "SUMME1\0"
  bCommonID : 1
  bPCRC : 0

- Reading output signal indicating whether the complete job has been concluded.
  Server service: **VarRead4**
  Variable in BAPS program: e.g. **ENDESIG**

  Data struct for transfer to server: **TReadVar**
  Contents of struct elements:
  PokeStatus : INIT_POKE
  szPath: : "c:\projekt\prod\0"
  szVarName : "ENDSIG\0"
  bCommonID : 1
  bPCRC : 0

All cyclical services must be started via **XTYP_ADVSTART**.

11 Index

1070 072 161-102 (97.08) GB • HB SP • AT/PLS • Printed in Germany