_connect.BRAIN

Operating instructions

as of program version 3.52

38.026.297.002 en

BIZERBA

Bizerba GmbH & Co. KG,
72336 Balingen

P.O. Box 10 01 64
72301 Balingen, Germany
Telephone (+49 7433) 12-0, Fax (+49 7433) 12-2696
Email: marketing@bizerba.com
Internet: http://www.bizerba.com

Contents

Page

1 About these instructions 8

1.1 Safe-keeping 8

1.2 Target group 8

1.3 Symbols used 8
1.3.1 How notes and information are depicted 8
1.3.2 Explanation of warnings 8
1.3.3 Viewing of menu call-up 9

2 About the software 10

2.1 Overview 10

2.2 Rights 10

2.3 Warranty 11

2.4 Virus protection 11

3 Licensing 12

3.1 License packages 12

3.2 Developer mode 12

3.3 Software protection 13

4 BizInfo - application information 14

4.1 Overview 14

4.2 Tab "Application" 14

4.3 Tab "Modul" 15

4.4 Tab "System" 16

4.5 "Directories" tab 16

4.6 "License" tab 17

4.7 "3rd party" tab 17

5 Device connection 18

5.1 Modem 18
5.1.1 Overview 18
5.1.2 Configuration 18
5.1.3 Connection establishment 18
5.1.4 Connection clear-down 19

5.2 ADDI-DATA I/O board MSX-E1516 19

6 Use on terminal servers 21

6.1 Citrix terminal server 21
6.1.1 Installing _connect.Brain on a Citrix terminal server 21
6.1.2 Configuring user accounts for _connectServer 21
6.1.3 Settings in the registry 22
6.1.4 Using a serial port in a Citrix session 22

6.2 Windows Terminal Server 22
6.2.1 Installing _connect.Brain on a Windows terminal server 22
6.2.2 Temporary network breakdown when using a serial interface 22
6.2.3 Using VirtualES when _connectServer has been installed as service 23

7 _connectServer 24

7.1 Overview 24

7.2 Configuration 24
7.2.1 Configuring _connectServer 24
7.2.2 Operating modes 24
7.2.3 Assigning user accounts 25
7.2.4 Configuring operation in the network 26

8 _connectConfig 29

8.1 Overview 29

8.2 Starting the program 29

8.3 Program structure 30

8.4 Menu bar and toolbar functions 31

8.5 Functions in the work area 34

8.6 _connectServer Configuration 34
8.6.1 "Common" tab 35
8.6.2 "Connections" tab 37
8.6.3 "Devices" tab 46

8.7 _connect2File / 2File Configuration 49
8.7.1 Naming log files 49
8.7.2 "Common" tab 49
8.7.3 "Files" tab 50
8.7.4 "Devices" tab 52

8.8 BHI Configuration 55
8.8.1 Adding and deleting master/slave devices and department 56

8.9 Wizard for creating devices ... 57
8.9.1 Creating new devices using a wizard ... 57
8.9.2 Copying devices using the wizard ... 58
8.10 Creating links to devices of other _connect.BRAIN clients ... 59

9 _connectDiagnostics ... 60
9.1 Overview ... 60
9.2 Starting the program ... 60
9.3 Program structure ... 61
9.4 Menu bar and toolbar functions ... 61

10 _connect2File ... 63
10.1 Overview ... 63
10.2 Starting the program ... 63
10.3 Program structure ... 64
10.4 Menu bar and toolbar functions ... 65
10.5 Functions in the connection windows ... 66
10.6 File transfer ... 66
10.6.1 Data transfer from host to _connect2File ... 67
10.6.2 Data transfer from _connect2File to host ... 68
10.6.3 Controlling data output by events ... 68
10.6.4 File structure ... 69
10.7 Troubleshooting ... 71

11 2File ... 73
11.1 Overview ... 73
11.2 Starting the program ... 73
11.3 Program structure ... 74
11.4 Menu bar and toolbar functions ... 75
11.5 Functions in the connection windows ... 76
11.6 File transfer ... 76
11.6.1 File transfer from host to 2File ... 77
11.6.2 File transfer from *2File* to host ... 77
11.6.3 Controlling data output by events ... 78

11.7 File formats ... 78
11.7.1 Text file ... 79
11.7.2 XML file ... 81
11.8 Troubleshooting ... 83

12 _connect2DB ... 85
12.1 Overview ... 85
12.2 Starting the program ... 85
12.3 Database setup ... 85
12.4 Database structure ... 87
12.5 Program structure ... 88
12.6 Menu bar and toolbar functions ... 88
12.7 Defining filters ... 90
12.7.1 Creating filter using the wizard ... 90
12.7.2 Creating filters manually ... 91
12.8 Renaming filters ... 92
12.9 Edit filter ... 92
12.10 Device menu ... 93
12.10.1 Change window devices ... 94
12.10.2 Device settings ... 95
12.11 Default data ... 96
12.12 Defining handling of complex device commands ... 96
12.13 Deleting filters and relevant database tables ... 98
12.14 Exporting data ... 98
12.15 Data export via command line ... 99
12.16 Configuring _connect2DB ... 101
12.17 Deleting database contents ... 103
12.18 Backing up and reloading data ... 103
12.19 Convert DBConvert database ... 104

13 _connect2SAP ... 106
13.1 Overview ... 106
13.2 Installation ... 106

13.3 Configuration ... 107
13.3.1 _connect2SAP configuration ... 107
13.3.2 SAP configuration ... 108
13.3.3 Packing table configuration ... 110
13.4 _connect2SAP Frontend ... 112
13.5 _connect2SAP Registry and _connect2SAP Spooler ... 113
13.6 _connect2SAP Viewer ... 114
13.6.1 Overview ... 114
13.6.2 Starting the program ... 114
13.6.3 Structure of the program ... 114
13.6.4 Menu bar and toolbar functions ... 115
13.7 Functional components ... 116
13.7.1 Overview of functions ... 116
13.7.2 Z_RFC_BCT ... 116
13.7.3 Z_RFC_BCT_MULTI ... 117
13.7.4 Z_RFC_BCT_PRINT ... 117
13.7.5 Z_RFC_BCT_SPOOLER ... 118
13.7.6 Z_BCT_LABEL_GLP ... 118
13.7.7 Z_BCT_DIMENSION ... 121
13.7.8 Z_BCT_REG ... 122
13.7.9 Z_BCT_NULLSTELLEN ... 126
13.7.10 CFB_RFC_BCT_MULTI ... 127

14 _connectScannerWI ... 128
14.1 Overview ... 128
14.2 Prerequisites ... 128
14.3 Installation ... 128
14.4 Configuration ... 129
14.5 Starting the program ... 129
14.6 Program structure ... 129
14.7 Functions in the context menu ... 130

15 VirtualES - View ... 131
15.1 Overview ... 131
15.2 VirtualES - Admin ... 131
15.2.1 Starting program ... 131
15.2.2 Structure of the program ... 132
15.2.3 Menu bar and toolbar functions ... 133
15.2.4 Defining new configuration ... 133

15.3 VirtualES - View ... 134
15.3.1 Start program ... 134
15.3.2 Program structure ... 134
15.3.3 Menu bar and toolbar functions ... 135
15.3.4 Display data records ... 136
15.3.5 Check signatures ... 136
15.3.6 Registration of weighing results ... 136

16 _edit.BRAIN ... 137
16.1 Overview ... 137
16.2 Starting the program ... 137
16.3 Program structure ... 137
16.4 Menu bar and toolbar functions ... 138
16.5 Functions in the context menu ... 141

17 LogPathConfig ... 142
17.1 Overview ... 142
17.2 Program structure ... 142
17.3 Menu bar and toolbar functions ... 142
17.4 Deleting log files ... 142

18 Background information ... 144
18.1 Device families ... 144
18.2 BxNet language ... 144
18.2.1 Telegram structure ... 145
18.2.2 Coding of the data description ... 145
18.2.3 BxNet data types ... 145
18.2.4 Coding of the useful data ... 146
18.2.5 Dimensionful data: ... 146
18.2.6 Coding of prices ... 146
18.2.7 Coding of weights ... 147

19 Program interfaces ... 149
19.1 _connectServer DCOM communication interface ... 149
19.1.1 Methods ... 149
19.1.2 Events ... 164
19.2 _connectServer DCOM information interface ... 165
19.2.1 Methods ... 165

19.3 _connectControl DCOM communication interface 172
19.3.1 Properties 173
19.3.2 Methods 173
19.3.3 Events 194
19.3.4 IsUnicodeDevice 196
19.4 BctFunctions 196
19.4.1 Conversion function 197
19.4.2 Parse functions 201

1 About these instructions

Read the operating manual through carefully before installing and using the program, to ensure that you fully utilize the quality and possibilities of application offered.

We offer training in relation to our products. Please contact your Bizerba consultant for details.

Our products undergo continuous further development and are subject to different country-specific regulations. Examples of pictures and graphics included in these instructions may vary from the version you have received.

1.1 Safe-keeping

The operating manual is an integral part of this software and must be kept easily accessible for all personnel. If selling or passing on the software to others, the complete operating manual must also be provided.

1.2 Target group

Basic knowledge of the MS-Windows user interface is advantageous for operation of the program.

1.3 Symbols used

The following symbols can be found in the manual:

⇨	Text with arrow prompts you to carry out an action.
①	Position number in figure.
<OK>	Text inside a < > refers to a key or softkey.
"Display"	Text inside a " " refers to display text.

Prerequirements are displayed with a gray background.

1.3.1 How notes and information are depicted

Notes and information are depicted as follows:

Observance of these notes is mandatory.

This information is provided for greater understanding.

1.3.2 Explanation of warnings

The signal word above the symbol indicates the risk level.

DANGER 	**Source of danger with high risk of imminent danger to persons!** **The consequences are:** – **life threatening injuries** – **severe damage to health** – Measures to avoid the danger are specified.

WARNING 	**Source of danger with medium risk with potentially threatening danger for personnel!** **The consequences can be:** – **serious injuries** – **damage to health** – **serious damage to property** – Measures to avoid the danger are specified.

CAUTION 	**Source of danger with slight risk with potentially threatening danger for personnel!** **The consequences can be:** – **Injuries** – **Damage to property** – Measures to avoid the danger are specified.

CAUTION	**Source of danger, improper use!** **Damage to property can result.** – Measures to avoid the danger are specified.

1.3.3 Viewing of menu call-up

In order to present the menu navigation clearly and concisely, this manual makes use of the following abbreviations (example):

➪ <Start> / "All programs" / "Bizerba" / "BspLicenseManager" / "BspLicenseManager"

2

2 About the software

With *_connect.BRAIN*, Bizerba industrial scales, marking systems and printers can be connected with a single communication software package. Regardless of the connected device types, these communicate via the Bizerba standardized BxNet language and so control all your processes. This communication tool permits all industrial scales, marking devices and printers to be connected very quickly and efficiently to a customer-specific EDP.

2.1 Overview

The software *_connect.BRAIN* includes the following programs:

_connectControl	Bizerba Communication Control
_connect2File	File-based device communication
_connectServer	Bizerba Communication Server
_connectConfig	Configuration tool
_connectDiagnostics	Diagnostic tool
_edit.Brain	Log view
_connect2DB	Data transfer to a database
_connect2SAP	Data transfer to the SAP system
_connectScannerWI	Connection and operation of a barcode scanner
VirtualES	Data transfer to a verifiable memory

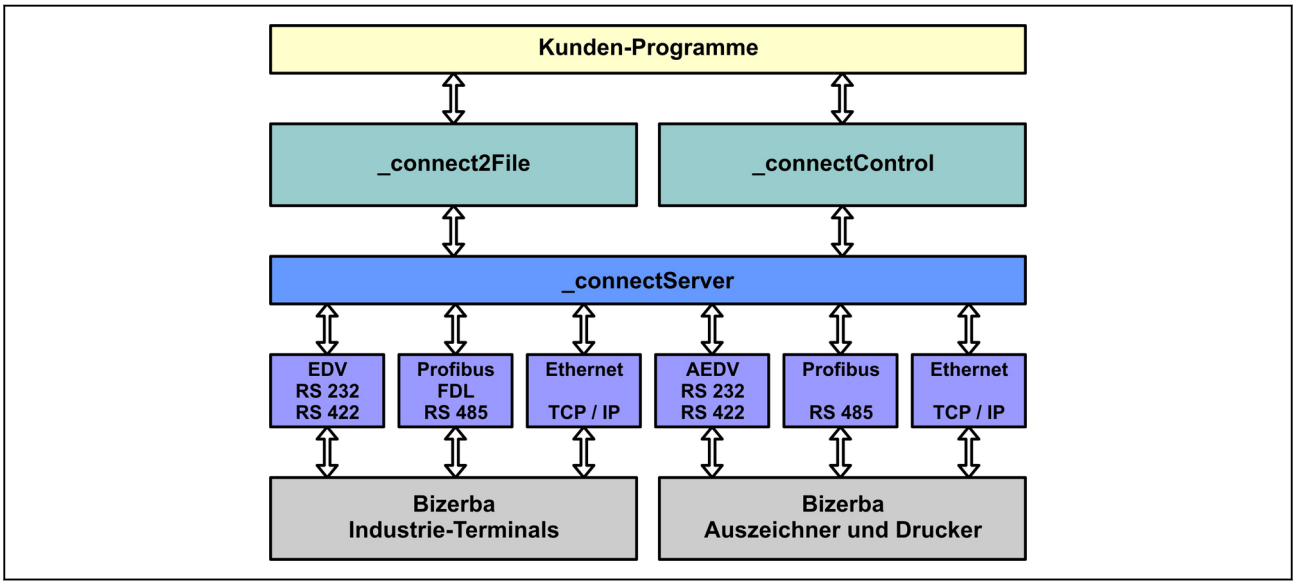


Illustration 1: Communication diagram

2.2 Rights

Proper purchase of the software licenses and instructions enables the programs to be used in accordance with the number of licenses. Copies on DVD are only permitted for data backup purposes (working copy).

2.3 Warranty

Despite all efforts made, errors cannot be completely ruled out of descriptions. We welcome your hints or comments at all times.

We are not responsible for damages caused by:

– Non-observance of these operating instructions.
– Incorrect electrical installation by customer.
– Changes to the operating system and configuration and to our software and configuration.

This warranty does not cover defects / damage caused by unauthorized persons.

Our products are constantly further developed and are subject to various country-specific regulations. Examples of pictures and graphics contained in these operating instructions may vary from the version which you received.

2.4 Virus protection

This software is manufactured and supplied without viruses based on the latest technology. However, there is always a possibility that a computer may be contaminated by computer viruses or other damaging software.

For your own safety, we recommend to run anti-virus software on your computer on a regular basis and after loading software. Furthermore, we recommend to check on viruses after loading software of other manufacturers or data carriers.

We recommend that you purchase an anti-virus program for virus analysis, which is constantly updated to the latest status.

3 Licensing

3.1 License packages

For the operation of *_connect.BRAIN*, the following license packages are available:

Premium				Professional				BasicPlus	Basic	Licenses for
CX	GX	IX	MX	CX	GX	IX	MX			
X	-	-	-	X	-	-	-	-	-	1 CX device
-	X	-	-	-	X	-	-	-	-	1 GX device
-	-	X	-	-	-	X	-	X	X	1 IX device
-	-	-	X	-	-	-	X	X	X	1 MX device
X				X				-	-	1 device in *_connect2DB*
X				X				-	-	1 device in *_connect2File*
X				X				-	-	1 device in *_connect2SAP*
X				-				X	-	1 device in *VirtualES*

Device assignment

License for CX device: CWM, CWE, CWD, CWP

License for GX device: GD, GH, GLP, GS, GV, GLM-I, GLM-E, GLM-B, GLM-P, GLM-L, GLF, GLM-E Retail

License for IX device: BT, EL, WM, CWL Eco, ITC1, ITC2, ITC-S, ITE, IST, ITL, ITU, MCE, MCI, ST, NTScale
Software: PSS
Other devices:
Siemens 3964R, scanner, terminal, ADDI-DATA IO card, ADDI-DATA IO board

License for MX device: BVS - Bizerba Vision System

3.2 Developer mode

The following limitations apply to the developer mode:

– A maximum of 50 telegrams can be transmitted
– 1000 ms delay during telegram transmission
– When opening the connection, standby time of 5 s before interface opens
– Communication possible with one device only

3.3 Software protection

The *_connect.BRAIN* software is protected by means of cryptographic functions. Software and device licenses are activated by a license key. They can be obtained from the sales or support department. Each connected device and, if needed, specific functions require appropriate licenses.

For the operation of *_connect.Brain*, the *Bizerba Software Protection (BSP)* has to be installed. *_connect.BRAIN* communicates with BSP and allows access to enabled devices and functions.

The license manager *BspLicenseManager* allows access to the license management.

Call up license manager

➪ <Start> / "All programs" / "Bizerba" / "BspLicenseManager" / "BspLicenseManager"

4 BizInfo - application information

4.1 Overview

Information about the program and its environment can be viewed via menu item "?" of a program and submenu item "Info about…". The information is found on different pages, which can be selected clicking the corresponding tabs.

Independent from the selected tab, the following buttons are always available.

	Close the information window.
	Save complete information in a zipped file.
	Opens the *Netviewer* user dialog.

4.2 Tab "Application"

Tab "Application" contains program information.

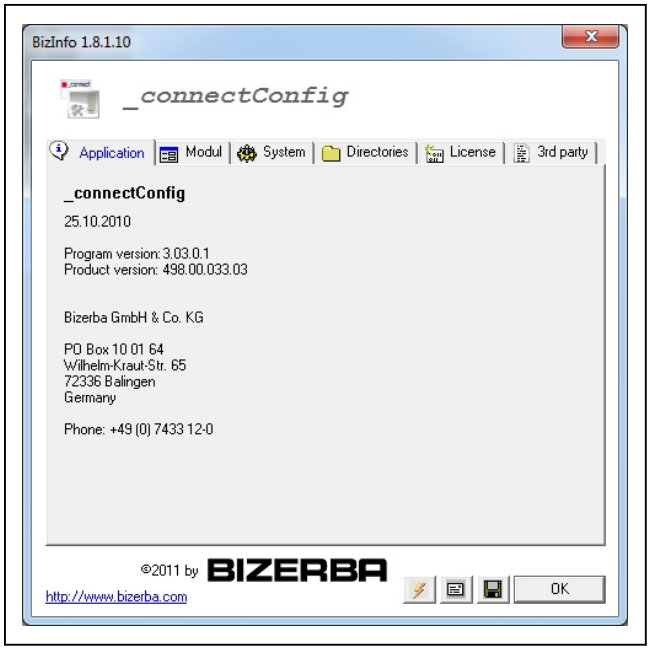


Illustration 2: Tab "Application"

4.3 Tab "Modul"

Tab "Modul" lists all programs required by the application. Missing files are marked with a red icon.

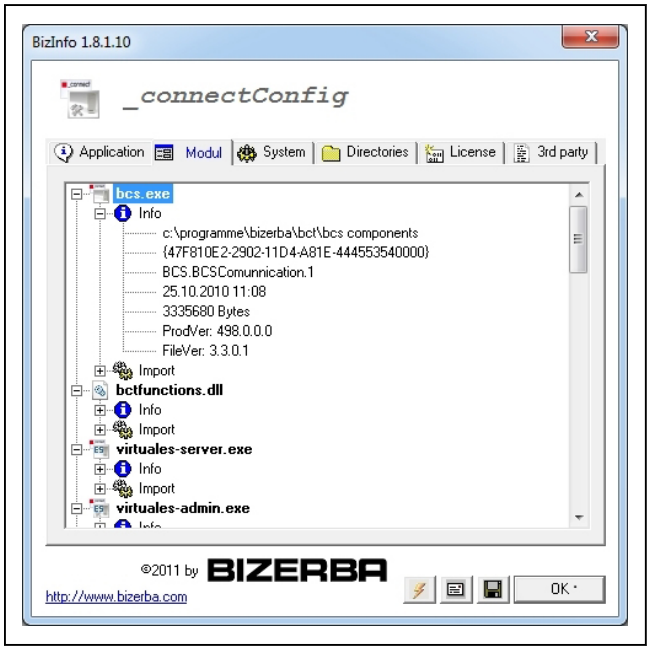


Illustration 3: Tab "Modul"

Information	Description
C:\Programme\Bizerba\…	Folder-/Directory path to file
{C50E6FA0-474E-4F66-9720-7C7637047214}	ActiveX component license key
BCS.BCSCommunication.1	Name of ActiveX license key
23.06.2006 11:45	Time of creation of program or component
3444798 Bytes	File size in bytes
ProdVer: …	Product version
FileVer: …	File version

4.4 Tab "System"

The "System" tab contains the following information:

– Information on operating system
– Information on network
– Information on processor (CPU)
– Information on system memory
– Information on the hard disks

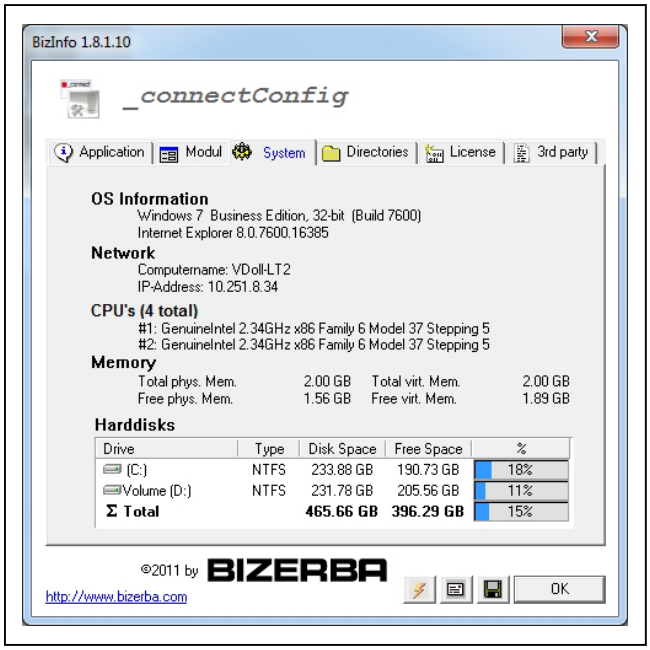


Illustration 4: Tab "System"

4.5 "Directories" tab

The "Directories" tab contains the read-only attributes of the single directories.

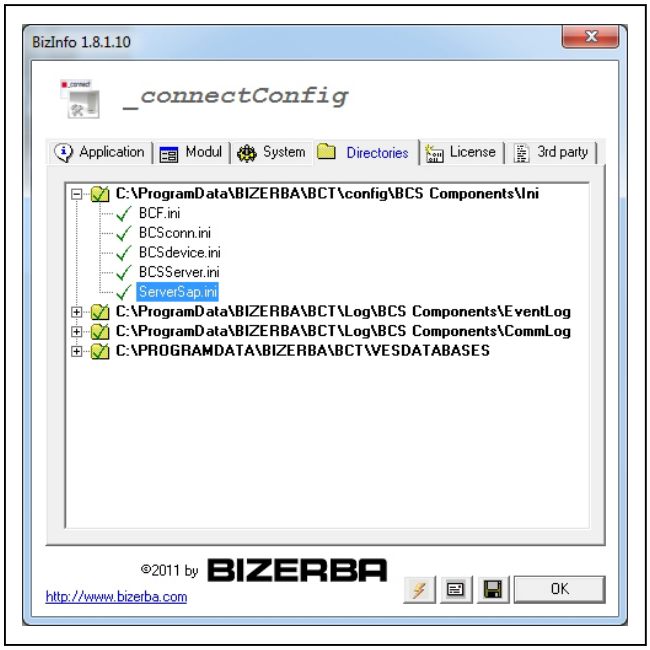


Illustration 5: "Directories" tab

C:\Dokumente und Einstellungen\...	Read-only directory.
test.txt	File cannot be read or edited.

BCSdevice.ini	Read-only file.
BCF.ini	File/directory cannot be read or edited.

4.6 "License" tab

The "License" tab contains information on all modules available within this program and on available and unused licenses. Furthermore, the "BspLicenseManager" button allows to open the BSP license manager. To activate the button, select a main item.

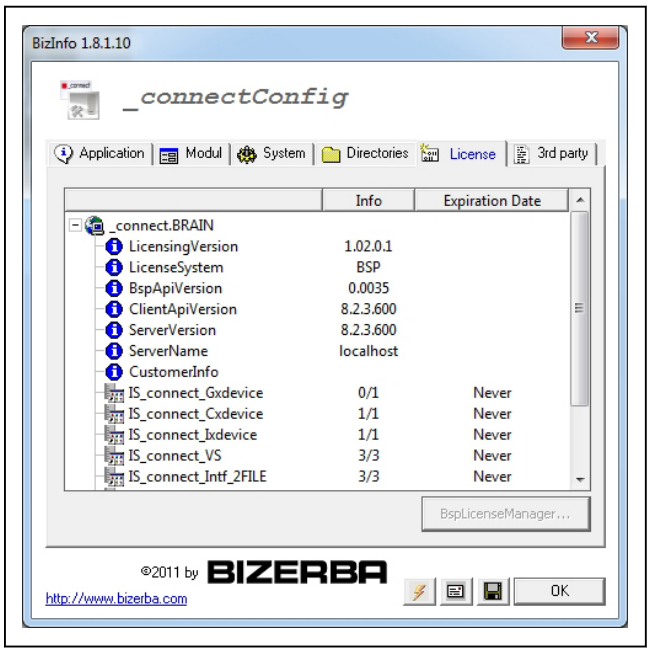


Illustration 6: "License" tab

4.7 "3rd party" tab

The "3rd party" tab contains all third party applications that are installed together with the *_connect.BRAIN* setup.

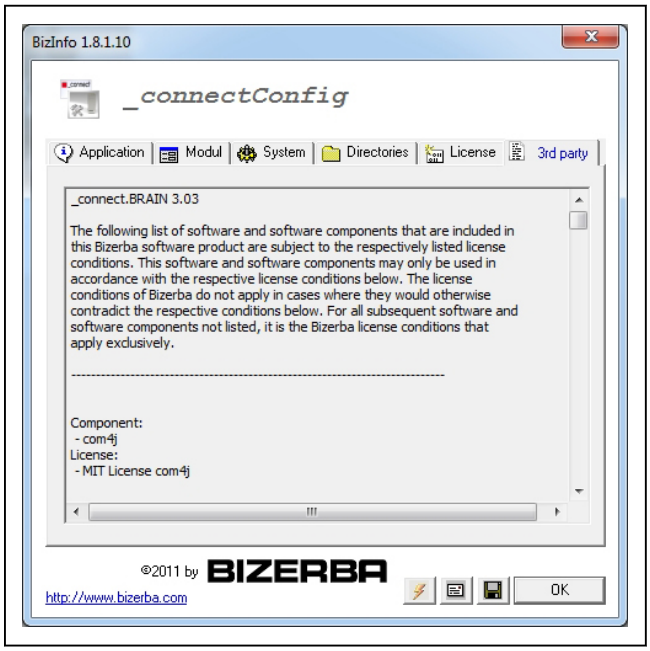


Illustration 7: "3rd party" tab

5 Device connection

5.1 Modem

5.1.1 Overview

For modem connection, two different pre-configured modems are required. The PC-side modem is called the transmitting modem, the device-side modem is the receiving modem.

5.1.2 Configuration

Create transmitting modems in the systems in modem pools and allocate them to the devices via modem connections. When opening a modem connection, *_connectServer* uses the next free transmitting modem of the assigned modem pool.

Configure transmitting modems, modem pools and modem connections in *_connectConfig*, see page 43.

5.1.3 Connection establishment

_connectServer configures the serial interface to which the modem is connected according to the settings in *_connectConfig*. It opens the serial interface and initializes the modem. The initialization process is determined by the parameters set in the modem and modem connection configurations, see page 43.

Three attempts are initially made to actuate the modem. Procedure of each attempt:

1. Wait for "Serial-Com-Delay" time (modem).
2. Send `AT\r\n` sequence to the modem.
3. Wait for positive answer. Timeout for receiving an answer is the "Response-Timeout" time (modem connection).

If the modem cannot be actuated after three attempts, the connection establishment process is terminated.

If a modem connection can be established, the parameters are sent to the modem. Not set parameters are ignored. Procedure:

1. Transmit "Pre-Dial-String" (modem connection).
   Timeout is the "Response-Timeout" (modem connection).
2. Transmit "Pre-Dial-String2" (modem connection).
   Timeout is the "Response-Timeout" (modem connection).
   For Bizerba modem configurations, this string is normally not required. It has been created because some modem commands (e.g. `reset`, `template loading`) can only be separately sent to the modem and not together with other commands.
3. Transmit "Pre-Dial-String" (modem).
   Timeout is the "Response-Timeout" (modem connection).
4. Transmit "Pre-Dial-String2" (modem).
   Timeout is the "Response-Timeout" (modem connection).
5. Transmit "Dialstring" (modem connection).

6. Timeout is the "Dial-Timeout" (modem connection).
7. Transmit "Post-Dial-String" (modem connection).
   Timeout is the "Response-Timeout" (modem connection).
   The "Post-Dial-String" of the modem connection is not a modem command (no leading AT). It can be used to configure a serial switch after the modem, for example.
8. Transmit "Post-Dial-String" (modem).
   Timeout is the "Response-Timeout" (modem connection).

5.1.4 Connection clear-down

Before closing the modem connection, the below mentioned commands are transmitted to the modem. Not set parameters are ignored.

1. Wait for "Silent-Delay" (modem).
2. Transmit `+++`.
3. Wait for 200 ms.
4. Transmit `HO`.
   Timeout is the "Response-Timeout" (modem connection).
5. Wait for "Hook-Off-Delay" (modem).
6. Transmit "Post-Hook-String" (modem connection).
   Timeout is the "Response-Timeout" (modem connection).
7. Transmit "Post-Hook-String" (modem).
   Timeout is the "Response-Timeout" (modem connection).

The serial interface is then closed.

5.2 ADDI-DATA I/O board MSX-E1516

_connect.BRAIN supports the ADDI-DATA I/O board *MSX-E1516*. *_connectServer/Digital-IO* and the I/O board communicate via Ethernet.

According to ADDI-DATA, the I/O board must not be used as safety resource. This means, that inputs or outputs in safety relevant areas must not be controlled by this I/O board.

About installation and configuration

When installing and configuring the I/O board, make sure that only one computer communicates with the I/O board.

Access of more computers or *_connect.BRAIN* installations to the I/O board is not supported. It may cause problems when controlling outputs or signaling inputs.

About operation

Make sure that the network load between _*connectServer/DigitalIO* and the I/O board does not become excessive during operation.

If the network load is excessive, inputs and outputs may be controlled too slowly and the I/O board watchdog (monitoring program) will report an error. In this case input and output control is no longer reliable.

6 Use on terminal servers

6.1 Citrix terminal server

To ensure that *_connect.Brain* can work with the licensing, the client must be located in the same domain as the server. Otherwise *_connect.Brain* works in the developer mode only.

6.1.1 Installing _connect.Brain on a Citrix terminal server

To install *_connect.Brain* on a Citrix terminal server, proceed as follows:

- "Start" / "Settings" / "System control" / "Software" / "Install"
- Answer the question whether *_connect.BRAIN* shall be available on all clients with Yes or No.

In the event of auto installation, *_connect.BRAIN* has to be registered on the client again.

6.1.2 Configuring user accounts for _connectServer

Under *DCOM configuration*, the *BCS* configuration has to be set to "user starting application".

- "Start" / "Settings" / "System control" / "Administration" / "Component services"
- "Component services" / "Computer" / "Workstation" / "DCOM configuration"
- Call up context menu for "BCS".
- Select "Properties".
- Call up "Identity" tab.
- Activate "User starting the application".

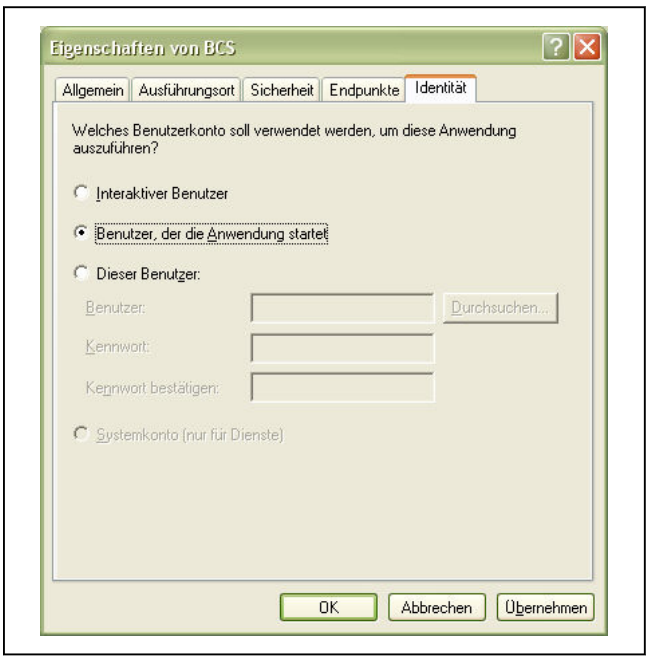


Illustration 8: *BCS* properties

6.1.3 Settings in the registry

If the BCS is always active on the terminal server, the following key can be entered in the registry, so that the last Citrix application ends the BCS:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Citrix\wfshell\TWI`
A new character string, called "LogoffCheckSysModules" is created, whose value is the name of the task. In this case: "*BCS.exe*".

6.1.4 Using a serial port in a Citrix session

A batch file must be executed in the Citrix session, which maps the COMx port, for example: "net use COM1: \\client\COM1"). Subsequently, "BCS" is able to access the client serial port.

6.2 Windows Terminal Server

Installation under Windows Terminal Server is supported for Microsoft Windows 2000 and 2003 servers.

6.2.1 Installing _connect.Brain on a Windows terminal server

To install the program on a Windows terminal server, proceed as follows:

When installing or changing an installation, ensure that no user or system service is using any part of the installation. Access can be monitored in the computer administration under "System" / "Released folder" / "Open files".

If, once applications have been properly closed and services ended, there are still files being accessed, it is possible to cancel access here.

Installation via the Windows software administration

➪ Open system administration via the Windows start menu.

➪ Install program using "Neue Programme hinzufügen".

Installation via the command lines window

➪ Open command lines window via the Windows start menu.

➪ Execute `change user / install`.

➪ Start installation program using `setup.exe`.

➪ Execute `change user / execute`.

This procedure allows to install components that are not only for the present user but for all users. This information is communicated to the system.

6.2.2 Temporary network breakdown when using a serial interface

When using a serial interface at the client computer during a Terminal session, please note that if the network temporarily breaks down, *connectServer* can no longer access the serial port.

This problem can only be resolved when the connection from the application program to *connectServer* is closed and subsequently reestablished. Close the connection as follows: Call up "Close" or stop devices via *_connect2File*.

The operating systems reports that the "device" is no longer available. In this case, "Device" stands for COM port.

6.2.3 Using VirtualES when _connectServer has been installed as service

If the VirtualES server, view, or admin displays an authorization error, the access rights set on the VirtualES server (*BizMemServer*) have to be adjusted using the *dcomcnfg* Windows program.

- ➪ Open the "Component services" windows via the *Windows* system administration.
- ➪ Open "Computer" / "Workstation" / "DCOM configuration".
- ➪ Call up the "Properties" of *BizMemServer* via the context menu.
- ➪ Open the "Security" tabs and, via <Edit...>, adapt the settings for the "Start and activation rights" and "Access rights".

  User name: SYSTEM

  Authorizations: Permit

7 _connectServer

7.1 Overview

_connectServer (BCS.Exe) is the system control center for communication between programs and devices. *_connectServer* is installed as DCOM object which can be accessed by means of the DCOM interface.

DCOM (Distributed Component Object Model) is a Microsoft proprietary technology for communication among software components distributed across networked computers. It is an extension of COM (Component Object Model) which permits standardized access to software components no matter which programming language is used.

To access the *_connectServer*, you can integrate the ActiveX control element *_connectControl* (BCC.OCX) in own proprietary programs. The properties, methods and events that can address *_connectServer* are described in a separate chapter, see page 149.

7.2 Configuration

7.2.1 Configuring _connectServer

Configure *_connectServer* via the *_connectConfig*, see page 34.

The settings are stored in the following ini files:

BCSconn.ini:	Settings in the "Connections" tab.
BCSdevice.ini:	Settings in the "Devices" tab.
BCSserver.ini:	Settings in the "Common" tab.

7.2.2 Operating modes

_connectServer can be operated as application or service. By default, the program is installed as application. It is possible to change the operating mode via the command line window.

You need administration rights to change the operating mode via the command line window.

Application: *_connectServer* is activated as soon as another program establishes connection via the DCOM or COM interface. *_connectServer* is deactivated as soon as the program that established the last connection closes the connection. Since *_connectServer* is only active when used by other programs, this operating mode facilitates changing the configuration and updating.

Service: *_connectServer* becomes active when starting the operating system and remains active until the operating system is shut down. Since there will not be any delays due to repeated opening and closing of the program, this operating mode offers a better performance. The automatic monitoring of services ensures that the program will be automatically re-started after a program crash. The operation as service makes changing the configuration and updating more difficult because the service has to be manually closed and re-started.

Changing the operating mode via the command line window

- ➪ Call up the command line window via the start menu.
- ➪ In the *_connect.BRAIN* installation directory, go to the *BCT\BCS Components* subdirectory.
- ➪ Change the operating mode. The following commands are available:

`BCS.exe -unregserver`	Delete the set operating mode. A new operating mode can be set then.
`BCS.exe -regserver`	Operate program as application.
`BCS.exe -service`	Operate program as service.

7.2.3 Assigning user accounts

Use the *dcomcnfg* Windows program to allocate the user account with which *_connectServer* will be started. The possible settings depend on the set operating mode of the program.

- ➪ Open the "Component services" windows via the *Windows* system administration.
- ➪ Open "Computer" / "Workstation" / "DCOM configuration".
- ➪ Call up the "Properties" of *BCS* via the context menu.

➪ Open the "Identity" tab and set the user account under which the program will be started.

Possible settings with operation as application:

"The interactive user.":	Use the account of the user that is currently logged on the operating system.
"User starting the application.":	Use the account of the user who starts the application. This can be a different user than the one currently logged on the system if e.g. the application is started with `RunAs` in another user context.
"This user.":	Use a preset user, no matter which application requires access.

Possible settings with operation as service:

"This user.":	Use preset user, however individually defined.
"System account (only for services).":	Use user account of operating system. Default setting for service.

7.2.4 Configuring operation in the network

The following configuration modifications are required to allow remote access to the *_connectServer* via a network.

Make the settings centrally using the domain controller or individually on every computer connected in the network.

– Create a new user group (hereinafter referred to as *Bizerba*) and a new user (hereinafter referred to as *_connect.BRAIN*). Allocate the new user to the user groups *User*, *Main user* and *Bizerba*.
  When creating a user, please observe the following:
  – The user is not authorized to change the password.
  – The password must not expire.
  – The password must be active.

As an example, the following description uses the *Bizerba* user group and the *_connect.BRAIN* user. If other names are assigned, the following descriptions apply accordingly to the user group and the user saved with the selected name.

– Allocate existing and new users that shall have access to *_connectServer* to the *Bizerba* user group.
– Adapt the DCOM configuration to the workstation and *_connectServer* as described below.

Adapt DCOM configuration to workstation:

➪ Open the "Component services" windows via the *Windows* system administration.
➪ Open "Computer".
➪ Open the "Properties" via the context menu of the workstation and make the following settings:

Settings in the "Standard properties" tab	
"Activate DCOM (Distributed COM) on this computer"	Activate.
"Standard authentication level"	Set "Connect".
"Standard identity change level"	Set "Identify".

Settings in the "COM security" tab	
"Access rights"	Via <Edit limits>, add the *Bizerba* user group with all activated rights.
"Start and activation rights"	Via <Edit limits>, add the *Bizerba* user group with all activated rights.

Adapt DCOM configuration to *_connectServer* (*BCS*):

➪ In the "Component services" window, open "Computer" / "Workstation" / "DCOM configuration".

➪ Via the *BCS* context menu, call up "Properties" and make the following settings:

Settings in the "General" tab	
"Authentication level"	Set "Standard".

Settings in the "Security" tab	
"Start and activation rights"	Select "Adapt" and, via <Edit>, add the *Bizerba* user group with all activated rights.
"Access rights"	Select "Adapt" and, via <Edit>, add the *Bizerba* user group with all activated rights.
"Configuration rights"	Select "Adapt" and, via <Edit>, add the *Bizerba* user group with all activated rights.

Settings in the "Identity" tab
Select "This user." and enter the user *_connect.BRAIN* with the corresponding password.

8 _connectConfig

8.1 Overview

_connectConfig is used to configure the following applications:

– *_connectServer* - Bizerba communication server (BCS)
– *_connect2File* - File based communication interface
– *BHI* - Bizerba host interface

8.2 Starting the program

➪ Call up *_connectConfig* via the start menu.
➪ Answer the question "Load Data from _connectServer ?" with <yes> to download the current configuration from the *_connectServer* (BCS).

or

➪ Answer the question with <no> to create a new configuration.
➪ If necessary, confirm the message that the new configuration has been successfully imported.

The start window of the program appears.

8.3 Program structure

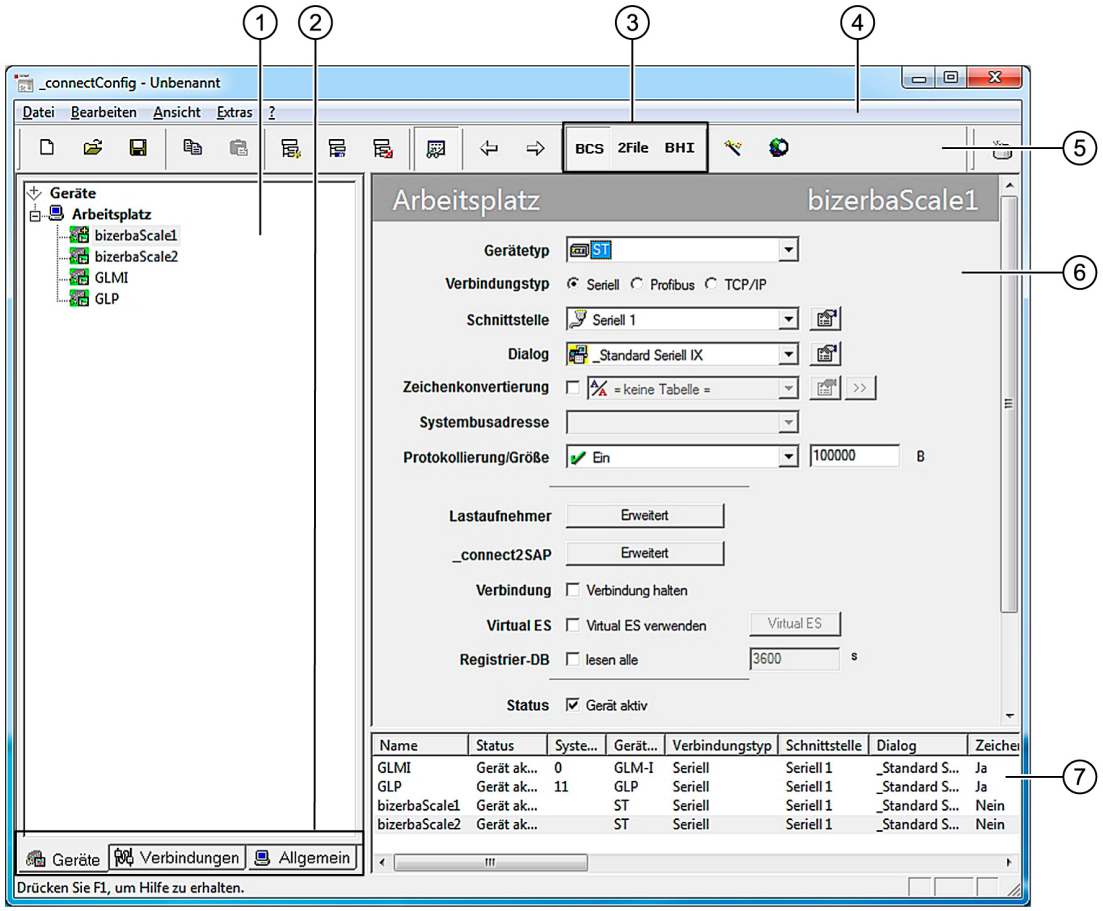


Illustration 9: User interface *_connectConfig*

1. Navigation area
2. Tabs in the navigation area
3. Buttons for configuration selection
4. Menu bar
5. Tool bar
6. Work area
7. Overview area

The user interface contains the following areas, in addition to typical *Windows* elements:

Navigation area:	Select content for editing. The navigation area varies according to the selected configuration and is split up in tabs.
Work area:	Configure devices and connections.
Overview area:	Display settings of similar devices or connections in a tabular structure. Whilst editing a serial interface for example, the overview area shows a table with the settings of all created serial interfaces.

Select one of the three possible configurations via the toolbar buttons:

BCS	<BCS> Configuration *_connectServer* - Bizerba communication server
2File	<2File> Configuration *_connect2File* - File based communication interface
BHI	<BHI> Configuration *BHI* - Bizerba host interface

8.4 Menu bar and toolbar functions

Functions for creating new item as well as rename, delete, copy, and paste functions are available via the corresponding context menu.

Functions in the "File" menu

"New"		Create new configuration.
"Open"		Open existing configuration.
"Save"		Save open configuration under current name.
"Save as..."		Save open configuration under a freely selected name.
"_connect.BRAIN Import/Export" / "data from _connect-Server"		Establish server connection and load current server configuration in the *_connectConfig* work area.
"_connect.BRAIN Import/Export" / "data to _connectServer"		Establish server connection and send the configuration that is currently being edited in *_connectConfig* to the server.
"Last file" or for example "1: C:\Bizerba\...\Beispiel.BCS"		Open one of the last saved configurations. To this end, the menu contains up to four items with progressive number, part of the path and the file name. If there is not any configuration available, it contains only the grayed out "Last file" menu item.
"Exit"		Close *_connectConfig*.

Functions in the "Edit" menu

"New Ins"		Add new device or new connection. The function is available when a structural level is marked in the navigation area, e.g. "Serial".
"Rename F2"		Rename device or connection.
"Delete Del"		Rename device or connection. The function is available when a device or connection is marked in the navigation area. Before deletion, a query appears.
"Copy CTRL+C"		Copy device or connection to clipboard. The function is available when a device or connection is marked in the navigation area.
"Paste CTRL+V"		Paste device or connection from clipboard into work area. The function is available as soon as a device or connection has been copied to the clipboard.

Functions in the "View" menu

"Toolbar"		Show or hide toolbar.
"Status bar"		Show or hide status bar.
"Overview"		Show or hide overview below the work area.
"Tabpages" / "_connect2File-Files"		Show or hide "Files" tab in the *_connect2File* configuration, see page 50.

Functions in the "Extras" menu

"Language"		Set user language. The language information is saved as text files in the *..\Programme\Bizerba\BCT\BCS Configuration\language* directory. The "Language" drop down list contains the names of these text files as options.
"Wizard"		Call up assistant for creating devices, see page 57.
"Import devices"		Create links to devices of other *_connect.BRAIN* clients, see page 59.
"Conversion tables" / "Import from file"		Import conversion tables from a file, see page 42.
"Conversion tables" / "Export into file"		Export conversion tables to a file, see page 42.

Functions in the "?" menu

"Info about _connectConfig"	Show program information, see page 14.

Functions that are only available via the toolbar

	Back to the previous tab.
	Go to the next tab.
	Configure _connectServer (BCS), see page 34.
	Configure _connectServer (BCS), see page 49.
	Configure BHI, see page 55.
	Test connection, see page 37.
	Display of modifications to the configuration. Symbol is grey: Configuration not modified since last saving. Symbol is blue: Configuration modified since last saving.

8.5 Functions in the work area

The following buttons are available in the work area at different program positions.

	<Configure object> **Example:** Configure the interface that is assigned to a modem.
<...>	Open "File-Selection" window.
<Back>	After configuring the assigned object, return to the original work area. **Example:** After configuring an interface, return to the modem configuration the interface is assigned to.
<Default>	Restore default settings in all fields of the current work area.
<Advanced>	Show extended settings of an object.

8.6 _connectServer Configuration

Use the <BCS> buttons to configure the *_connectServer* program.

8.6.1 "Common" tab

Application

Make individual settings for *_connectServer* here.

Settings	
"PC-Address:"	PC address of PC that communicates with devices, functioning as *_connectServer*. Value range: 1-127
"Error log level:" / "Size"	Kind of errors to be logged (left) and maximum size of log file (right). Value range for dropdown list (left): "User": All error messages are displayed. "Normal": Error message of the "Normal" level are displayed. "Debug": Error message of the "Debug" level are displayed. "Warning": Error message of the "Warning" level are displayed. Value range for input field (right): 1-10,000,000 bytes
"Delete Protocolfile"	Number of days after that commlog files will be deleted. With activated registration, commlog files save the data transfer between *_connectServer* and the device. Value range: 1-90 days
"_connect.BRAIN-language:"	Language in that *_connectServer* logs and outputs error messages. The dropdown list may contain customer-specific languages in addition to standard languages.

_connect2SAP

Make the settings for the *SAP* system interface here. The *SAP* system is accessed via three levels that can be individually configured:

– Frontend (*BCT2SAP Frontend*)
– Registry (*BCT2SAP Registry*)
– Spooler (*BCT2SAP Spooler*)

Settings	
"Device for function-module Z_RFC_BCT"	Settable for "Frontend" and "Registry". Selection of a device created on *_connectServer* for assignment in the *SAP* system. If "No device" is set, the device is unknown in the *SAP* system.
"SAP Destination"	Settable for "Registry" and "Spooler". Name of functional component in the *SAP* system (program ID in the RFC destination (SM59)).
"SAP Host"	Settable for "Registry" and "Spooler". IP address of the *SAP* host.
"SAP Gateway"	Settable for "Registry" and "Spooler". Gateway of the *SAP*system.
"SAP Reconnect Time"	Settable for "Server" and "Spooler". Time interval in seconds between two attempts to establish a connection to the *SAP* system.
"delete SAP Trace"	Settable for "Server" and "Spooler". Delete or do not delete *SAP* trace. Specify in the SAP system whether trace files have to be created or not.

_connectScannerWI

With *_connectScannerWI*, a scanner can be used in the keyboard wedge so that data is transferred to the active program via keystrokes (e.g.*Notepad*), see page 128.

Make the settings for these scanning operations here.

Settings	
"Scanner"	Select scanner.
"End of Line"	Select control character for marking the line end.

Settings	
"EraseChar"	Select the control character that is to be deleted from the data flow.
"Replace text %2u"	Each text contained in the left column is substituted by the control character selected in the same line of the right column.

Testing connections

This function is used to check connection to single devices or all connected devices. Prior to checking, the current configuration has to be send to the *_connectServer*.

Start the connection test via the menu or using the following icons of the toolbar:

	Test connection.

➪ In the navigation area of the "Common" tab, call "Connection test".

or

➪ Click the corresponding icon of the toolbar.

➪ Answer the "Save changed document to _connectServer-Server ?" question with <Yes>.

➪ Enter password.

➪ <Connect>

➪ Confirm the message stating that export was successful.

➪ In the editing area, select the device whose connection is to be tested.

or

➪ Activate the "All devices" control field.

➪ <Test>

The connection test result is shown in the "Common" table next to the relevant device.

8.6.2 "Connections" tab

The "Connections" tab is used to define and configure connections, dialogs, conversion tables, modem connections, and local IP addresses for device connection.

Configuring serial connections

Settings	
"Assignation"	Assignment to a device family. The device-specific default settings and restrictions of this device family are taken in consideration.
"Port"	Assignment of a physically existing serial interface. **The name of the serial connection that was created in *_connectConfig* can be freely selected and does not need to match the name of the physical interface.**

Settings	
"Baudrate"	Speed of the interface.
"Databits"	Number of data bits. Value range: 4; 5; 6; 7; 8
"Parity"	Parity. Value range: – no parity – odd – odd – Mark, i.e. always 1 – Space, i.e. always 0
"Stopbits"	Number of stop bits. Value range: 1; 1,5; 2
"Protocol"	Selection of a protocol for controlled data transfer. Value range: – "no handshake" (no protocol) – "Xon/Xoff" (software protocol) – "RTS/CTS" (hardware protocol) – "Xon/Xoff + RTS/CTS" (mixed protocol)

Configuring profibus connections

Settings	
"Assignation"	Assignment to a device family so that device-specific default settings and restrictions are taken in consideration. **Use the GX device family also for CX devices, see page 144.**
"Card"	Assignment of a Profibus card.
"Channel"	Assignment to a Profibus channel. Value range: 1: Price labeler 1, 2: Industrial devices
"Baudrate"	Speed of the interface.

Configuring TCP/IP connection

Settings		
"Assignation"	Assignment to a device family so that device-specific default settings and restrictions are taken in consideration.	
"Type"	Transmission type. TCP_SERVER_CLIENT=client/server	
"Device-IP"	IP address of the device used.	
"Hostname"	Specification of the host name instead of the IP address. If the host name is known to the DNS (Domain Name Server), the corresponding IP address can be displayed in the "Device-IP" field by means of the <IP> button.	
"Port"	Local TCP/IP port for communication with the device. Use port number 1024 or higher because the area 0..1023 is reserved for standard programs.	
"Local IP address"	IP address that is locally to be used for communication with the device. If the entered IP address is selected whilst establishing a TCO/IP connection, communication with the device is performed via this IP address. If nothing is defined, the first found IP address is used. **Indication of the local IP address is necessary for error-free communication between the device and the PC when the PC has several IP addresses.** **When using the "Local IP address" setting, make sure that a local IP address has been configured for all devices with Ethernet configuration. Otherwise Ethernet communication problems may occur.** The configuration can be checked in the overview area, see page 30.	
"use DHCP"	activated:	Use DHCP (Dynamic Host Configuration Protocol) for automatic determination of the device IP address. This setting can also be used without DHCP. In this case the device name must be stored in the network, e.g. in DNS (Domain Name System). Otherwise the device IP address cannot be determined.
	not activated:	No automatic determination of the IP address.
"Check of valid connection to device (ping)"	Function for service technician, cannot be set via the *_connectConfig* program.	

Interface dialogs

Standard interface dialogs can be viewed but not configured with the exception of modem-specific settings in case of retail scales.

Configuration of newly created interface dialogs and of copies of the standard interface dialog can be customized.

Settings for GX and IX device families	
"Assignation"	Assigned device family.
"Connection-type"	Connection type. Value range: – "Serial" – "Profibus" – "TCP/IP"
"Dialog-type"	Type of communication with the device. Value range: "no dialog": Data transfer and receipt without frame protocol. "S-dialog": Data transfer with positive (ACK) or negative (NAK) confirmation of receipt. "N-dialog": Enquiry (ENQ) prior to transmission whether data can be sent (ACK) or not (NAK). In case of permission, data transfer with confirmation of receipt (ACK or NAK).
"Blockcheck"	Block check type. Value range: "no check": No block test. "with LRC": Longitudinal redundancy check (LRC).
"Enquirychar"	ASCII code for sending enquiry in N dialog.
"Pos/Neg acknowledge"	ASCII codes for positive (ACK) or negative (NAK) confirmation of receipt.
"Start characters 1/2"	ASCII codes for marking a block start.
"End characters 1/2"	ASCII codes for marking a block end.
"Delimiter"	ASCII code for a delimiter.
"IgnoreChar"	**With IX devices only.** ASCII code for identification of characters to be ignored.

Settings for GX and IX device families	
"Retry"	Number of repetitions in the event of connection errors.
"Timeout in ms:"	Maximum waiting time when establishing connection. Value range: 1000…10.000 ms

LW device family settings	
"Assignation"	Display of the assigned device family.
"Dialog-type"	Dialog type. Value range: "SCHDLC": HDLC conform dialog. "SCTCP": TCP/IP conform dialog.
"Init-command"	Command character for initialization.
"Init-name"	Identifier.
"Retry"	Number of repetitions in the event of connection errors.
"Modem-Type"	**With serial connection only (dialog type "SCHDLC").** Modem type. Value range: 0…99
"Phone-No."	**With serial connection only (dialog type "SCHDLC").** Telephone number for modem connection.
"Password"	**With serial connection only (dialog type "SCHDLC").** Password for protection of setting.
"Blockcheck"	**With serial connection only (dialog type "SCHDLC").** Block check type. Value range: "no check": No block test. "LRC": Longitudinal redundancy check (LRC). "16-Bit CRC": Cyclic redundancy check (CRC).

Creating and editing conversion tables

Character replacements for conversion operations are defined in character conversion tables. Characters that are not listed in the conversion table are transferred unchanged. Conversion tables for the most important standard conversions are part of the program.

Each character conversion table consists of two parts:

<_connectServer >> Device>:	Conversion from PC to device
<Device >> _connectServer>:	Conversion from device to PC

Creating a new character conversion table

➪ In the navigation area, press "conversion tables".

➪ In the <Edit> menu, call up the <New> menu item.

or

➪ In the context menu, call up the "New Conversion table".

A new conversion table is displayed in the navigation and work area.

8

Creating character conversion table

➪ Select the device familiy for which the conversion table shall be valid in the <Assignation> dropdown list.

➪ Enter the required character replacements for both conversion directions.

Entering character replacements

➪ Add a line to the table using <Add>.

➪ Select the character to be replaced in the dropdown list on the left below the edited table part.

➪ Select the replacement character in the dropdown list on the right below the edited table part.

Deleting lines from a conversion table

➪ Select the line and press <Delete>.

Exporting and importing conversion tables

Conversion tables can be exported and imported. The conversion tables are saved as *.CNV* files in the system and imported from there. These files can contain several conversion tables.

This feature makes it possible to maintain different conversion settings under the same table name for international applications.

Exporting conversion tables

➪ "Extras" / "Conversion tables" / "Export into file"

➪ Select the conversion table to be exported and click <OK>.

➪ Select a path in the "File-Selection" window and enter a file name.

Several exported conversion tables are saved in the same file. When reimporting this file, you'll obtain single conversion tables again.

Importing conversion tables

➪ "Extras" / "Conversion tables" / "Import from file"

➪ Select the file to be imported in the "File-Selection" window.

➪ Import using <Open>.

The conversion table appears in the *_connectServer* configuration on the "Connections" tab. It can be edited and assigned to devices.

Update the display to be able to see the imported conversion table. To do so, change the tab and return to previous one.

Creating and editing modems, modem pools and modem connections

Assign a modem connection to each device connected by means of a modem. The modem connection itself refers to a modem pool. When establishing connection to a device, *_connectServer* uses the next free modem of the assigned modem pool.

Each modem pool consists of at least one modem. Each modem is assigned to a modem pool.

To be able to establish a modem connection, you have to create at least one modem pool with at least one modem first.

Modem settings	
"Serial-Com-Delay"	Delay time in ms for transmission.
"Hook-Off-Delay"	Delay time in ms until acceptance of a connection.
"Silent-Delay"	Delay time in ms when closing the modem connection to set the modem to the command mode.
"Pre-Dial-String"	Prefix for dialing
"Pre-Dial-String2"	Further prefix for dialing if the first prefix is a separate command which must be followed by another command.
"Dial-Timeout"	Postfix for dialing.

Modem settings	
"Post-Hook-String"	Command after accepting a connection.
"Interface"	Serial interface.

Modem connection settings	
"Log messages"	Allow or refuse event logging.
"Dialstring"	Command for selecting a phone number.
"Dial-Timeout"	Maximum waiting time until successful establishment of connection in ms.
"Response-Time-out"	Maximum waiting time until successful initialization of connection with remote station in ms.
"Pre-Dial-String"	Prefix for dialing
"Pre-Dial-String2"	Further prefix for dialing if the first prefix is a separate command which must be followed by another command.
"Post-Dial-String"	Postfix for dialing.
"Post-Hook-String"	Command after accepting a connection.
"Modem-Pool"	Modem pool used.

Creating modem in new modem pool

➪ In the navigation area, press "Modem-Pools".

➪ In the "Edit" menu, call up the "New" menu item.

or

➪ In the context menu, call up the "New Modempool".

A new modem pool with a new modem is created.

Creating modem in existing modem pool

➪ Select the modem pool in the navigation area.

➪ In the "Edit" menu, call up the "New" menu item.

or

➪ In the context menu, call up the "New Modem".

A new modem is created.

Creating a modem connection

➪ In the navigation area, press "Modem-connections".

➪ In the "Edit" menu, call up the "New" menu item.

or

➪ In the context menu, call up the "New Modem-connection".
A new modem connection is created.

The newly created items in the navigation area can be renamed, deleted and copied using the corresponding menu functions, see page 31.

Creating and editing local IP addresses

Settings	
"Local IP address"	IP address that is locally to be used for communication with the device. If the entered IP address is selected whilst establishing a TCO/IP connection, communication with the device is performed via this IP address. If nothing is defined, the first found IP address is used. **Indication of the local IP address is necessary for error-free communication between the device and the PC when the PC has several IP addresses.**

Configure inputs and outputs

Configure here the inputs and outputs of the board. The board has 16 ports forming a channel in pairs. Each of the 8 channels can be individually set.

The circles filled in black on the left show the set inputs and on the right the set outputs. In order to change the configuration click into opposite circle per channel.

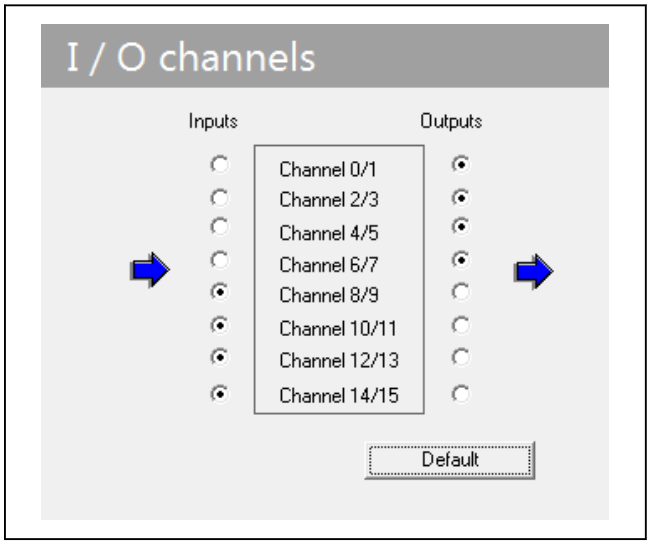


Illustration 10: Inputs and outputs

8.6.3 "Devices" tab

Assign existing connections during device configuration. Depending on the device type or family, the following settings can be made.

The CX (CWM, CWE, CWP) device family is based on the GX family. The CX device settings correspond to those of the GX devices and are not managed separately by the program. Consequently, you find CX devices in *_connectBRAIN* always under the GX device family.

Settings	
"Devicetype"	Specification of the device type so that device-specific default settings and restrictions are taken in consideration.
"Connection-type"	Kind of physical connection between PC and device. Value range according to device family: GX: "Serial", "Profibus", "TCP/IP", "Modem" IX (except for NT scale): "Serial", "Profibus", "TCP/IP" IX (NT scale): "DCOM" LX (except for CE): "Serial", "TCP/IP" LX (CE): "RAS", "Ethernet" SX: "DCOM" FX (except for ADDI-Data I/O card): "Serial", "TCP/IP" FX (ADDI-Data I/O card): "DCOM"
"Interface"	**GX, IX, LW, FX:** Select configured connection via dropdown list. The list is available after entering the connection type. Via the button on the right next to list field, the selected connection can be configured, see page 37.
"Dialog"	**GX, IX, LW, FX:** Select device dialog. The device dialog is the protocol between PC and device. The dialogs shown in the dropdown list depend on connection and device type. Via the button on the right next to list field, the selected dialog can be configured, see page 37.
"I / O channels"	**only ADDI data MSX E-1516 IO board** Select configuration of inputs and outputs from list of available configurations. You directly get to the configuration dialog for inputs and outputs via the button on the right next to the selection.

Settings	
"Token-Conversion"	**GX, IX:** Activate or deactivate conversion via the control field. When conversion is activated, select a conversion table in the drop-down list. The displayed conversion tables depend on the device type. Via the button on the right next to list field, the selected conversion table can be configured, see page 37. The second button on the right next to the list field opens the "Advanced properties for Token-conversion" dialog. Activate automatic conversion, when sequences that start with a backslash have to be converted automatically. Automatic conversion has the following effects, amongst others: "\n": This sequence stands for a line break and is converted to the GxNet compatible form "@0A" (ASCII character 10 or 0x0A). "\\": This sequence stands for the backslash character. Automatic conversion to "\" is performed only in the case of data transfer from the PC to the device and not vice versa.
"Systembusaddress"	**GX, IX (außer NT Scale), LW:** Select device system bus address.
"Logging/size"	**For all device types.** List field on the left: Logging type. **LW:** Deactivate logging or set a logging mode. The higher the consecutive number of the logging mode, the more information is logged. The size specification in bytes defines from which file size a new log file is created and the old file is renamed. **Others:** Activate or deactivate logging. Input field on the right: Maximum size of a logging file in byte. If this size is exceeded, a new log file is generated and the old file is renamed.
"Load Receivers"	**IX (except for NT scale):** Open "Advanced settings for Load Receivers" window. Add up to 99 load receptors with number and individual name. Edit or delete table entries.
"Software-Vol."	**GX:** Firmware version of the device. **If the program version was incorrectly indicated, problems may occur during controlling or querying of licenses which prevent correct functioning of the program.**

Settings	
"Header-Format"	**GX, FX:** Select GxNet transfer format. The following formats are available for Gx and Fx devices: – "Gx/Ix-Net(A!, I!, A?, I?)": new format, standard for all devices – "Gx-Net (!,?)": old format for compatibility with previous program *GxTools*
"_connect2SAP"	**For all device types:** Open "Advanced Settings for _connect2SAP" window. Value range: Access: Enable multiple or only single access to the device via other programs. Spontaneous telegrams: Log or do not log spontaneous telegrams of the device. Logging: Activate or deactivate logging of *SAP* interface access. Repetition attempts: Define number of repetitions in the event of connection errors, before attempts are terminated with an error message. HUPast: Select gross or netto weight type for display, see page 110.
"Connection"	**GX, IX, SX, FX:** Set via "Hold Connection" whether the device connection is to be permanently maintained or only established if required. For connection to *SAP*: Activate "Hold Connection". On terminal servers with *_connectServer* as server: deactivate "Hold Connection".
"Virtual ES"	**IX, SX:** Enable access to *VirtualES*. Program *VirtualES - Admin* can be started via <Virtual ES>. First, program *_connectConfig* sends current settings to the *_connectServer*.
"Registry-DB"	**IX:** Enable or block access to the registry database for ST devices. If required, define readout cycle in seconds. *_connectServer* reads the data from the database and forwards it as spontaneous data to the higher ranking program.
"Status"	**For all device types:** Activate check-box to enable the use of the device.

8.7 _connect2File / 2File Configuration

Use button <2File> to configure programs *_connect2File* and *2File* for file-based device communication. Scope of functions and operation of these programs are documented in a separate chapter of these instructions.

– For scope of functions and operation of *_connect2File* see page 63
– For scope of functions and operation of *2File* see page 63

The configuration of *2File* differs in a few aspects from *_connect2File*. Differences are clearly marked.

8.7.1 Naming log files

Select the log file names so that they provide information on the content type and the relevant device.

Example: *record01.txt*
The first part of the file name stands for the content type, for example
– *record* for normal data
– *error* for errors

The second part of the file name stands for the device address (01, 02, ...) or the device name.

8.7.2 "Common" tab

Here you carry out basic settings of programs *_connect2File* and *2File*.

Settings		
"Error log level:"	Define the log depth for the log file. Value range: "User": "Normal": "Debug": "Warning":	 Log all error messages. Log the error messages of the "Normal" level. Log the error messages of the "Debug" level. Log the error messages of the "Warning" level.
"Show errors as dialog"	activated: not activated:	The logged error messages are also interactively displayed. The error messages appear only in the log file.

Settings	
"GX-SendChannel"	Transmission channel for transmitting device information when the device is stopped. If the transmission channel is missing, the device cannot send any data after being stopped and reports an error. Program *_connect2File* activates the set transmission channel when establishing the connection to the GX device allowing the GX device to transmit data records to the *_connect2File* PC. When closing the connection to the GX device, the transmission channel is deactivated preventing the GX device from reporting transmission errors while *_connect2File* is not active. **The same transmission channel for sending information has to be defined in the device.**
"Files"	Specify a process for transferring received data in a regular file. Data received from the device are initially saved in a temporary file. Only after releasing the data, the temporary file is converted to a regular file. The following processes are available for this: "Rename with Move (Win-NT/2000)": The temporary file is relocated and renamed. "Rename with Copy/ Delete (Win-9x, ME)": Process for older systems. The temporary file is copied and then deleted.
"Do not move empty files"	activated: Empty reception data files are directly deleted. not activated: Empty reception data files are converted in regular files.

8

8.7.3 "Files" tab

This tab is only displayed when the "_connect2File-Files" menu item is activated in the "View" menu under "Tabpages".

Programs *_connect2File* and *2File* manage log data of the connected devices. External access to the primary log data is disabled. *_connect2File* and *2File* manage output files for access via external software. Output files are called "Backup" in the program.

Creation and management of output files are configured on tab "Files". In the navigation area you can create and configure separate log files for a classical *_connect2File* system / *2File* system and *_connect2File / 2File* systems on the clients of a terminal server.

Settings	
"Log-Path"	Path for storing log files. Enter absolute path directly or select folder via <…>.
"Filetype"	File name extension of log files.

Settings	
"Backup-Path"	Path for storing output files (output directory). Enter absolute path directly or select folder via <…>.
"Filetype"	File name extension of output files.
only *2File*	
"File format"	File format of output file in which it is stored: – **Default:** files are stored as ANSI/ASCII file. – **Unicode:** is stored as UCS-2 file and UTF16 BigEndian. – **XML:** file is stored as XML file in standard UTF-8 format.

Use the <Advanced> button to open a window with additional settings for copying and data backup ("Copy options" tab) and for accessing the output file ("Office-Applications" tab).

Settings in the window "Advanced properties for log-files" **Tab "Copy options"**		
"Max-Entry"	Number of entries after that the log data will be moved to the output file.	
"Max-Time"	Time interval after that log data will be moved to the output file.	
"Copy at"	Time at that log data will be moved to the output file. If an output file with an identical name already exists in the output directory and cannot be overwritten due to the configuration (see settings in area "Backup-File") or if *_connect2File*is not active at the set time, no action will take place. Not before the set time is reached again, a new attempt is made to move the log data.	
"Retry"	activated: not activated:	Repeat copying cyclically after errors. Do not repeat copying after errors.
"Retry-Delay"	Time interval between two copying attempts in ms.	
"Header-Line"	activated: not activated:	Move log data to the output file after each header change. Move log data to output file independently from header change.
Area "Log-File" **(only _connect2File)**		

Settings in the window "Advanced properties for log-files" **Tab "Copy options"**		
"Update from the System"	Select process for updating the log file. Value range:	
"Update after every entry"	"Update from the System":	The log data are buffered in the system. The collected data are converted to the log file later. In comparison to updating at each new entry, the system is less loaded and the log file is less updated.
	"Update after every entry":	The log file is updated after each write access (flush). Thus, it is always updated. However, this process may reduce total system speed.
Area "Backup-File"		
"no overwriting of file"	Enable or disable overwriting of existing output files. If overwriting of output files is enabled, not collected output files can be overwritten with more recent data. The previous state of the output file cannot be called up any longer.	
"overwriting of file"		

Settings in the window "Advanced properties for log-files" **Tab "Office-Applications"** **(only _connect2File)**	
"Application"	Program that is used to access the output file. Value range: – "none" – "Excel" – "Word" – "Access"
"File"	File that is used to access the output file (excel, word or access file with macro). Enter file name directly or select file via <…>.
"Macro"	Name of the macro that is used to access the output file.

8.7.4 "Devices" tab

Device specific settings are split up into two areas:

Conventional area: The classical area includes a *_connect2File* system and a *2File* system. The here created devices are visible for all clients and can be used on all clients.

Terminal server area: The terminal server area contains a *_connect2File* system and a *2File* system for terminal server clients. The here created devices are only visible on the respective client and can only be used by this client.

Assigning devices or changing assignments

➪ In the navigation area select main entry "_connect2File-System" or "2 file system" or one of the clients created under main entry "_connect2File system terminal server" or "2file system terminal server clients".

➪ <Change>

Assign devices

➪ Select one or more devices in the "Available Elements" area.

➪ Assign devices with < **>>** >.

Remove assignment

➪ Select one or more devices in the "Selected Elements" area.

➪ Remove assignment of devices with < **<<** >.

Save changes

➪ <OK>

Configuring device - "Input files" tab

Configure the communication between PC and device here. The communication is done via one of the following programs:

– *_connect2File* / *2File* (default)
– *BLD* (first user-defined program)
– *WinCWS* (second user-defined program)

Settings	
Settings of selected program *_connect2File*, *2File*, *BLD* or *WinCWS*	
"Input"	Path and name of the file that contains the data to be sent to the device.
"Error"	Path and name of the file that records defective data records, e.g. transmission and syntax errors, or header not detected, see page 71. The defective data records are not sent to the device.
"Dummy-files"	Reserved for future application.
only *2File*	
"File format"	File format of output file in which it is stored: – **Default:** files are stored as ANSI/ASCII file. – **Unicode:** is stored as UCS-2 file and UTF16 BigEndian. – **XML:** file is stored as XML file in standard UTF-8 format.

Generally valid settings	
"Read-Interval"	Time interval between two read accesses to the input file in s.
"Wait-Time"	Waiting time between two data records to be transmitted to the device in ms.

Generally valid settings		
"Error-File"	activated:	Create error file only in the event of error.
	not activated:	Always create error file.
	Even empty error files can be used to determine when-on the basis of the moment of their creation-processing of an input file started.	
"Devicestop"	activated:	In the event of a device stop, the input file that is being processed will be deleted.
	not activated:	In the event of a device stop, the input file that is being processed will be kept.

Configuring device - "Outputfiles" tab

Select the data that the device will send to the PC. The setting is made column per column for the log files created on the "Files" tab, see page 50. Distinguish between output and event file settings by selecting the relevant tab. Set for the output file which device data will be transmitted to the PC. Make the settings for the device data that activates the log file relocation in the output directory, in the relevant event file.

Use the following buttons for further processing:

<Files>:	Log file configuration, see page 50
<Replace assignation>	Transmit assignment of output data from one log file to another one. Assignments can only be transmitted to log files without proper assignment. During transmission, assignments are deleted from the original file.

Replacing the assignment

This function is only available when at least one log file with assignments and one log file without assignments have been created.

➪ Click the <Replace assignation> button in the "Outputfiles" tab.

➪ In the "Current assignation" column of the "Replacement of files" window, select the log file from which assignments have to be transmitted.

➪ <Replace assignation>

➪ In the "File-Selection" window, select the log file in which assignments have to be transmitted.

➪ <OK>

The selected file appears in the "Replacement of files" window under "New assignation". If required, remove the assigned log file from the view using <Release assignation>.

➪ <OK>

The assignments are removed from the original log file and transmitted to the selected log file.

Configuring device - "Application parameters" tab

Make here general settings for handling devices.

Settings		
"Start-Options"	activated:	After starting *_connect2File*, establish automatically a connection to the device.
	not activated:	The connection to the device has to be started manually.
"Telegramms"	activated:	Receive also events initiated by the device (spontaneous telegrams).
	not activated:	Receive only data explicitly requested from the device.
"Memocard"	activated:	Read the storage medium in the device cyclically. Enter the reading interval in the input field in s.
	not activated:	Do not read the storage medium in the device.
"Autostop"	activated:	Stop device when receiving a total telegram of the relevant total. Then the device has to be enabled by the superordinate program.
	not activated:	Do not stop device when receiving a total telegram of the relevant total.

8.8 BHI Configuration

Via the <BHI> interface, the Bizerba host interface *BHI* is configured.

BHI is a host interface for *Windows* PCs for connecting the following devices:

– SC retail scales
– SW retail scales
– BS retail scales
– CE retail scales
– GX price labelers

Host definition and program sequences are created by the program *BHE* (Bizerba Host Editor) that is not documented here. The host data can be processed in ASCII, CSV or XML file format. *BHE* contains data converters and data transmission programs (BCS-light and SXCom).

BHI is only required in connection with the *WinCWS* program and has to be configured only when this program is used.

Settings	
"System-Type/No."	The system type defines the device types that can be contained in a BHI system. Value range: SC/SW/BS, CE, GX The system number is the number of a subsystem. A subsystem is a logic unit comprising a single device or a group of retail scales or price markers, which are supplied with data or from which data are called up separately during communication. Value range: 1-999
"Master devices"	The master device is a device in a subsystem, via which communication takes place with *BHI*. If more master devices are created, these will be used as master devices for communication in the event of failure of the first master device. Value range: All devices of the selected system type that are created in *_connect.BRAIN*.
"Slave devices"	Other devices of the subsystem for which device-specific data shall be transmitted. If no device-specific data are to be transmitted, the list of slave devices remains empty. Value range: All devices of the selected system type that are created in *_connect.BRAIN*.
"Departments"	Department-related data (e.g. PLU data) can be filtered by department Only department-related data whose department number is specified in the department list are transmitted. If the department list is empty, all data records present in the host file are transmitted to the subsystem. Value range: 0-999

8.8.1 Adding and deleting master/slave devices and department

The procedure is for master/slave devices and departments is largely identical and corresponds to the following description, by way of example. For editing slave devices or departments, select the respective tab.

Adding master device

- Select the *BHI* system to be edited in the navigation area.
- Select the "Master devices" tab in the work area.
- Use the <Configure object> button or the enter key to open the "Master devices" window.

➪ Select the device that shall be defined as master device in the "Available Elements" area. Only devices that have been created in the system and correspond to the defined system type are displayed.

➪ < **>>** >

Adding other master devices

➪ If required, transfer other devices in the "Selected Elements" area in the same way. Alternatively, these devices may be used as master device.

Removing master devices from list

➪ Select the device to be deleted in the "Selected Elements" area and < **<<** >

Terminating editing

➪ <OK>

8.9 Wizard for creating devices

8.9.1 Creating new devices using a wizard

New devices can be generated using a wizard. The wizard guides you through the device-specific settings to the *_connectServer* and possibly to *_connect2File* and *2File*.

➪ "Extras" / "Wizard"

➪ <Next >>

➪ Select function "Create a new device" and click <Next >>.

Enter *_connectServer* settings

➪ Enter *_connectServer* settings on the following pages, see page 46. Close each page using <Next >>.

Decide whether and how the device is to be assigned to *_connect2File* or *2File*

➪ Select if the device should be added to *_connect2File* or *2File* or to none of both and click <Next >>.

The following responses are possible:

"_connect2File":	The device is added to *_connect2File*.
"2File":	The device is added to *2File*.
"no":	The device is created in *_connectServer* but is not added to the *_connect2File* or *2File* configuration. In the further procedure of the wizard the settings for *_connect2File* or *2File* are omitted.

➪ Answer the question which environment should be used and click <Next >>.

The following responses are possible:

"yes":	The device is created in the *_connectServer* configuration and assigned in the *_connect2File / 2File* configuration to the associated standard *_connect2File / 2File* system.
"yes, add for existing terminal server client":	The device is created in the *_connectServer* configuration and assigned in the *_connect2File* or *2File* configuration to the terminal server client selected from the dropdown list.
"yes, enter name of new terminal server client":	The device is created in the *_connectServer* configuration. A new terminal server client with the name entered in the associated input field is created in the *_connect2File* or *2File* configuration. The new device is assigned to the new terminal server client.

If necessary, enter *_connect2File* / *2File* settings

➪ On the following pages of the wizard enter input files, output files and application parameters for the *_connect2File* configuration, see page 52. Close each page using <Next >>.

Completing device

➪ Check the summed-up settings on the last page of the wizard. Use <Back> to return to the previous page and correct settings.

➪ Exit wizard with <Complete>.

8.9.2 Copying devices using the wizard

Devices can be copied using the wizard.

➪ "Extras" / "Wizard"

➪ <Next >>

➪ Select function "Copy an existing device".

➪ Select the device to be copied in the list field.

➪ <Next >>

➪ Enter the device name.

➪ <Next >>

➪ Make settings for connection, see page 37.

➪ <Next >>

➪ If the copied device is assigned to *_connect2File* record input files, see page 53.

or

➪ If the copied device is assigned to *2File*, record input files, see page 53.

If the device is assigned to both systems, the device is only copied for the *2File* system! If needed, manually add copied device after copy process to *_connect2File* system.

➪ <Next >>

- If the copied device is assigned to *_connect2File* or *2File*, assign output files, see page 54.
- <Next >>
- Check the summed-up settings on the last page of the wizard. Use <Back> to return to the previous page and correct settings.
- Exit wizard with <Complete>.

8.10 Creating links to devices of other _connect.BRAIN clients

With function <Import devices> you create references to devices of other *_connect.BRAIN* clients in your current configuration. This allows a *_connect.BRAIN* client (*_connectControl*, *_connect2File*, additional program) to work in a simple way with a distributed system.

- "Extras" / "Import devices"
- Record name of *_connect.BRAIN* client from which devices are to be imported in input field "Computername" or select via button <..>.
- <Show Devices>
- Select devices for import.
- <OK>

The devices appear in the *_connectServer* configuration on tab "Devices" under main entry "Network environment" and the name of the computer. They can be viewed and used, but not configured.

9 _connectDiagnostics

9.1 Overview

_connectDiagnostics provides you with an overview of the configuration and the use of the *_connectBRAIN* program package. It accesses the *_connectServer* via the DCOM interface and finds out information about:

– Program version of *_connectServer* and other installed components
– Activated licenses for *_connect.BRAIN*
– Number of users currently connected to the server with user identification and name of the interface used
– Created devices with category, type and status

9.2 Starting the program

➪ Call up *_connectDiagnostics* via the start menu.

➪ If the start window is not displayed and the program appears only as symbol in the information area of the task bar: Click the "_connectDiagnostics" symbol using the right button of the mouse and select "Wiederherstellen" in the context menu.

The start window of the program appears.

9.3 Program structure

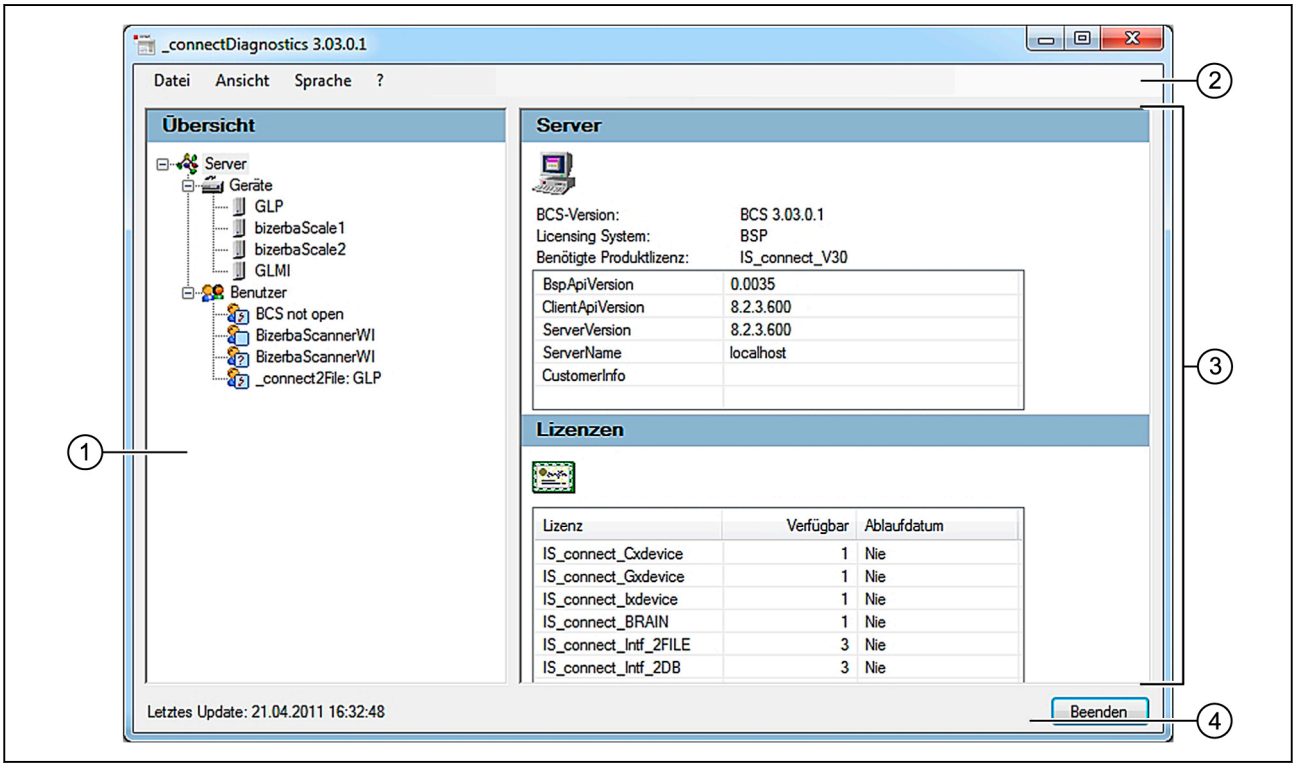


Illustration 11: User interface *_connectDiagnostics*

(1) Navigation area

(2) Menu bar

(3) Display area

(4) Status bar

The user interface contains the following areas, in addition to typical *Windows* elements:

Navigation area: Shows the devices and users logged on *_connectServer*.

Display area: Shows information on the element that was selected in the navigation area:

- Information on server (version) and licenses
- Information on device
- Information on user and interface

Status bar: Shows time of the last update.

9.4 Menu bar and toolbar functions

Functions in "File" menu

"Exit"	Terminate *_connectDiagnostics*.

Functions in menu "View"

"Refresh"	Update view to show modifications.

Functions in menu "Language"
Select here a language for the user interface.

Functions in "?" menu

"Info"	Show program information, see page 14.

10 _connect2File

10.1 Overview

_connect2File is a user-friendly file interface for transferring BxNet data and control commands in the form of ASCII files.

_connect2File is configured via program *_connectConfig*, see page 49.

Scope of functions of *_connect2File*:

- Receive spontaneous telegrams
  - Package-synchronous data (weight, price, date, time, ...)
  - Remote data (send on change)
  - Softkey entries from user-defined mode level or authorization level
  - Statistics data (totals, quantity,...)
- Acknowledge reception data according to telegram
  - The data content of a telegram determines the log file(s) to which the reception data are written. The automatic acknowledgement of total data can be suppressed for totals 1, 2 and 2 and carried out later by means of a release command, see page 55.
- Prepare log files for data transfer in event-controlled mode
- Call up macros in Microsoft Word and Excel

10.2 Starting the program

➪ Call up *_connect2File* via the start menu.

The start window of the program appears.

Recognition of the terminal server mode can be deactivated by means of the following call-up parameter: `BCF.exe /NoWtsDetection`

This is necessary, when the Windows terminal server console is accessed via *remote desktop connection* and *_connect2File* is to be run on the console. If the parameter has not been specified, *_connect2File* runs within a *remote desktop connection* on the client PC and not on the server console.

10.3 Program structure

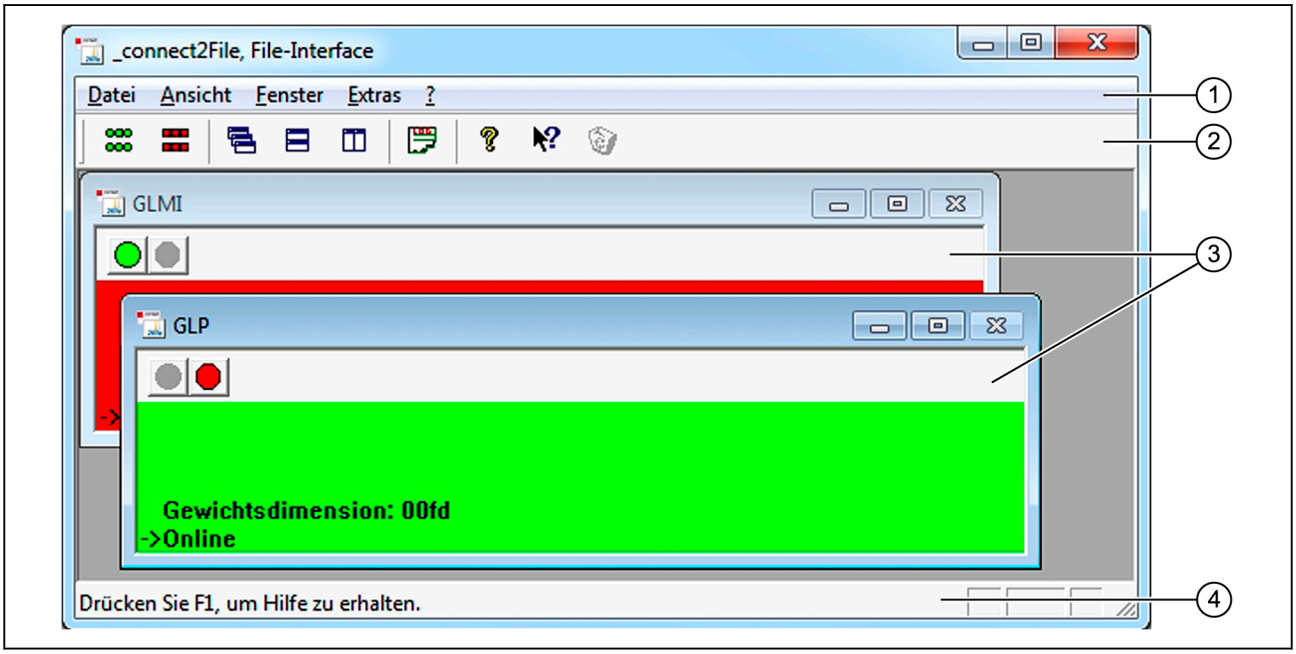


Illustration 12: User interface *_connect2File*

① Menu bar

② Tool bar

③ Connection window

④ Status bar

The user interface contains the following areas, in addition to typical *Windows* elements:

Connection window: Communication of each device is displayed in a connection window. The connection window shows the connection status and the data transfer. The background color varies according to the status:

- Yellow: establishing connection
- Cyan: waiting for switch-on message
- Green: connection open (online)
- Gray: closing connection
- Red: connection closed (offline)

10.4 Menu bar and toolbar functions

Functions in "File" menu

"Start for all devices"		Initialize access to all configured devices. Open connections and and start data exchange. Any devices that have already been started are not re-initialized.
"Stop for all devices"		Terminate data transfer for all devices. Close all connections.
"test online"		Check connection to configured devices and report result for each device.
"start"		Open connection to selected device and start data exchange.
"stop"		Terminate data transfer to selected device and close connection.
"Exit"		Close all connections and terminate *_connect2File*.

Functions in menu "View"

"logtape"		Show *_connect2File* protocol from application start.

Functions in "Window" menu

"overlapping"		Display windows in cascade format.
"arrangement vertical"		Display windows next to each other.
"arrangement horizontal"		Display windows next to each other.
"arrange symbols"		Arrange minimized connection windows at the bottom of the window.
example "1 GLP"		Consecutive number and device name of displayed windows. The active window is checkmarked. Select desired window.

Functions in menu "Extras"

"Language"	Select user interface language. The modification does not become active until the next application start.

Functions in menu "?"

"info about _connect2File"		Show application information, see page 14.

10.5 Functions in the connection windows

	Green symbol: Start device.
	Red symbol: Stop device.

10.6 File transfer

An application that provides data for the devices, runs on the host. The host application and the *_connect2File* program communicate by exchanging files in ASCII format. Data is transferred via the hard disk of the *_connect2File* computer according to a defined protocol.

The *WinCWS* program is an example of a host application. This program provides the GX device family with a master data ASCII file that is transferred to the GX device by *_connect2File*.

Sample file that transfers the text "hello GX" in the text field 1 of a GX device:

Line 1: `A!GT01`
Line 2: `hello GX`

More examples can be found under *C:\Programme\Bizerba\BCT\Samples\BCF Files*.

10.6.1 Data transfer from host to _connect2File

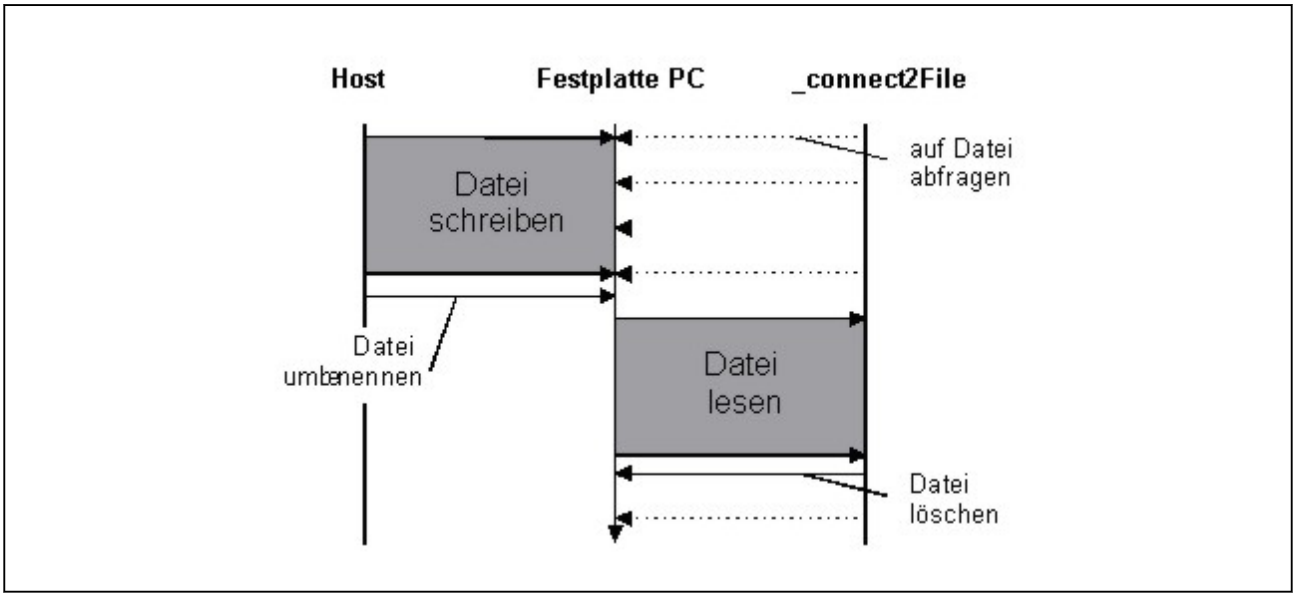


Illustration 13: Data transfer from host to *_connect2File*

1. The host generates a file in a directory the *_connect2File* computer has access to. To avoid access via *_connect2File* during generation, the file initially gets a name unknown to *_connect2File*.
   or
   The host generates the file in a directory the *_connect2File* computer has not access to.
2. After completing transfer, the file is renamed or relocated in the *_connect2File* input directory so that *_connect2File* can access it.
3. *_connect2File* reads and processes the file. In the event of transfer errors, *_connect2File* generates an error file, see page 71. After successful processing, *_connect2File*deleted the input file.
4. Only after deleting the input file, the host can provide a new input file by means of renaming or relocating. Until then the host collects the data in another directory or saves it with another file name (cf. step 1).

10.6.2 Data transfer from _connect2File to host

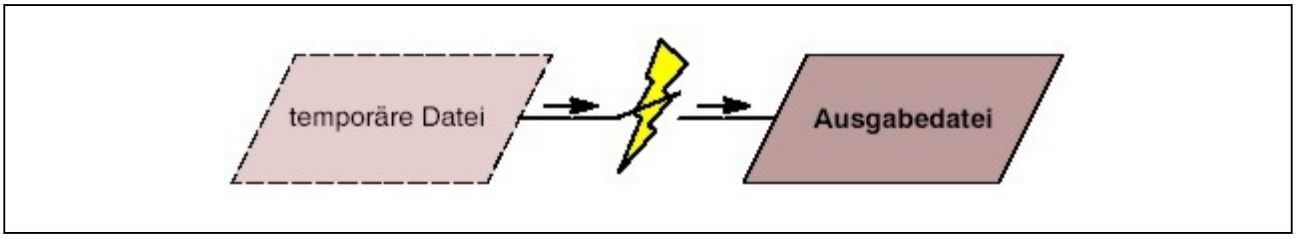


Illustration 14: Data transfer from _connect2File to host

1. First *_connect2File* collects the device data in a temporary file.
2. Triggered by an event such as changing the header, reaching a specified number of lines or receiving certain data, *_connect2File* moves the file to the output directory and supplies it to the host.
3. The host checks the output directory cyclically (polling). If it finds an output file, it copies or relocates the file and deletes it from the output directory.
4. New data cannot be added to files located in the output directory. Only after deleting the old file, *_connect2File* can move a file with new device data to the output directory.

10.6.3 Controlling data output by events

_connect2File relocates its temporary log file in the output directory as soon as a defined event occurs. This event may be related to files or received data. File-related events are the following: a defined number of entries is reached, a defined period of time has elapsed, a set time is reached or a header changed within the file. Configure these events in *_connectConfig* on the "Files" tab of the *_connect2File* configuration, see page 50.

In addition to file-related events, receipt of certain input data causes relocation in the output directory. Define this input data in *_connectConfig* by configuring the event file, see page 54.

10.6.4 File structure

The transfer files consist of headers and user data. Headers always contain information on the data format, its attributes and the following user data. To distinguish headers from user data lines, they start with the following identifiers:

`A!` Write access to labeler, new header format
Gx/Ix-Net(A!, I!, A?, I?)

`I!` Write access to industrial device, new header format
Gx/Ix-Net(A!, I!, A?, I?)

`A?` Read access to labeler, new header format
Gx/Ix-Net(A!, I!, A?, I?)

`I?` Read access to industrial device, new header format
Gx/Ix-Net(A!, I!, A?, I?)

The following formats are no longer supported:

`!` Write access, old header format
Gx-Net (!,?)

`?` Read access, old header format
Gx-Net (!,?)

A header is followed by one or more data records (user data). If the user data format changes, a new header is added.

Example:
Header for data format x
User data in data format x
User data in data format x
Header for data format y
User data in data format y
User data in data format y
User data in data format y

Example of a file in the new format Gx/lx-Net(A!, I!, A?, I?), headers highlighted

```
A!PV04|PW02|GW09|GL19|GL1A|GL16|PD00|GL2B|GL2C CR/LF
 1|2|4711|0|1|KG;-3;100|20997|1545 CR/LF
 1|2|4711|0|2|KG;-3;100|20997|1545 CR/LF
 1|2|4711|0|3|KG;-3;100|20997|1545 CR/LF
 1|2|4711|0|4|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|5|KG;-3;100|20997|1546 CR/LF
A!PV01|PW02|PW00|PL00|PD03|PD14|GL17|GL19|GL1A|GD02|GL2B|GL2C CR/LF
 8|1|5|KG;-3;500|0|650|2|4711|0|0|20997|1546 CR/LF
A!PV04|PW02|GW09|GL19|GL1A|GL16|PD00|GL2B|GL2C CR/LF
 1|2|4711|0|1|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|2|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|3|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|4|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|5|KG;-3;100|20997|1546 CR/LF
A!PV01|PW02|PW00|PL00|PD03|PD14|GL17|GL19|GL1A|GD02|GL2B|GL2C CR/LF
 8|1|5|KG;-3;500|0|650|2|4711|0|0|20997|1546 CR/LF
 8|2|10|KG;-3;1000|0|650|2|4711|0|0|20997|1546 CR/LF
 8|10|10|KG;-3;1000|0|650|2|4711|0|0|20997|1547 CR/LF
```


Example of a header in the new format Gx/lx-Net(A!, I!, A?, I?) with explanation

```
A!PV04|PW02|GL19|GL16|PD00|PD10|LX02 <CR/LF>
```

A!	Write access to a labeler.
PV04	!PSV_DATA Block command for transmission of configurable and cancellation-adjusted (i.e. labeled error-free) package-synchronous labeling data to a higher-level EDP. No parameters in the data line.
PW02	#PSW_PCKHDL Package handle for identification of a package in the labeling system. One parameter in the data line: `7`
GL19	GGL_PLUNR Currently labeled PLU no. One parameter in the data line: `1`
GL16	GGL_EINZEL_NUMERATOR Numerator for the individual label. One parameter in the data line: `1`
PD00	#PSD_GEW_NETTO_EINZEL Printed net weight with weight unit, value of exponent and weight value. Three parameters in the data line: `KG`, `-3`, `1064`
PD10	PSD_PRS_VKPREIS Printed selling price with country code and value. Two parameters in the data line: `0`, `106`
LX02	LGX_CLOSE Logical command to close a block command. No parameters in the data line.

10.7 Troubleshooting

_connect2File processes an input file provided by the host record-by-record and sends the single data records to the *_connectServer*. *_connectServer* transfers the data records to the relevant device. Errors that occur during transfer are logged in an error file, see page 53.

Transmission error ("acknowledgement timeout error")
Transmission errors result from a faulty communication between _connect2File or _connectServer and the device. A transmission error occurs, for example, when the device is switched off or when it is not ready to receive for other reasons.

Format:

```
[TT:MM.JJJJ] [HH:MM:SS] [acknowledgement timeout error:][Header];[Datenzeile]
```

Example:

```
05:05.2011 16:59:16 acknowledgement timeout error: !GT02;Test
```

Data error ("parsing error:")
Data errors are syntactical or semantic errors during transmission, e.g. a missing header.

Format:

```
[TT:MM.JJJJ] [HH:MM:SS] [parsing error:] [Datenzeile]
```

Example:

```
05:05.2011 17:03:28 parsing error: abcdefg
```

11 2File

11.1 Overview

Program *2File* is a further development of *_connect2File* and also provides a user-friendly file interface for the transfer of BxNet data and control commands in form of files.

New functions in *2File*:

– Operating mode as service (with viewer) or as application.
– Input and output files can be configured as text file or XML file.
– Input and output text files can be coded as ASCII or as Unicode.
– The old GxNet format w/o category (example: `?GD01`) is not supported.

The macro support of Microsoft *Word* and *Excel* is no longer available. The deactivation of the terminal server mode detection is also no longer available.

11.2 Starting the program

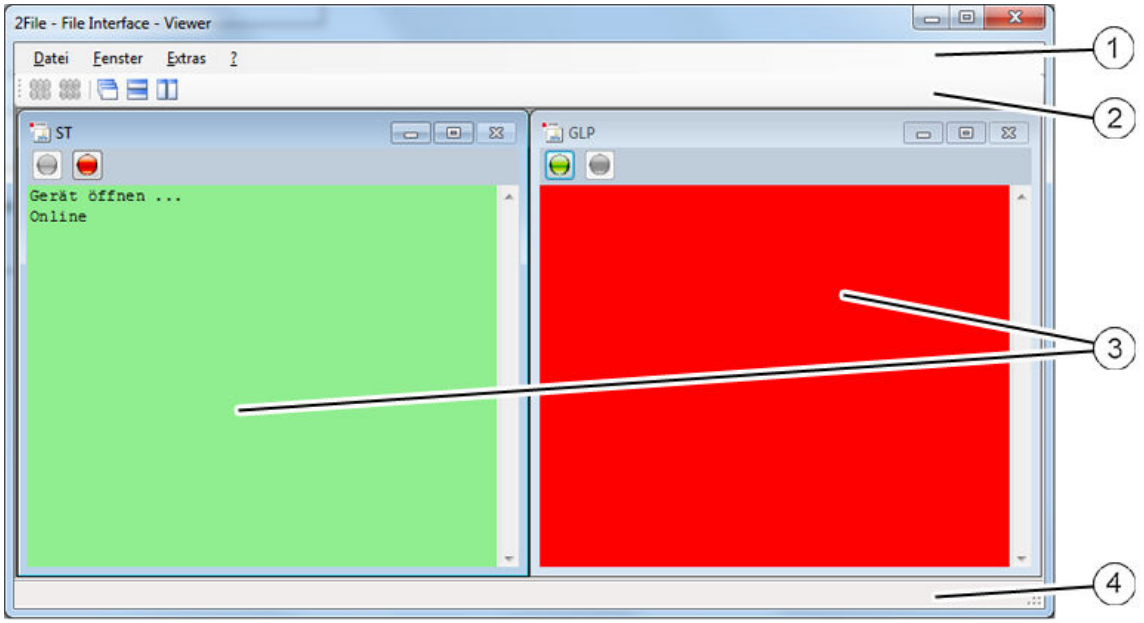
➪ Call up program *2File* via start menu.

If *2File* is installed as a service it automatically starts in the background during start of the operating system. During execution of the link the user interface of *2File* opens and connects to the service. As from the time of the connection establishment all information regarding running equipment is displayed. The devices can also be started and stopped here.
When the viewer is completed the service continues to run in the background.
If the service should not run it is automatically started during call-up of the viewer.

Changes to the configuration in *_connect.Config* become effective after restart of the service. The connection to all connected devices must be closed beforehand.

If the program runs in terminal server mode, *2File* uses as application the *_connectConfig* configuration which is the configuration set up for this terminal server client. If *2File* is set up as a service on a terminal server, the service runs with the local configuration. If in a terminal server session there will be a connection to this service with the viewer, this local configuration will be displayed.

11.3 Program structure



11

Illustration 15: User interface*2File*

① Menu bar

② Toolbar

③ Connection window

④ Status bar

Besides typical *Windows* elements the user interface contains the following areas:

Connection window: Communication of each device is displayed in a connection window. The connection window shows the connection status and the data transfer. The background color varies according to the status:

- Yellow: Connection establishment
- Cyan: wait for switch-on message
- Green: connection open (online)
- Gray: closing connection
- Red: connection closed (offline)

11.4 Menu bar and toolbar functions

Functions in the "File" menu

"Start for all devices"		Initialize access to all configured devices. Open connections and and start data exchange. Any devices that have already been started are not re-initialized.
"Stop for all devices"		Terminate data transfer for all devices. Close all connections.
"Exit"		Close all connections and end *2File*.

Functions in the "Window" menu

"overlapping"		Display windows in cascade format.
"arrangement vertical"		Display windows next to each other.
"arrangement horizontal"		Display windows next to each other.
"arrange symbols"		Arrange minimized connection windows at the bottom of the window.
example "1 ST"		Consecutive number and device name of displayed windows. The active window is checkmarked. Select desired window.

Functions in the "Extras" menu

"Language"	Select user interface language. The modification does not become active until the next application start.

Functions in the "?" menu

"2File"		Show application information, see page 14.

11.5 Functions in the connection windows

	Green symbol: Start device.
	Red symbol: Stop device.

11.6 File transfer

An application providing data for the devices runs on the host. Host application and *2File* program communicate via file exchange. For file formats see page 78. Files are transferred via hard disk of the *2File* computer based on a defined protocol.

Program *WinCWS* is an example of a host application. For the GX device family this program provides master data as ASCII file which is then transmitted to the GX device by *2File*.

Sample file transmitting text "hello GX" to text field 1 of a GX device.

Line 1: `A!GT01`

Line 2: `hello GX`

For more examples see *C:\Programme\Bizerba\BCT\Samples\BCF Files*.

11.6.1 File transfer from host to 2File

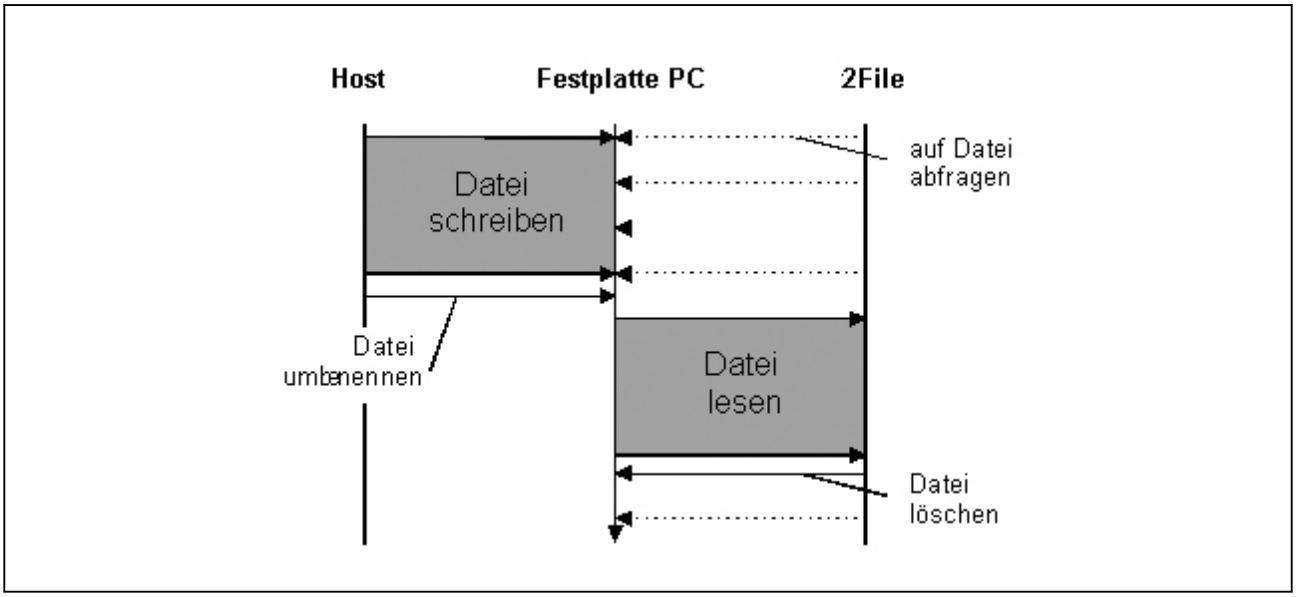


Illustration 16: File transfer from host to *2File*

1. The host generates a file in a directory which can be accessed by the *2File* computer. To prevent access via *2File* during the creation process the file receives a name unknown to *2File*.
   or
   The host generates the file in a directory which cannot be accessed by the the *2File* computer.
2. After complete transfer the file is renamed or moved to the *2File* input directory allowing *2File* access.
3. Program *2File* reads and processes the file. In case of transmission errors *2File* creates an error file, see page 53. After successful processing *2File* deletes the input file.
4. Only after deleting the input file, the host can provide a new input file by means of renaming or relocating. Until then the host collects the data in another directory or saves it with another file name (cf. step 1).

11.6.2 File transfer from *2File* to host

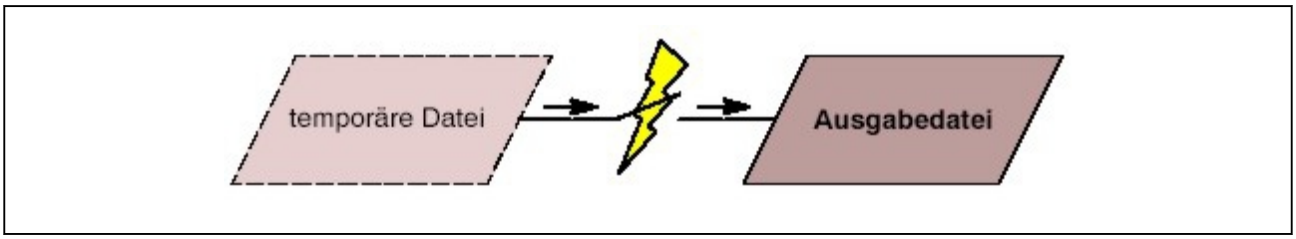


Illustration 17: File transfer from 2File to host

1. Program *2File* collects device data in a temporary file.
2. Triggered by an event such as a header change, reaching a specified number of lines or receiving certain data, *2File* moves the file to the output directory and provides it for the host.

3. The host checks the output directory cyclically (polling). If it finds an output file, it copies or relocates the file and deletes it from the output directory.
4. New data cannot be added to files located in the output directory. Only after deleting the old file, *2File* can move a file with new device data to the output directory.

11.6.3 Controlling data output by events

Program *2File* moves its temporary log file to the output directory as soon as there is a defined event. This event may be related to files or received data.

File-related events are:

– Reaching of a defined number of entries.
– Elapsing of a defined time interval.
– Occurring of a defined time.
– A header change in the file.

Configure these events in *Config* on tab "Files" of the *2File* configuration, see page 50.

In addition to file-related events, receipt of certain input data causes relocation in the output directory. Specify input data in *_connectConfig* with the event file configuration, see page 54.

11.7 File formats

The data transfer between host and *2File* is effected via files. Input and output files can be coded as ASCII or as Unicode text file or as XML file. If the coding of the input file no longer matches the coding specified in *_connect.Config*, the input file does not open and remains in the directory, see page 53.

Additionally, an error message in the viewer is issued.

11.7.1 Text file

File structure

The transfer files consist of headers and user data. Headers always contain information on the data format, its attributes and the following user data. To distinguish headers from user data lines, they start with the following identifiers:

`A!` Write access to labeler, new header format
Gx/Ix-Net(A!, I!, A?, I?)

`I!` Write access to industrial device, new header format
Gx/Ix-Net(A!, I!, A?, I?)

`A?` Read access to labeler, new header format
Gx/Ix-Net(A!, I!, A?, I?)

`I?` Read access to industrial device, new header format
Gx/Ix-Net(A!, I!, A?, I?)

The following formats are no longer supported:

`!` Write access, old header format
Gx-Net (!,?)

`?` Read access, old header format
Gx-Net (!,?)

A header is followed by one or more data records (user data). If the user data format changes, a new header is added.

Example:
Header for data format x
User data in data format x
User data in data format x
Header for data format y
User data in data format y
User data in data format y
User data in data format y

Example of a file in the new format Gx/lx-Net(A!, I!, A?, I?), headers highlighted

```
A!PV04|PW02|GW09|GL19|GL1A|GL16|PD00|GL2B|GL2C CR/LF
 1|2|4711|0|1|KG;-3;100|20997|1545 CR/LF
 1|2|4711|0|2|KG;-3;100|20997|1545 CR/LF
 1|2|4711|0|3|KG;-3;100|20997|1545 CR/LF
 1|2|4711|0|4|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|5|KG;-3;100|20997|1546 CR/LF
A!PV01|PW02|PW00|PL00|PD03|PD14|GL17|GL19|GL1A|GD02|GL2B|GL2C CR/LF
 8|1|5|KG;-3;500|0|650|2|4711|0|0|20997|1546 CR/LF
A!PV04|PW02|GW09|GL19|GL1A|GL16|PD00|GL2B|GL2C CR/LF
 1|2|4711|0|1|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|2|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|3|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|4|KG;-3;100|20997|1546 CR/LF
 1|2|4711|0|5|KG;-3;100|20997|1546 CR/LF
A!PV01|PW02|PW00|PL00|PD03|PD14|GL17|GL19|GL1A|GD02|GL2B|GL2C CR/LF
 8|1|5|KG;-3;500|0|650|2|4711|0|0|20997|1546 CR/LF
 8|2|10|KG;-3;1000|0|650|2|4711|0|0|20997|1546 CR/LF
 8|10|10|KG;-3;1000|0|650|2|4711|0|0|20997|1547 CR/LF
```

11

Example of a header in the new format Gx/lx-Net(A!, I!, A?, I?) with explanation

```
A!PV04|PW02|GL19|GL16|PD00|PD10|LX02 <CR/LF>
```

A!	Write access to a labeler.
PV04	!PSV_DATA Block command for transmission of configurable and cancellation-adjusted (i.e. labeled error-free) package-synchronous labeling data to a higher-level EDP. No parameters in the data line.
PW02	#PSW_PCKHDL Package handle for identification of a package in the labeling system. One parameter in the data line: `7`
GL19	GGL_PLUNR Currently labeled PLU no. One parameter in the data line: `1`
GL16	GGL_EINZEL_NUMERATOR Numerator for the individual label. One parameter in the data line: `1`
PD00	#PSD_GEW_NETTO_EINZEL Printed net weight with weight unit, value of exponent and weight value. Three parameters in the data line: `KG`, `-3`, `1064`
PD10	PSD_PRS_VKPREIS Printed selling price with country code and value. Two parameters in the data line: `0`, `106`
LX02	LGX_CLOSE Logical command to close a block command. No parameters in the data line.

File coding

Text files can be saved as ASCII/ANSI or as Unicode file. ASCII characters are coded in expanded 8 bit format. This also includes special characters depending on the code page such as an Umlaut for a German system.

11.7.2 XML file

File structure

The XML file structure applies to both input files and output files. An XML file for *2File* has got at least the following structure (empty XML file, i.e. without specifying any command).

```
<?xml version="1.0" encoding="UTF-8"?>
<DataSets>
</DataSets>
```

Illustration 18: Example XML file

The first line includes the XML declaration. This is followed by the top element of the XML file. This element must include all commands. Here, the structure header and user data line no longer exists.

Each command consisting of one or more headers and the user data is subsequently added as one or more elements. The command creates the start and end tag of the element and the data of the command the element content.

Comparison between text file and XML file

Here, the command consists of only one header. The first header of a command includes information if this is a read or a write command and from which category the command comes from.

This information is saved as attribute.

```
I?GD04
kg;0;0
```

```
<?xml version="1.0" encoding="UTF-8"?>
<DataSets>
    <GD04 CMD="read" CAT="I">kg;0;0</GD04>
</DataSets>
```

Illustration 19: Example text/XML

Attribute CMD indicates if this is a read or a write command. Attribute CAT indicates the category such as I. The abbreviation preceding a header in a text file is defined this way in the XML file.

11

The start tag is here <GD04>. This is header GD04.

Between start and tag is the user data of the command, here kg;0;0.

For commands w/o user data they are simply omitted (example: <RX01></RX01>).

Each header forms an own element below the top element <DataSets>.

Example text file	Example XML file
I?GD04 kg;0;0 I?GT04 Text	<DataSets> <GD04 CMD="read" CAT="I">kg;0;0</GD04> <GT04 CMD="read" CAT="I">Text</GT04> </DataSets>

Block commands include other commands. Here, included commands create the element content of a block command until its closing header.

Example block command

```
I!LV01|GD01|GD02|GV02|GW04|GT02|LX02|GL0A|LX02

kg;-3;3755|kg;-3;0|01|123|000001
```

Example XML file

```
<LV01 CMD="write" CAT="I">
        <GD01>kg;-3;4030</GD01>
        <GD02>kg;-3;0</GD02>
        <GV02>
            <GW04>01</GW04>
            <GT02>123</GT02>
        </GV02>
        <GL0A>000001</GL0A>
</LV01>
```


The first block command <LV01> opens the sequence. Since this is the first command of the header it includes relevant attributes. Its element content now create all other commands of the header.

In the text file the block command is completed with LX02. In the XML file the end tag of the block command indicates the end </LV01>. This is representing LX02. If the commands are extracted from the XML file and converted to the text file style the end tag of a block command must be replaced by LX02!

Block commands can include other block commands. Also here their element content form all included commands, see GV02. The included commands should be shown indented for better readability.

11.8 Troubleshooting

Program *2File* processes an input file provided by the host record-by-record and sends individual data records to the *_connectServer*. The *_connectServer* transfers the data records to the relevant device. Errors that occur during transfer are logged in an error file, see page 53.

Transmission error ("acknowledgement timeout error")
Transmission errors result from a faulty communication between *2File* or *_connectServer* and the device. A transmission error occurs, for example, when the device is switched off or when it is not ready to receive for other reasons.

Format:

```
[TT:MM.JJJJ] [HH:MM:SS] [acknowledgement timeout error:][Header];[Datenzeile]
```

Example:

```
05:05.2011 16:59:16 acknowledgement timeout error: !GT02;Test
```

Data error ("parsing error:")
Data errors are syntactical or semantic errors during transmission, e.g. a missing header.

Format:


```
[TT:MM.JJJJ] [HH:MM:SS] [parsing error:] [Datenzeile]
```

Example:

```
05:05.2011 17:03:28 parsing error: abcdefg
```

12 _connect2DB

12.1 Overview

By means of *_connect2DB* you transmit device data to a *MS Access* database or to a *MS SQL-Server*. Freely definable filters allow you to limit the data volume according to requirements. Device data can be exported from the database and made available for other programs. Unicode characters are supported.

12.2 Starting the program

➪ Call up *_connect2DB* via the start menu.

The start window of the program appears. When configured accordingly, the connections to the devices created on the *_connectServer* are established automatically.

12.3 Database setup

When using *_connect2DB* for the first time, the database setup wizard starts automatically. If during start of *_connect2DB* program *DBConvert* is called up, the SQL database must be prepared for Unicode, see page 104.

The wizard can also be directly started via *C:\Programme\Bizerba\BCT \BCT2DB\DBConfig.exe*.

When setting up the database, choose between the following database types:

Database type	Benefits	Disadvantages
Microsoft Access	– Easy operation – Easy administration – Easy creation of evaluations	– Only suitable for small data quantities (slow from 20,000 data records, performance no longer acceptable from 50,000 data records) – High storage requirements on local computer – Data security not as high as with *MS SQL Server* – Access to the data stock by more than one system problematic
Microsoft SQL Server (recommended)	– Quick data access even with high volume of data – simplest version (*Express*) available free of charge – Access to a database from different programs possible – Load distribution by installing database and programs on different computers – Database more stable than *MS Access*	– Higher administrative cost – More complex training as with *MS Access*; but it is possible to access data stock via *MS Access*.

It is recommended to use *MS SQL Server* as from version 7.0. This version is easily scaleable. This will guarantee data security and access speed even with large data volumes.

Before setting up a *MS SQL Server* database, the *MS SQL Server* has to be installed and set up.

➪ Select the database type in the first step of the wizard.

➪ <Next >>

Set up a *MS Access* database

➪ Confirm the available storage space display.

If there is enough storage space, a *MS Access* database called *BCT2DB.dat* is created in directory *C:\ProgramData\Bizerba\BCT\db*.

Set up a *MS SQL Server* database

➪ Select server. Enter user name and password for establishing the connection to the *MS SQL Server*.

Database *BCT2DB* is set up on the *MS SQL Server*.

Completing database setup (for both database types)
➪ <Complete>

12.4 Database structure

The database structure is identical for *MS Access* and *MS SQL Server*:

– Each filter corresponds to a database table with the same name.
– Each filter element correspond to a column of the database table. Column name is the underlying device command. The columns are of type nvarchar(255) for a SQL table and for a *MS Access* table of type varchar(255) since for *Access* all characters are automatically saved as Unicode. Whether empty table columns are created or not depends on the filter configuration, see page 90.
– Each table contains also the following columns:
  – Column "INSERT_TIMEDATE" of type "datetime" and "date/time". This column contains the moment of time when the relevant line has been written in the database.
  – Column "DEVICE" of type nvarchar(255) and varchar(255). This column contains the device name as configured in *_connectConfig* where the data package or device commands originate from.
– Each line contains a data packet from a device.

12.5 Program structure

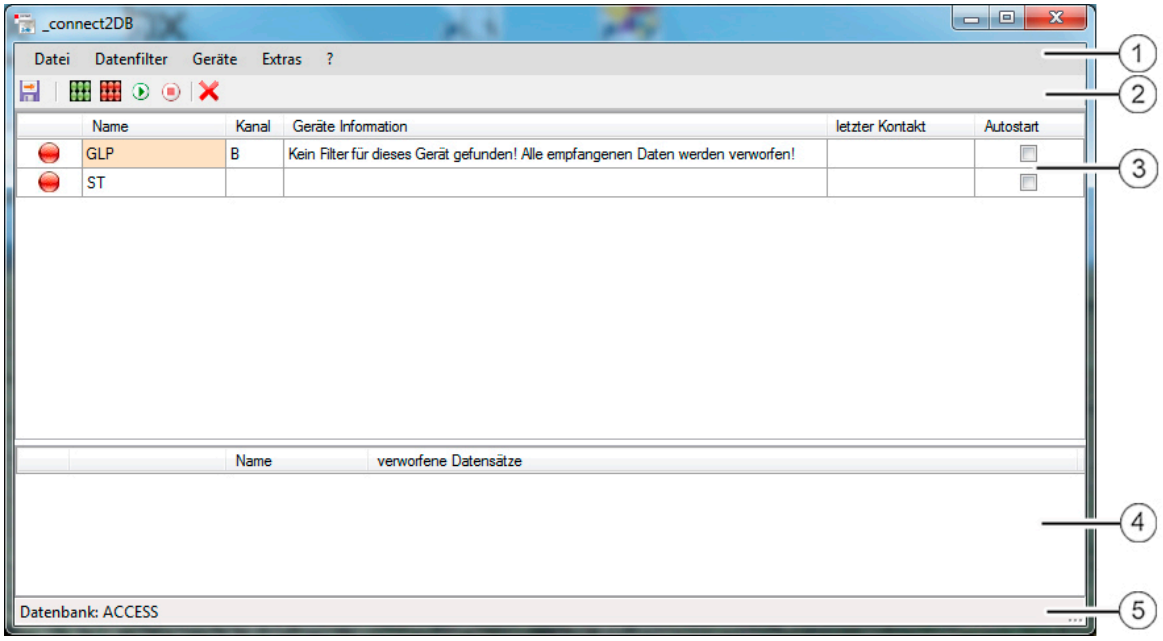


Illustration 20: User interface_*connect2DB*

12

① Menu bar

② Toolbar

③ Display area

④ Error area

⑤ Status bar

Besides typical *Windows* elements the user interface contains the following areas:

Display area:	Shows status information of devices and filters.
Error area:	Can be switched on or off. Lists rejected data records.
Status bar:	Displays if *MS Access* or *MS SQL Server* is used.

12.6 Menu bar and toolbar functions

Functions in the "File" menu

"Export..."		Export data, see page 98.
"Exit"		Terminate all data transfer, close connections and terminate _*connect2DB*.

Functions in the "Data filter" menu

"Add..."	Set new filters, see page 90.
"Manage..."	Change existing filter, see page 92.
"Default data"	Set-up of default data for GX and IX equipment.

Functions in the "Devices" menu

"Manage..."	Add to or remove devices from *_connect2DB* configuration and set devices.

Functions in the "Extras" menu

"Options..."	Make *_connect2DB* settings, see page 101.
"Clear contents of database..."	Delete database contents of defined period, see page 103.
"Language"	Change user interface language.

Functions in the "?" menu

"Info..."	Show application information, see page 14.

Functions that are only available in the toolbar

	"Start all devices" Start data transfer for all devices and filters.
	"Stop all devices" Stop data transfer for all devices and filters.
	"Start device" Start data transfer only for selected device.
	"Stop device" Stop data transfer only for selected device.
	"Clear discarded data" Empties the display for discarded data. Only available if discarded data should be displayed.

12.7 Defining filters

Filters are used to collect selected data from the data flow of a connected device. A wizard is available for creating filters.

12.7.1 Creating filter using the wizard

The device used for determining data records has to be connected and ready for use. The device must be enabled for manual release of data records. With GX and CX devices provided with storage medium, it is also possible to read data records from the storage medium.

- ➪ End all connections to the device.
- ➪ "Data filter" / "Add..."
- ➪ Select a device for determining the data records to be filtered in the "Name" dropdown list.
- ➪ **Only with GX/CX devices:** In dropdown list "Channel Control" select setting "no channel control" or a channel for the transmission of data. The setting must match the device setting.
- ➪ In the dropdown list "IxNet Converting (@41 --> A)", select whether and how data has to be converted.

  The following settings are possible:
  - "No BxNet conversion"
  - "Ansi conversion (@41 -->A)"
  - "UTF8 conversion (@E2@82@AC --> €)"
- ➪ <Next >>
- ➪ Enter a comprehensive name for the new filter.

  The filter name is also used for the relevant database table.
  Permitted characters: upper case, lower case, numbers and spaces. Spaces or digits at the beginning of a name are not permitted.
  Characters not permitted: . / \ @

No differentiation is made between upper/lower case in filter names.

- ➪ <Next >>
- ➪ For registration, activate data transfer of the device, e.g. transmission of total information.

or

- ➪ **Only with GX/CX devices with storage medium:** Click <Memocard> when data records should be read from a storage medium instead of spontaneous telegrams.

  If one or more data records are read by the storage medium a selection window appears. Select here the desired data record and click <OK>. If the desired data record is not available, click <Cancel> and re-read storage medium.

➪ Select telegram or data record and confirm it with <OK>.

The commands set under "Options..." (default data, see page 101) and the commands that have been received during registration are available for saving in the database table.

➪ Deactivate check-box for data that is not to be saved in the database table.

➪ If only data records that contain the selected data in the specified order are to be written in the database table, activate the "Only evaluate data in the telegram, which correspond to the indicated sequence." check-box.

➪ Select the device whose data is to be filtered accordingly.

➪ **Only with GX/CX devices:** Select whether the channel control has to be used and specify the channel, if necessary.

➪ **Only with GX/CX devices:** By activating or deactivating the corresponding check-boxes, specify whether spontaneous telegrams from the device shall be received or the storage medium shall be read.

The "spontaneous telegrams" check-box is automatically activated for devices of other device families.

➪ Create filter with <Complete>.

The filter information is written to file *C:\ProgramData\Bizerba\BCT\config\BCT2DB \2DBRules.xml*. It is not saved user- specifically, but it is valid for all users. Files must not be changed by the user!

12.7.2 Creating filters manually

➪ "Data filter" / "Manage..."

In the "Data File Administration" window, the following buttons are available:

	<New Filter> Define new filter.
	<Save> Save all data filters.

➪ <New Filter>

A new filter is displayed in the navigation area.

➪ Rename filter, see page 92

➪ Assign device and make device-specific settings, see page 92.

➪ Create list of commands to be evaluated, see page 92.

➪ <Save>

➪ <File> / <Close>

12.8 Renaming filters

You can rename filters even if the table already includes data. A new empty table is created and the table with the old name remains. Changes become effective after clicking <Save> or close window and save changes. Use a comprehensive name. The filter name is also used for the relevant database table.

Permitted characters: upper case, lower case, numbers and spaces. Spaces and digits at the beginning of the name and spaces at the end of the name are not permitted.

Characters not permitted: . / \ @

No differentiation is made between upper/lower case in filter names.

➪ "Data filter" / "Manage..."
➪ Right-click the filter to be renamed.
➪ In the context menu select "Rename".
➪ Enter a new name and confirm using the enter key.

12.9 Edit filter

The filter management is used to edit existing filters and view the related database tables. The display is divided into the following tabs:

"Commands / Data":	List of device commands which the filter evaluates.
"Devices":	List of devices whose data the filter evaluates.
"Data Browser":	Display of already read table data.

The following buttons are available:

	<New Filter> Define new filter.
	<Save> Save all data filters.
	<Re-load table>

➪ "Data filter" / "Manage..."
➪ Select filter to be edited in the navigation area.
The related data is shown in the display area.

Assign devices and make device-dependent settings

➪ Select the "Devices" tab.
➪ Activate check-boxes of devices whose data the filter shall evaluate.

➪ Click category check-boxes in order for the selected equipment of this category to be transferred.

Only one category but multiple devices of the same category may be selected.

➪ Select device and make relevant settings.

Please, make device settings in menu "Devices" / "Manage...".

Settings	
"Channel Control"	**Only for GX/CX devices.** Deactivate channel control or select channel to be used for data transfer.
"use MemoCard"	**Only for GX/CX devices.** activated: Read out memory card regularly. not activated: Do not read out memory card.
"spontaneous telegrams"	activated: Receive data initiated by the device (spontaneous telegrams). not activated: Receive only requested data.
"IxNet Converting (@41 --> A)"	Deactivate conversion or select conversion type.

Editing list of commands to be evaluated by means of filter

➪ Select the "Commands / Data" tab.

On the tab, the following tabs are available for editing:

	Move line upwards.
	Move line downwards.

Editing lines

➪ Click the field to be edited on the "Commands / Data" tab and select the desired field name or command in the dropdown list.

Saving data filter management

➪ <Save>

12.10 Device menu

Here you can add to and remove devices from the *_connect2DB* configuration and set devices used.

➪ Click <Save> to save changes and to close the window.

12.10.1 Change window devices

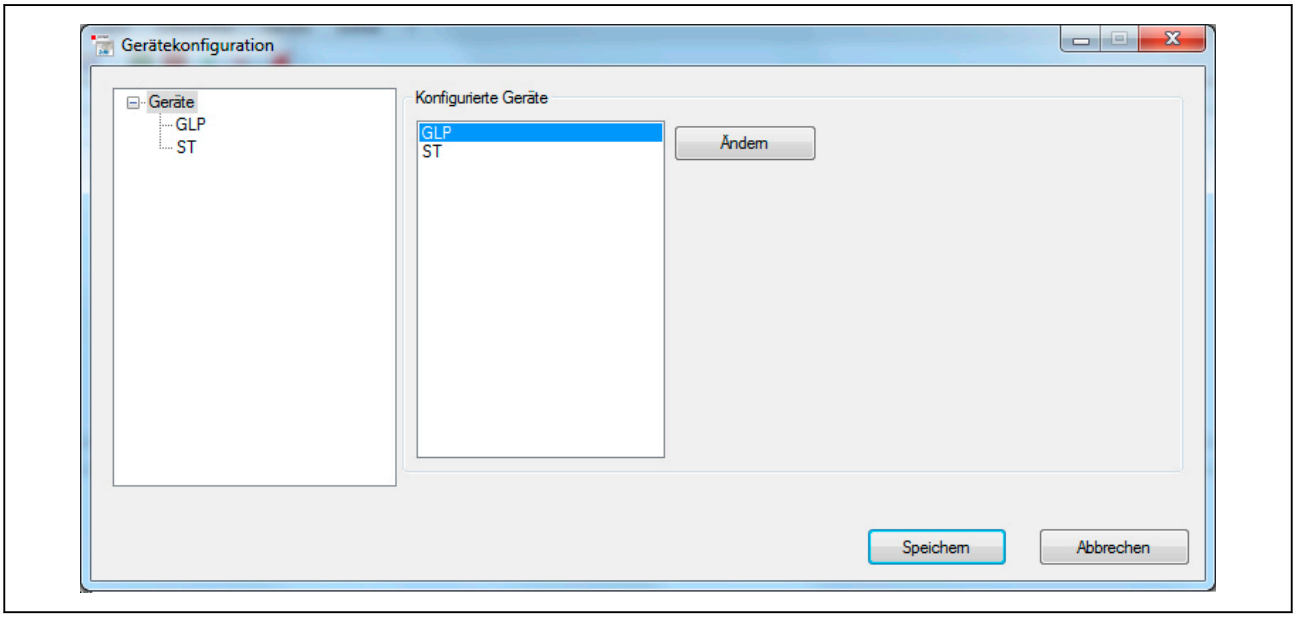


Illustration 21: Change devices

➪ Click <Change>.

A window opens in which you can add or remove devices in the *_connect2DB* configuration. A filter must not necessarily be assigned to a configured device. In this case all data of the device will be discarded.

➪ Select device from list.

Available devices are listed on the left and selected devices are listed on the right. You can select up to 40 devices.

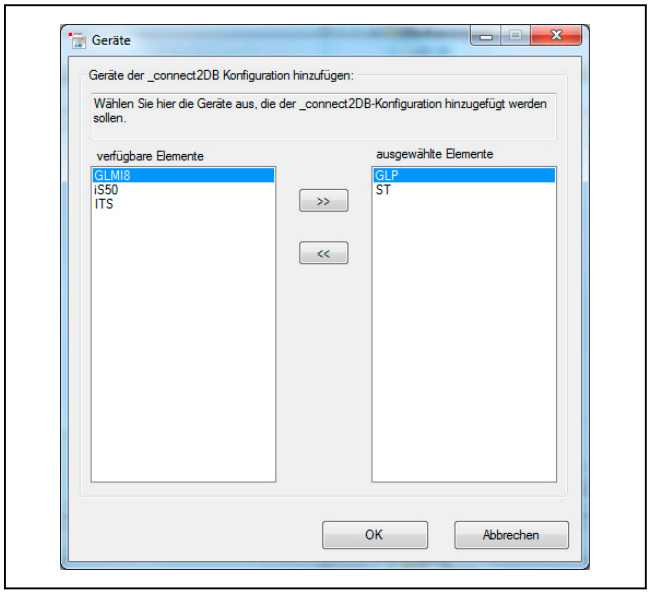


Illustration 22: Add / remove devices

Discarded data is saved in folder *C:\ProgramData\Bizerba\BCT\log \BCT2DB\Stdlog*.

12.10.2 Device settings

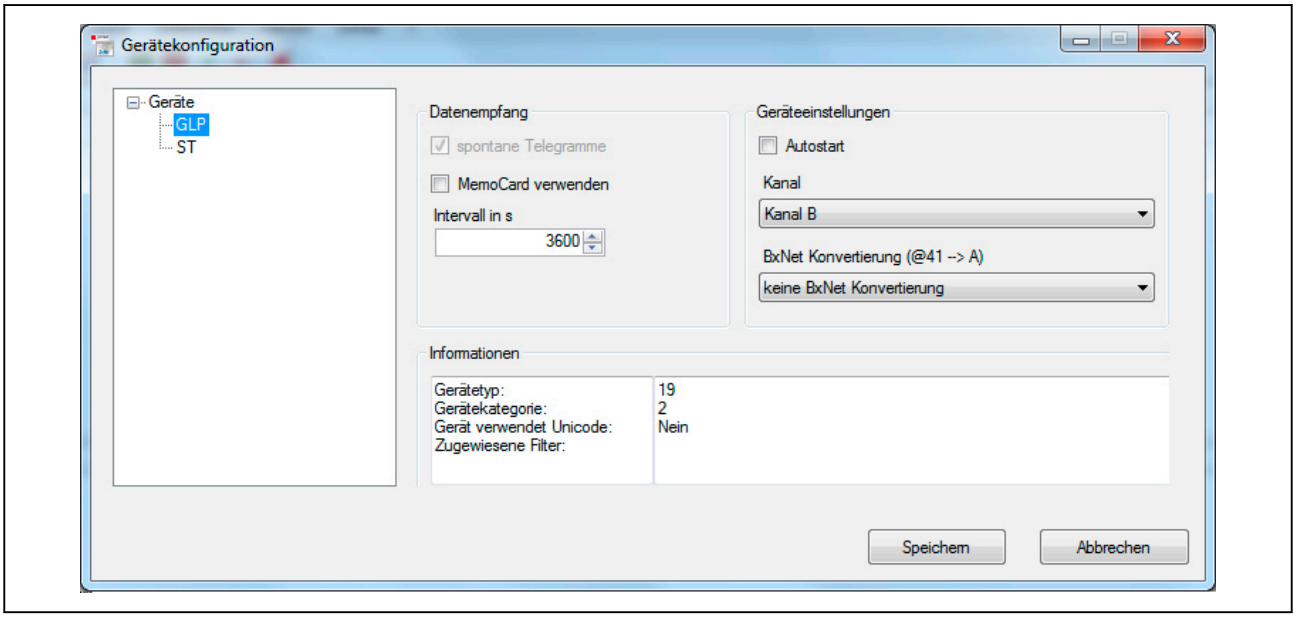


Illustration 23: Device settings

If a device is selected from the list, available settings will be displayed.

Channel control (only GX/CX)	Deactivate channel control or select channel to be used for data transfer.
Memory card + interval (only GX/CX)	activated: Read memory card on a regular basis with specific interval in seconds. not activated: Do not read out memory card.
BxNet conversion	Deactivate conversion or select conversion type.
Spontaneous telegrams	activated: Receive data initiated by the device (spontaneous telegrams). not activated: Receive only requested data.
AutoStart	activated: When starting the program, automatically establish connections to the device and start data transfer. not activated: No automatic start of data transfer.

12.11 Default data

In this menu you can define default data which should be offered as possible filter criteria when creating filters by means of the wizard. Add selected data from field "Available Elements" to field "selected elements" and back using the buttons. The default data is divided into IX and GX/CX category.

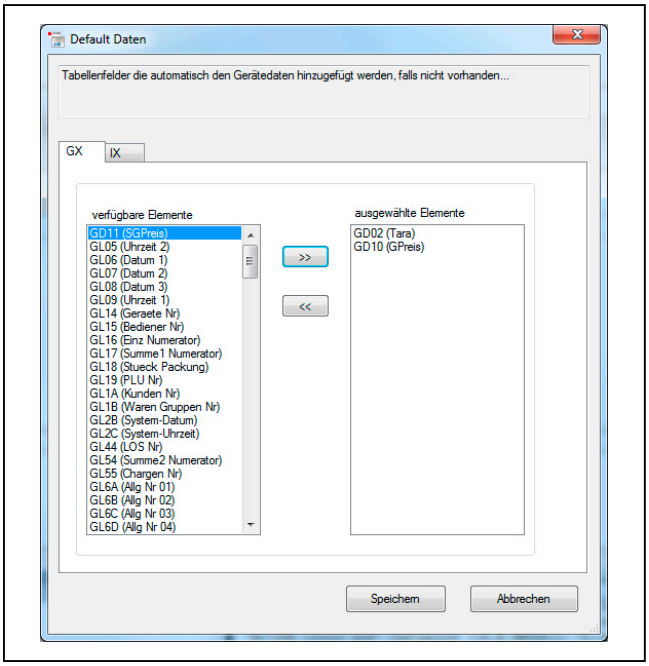


Illustration 24: Default data

12

"selected elements" Commands that are always available as possible filter criteria when defining filters (default data).

"Available Elements" Available commands that can be used as default data.

12.12 Defining handling of complex device commands

In the default setting, only simple device commands can be selected as filter criteria in *_connect2DB*. Device commands with a complex structure comprising a number of values and texts must first be made known to the program.

Define the handling of device commands with complex data structure or variable number of parameters in the *Collection_Command.dat* configuration file. Each line contains a replacement command related to a complex device command.

Example:

– `GV02|GW05|GT03|LX02;%HGW05%_%DGW05%;%DGT03%`

The lines contain the following elements that are separated by a semicolon:

– Header of the device command with a variable number of parameters for identification of the command in the data flow (`GV02|GW05|GT03|LX02`)
– Definition of a new header for designation of the column in the database table (`%HGW05%_%DGW05%`)
– Definition of the value that is written in the database (`%DGT03%`)

Commands with a complex structure have the letter *V* (`GV02`) at the second position as operand code.

Character strings that are enclosed in percentage signs are interpreted. Other characters are transferred unchanged.

The first character after the percentage sign defines the information type:

– `%H...` stands for a header.
– `%D...` stands for a value.

When receiving device commands, the program searches for headers that are defined in the configuration file. When a suitable command is found, the received data will be replaced according to the relevant command for storage in the database table.

Example 1: Resolution of a device command with a variable number of parameters into a replacement command with one parameter
Instead of saving the received device commands, this replacement command saves the new header and the new value that have been defined in the *Collection_Command.dat*.

Line in the *Collection_Command.dat* configuration file:
`GV02|GW05|GT03|LX02;%HGW05%_%DGW05%;%DGT03%`
Received data packet:
`GV02|GW05|GT03|LX02 ; 2|Hallo`
Storage in the database table:
`GW05_2 ; Hallo`

The name of the first parameter (`GW05`) and the value of the first parameter (`2`) are connected by an underscore to form the name of the replacement command. This is also the name of the filter element or of the column in the database table (`GW05_2`). The value of the second parameter (`Hallo`) is saved as value of the replacement commands or contents of the database column.

Example 2: Resolution of a device command with a variable number of parameters into more replacement command with several parameters
This replacement command lists the single commands with their respective parameters.

Line in the *Collection_Command.dat* configuration file:
`GV01|GW03|GD01|GD02|GD07|LX02 ; %HGW03%|%HGD01%|%HGD02%|%HGD07% ; %DGW03%|%DGW01%|%DGW02%|%DGW07%`
Received data packet:
`GV01|GW03|GD01|GD02|GD07|LX02 ; 1|1.0|2.1|3.4`
Storage in the database table:
`GW03|GD01|GD02|GD07 ; 1|1.0|2.1|3.4`

The database columns `GW03`, `GD02`, `GD02` and `GD07` receive the values `1`, `1.0`, `2.1` and `3.4`.

The replacement command of this example resolves the device command with a variable number of parameters without generating new pseudo commands. The following simple formula obtains the same result:

`GV01|GW03|GD01|GD02|GD07|LX02;*;*`
The asterisks cause only the sequences `GV01` and `LX02` to be deleted from the data packet. The remaining data is transferred unchanged.

12.13 Deleting filters and relevant database tables

Deleting filters

➪ "Data filter" / "Manage..."

➪ Right-click the filter to be deleted.

➪ In the context menu, select "Clear...".

The filter is deleted without query. If data related to the filter has already been read, the corresponding database table appears in the navigation area under "Tables w/o Filters" and can be viewed in the display area.

Deleting the relevant database table

➪ Right-click database table in the navigation area under "Tables w/o Filters".

➪ In the context menu, select "Clear...".

The table is deleted without query.

12.14 Exporting data

The data imported in *_connect2DB* can be exported in the following formats:

"Comma Separated Values (*.CSV)"	Export in a text file with a semicolon (;) as separator.
"Microsoft Excel (.XLS)"	Export in file format of *Microsoft Excel*.
"Microsoft Access (.MDB)"	Export in file format of *Microsoft Access*.

During export, data is copied and not deleted from *_connect2DB*.

➪ <Exporting data>

or

➪ Via the "File" menu, select "Export...".

Program *_connect2DB.Export* starts.

Program *_connect2DB.Export* can also be called up separately via the start menu.

➪ Select the export format in the "Type" dropdown list.

➪ <Forward >>

➪ Select at least one data filter or one database table as data source.

➪ <Forward >>

➪ Enter the period from which data has to be exported.

Depending on the selected data source, the devices from which the data to be exported originates are displayed.

➪ If only data of certain devices is to be exported, deactivate the check-boxes of the other devices.

➪ <Forward >>

➪ Enter path and name of the export file or search for them using the <...> button.

➪ Activate function "Change numeric IX values" if decimal numbers and values with unit have to be converted according to the configuration of *_connect2DB*, see page 101.

➪ Activate function "Export of weight values without unit" if the unit of the value should not be exported.

➪ Activate function "No header exported" if no header should be created. A header describes the following data records.

➪ "Separator for weight values with unit" set to desired value.

- `1`: used in the system
- `2`: decimal point
- `3`: point

➪ <Finish>

If no data is available, an error message is displayed. Otherwise export starts.

12.15 Data export via command line

Exporting of database tables can also be started via the command line. The following parameters are available:

Parameter	Description
`-export`	Program start without user interface (silent mode).
`-table`	Name of the table to be exported.
`-period`	Period from which data has to be exported. Value range: – `thisday`: All data with current date. – `yesterday`: All data of previous day. – `thisweek`: All data of current week (from Monday incl.). – `lastweek`: All data of previous week (Monday to Sunday). – `thismonth`: All data of current month. – `lastmonth`: All data of previous month. – `thisyear`: All data of current year. – `lastyear`: All data of previous year. Example: `-period yesterday` All data of the previous day will be exported.

Parameter	Description
`-from`	Period from which data has to be exported. Usable as an alternative to the `-period` parameter. Enter date and time in the format set under Windows. German format, for example: `-from 21.05.2010 14:30 -to 21.05.2010 14:50`
`-to`	
`-format`	Output format of data. Value range: `CSV`, `XLS`, `MDB`
`-file`	Path and name of output file. Example: `-file c:\temp\export.csv`
`-withoutUnit`	Same as option "Export of weight values without unit" Specify parameter to activate option. Default setting: not activated
`-noDimChange`	Same as option "Change numeric IX values" Specify parameter to prevent conversion of values! Default setting: Values are converted.
`-exportHeader`	Same as option "No header exported" Specify parameter if header should be exported! Default setting: Do not export header
`-seperator`	Same as option "Separator for weight values with unit": Value range: – `1`: used in the system – `2`: decimal point – `3`: point Default setting: `1`
`-showMessage`	After completion of export display success or error message. Specify parameter to activate option. Default setting: not activated

Example of an export call-up from the command line

```
DBExport.exe –export –table Registrierung – from 02.03.2005 01:23 –to
03.09.2011 05:17 –format CSV  -file c:\temp\export.csv –exportHeader –withoutU-
nit –separator 3 –showMessage
```

Export file *export.csv* is saved in directory *C:\temp* w/o start of the *DBExport* user interface. It contains all the data that has been imported in the mentioned period using the "Recording" filter. The headers of the table columns are exported. Dimensional values are exported w/o unit, with decimal separator "Point". After completion of the export a success or error message should be issued.

12.16 Configuring _connect2DB

Via menu item "Extras" / "Options" you configure program *_connect2DB*. The configuration is stored in file *C:\ProgramData\Bizerba\BCT\config\BCT2DB.xml*.

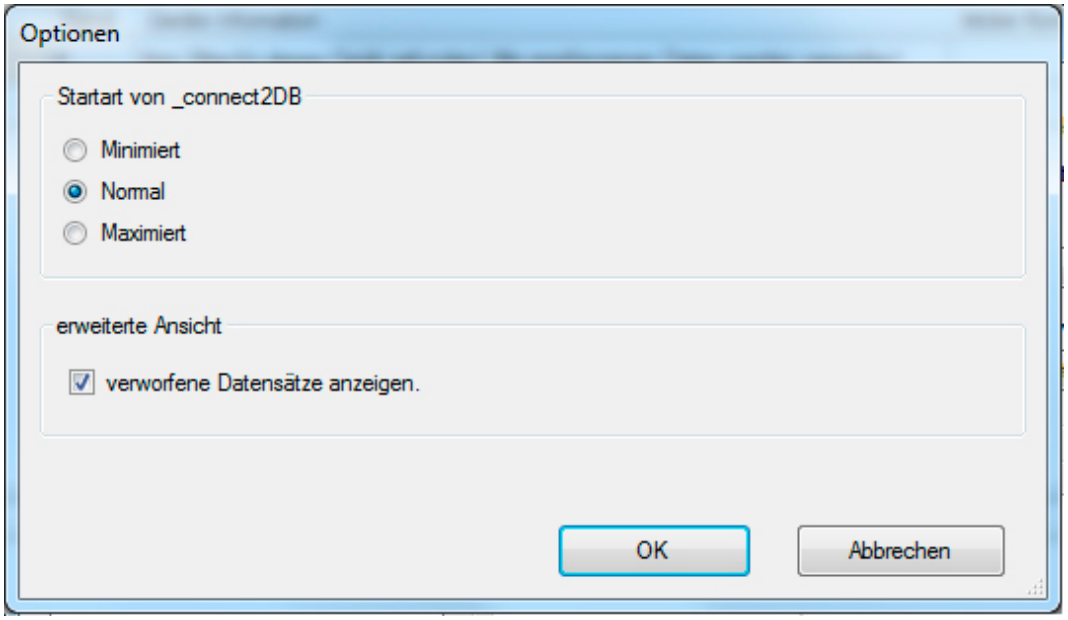


Illustration 25: Options

12

Settings on tab "window"	
"Start mode of _connect2DB"	Display of program window after program start. "Minimized": The program starts in a minimized window and can be displayed using the task bar. "Normal": The program starts in a window that can be moved and scaled on the screen. "Maximized": The program starts in a window maximized to the screen size.
"Display rejected data sets."	activated: The last three data records, that have not been saved in the database table due to the filter criteria, are listed in the lower part of the display area (error display). The list contains the rejected data records with the following data: – Date and time of transmission – Device from which data originates – Filters used – Data record

12.17 Deleting database contents

During data export, data is not deleted. The following options are available to delete database contents:

– Delete database by means of integrated mechanisms (especially when *MS SQL Server* is used).
– Delete database contents using the *_connect2DB.Delete* program. The program is available via the *_connect2DB* menu.

➪ "Extras" / "Clear contents of database..."
➪ Enter password.
➪ <Forward >>
➪ Enter the creation time of the database contents to be deleted.
➪ <Forward >>
➪ Select database tables whose data is to be deleted.
➪ <Complete>
➪ Check and confirm security query.

12.18 Backing up and reloading data

The *_connect2DB* program supports data backup when a *MS Access* database is used. Enter the settings for data backup and restore in the *_connect2DB* configuration, see page 101. Only data stored in the *MS Access* database is backed up. Filters are not saved.

With the *MS SQL Server*, use the corresponding Microsoft data backup tools.

➪ "Extras" / "Options"

Saving data manually

➪ Open the "Backup" tab.
➪ <Carry out backup now>

A backup file is saved in the configured target directory.

Saving data automatically

➪ Open the "Backup" tab.
➪ Activate "Backup database every day" and enter the time for daily backup, see page 101.

The data is automatically backed up at the set time every day.

Loading data backups

➪ Open the "Restore" tab.
➪ <Load backups>
➪ Select backup file.

➪ <Forward >>

When loading data backups, the current data gets lost. Save the current data before loading data backups.

➪ After checking the security query, click <Finish>.

➪ Enter password and confirm.

The selected data backup will be loaded.

12.19 Convert DBConvert database

For the configuration of an existing SQL database *_connect2DB* automatically opens a wizard to convert its table to Unicode. This conversion is necessary to allow error-free functioning of the program.

The wizard can also be directly started via *C:\Programme\Bizerba\BTC \BTC2DB\DBConvert.exe*.

This process may take several minutes or hours depending on the SQL server. It is recommended not to cancel this process.

Creation of a data backup

➪ Perform backup of existing database.

➪ Click <Next >>.

Overview of selected database

In this step the name of the database, user and existing tables are displayed.

➪ Click <Next >>.

Overview of conversion

In this window you can see which tables can be automatically converted and which tables can be manually converted at a later time, if needed.
Tables marked with a red "X" are too large (> 4.000.000 data records) to be processed using the wizard.

➪ Click <Next >>.

Tables processed one after another. In the last column you can see a progress bar. The conversion of a database table column may take several minutes depending on the number of data records.

Completion of conversion
As a last step you will see a success message and possibly not correctly converted tables as a list.
Reasons for an incorrect conversion

- SQL server timeout: The SQL server has sent back a timeout w/o explanation of any reasons. Reasons for a timeout could be:
  - Less performance of server computer
  - Many data in one data record
- Error in conversion: An error occurred during conversion. This error is displayed and it will be continued with the next table. Possibly appeared errors can be found in the log file under *C:\ProgramData\Bizerba\BCT\log\BCT2DB - Convert\Log*
  Possible errors:
  - Line length exceeds the maximum (8060 bytes)
  - Server can no longer be reached
  - Insufficient hard-disk space

13 _connect2SAP

13.1 Overview

_connect2SAP connects the Bizerba device world to the *SAP* environment. The program consists of the following components:

- *_connect2SAP Frontend*, see page 112
- *_connect2SAP Registry*, see page 113
- *_connect2SAP Spooler*, see page 113
- *_connect2SAP Viewer*, see page 114

13.2 Installation

If the *_connect2SAP* components have not been installed together with *_connect.BRAIN*, execute the *_connect.BRAIN* setup again to install this component. Select "Change", set the "_connect2SAP" option and follow the instructions on the screen.

Before installing *_connect2SAP*, make sure SAP GUI is installed. If SAP GUI has not been installed yet, install it prior to the *_connect2SAP* component.

The components *_connect2SAP Registry* and *_connect2SAP Spooler* can be installed as service or program. Installation as service has the advantage that the communication will be activated automatically whenever the PC is started. Otherwise the components have to be manually started via the start menu.

Please take into account that the services have a service name and a display name which both differ from the exe file name. To start or terminate services via the command prompt using `net start/stop`, the service name or display name may be used.

Service names:

- *Connect2SapRegistry*
- *Connect2SapSpooler*

Display names:

- *Bizerba _connect.BRAIN SapRegistry*
- *Bizerba _connect.BRAIN SapSpooler*

Exe files:

- *BCT2SAPRegistry.exe*
- *BCT2SAPSpooler.exe*

Changing operating mode later (service/program)

➪ Call up the command line window via the start menu.

➪ Go to the installation directory (usually: *C:\Programme\Bizerba*) and there to the *..\BCT\BCT2SAP* subdirectory.

Setting as service

➪ `BCT2SAPRegistry.exe -service`

or

➪ `BCT2SAPSpooler.exe -service`

Setting as program

➪ `BCT2SAPRegistry.exe -regserver`

or

➪ `BCT2SAPSpooler.exe -regserver`

Setting to "Automatic" for "Start type" service

➪ Open the "Services" windows via the system administration.

➪ Open the "Properties" of the service via the context menu.

➪ Set "Start type" to "Automatic".

Terminating service and deleting entries from the registry

➪ <Start> / "Administration" / "Services"

➪ Terminate the service via the context menu.

➪ Go to the command line window.

➪ `BCT2SAPRegistry.exe -unregserver`

or

➪ `BCT2SAPSpooler.exe -unregserver`

➪ Close command line window.

13.3 Configuration

13.3.1 _connect2SAP configuration

Configure *_connect2SAP* using the *_connectConfig* program, see page 36.

13.3.2 SAP configuration

To control the *_connect2SAP* server from the *SAP* system, the connection data on the R/3 side has to be maintained. The technical properties of the RFC connection are summed up under a logical name that is indicated in a program that was edited in the *SAP* programming language *ABAP*. Thus, when developing the *ABAP* client, the large amount of connection data must not be manually entered any longer. The connection data is exclusively maintained using the *SAP* transaction *SM59*. First create a RFC destination as described below. There are two possibilities for activation:

<Start-up>: When using the <Start-up> activation mode, communication between *SAP* and *_Connect.BRAIN* is established via *_connect2SAP Frontend*.

<Recording>: Create a registered server using <Recording>. The communication is carried out via the program components *_connect2SAP Registry* and *_connect2SAP Spooler*. In the "Program ID" field, specify the name used by *_connect2SAP Registry* or *_connect2SAP Spooler* when logging on *SAP* gateway. This is the "SAP Destination" registered in *_connectConfig*, see page 36.

➪ In the *SAP* system, open the "SAP Easy Access" windows.

➪ Select "SAP menu" / "Tools" / "Administration" / "Management" / "Network" / "SM59 - RFC destinations".

13

➪ Select "TCP/IP connections" in the "Display and maintenance of RFC destinations" window and click <Create>.

➪ Create a RFC destination with the following properties:

Input field	Value	Description
RFC destination	in the examples: DEST_Z_RFC_BCT_FRONTEND or DEST_Z_RFC_REG	Name of RFC destination. *SAP* applications use this name to access the RFC destination, see page 110. The name is freely selectable.
Connection type	T	TCP/IP connection

Deactivate the "Trace" checkbox for standard operation to avoid that the system will be excessively loaded due to the large amount of incoming trace data. Activate it only when you need trace files for troubleshooting. Delete trace files generated by the *SAP* system via the "delete SAP Trace" checkbox in *_connectConfig*.

Activation using start-up

➪ <Start-up>

➪ <Front end workstation>

➪ In the "Program" field, specify the *_connect2SAP Frontend* program with the complete path.

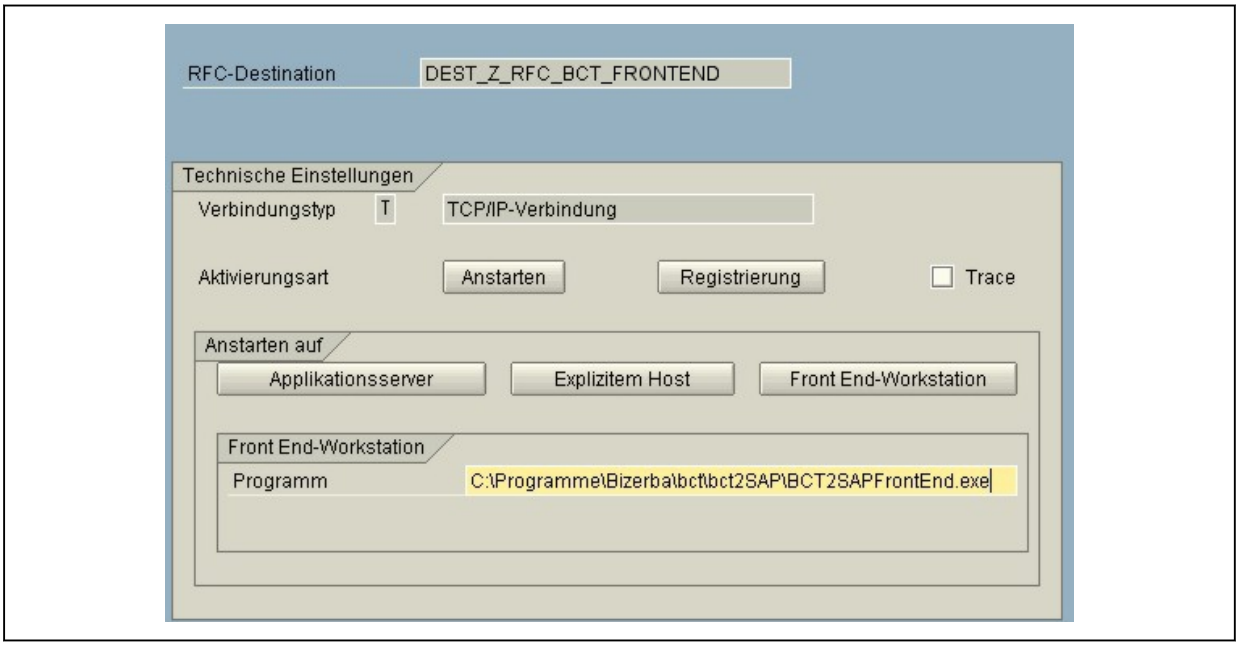


Illustration 26: RFC destination DEST_Z_RFC_BCT_FRONTEND, activation using start-up

Activation using registration

➪ <Recording>

➪ As "Program ID", enter the "SAP Destination" that has been registered in *_connect-Config*, see page 36.

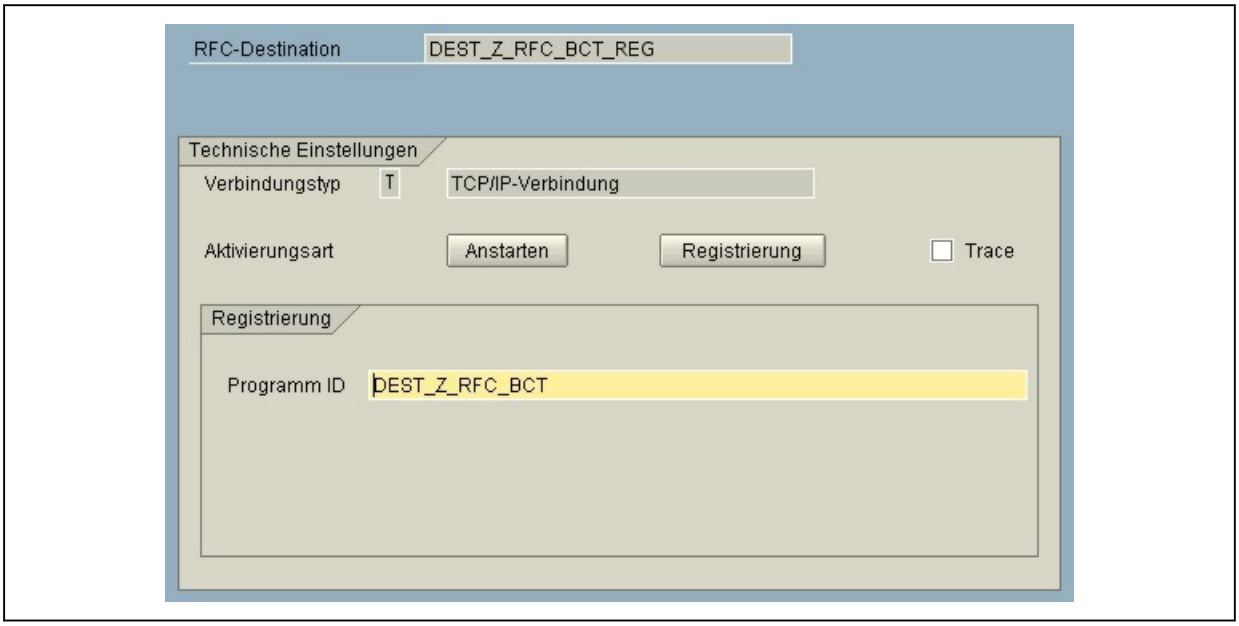


Illustration 27: RFC destination DEST_Z_RFC_BCT_REG, activation using registration

➪ Start *_connect2SAP Registry* or *_connect2SAP Spooler* and test the connection using the *SM59* transaction.

When the connection is established, a log with the duration of the transmission time for the individual data packets will be generated.

13.3.3 Packing table configuration

The configuration described below is required for using the *Packing table* function (transaction *Hupast*) in *SAP* and connecting it to *_connect.BRAIN*. This function is used to register the weight of incoming deliveries.

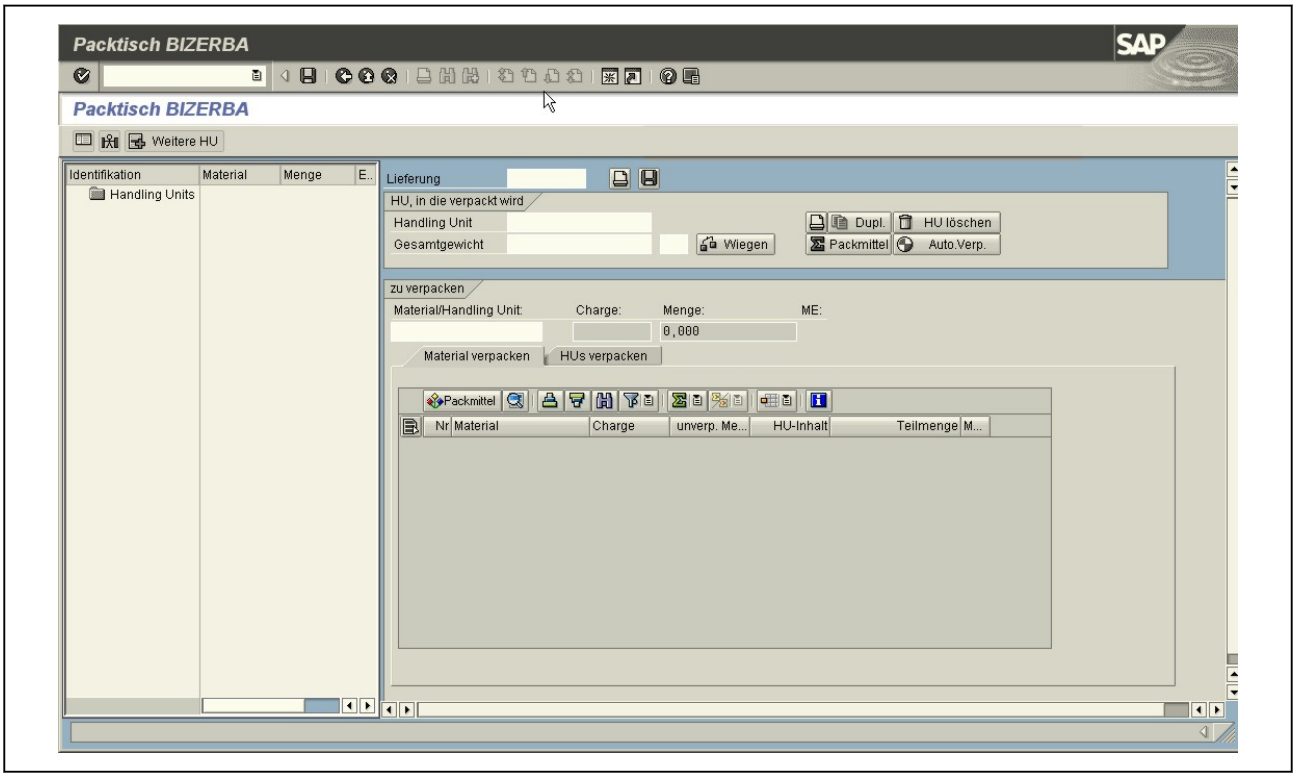


Illustration 28: *SAP* packing table (*Hupast* transaction)

The packing table data is maintained in the *Hupast_C* transaction. Complete the scale as described below.

I *_connectConfig*, the "Hold Connection" checkbox has to be activated for the specified scale, see page 46.

➪ In *SAP*, open the packing table profile (*Hupast_C* transaction)

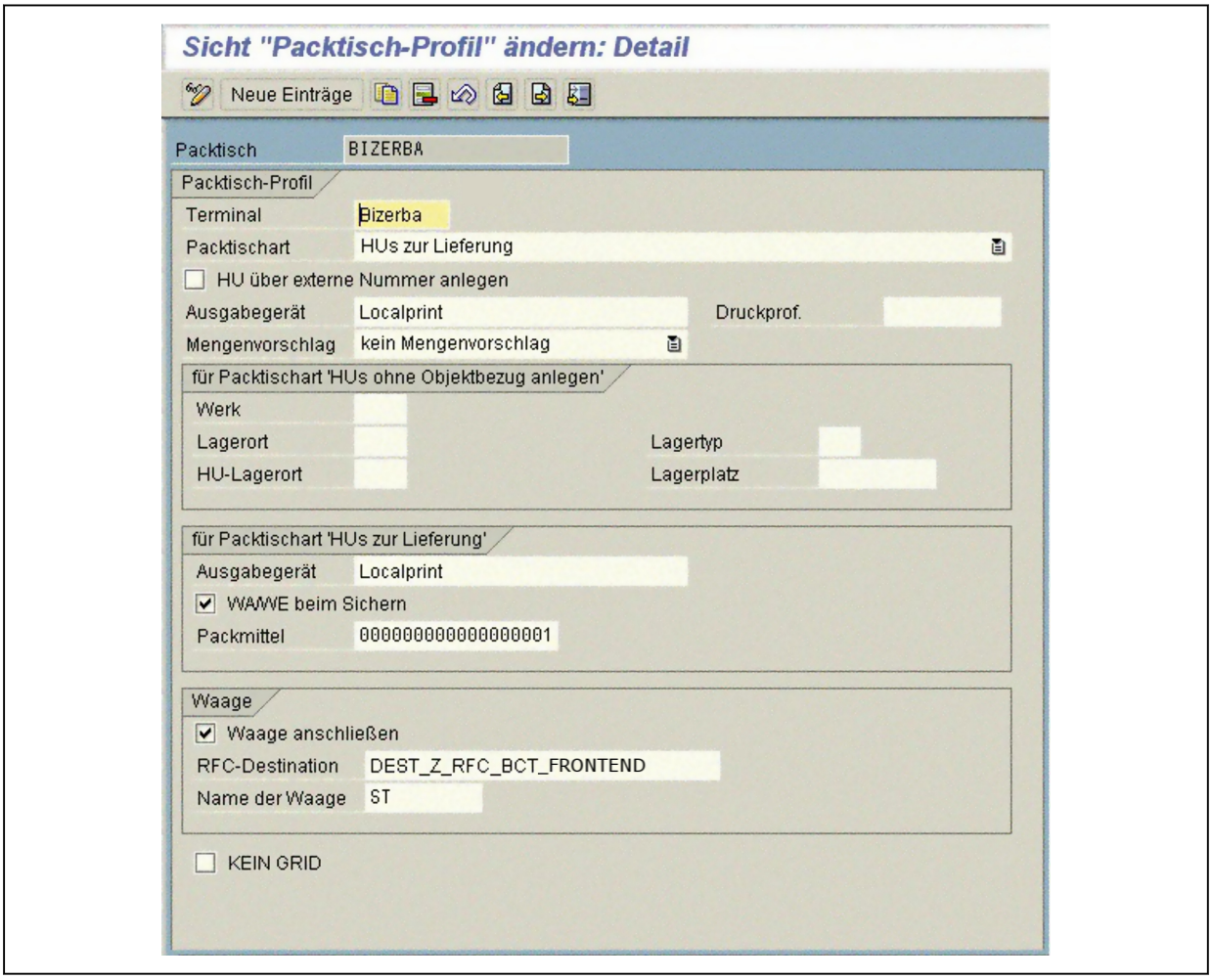


Illustration 29: *SAP* packing table profile (*Hupast_C* transaction)

13

➪ Make the following settings in the "scale" area.

Pay attention to upper and lower case when entering the scale name and the RFC destination.

Input field	value	Description
Connect scale	activating	
RFC destination	In this example: DEST_Z_RFC_BCT_FRONTEND	Name of RFC destination created in *SM59* which has to be accessed. Select RFC destination with <Start-up> setting via <Front End-Workstation>, see page 108.
Scale name	In this example: ST	Name of the scale as created in *_connectConfig*.

13.4 _connect2SAP Frontend

13

_connect2SAP Frontend supports communication between *SAP* and *_connect.BRAIN* via Frontend. The advantage of this program is that all settings are identical for each PC in *SAP* even if different devices are connected to each PC.

To work with the program, use start-up via the Frontend workstation for activation in the RFC destination (*SM59* transaction), see page 108. If *SAP* accesses the RFC destination, *_connect2SAP Frontend* starts automatically and handles communication with *_connect.BRAIN*. After transmitting the data, the program sends the result to *SAP* and ends.

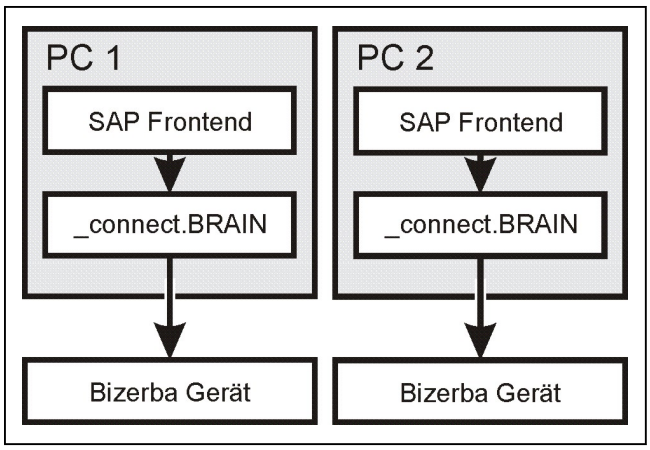


Illustration 30: Communication via Frontend

Possible functional components
– Z_RFC_BCT_MULTI, see page 117. – Z_RFC_BCT, see page 116.

13.5 _connect2SAP Registry and _connect2SAP Spooler

The programs *_connect2SAP Registry* and *_connect2SAP Spooler* support communication between *_connect.BRAIN* and *SAP* via a RFC destination. They can be installed as service or as program. During start, these components log on the *SAP* system using the "SAP Destination" specified in *_connectConfig*, see page 36. In *SAP*, "SAP Destination" is registered as program ID in the RFC destination. As soon as *SAP* uses the RFC destination, *SAP* tries to establish a connection, transfer data to the program and return the results to the *SAP* system by using the program ID.

The advantage of this procedure is that only one PC is used for communication with the devices. All *SAP* programs can establish a connection to this PC.

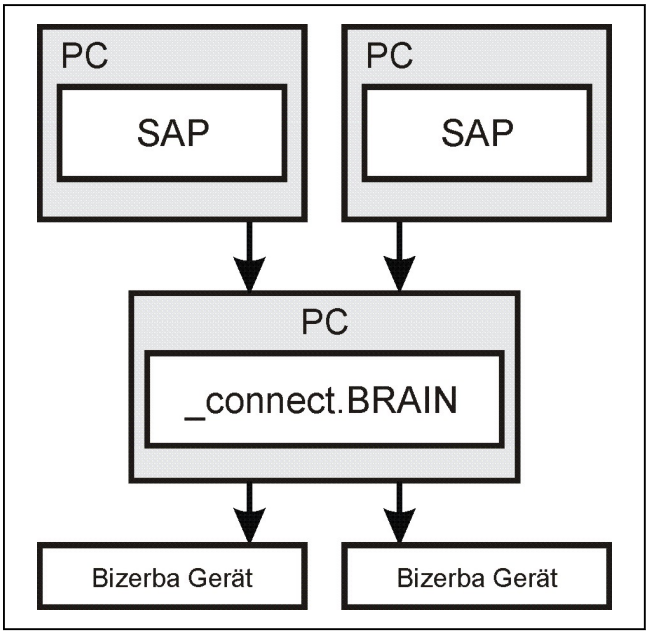


Illustration 31: Communication using *_connect2SAP Registry* and *_connect2SAP Spooler*

_connect2SAP Registry

Via *_connect2SAP Registry*, the *SAP* system can communicate with the devices, e.g. to determine weight values of a scale.

Possible functional components
– Z_RFC_BCT_MULTI, see page 117 – Z_RFC_BCT, see page 116

_connect2SAP Spooler

_connect2SAP Spooler supports communication between *SAP* and *_connect.BRAIN* when printing. *SAP* uses this component to send print jobs to the server. The spooler manages the print jobs and handles errors. Print jobs can be deleted using the corresponding *_connect2SAP Viewer* monitoring program.

Example of a print job

```
szHeader = I|PV01|PW00|PW02|GW09|GT01|LX02
szData = 0|1|2|Testlabel|ST
```

PV01:	Print command
PW00:	Label type
PW02:	Handle
GW09:	Labeling mode
GT01:	Text field 1
GT02:	Text field 2
LX02:	End of a V command

Possible functional components
– Z_RFC_BCT_PRINT, see page 117 – Z_RFC_BCT_SPOOLER, see page 118

13.6 _connect2SAP Viewer

13

13.6.1 Overview

The program lists the print jobs sent by the *SAP* system. Print jobs can be cancelled and deleted here. The program is a client application and does not function without server.

13.6.2 Starting the program

➪ Call up *_connect2SAP Viewer* via the start menu.

The start window of the program appears.

13.6.3 Structure of the program

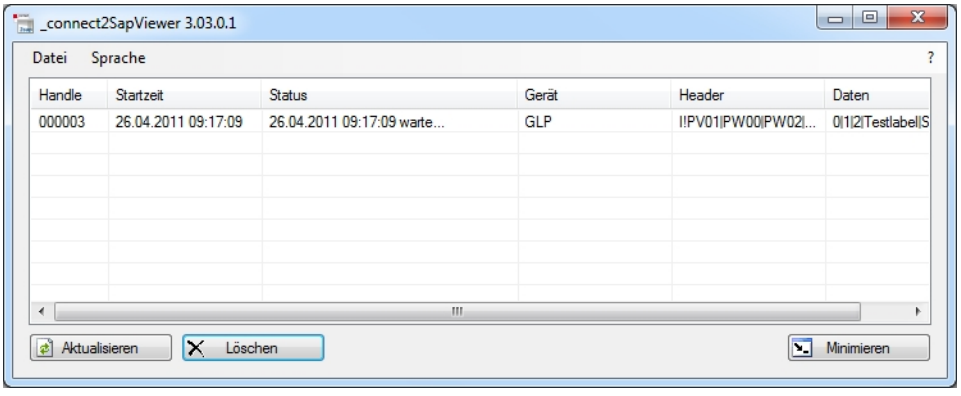


Illustration 32: User interface *_connect2SAP Viewer*

In addition to typical Windows elements, the user interface contains a table with the print jobs in progress.

13.6.4 Menu bar and toolbar functions

"File" menu

"Create testjob..."	Create print job manually and display it in *_connect2SAP Viewer*. Enter the following print job data in the "Create testjob" window. – Device – Number of labels to be printed – Print job header – Data line
"Refresh"	Update print job list. The function is also available as button with the same name.
"Minimize"	Minimize program. The program appears as symbol in the right part of the task bar and can be restored or terminated via the context menu. The function is also available as button with the same name.
"Exit"	Close program.

"Language" menu
Select here a language for the user interface.

Additional functions

<Delete>	Cancel and delete currently selected print job.
	Show application information, see page 14.

13.7 Functional components

13.7.1 Overview of functions

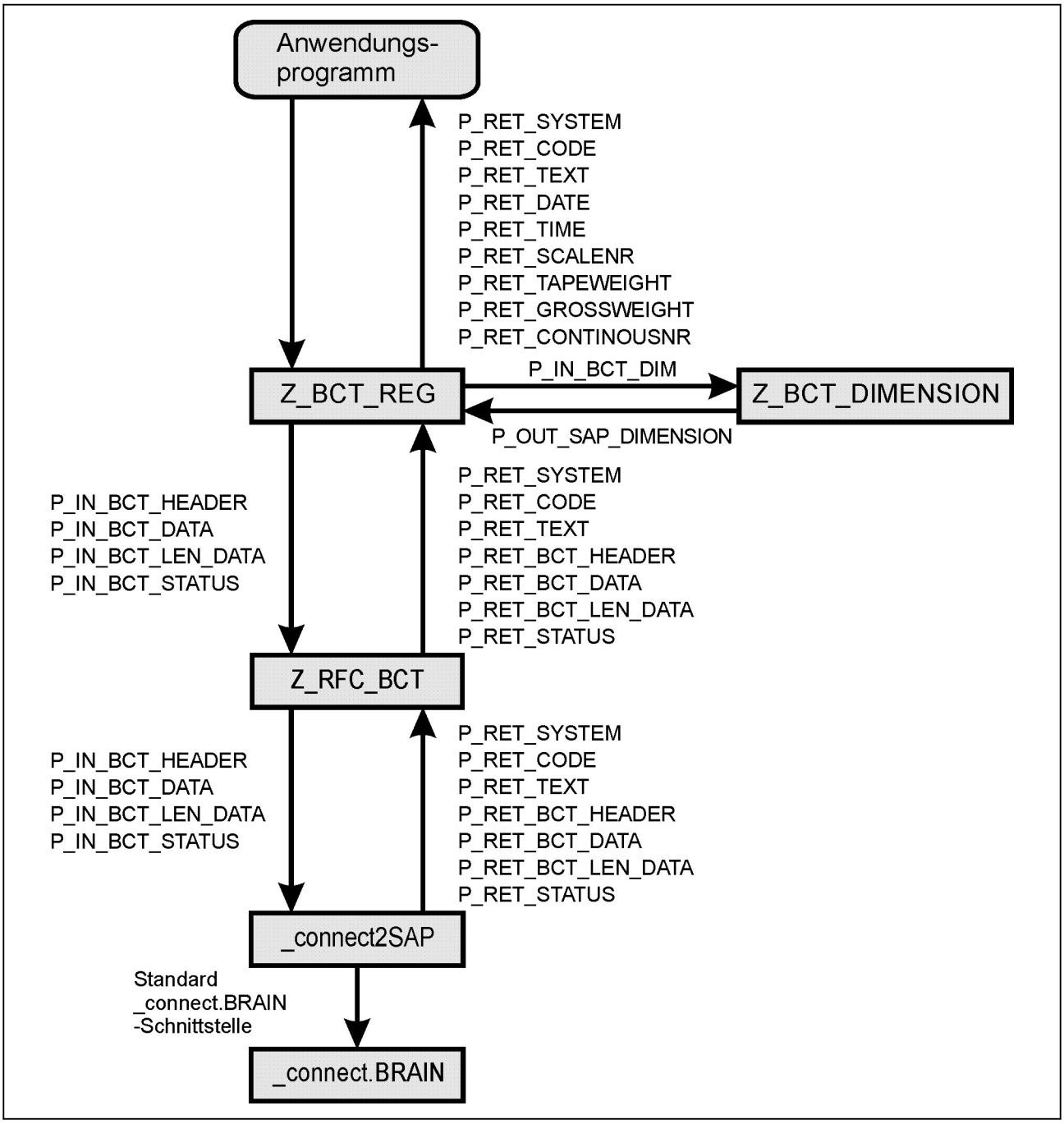


Illustration 33: Overview of *SAP* functions

13.7.2 Z_RFC_BCT

```
FUNCTION Z_RFC_BCT.
*"----------------------------------------------------------------------
```

""Local interface:

```
*"  IMPORTING
*"    VALUE(P_IN_BCT_HEADER) TYPE CHAR1024
```

```
*"    VALUE(P_IN_BCT_DATA) TYPE CHAR1024
*"    VALUE(P_IN_BCT_LEN_DATA) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_IN_BCT_STATUS) LIKE BAPIRET2-MESSAGE
*"  EXPORTING
*"    VALUE(P_RET_SYSTEM)LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_CODE) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_TEXT) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_HEADER) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_DATA) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_LEN_DATA) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_STATUS) LIKE BAPIRET2-MESSAGE
*"  EXCEPTIONS
*"    COMMUNICATION_FAILURE
*"    SYSTEM_FAILURE
*"----------------------------------------------------------------------
ENDFUNCTION.
```

13.7.3 Z_RFC_BCT_MULTI

```
FUNCTION Z_RFC_BCT_MULTI.
*"----------------------------------------------------------------------
```

""Local interface:

```
*"  IMPORTING
*"    VALUE(P_IN_BCT_HEADER) TYPECHAR1024
*"    VALUE(P_IN_BCT_DATA) TYPECHAR1024
*"    VALUE(P_IN_BCT_LEN_DATA) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_IN_BCT_STATUS) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_IN_BCT_DEVICE) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_IN_BCT_COUNT) LIKEBAPIRET2-MESSAGE
*"  EXPORTING
*"    VALUE(P_RET_SYSTEM) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_RET_CODE) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_RET_TEXT) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_HEADER) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_DATA) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_LEN_DATA) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_STATUS) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_DEVICE) LIKEBAPIRET2-MESSAGE
*"  EXCEPTIONS
*"    COMMUNICATION_FAILURE
*"    SYSTEM_FAILURE
*"----------------------------------------------------------------------
ENDFUNCTION.
```

13.7.4 Z_RFC_BCT_PRINT

```
FUNCTION Z_RFC_BCT_PRINT.
*"----------------------------------------------------------------------
```

""Local interface:

```
*"  IMPORTING
*"    VALUE(P_IN_BCT_HEADER) TYPE CHAR1024
*"    VALUE(P_IN_BCT_DATA) TYPE CHAR1024
*"    VALUE(P_IN_BCT_LEN_DATA) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_IN_BCT_STATUS) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_IN_BCT_DEVICE) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_IN_BCT_COUNT) LIKE BAPIRET2-MESSAGE
```

```
*"  EXPORTING
*"    VALUE(P_RET_SYSTEM) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_CODE) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_TEXT) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_HEADER) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_DATA) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_LEN_DATA) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_STATUS) LIKE BAPIRET2-MESSAGE
*"    VALUE(P_RET_BCT_DEVICE) LIKE BAPIRET2-MESSAGE
*"  EXCEPTIONS
*"    COMMUNICATION_FAILURE
*"    SYSTEM_FAILURE
*"----------------------------------------------------------------------
ENDFUNCTION.
```

13.7.5 Z_RFC_BCT_SPOOLER

```
FUNCTION Z_RFC_BCT_SPOOLER.
*"----------------------------------------------------------------------
```

""Local interface:

```
*"  IMPORTING
*"    VALUE(P_IN_BCT_DEVICE) LIKEBAPIRET2-MESSAGE
*"    VALUE(P_IN_BCT_STATUS) LIKEBAPIRET2-MESSAGE
*"  EXCEPTIONS
*"    COMMUNICATION_FAILURE
*"    SYSTEM_FAILURE
*"----------------------------------------------------------------------
```

*status 1 = Start spooler.

*status 2 = Stop spooler.

*status 3 = Delete all active orders.

```
ENDFUNCTION.
```

13.7.6 Z_BCT_LABEL_GLP

The layout *WE_Chargenlabel.lay* must be present on the GLP.

```
*
```

* Label printing report on Bizerba GLP printer via RFC-FB.

```
*
report z_bct_label_glp message-id z0 line-size 255.
*
```

* Tables:

```
*
tables:
  mseg,
  makt.
*
```

* Number of labels

```
*
parameters:
```

```
  p_anz type i default 1.
 *
```

* Communication structure for FB Z_RFC_BCT:

```
*
data:
  p_in_bct_header like bapiret2-message,
  p_in_bct_data like bapiret2-message,
  p_in_bct_len_data like bapiret2-message,
  p_in_bct_status like bapiret2-message,
  p_in_bct_device like bapiret2-message,
  p_ret_system like bapiret2-message,
  p_ret_code like bapiret2-message,
  p_ret_text like bapiret2-message,
  p_ret_bct_header like bapiret2-message,
  p_ret_bct_data like bapiret2-message,
  p_ret_bct_len_data like bapiret2-message,
  p_ret_bct_status like bapiret2-message,
  p_ret_bct_device like bapiret2-message.
 *
```

* Help fields:

```
 *
data:
  zerf_menge(13) type c,
  fl type i.
```

* GxNet	Parameter-Name	Parameters
* GT01	GGT_ATX1	Character data
*	(PLU memory 1: part designation)	
* GT02	GGT_ATX2	Character data
*	(PLU memory 2: measuring unit)	
* GT21	GT_CAB1	Barcode with structure rule
*	(material number)	
* GT22	GGT_CAB2	Barcode with structure rule
*	(quantity)	
* GT23	GT_CAB3	Barcode with structure rule
*	(batch data)	
* GW09	GGW_AUSZEICH_ART	2 (fixed weight)
*	(Labeling mode)	
* LV01	LGV_SEQUENZ	-
*	(Device configuration)	
* LX02	LGX_CLOSE	-
*	(logical end of command)	
* PV01	PSV_PCK	-
*	(package synchronous labeling data)	

```
* PW00        PSW_ETIKETTYP                                  0 (Normal label)
*             (Specification of label data)

* PW02        PSW_PACKHDL                                    0-100
*             (package handle: cons. number)

*
```

* Fill data:

```
  mseg-matnr = '000000071432080001'.
  mseg-charg = '1234567890'.
  mseg-erfmg = '1542.380'.
  mseg-meins = 'ST'.
  makt-maktx = 'Testlabel'.
write mseg-erfmg to zerf_menge no-sign no-grouping.
if zerf_menge+9(4) = ',000'.
  clear zerf_menge+9(4).
endif.
shift zerf_menge left deleting leading space.
translate zerf_menge using ',.'.
 fl = strlen( zerf_menge ).
do p_anz times.
*
```

* send barcodes:

```
*
  p_in_bct_header = '!LV01|GT21|GT22|GT23|LX02'.
  concatenate
    '0001{300502000200000070}0002{'mseg-matnr '}|'
    '0001{300502000200000070}0002{'zerf_menge(fl) '}|'
    '0001{300502000200000070}0002{'mseg-charg '}'
    into p_in_bct_data.
  perform call_rfc_bct.
*
```

* Send text data:

```
*
  p_in_bct_header ='!PV01|PW00|PW02|GW09|GT01|GT02|LX02'.
  concatenate
    '0|1|2|' makt-maktx '|' mseg-meins
    into p_in_bct_data.
  perform call_rfc_bct.
enddo.
* ============================================
```

* Call RFC functional component Z_RFC_BCT_MULTI:

* Starts destination DEST_Z_RFC_BCT.

* The PC program is linked with this destination.

```
* ============================================
form call_rfc_bct.
*
```

* Determine field length:

```
*
  p_in_bct_len_data = strlen( p_in_bct_data ).
  p_in_bct_status = '0'.
  p_in_bct_device = 'GLP1'.
```

```
*
```
* Call RFC functional component:
```
*
  call function 'Z_RFC_BCT_MULTI' destination'DEST_Z_RFC_BCT'
   exporting
     p_in_bct_header = p_in_bct_header
     p_in_bct_data = p_in_bct_data
     p_in_bct_len_data =p_in_bct_len_data
     p_in_bct_status = p_in_bct_status
     p_in_bct_device = p_in_bct_deviceimporting
     p_ret_system = p_ret_system
     p_ret_code = p_ret_code
     p_ret_text = p_ret_text
     p_ret_bct_header =p_ret_bct_header
     p_ret_bct_data = p_ret_bct_data
     p_ret_bct_len_data =p_ret_bct_len_data
     p_ret_bct_status =p_ret_bct_status
     p_ret_bct_device =p_ret_bct_device
   exceptions
     communication_failure = 1
     system_failure = 2
     others = 3.
skip.
write: /'p_in_bct_header ', p_in_bct_header(200),
     /'p_in_bct_data ',p_in_bct_data(200),
     /'p_in_bct_len_data ',p_in_bct_len_data(200),
     /'p_in_bct_status ',p_in_bct_status(200),
     /'p_in_bct_device ',p_in_bct_device(200).
write: /'p_ret_system ', p_ret_system(200),
     /'p_ret_code ', p_ret_code(200),
     /'p_ret_text ', p_ret_text(200),
     /'p_ret_bct_header ',p_ret_bct_header(200),
     /'p_ret_bct_data ',p_ret_bct_data(200),
     /'p_ret_bct_len_data ',p_ret_bct_len_data(200),
     /'p_ret_bct_status ',p_ret_bct_status(200),
     /'p_ret_bct_device ',p_ret_bct_device(200),
     /'sy-subrc ', sy-subrc.
*
```
* Initialize input fields for next call:
```
*
  clear p_in_bct_header.
  clear p_in_bct_data.
  clear p_in_bct_len_data.
  clear p_in_bct_status.
endform.
```

13.7.7 Z_BCT_DIMENSION

```
FUNCTION Z_BCT_DIMENSION.
*"----------------------------------------------------------------------
```
"" Local interface:
```
*"IMPORTING
*"   REFERENCE(P_IN_BCT_DIM) TYPE STRING
*"EXPORTING
*"   REFERENCE(P_OUT_SAP_DIMENSION) TYPESTRING
```

```
*"   REFERENCE(P_OUT_SAP_VALUE) TYPE STRING
*"----------------------------------------------------------------------
DATA :
            ilen                    TYPE i,
            dimensiontab            TYPE TABLE OF string,
            szeinheit               TYPE string,
            szstellen               TYPE string,
            szvalue                 TYPE string,
            szvorkomma              TYPE string,
            sznachkomma             TYPE string,
            nhelp                   TYPE i.
```

* Split up data from ;-3;123 kg

```
   SPLIT p_in_bct_dim AT ';' INTO TABLEdimensiontab.
   READ TABLE dimensiontab INDEX 1 INTOszeinheit.
   READ TABLE dimensiontab INDEX 2 INTOszstellen.
   READ TABLE dimensiontab INDEX 3 INTO szvalue.
```

* Check whether decimal places are present:

```
  IF szstellen < 0.
```

* Check whether filling is required at front

```
    ilen = strlen( szvalue ).
    nhelp = abs( szstellen ) + 1.
    WHILE ilen < nhelp .
      ilen = ilen + 1.
      CONCATENATE '0' szvalue INTOszvalue .
    ENDWHILE.
```

* Insert point:

```
    ilen = strlen( szvalue ).
    ilen = szstellen + ilen.
    szvorkomma = szvalue+0(ilen).
    sznachkomma = szvalue.
    SHIFT sznachkomma BY ilen PLACES.
    CONCATENATE szvorkomma '.' INTO szvalue.
    CONCATENATE szvalue sznachkomma INTOszvalue .
  ELSE.
```

* Fill with zeros at back:

```
    WHILE szstellen > 0.
      szstellen = szstellen - 1.
      CONCATENATE szvalue '0' INTOszvalue .
    ENDWHILE.
  ENDIF.
* CONCATENATE szValue szEinheit into szValue SEPARATED BY SPACE.
* write: / 'Dimension ' , szValue.
    p_out_sap_value = szvalue.
    p_out_sap_dimension = szeinheit.
ENDFUNCTION.
```

13.7.8 Z_BCT_REG

The function divides a logon data record received by *_connect.BRAIN* in *SAP* fields.

Example:

Recording data set:

```
 I?LV01|RX04|LX02 Postenregistrierung addierend
```

Received data:

```
I!GV01|GW01|GT08|GT0A|GT11|GL0E|GL0C|GT0E|GL0A|GL15|GD07|GD01|GD02|LX02-1|
Scale_49|11000000|100000000+0|02122002|1618|19596352|101|101|kg;-3;75|kg;-3;75|
kg;-3;0
```

Data to SAP:

P_RET_SYSTEM	System number
P_RET_CODE	Error code
P_RET_TEXT	Error text
P_RET_DATE	Date
P_RET_TIME	Time
P_RET_SCALENR	Scale number
P_RET_TAREWEIGHT	Tare weight
P_RET_GROSSWEIGHT	Gross weight
P_RET_CONTINUOUSNR	Verification number

```
 FUNCTION Z_BCT_REG.
```

""Local interface:

```
*"----------------------------------------------------------------------
 *" EXPORTING
*"   REFERENCE(P_RET_SYSTEM) TYPE STRING
*"   REFERENCE(P_RET_CODE) TYPE STRING
*"   REFERENCE(P_RET_TEXT) TYPE STRING
*"   REFERENCE(P_RET_DATE) TYPE D
*"   REFERENCE(P_RET_SCALENR) TYPE STRING
*"   REFERENCE(P_RET_TAREWEIGHT) TYPE STRING
*"   REFERENCE(P_RET_GROSSWEIGHT) TYPESTRING
*"   REFERENCE(P_RET_CONTINUUSNR) TYPESTRING
*"   REFERENCE(P_RET_TIME) TYPE T
*"   REFERENCE(P_RET_DIMENSION) TYPE STRING
*" EXCEPTIONS
*"   COMMUNICATION_FAILURE
*"   SYSTEM_FAILURE
*"----------------------------------------------------------------------
```

```
DATA:
            text                    TYPE string,
            datatab                 TYPE TABLE OF string,
            headertab               TYPE TABLE OF string,
            ni                      TYPE i,
            szheader(220)           TYPE c,
            szheader(220)           TYPE c,
            szanswerheader(220)     TYPE c,
            szanswerdata(220)       TYPE c,
            ctyp                    TYPE char1,
            szdimensions            TYPE string,
            szretsystem(220)        TYPE c,
            szretcode(220)          TYPE c,
            szrettext(220)          TYPE c,
            szlendata(220)          TYPE c,
            szstatus(220)           TYPE c,
            szvalue                 TYPE string.

 szheader = 'I?LV01|RX04|LX02'.
szdata = ''.
szvalue = strlen( szdata ).
CALL FUNCTION 'Z_RFC_BCT' DESTINATION 'DEST_Z_RFC_BCT'
  EXPORTING
    p_in_bct_header = szheader
    p_in_bct_data = szdata
    p_in_bct_status = '0'
    p_in_bct_len_data = szvalue
  IMPORTING
    p_ret_system = szretsystem
    p_ret_code = szretcode
    p_ret_text = szrettext
    p_ret_bct_header = szanswerheader
    p_ret_bct_len_data = szlendata
    p_ret_bct_status = szstatus
    p_ret_bct_data = szanswerdata
  EXCEPTIONS
    communication_failure = 1
    system_failure = 2
    OTHERS = 3.
CASE syst-subrc.
  WHEN 0.
   WRITE : 'header', szanswerheader.
   WRITE : 'data', szanswerdata.
   WRITE : 'szRetSystem', szretsystem.
   WRITE : 'szStatus', szstatus.
   WRITE : 'szLenData', szlendata.
   WRITE : 'szRetText', szrettext.
   WRITE : 'szRetCode', szretcode.
   CONDENSE szretcode.
   IF szretcode = '0'.
    WRITE : 'ok'.
```

```
* Customer program
* Separate header I! or I?
    SHIFT szheader BY 2 PLACES.
* Data in table:
    SPLIT szanswerheader AT '|' INTO TABLEheadertab.
    SPLIT szanswerdata AT '|' INTO TABLEdatatab.
* Evaluate data:
    ni = 1.
    LOOP AT headertab INTO text.
     CASE text.
       WHEN 'GL15'.
         READ TABLEdatatab INDEX ni INTO P_RET_CONTINUUSNR.
       WHEN 'GT08'.
         READ TABLEdatatab INDEX ni INTO P_RET_SCALENR.
       WHEN 'GL0E'.
* Date must be indicated like this: year day month:
         READ TABLEdatatab INDEX ni INTO szvalue.
         CONCATENATEszvalue+4(4) szvalue+0(4) INTO szvalue.
         P_RET_DATE= szvalue.
       WHEN 'GL0C'.
         READ TABLEdatatab INDEX ni INTO P_RET_TIME.
       WHEN 'GD02'.
         READ TABLEdatatab INDEX ni INTO szvalue.
         CALLFUNCTION 'Z_BCT_DIMENSION'
           EXPORTING
             p_in_bct_dim= szvalue
           IMPORTING
             p_out_sap_value= szvalue
             p_out_sap_dimension= szdimensions.
         P_RET_TAREWEIGHT= szvalue.
       WHEN 'GD07'.
         READ TABLEdatatab INDEX ni INTO szvalue.
         CALLFUNCTION 'Z_BCT_DIMENSION'
           EXPORTING
             p_in_bct_dim= szvalue
           IMPORTING
             p_out_sap_value= szvalue
             p_out_sap_dimension= szdimensions.
         P_RET_GROSSWEIGHT= szvalue.
         TRANSLATEszdimensions TO UPPER CASE.
         P_RET_DIMENSION= szdimensions.
     ENDCASE.     " text
* The data type determines whether data is included or not:
     SHIFT text BY 1 PLACES.
     ctyp = text.
     CASE ctyp.
       WHEN 'L' OR 'T' OR 'W'OR 'D'.
         ni = ni +1.
     ENDCASE.     "ctyp
    ENDLOOP.     "headertab
   ENDIF.     "szretcode = 0
* Fill return:
   P_RET_SYSTEM = szretsystem.
```

```
   P_RET_CODE = szretcode.
   P_RET_TEXT = szrettext.
  WHEN 1 OR 2.
* 'communication_failure' or 'system_failure'
   RAISE COMMUNICATION_FAILURE.
  WHEN OTHERS.
```

* Other errors:

```
 ENDCASE.     " syst-subrc'Z_RFC_BCT'
ENDFUNCTION.
```

13.7.9 Z_BCT_NULLSTELLEN

```
FUNCTION Z_BCT_NULLSTELLEN.
*"----------------------------------------------------------------------
```

""Local interface:

```
*" EXPORTING
*"   REFERENCE(SYSTEM_NR) TYPE ZFC_SYSTEM_NR
*"   REFERENCE(SUBRC) TYPE ZFC_SUBRC
*"   REFERENCE(SUBRC_TXT) TYPE ZFC_SUBRC_TXT
*" EXCEPTIONS
*"   ERROR
*"----------------------------------------------------------------------
DATA:

            szheader(220)           TYPE c,
            szdata(220)             TYPE c,
            szanswerheader(220)     TYPE c,
            szanswerdata(220)       TYPE c,
            ctyp                    TYPE char1,
            szdimensions            TYPE string,
            szretsystem(220)        TYPE c,
            szretcode(220)          TYPE c,
            szrettext(220)          TYPE c,
            szlendata(220)          TYPE c,
            szstatus(220)           TYPE c,
            szvalue                 TYPE string.


szheader = 'I!GX02'.
szdata = ''.
szvalue = strlen( szdata ).
CALL FUNCTION 'Z_RFC_BCT' DESTINATION 'DEST_Z_RFC_BCT'
  EXPORTING
    p_in_bct_header = szheader
    p_in_bct_data = szdata
    p_in_bct_status = '0'
    p_in_bct_len_data = szvalue
  IMPORTING
    p_ret_system = szretsystem
    p_ret_code = szretcode
    p_ret_text = szrettext
    p_ret_bct_header = szanswerheader
    p_ret_bct_len_data = szlendata
```

```
    p_ret_bct_status = szstatus
    p_ret_bct_data = szanswerdata
  EXCEPTIONS
    communication_failure = 1
    system_failure = 2
    OTHERS = 3.
 CASE syst-subrc.
  WHEN 0.
    system_nr = szretsystem.
    subrc = szretcode.
    subrc_txt = szrettext.
  WHEN 1 OR 2.
* 'communication_failure' or 'system_failure'
    RAISE error.
  WHEN OTHERS.
* other failure
 ENDCASE.     " syst-subrc'Z_RFC_BCT'
ENDFUNCTION.
```

13.7.10 CFB_RFC_BCT_MULTI

```
FUNCTION CFB_RFC_BCT_MULTI.
*"----------------------------------------------------------------------
```

""Local interface:

```
*"  IMPORTING
*"    VALUE(P_IN_BCT_HEADER) TYPE  TEXT_512
*"    VALUE(P_IN_BCT_DATA) TYPE  STRING
*"    VALUE(P_IN_BCT_LEN_DATA) TYPE  TEXT_512
*"    VALUE(P_IN_BCT_DEVICE) TYPE  TEXT_512
*"  EXPORTING
*"    VALUE(P_RET_SYSTEM) TYPE  TEXT_512
*"    VALUE(P_RET_CODE) TYPE  TEXT_512
*"    VALUE(P_RET_TEXT) TYPE  TEXT_512
*"    VALUE(P_RET_BCT_HEADER) TYPE  TEXT_512
*"    VALUE(P_RET_BCT_DATA) TYPE  TEXT_512
*"    VALUE(P_RET_BCT_LEN_DATA) TYPE  TEXT_512
*"    VALUE(P_RET_BCT_STATUS) TYPE  TEXT_512
*"    VALUE(P_RET_BCT_DEVICE) TYPE  TEXT_512
*"  EXCEPTIONS
*"    COMMUNICATION_FAILURE
*"    SYSTEM_FAILURE
*"----------------------------------------------------------------------
ENDFUNCTION.
```

14 _connectScannerWI

14.1 Overview

_connectScannerWI is used to connect an external barcode scanner to any program that processes text and numeric information. For this, the scanner is connected to the PC where the processing program runs via a serial interface. A service running in the background records the data and sends it to the active program for processing.

_connectScannerWI transfers the scanned data in form of simulated keyboard inputs. The data supplied by the scanner is therefore processed as if the barcode had been input using a keyboard. To control processing, the barcode sequence can be completed by means of additional keyboard commands (e.g. forwarding to a new input field or to the next input dialog).

14.2 Prerequisites

The following prerequisites apply when connecting a barcode scanner:

– The barcode scanner can be connected to a PC serial interface. It can transmit successfully read barcodes in form of numbers and separators to a PC.
– The barcode scanner is set by default. That means, it only transmits correctly read barcode data using CR+LF (carriage return and line feed) as separators. It does not transmit any incorrect or unidentified barcode data, nor does it transmit meta information via barcode type, etc.
– The PC has a free serial interface, to which the barcode scanner can be connected.

14.3 Installation

Select *_connectScannerWI* as component to be installed when installing *_connect.BRAIN*. If *_connect.BRAIN* has been installed without *_connectScannerWI*, proceed as follows:

➪ Open *Windows* system control.
➪ Select "Deinstall program".
➪ Select "Change" in the context menu for *_connect.BRAIN*.
➪ Select "Change program" in the installation program.
➪ <Forward >>
➪ Activate the "_connectScannerWI" check-box.
➪ <Forward >>
➪ Follow the instructions on the screen to complete installation.

During installation, *_connectScannerWI* registers in the Windows autorun path so that it is automatically started with each user login.

Registry path: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`

14.4 Configuration

Configure *_connect2File* using the *_connectConfig* program.

➪ Create device with the following settings, see page 46.

Device settings for the barcode scanner in _connectConfig	
"Devicetype"	"FX Other devices" / "Scanner"
"Connection-type"	"Serial"
"Interface"	Set to the interface the scanner is connected with. Configure the interface according to the scanner parameters (see manual or scanner configuration program). The following parameters are common: – "Baudrate": 9600 – "Databits": 8 – "Parity": "none" – "Stopbits": 1 – "Protocol": "no handshake"
"Dialog"	"_Standard serial IX"
"Logging/size"	Normal operation does not require reporting. Logging of all data is advisable for faultfinding.
"Header-Format"	"Gx/Ix-Net(A!, I!, A?, I?)"
"_connect2SAP"	Save default settings.
"Connection"	Do not activate the "Hold Connection" check-box.
"Status"	Activate the "Device active" check-box.

➪ Adapt settings to the operation of *_connectScannerWI*, see page 36.

14.5 Starting the program

_connectScannerWI is automatically started with each user login. The started program appears as icon in the right part of the task bar. After closing the program, it can be restarted manually via the start menu.

14.6 Program structure

After starting the program, *_connectScannerWI* appears only as icon in the right part of the task bar. The user interface consists of a context menu and a status window. Open the status window using the "Display window" function of the context menu. The program activities and the time are displayed.

14.7 Functions in the context menu

The following functions are available via the context menu of the program symbol:

"Display window"	Show status window.
"Hide Window"	Hide status window.
"Start Scanner"	Start accessing barcode scanner.
"Stop Scanner"	Stop access to barcode scanner. This is necessary for changing a *_connect.BRAIN* configuration.
"Close"	Close *_connectScannerWI*.

15 VirtualES - View

15.1 Overview

The *VirtualES* program is a verifiable memory for weighing. The data are protected by check sums and are protected against manipulation.

The VirtualES software component consists of two programs:

Admin	to configure the verifiable memory.
View	to read the verifiable memory content.

In order to be able to operate devices with a verifiable memory, access to *VirtualES* must be activated in the device configuration. Activation is done in *_connectConfig* in the BCS configuration of the devices. The "Use Virtual ES" option must be activated here. The *Admin* administration tool can also be called from here.

15.2 VirtualES - Admin

The *VirtualES* administration tool serves to configure the verifiable memory. It can be used to define the properties that will be stored in the verifiable memory for each weighing, for each configured device.

15.2.1 Starting program

➪ Call up *VirtualES - Admin* via the Start menu.
The start window of the program appears.

15.2.2 Structure of the program

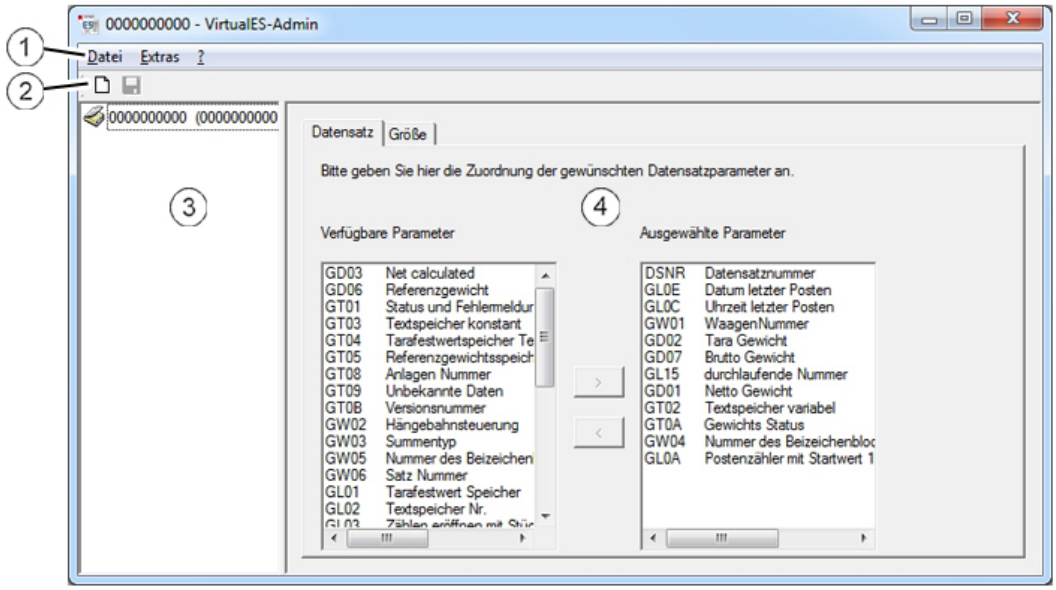


Illustration 35: User interface VirtualES - Admin

(1) Menu bar

(2) Tool bar

(3) Navigation area

(4) Display area

15

The user interface contains the following areas, in addition to typical Windows elements:

Navigation area	The databases and devices configured until now are displayed here. When you select a database, the set parameters appear in the display area.
Display area	The parameter settings of the database or the device selected in the navigation area are displayed here.

Information is distributed to two tabs in the display area:

Dataset	This displays the assignments relating to which parameters are saved within the data records and which can be additionally selected. Compulsory parameters cannot be deselected and are always a constituent of a data record.
Size	The number of data records is specified here.

15.2.3 Menu bar and toolbar functions

Functions in menu "File"

"New"		Create new configuration.
"Save"		Save current configuration.
"Exit"		Close *VirtualES - Admin*.

Functions in menu "Extras"

"Language"	Allows the language to be changed in the user guidance.
"Change name of DB"	Allows to rename an existing database.

Functions in menu "?"

"Information about VirtualES-Admin"	Displays the program version and installation environment.
"Signatures"	Displays the digital signatures of the configuration program and the background server application. This allows you to check whether changes have been made to *VirtualES* or parts of it.

15.2.4 Defining new configuration

➪ "file" / "New"

or

➪ Click the corresponding icon of the toolbar.

➪ Select a device and go to the next selection window using "Forward >".
Only one device can be selected.

➪ Select further parameters that are to be transferred to the protected memory.
Compulsory data (gross, net/tare, data record number, etc.) cannot be deselected.

15.3 VirtualES - View

The *VirtualES - View* display program serves for visualization of the saved data. Data records:

– Data record number
– Date
– Time
– Scale number
– Tare weight
– Gross weight
– Consecutive number

15.3.1 Start program

➪ Call up *VirtualES - View* via the Start menu.

The start window of the program appears.

15.3.2 Program structure

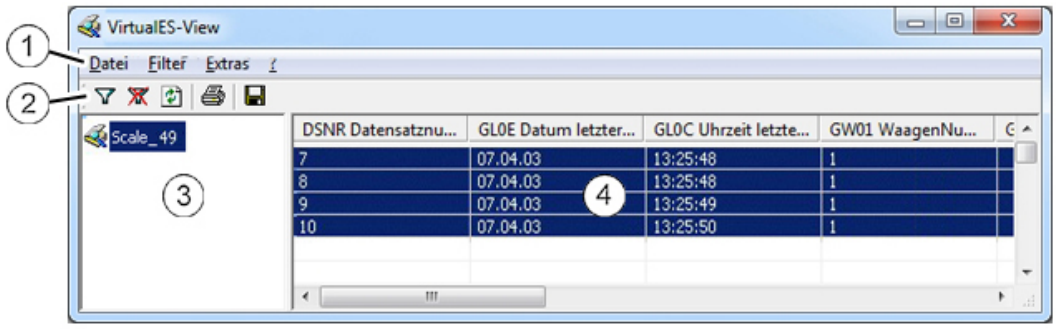


Illustration 36: VirtualES view user interface

① Menu bar
② Tool bar
③ Navigation area
④ Display area

The user interface contains the following areas, in addition to typical Windows elements:

Navigation area	The configured databases and devices are displayed here. When you select a device, the saved data appears in the display area.
Display area	The parameters of the database selected in the navigation area are displayed here. The data is displayed in a tabular structure. If a filter is set, only the filtered data records appear. Otherwise all saved data appears. All or part of the data in the display area can be selected, e.g. for printing.

15.3.3 Menu bar and toolbar functions

Functions in the "File" menu

"Refresh"		Updates the current display by reading out the verifiable memory again.
"Save as"		Saves all data of the selected devices in an XML file. For this purpose a file dialog is called up, in which you can define the name and location of the file to be written.
"print"		Prints the selected area or the entire document.
"sideview"		Displays a preview of the selected area or the entire document. A printout can then be made from this preview.
"Printer settings..."		Allows the configuration of a printer for the print function.
"Close"		Close *VirtualES - View*.

Functions in the "Filter" menu

"Activate Filter..."	Opens a dialog window, in which up to 4 filter options can be set: The filter can be configured by selecting a parameter from the verifiable memory, an operator (less than, less than/equal to, equal to, etc.), a reference value and a link to the other filter options.
"Remove Filter"	This allows a set filter to be deactivated again, so that all data is displayed again.

Functions in the "Extras" menu

"Language"	Allows the language to be changed in the user guidance.

Functions in the "?" menu

"Information about VirtualES-View"	Displays the program version and installation environment.
"Show signatures"	Displays the digital signatures of the configuration program and the background server application. This allows you to check whether changes have been made to VirtualES or parts of it.

15.3.4 Display data records

The data records of the verifiable memory can be viewed, searched and printed out.

➪ Click on the selected scale.

The data records are displayed.

15.3.5 Check signatures

The signatures must match the data in test certificate no. D09-02.22.

VirtualES - View is opened.

Show signatures

➪ "?" / "Show signatures"

➪ Read signatures and compare with test certificate.

Close window

➪ "OK"

The BCS server identification number is only displayed if the connection with the Bizerba Communication Server (BCS) is activated and registration <RX04> has already taken place.

15.3.6 Registration of weighing results

VirtualES saves the weighing results of the individual verified scales in a verifiable memory. The verifiable module and the saved data are protected by check sums and cannot be changed unnoticed.

➪ Select scale with right mouse button.

➪ "VirtualES function test"

A pop-up window is opened.

➪ "Registration"

The weighing results of the scale are registered and saved in VirtualES.

➪ "Close"

The pop-up window is closed.

The saved data is displayed highlighted in the display area.

16 _edit.BRAIN

16.1 Overview

_edit.BRAIN is a program for viewing log files (*.log or *.commlog) created by other *_connect.BRAIN* applications.

16.2 Starting the program

➪ Call up *_edit.BRAIN* via the start menu.

The start window of the program appears.

16.3 Program structure

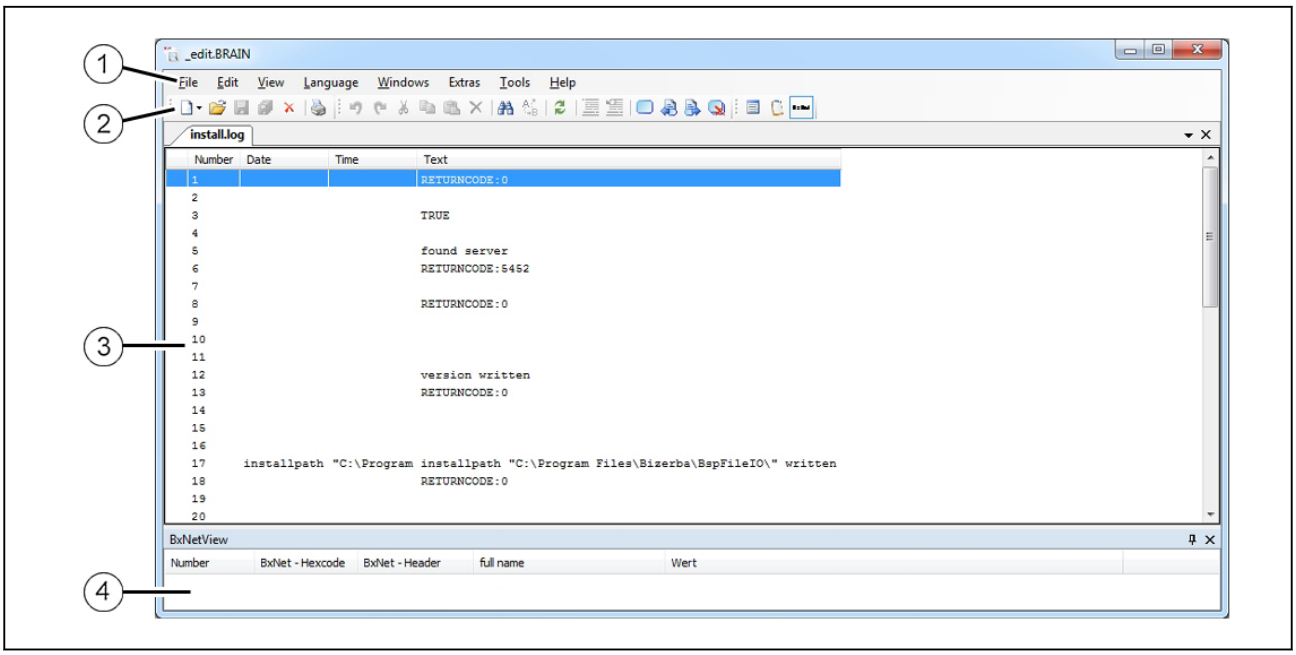


Illustration 37: User interface _edit.BRAIN

(1) Menu bar

(2) Tool bar

(3) Display and editing area

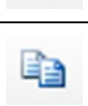
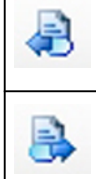
(4) Bx-Net View

16.4 Menu bar and toolbar functions

Functions in the "File" menu

"New"		Create a new text file.
"Open"		Open an existing *.log, *.commlog or text file.
"Save"		Save an existing *.log, *.commlog or text file.
"Save as..."		Save an open *.log, *.commlog or text file with a freely selectable name.
"Save all"		Save all open *.log, *.commlog or text files.
"Close"		Close an open *.log, *.commlog or text file.
"Close all"		Close all open *.log, *.commlog or text files.
"Print"		Print an open *.log, *.commlog or text file.
"Send as E-Mail"		Send an open *.log, *.commlog or text file as e-mail.
"Last opened files"		Open last open file again.
"Exit"		Close *_edit.BRAIN*.

Functions in the "Edit" menu

"Undo"		Undo last action.
"Redo"		Repeat last action.
"Cut"		Cut selected section.
"Copy"		Copy selected data to clipboard.
"Paste"		Paste data from clipboard.
"Delete"		Delete marked section.
"Select all"		Select all.
"Find"		Find data.
"Replace"		Replace data.
"Find next"		Continue search with same settings.
"Refresh"		Reload data.
"More"	Open further editing tools:	
		"Set comment" Set comment symbol.
		"Delete comment " Delete comment symbol.
		"Set bookmark" Set read symbol.
		"Previous bookmark " Go to previous read symbol.
		"Next bookmark" Go to next read symbol.
		"Delete bookmark " Delete read symbol.

16

Functions in the "View" menu

"Output"		Opens a display area for the data to be output at the bottom of the main window.
"Clipboard"		Opens the clipboard.
"BxNet View"		Opens the "BxNetView" display area at the bottom of the main window.
"Toolbars"		Show or hide toolbars: – View – Edit – File

Functions in the "Language" menu

"Deutsch" Sets the program language to German.

"English" Sets the program language to English.

Functions in the "Windows" menu

"Windows..." It is used to change between the open *.log, *.commlog or text files.

Functions in the "Extras" menu

"Synchronisation..." Commlogs or logs can be synchronized between themselves or with event logs. To do so, the time stamps are compared and chronologically arranged in a temporary file.

16

Functions in the "Tools" menu

"Notepad" Opens the Windows notepad.

"BCT" Log files of other Bizerba applications can be opened or deleted.

Functions in the "Help" menu

"Info about..." A window with the current program information appears.

16.5 Functions in the context menu

After opening the *.commlog files, the following features are available.

"Detail"	It is used to display details. The single BxNet commands of a telegram are broken down and displayed in the BxNetView window. *_edit.Brain* picks up the information from the Gx database in the application folder under the BxNet folder.
"Find"	It is used to find special terms (full text search). The results are displayed in the output window.
"Hexcode"	The selected line is displayed as hexadecimal value.
"Copy to Notepad"	Copy to notepad: Telegrams can be copied to the notepad for further processing in other programs or a better overview. The selected lines or the complete telegram with mixed or separated header/data can be copied.
"Invisible Row(s)"	Selected lines will be hidden to get a better overview of the document.
"Visible Row(s)"	Hidden lines will be displayed again.
"Fonts"	It is used to set font types.

After opening the *.log files, the following features are available.

"Detail"	It is used to display details. The single BxNet commands of a telegram are broken down and displayed in the BxNetView window. *_edit.Brain* picks up the information from the Gx database in the application folder under the BxNet folder.
"Copy to Notepad"	Copy to notepad: Telegrams can be copied to the notepad for further processing in other programs or a better overview. The selected lines or the complete telegram with mixed or separated header/data can be copied.
"Invisible Row(s)"	Selected lines will be hidden to get a better overview of the document.
"Visible Row(s)"	Hidden lines will be displayed again.
"Decrypt line"	This menu point contains raw hex data as saved in the file. Only one telegram is displayed at a time.
"Syntax validation"	It is used to verify the syntax of the entire IX command.

To avoid error messages, all commands from the first to the last command of an IX string have to be selected.

"Fonts"	It is used to set font types.

17 LogPathConfig

17.1 Overview

LogPathConfig is used to modify the log file standard directory or to delete entries. Then all components write the related log files in the specified directory.

17.2 Program structure

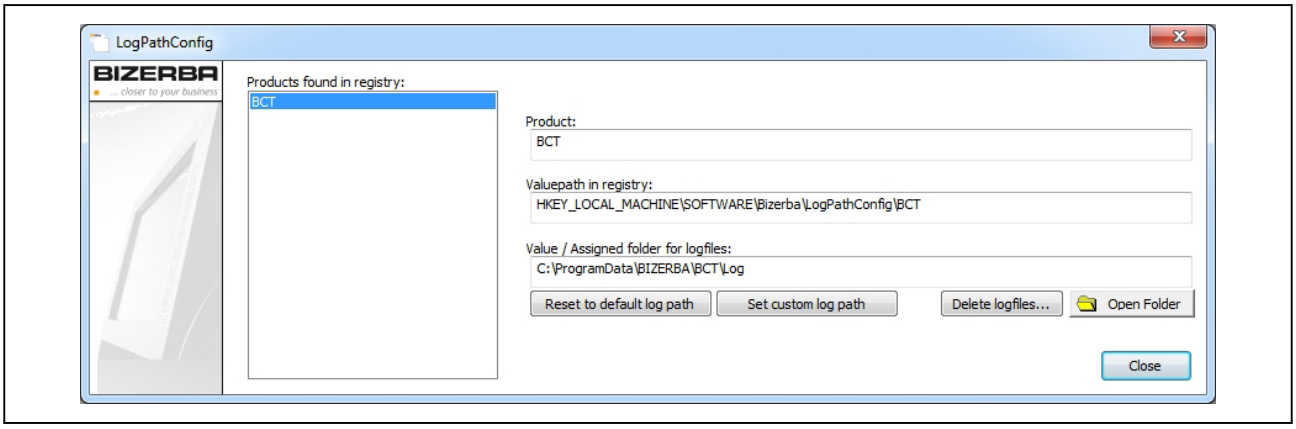


Illustration 38: *LogPathConfig* user interface

In addition to typical Windows elements, the user interface contains a display area with the paths to the program registry entry and to the log file storage location.

17.3 Menu bar and toolbar functions

"Return to default log path"	Settings are reset to predefined path.
"Set custom log-path"	A window containing the Window directory tree for selecting the storage location appears.
"Delete logfiles ..."	Log files can be deleted using a filter criteria.
"Open Folder"	The specified storage folder is opened.

17.4 Deleting log files

To delete selected log files, proceed as follows:

➪ Call up filter mask via *Delete log files ...*.

- Select date.
- Enter extension.

  The extension (zip, log, commlog, etc.) can be displayed as list, separated by semicolon.
  The "Use recycle bin" checkbox allows to specify whether the Windows trash shall be used when deleting files.
- Confirm input with <OK>.
- Answer the following security query with <Yes> or <No>.

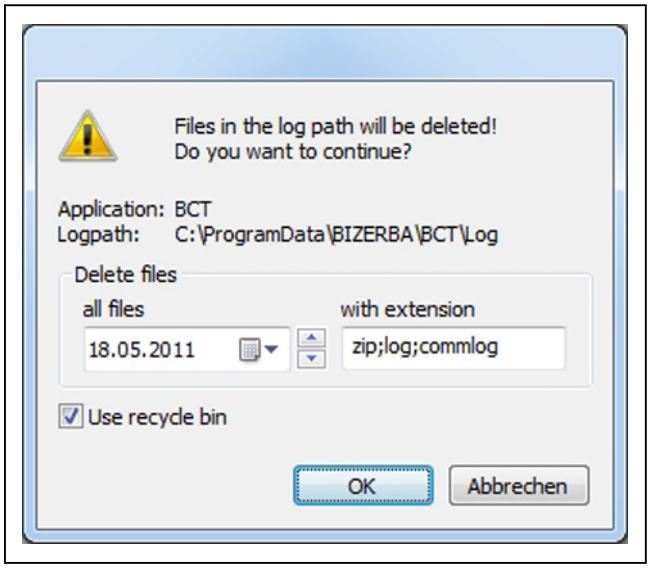


Illustration 39: Deleting filter mask log files

17

18 Background information

18.1 Device families

The Bizerba syntax distinguishes between the following device families:

CX:	Checkweigher (CWE, CWM, ...)
GX:	Labeler (GLM-I, GLM-E, GLP, ...)
IX:	Industrial devices (WM, CWL Eco, ST, ...)
LW:	Retail scales (SC, SW, ...)
SX:	Software (PSS, WinCis Count, ...)
FX:	Other devices (scanner, terminal, ...)
MX:	Inspection systems (BVS - Bizerba Vision System, …)

The CX family is based on the GX family. The CX device settings correspond to those of the GX devices and are not managed separately in *_connectBRAIN*. Thus, the CX device are found in *_connectBRAIN* under the GX device family.

18.2 BxNet language

BxNet is the generic term used for the interface languages GxNet and IxNet that alllow data exchange with Bizerba devices. BxNet is unique within the language group and therefore version-independent. The language scope is upgraded with each new function in the devices. BxNet depends on the physical interface used and on the conversion type.

A data record in BxNet consists of one or more subfunction identifiers with the relevant data. The languages belonging to BxNet have the same structure, but distinguish themselves by the subfunction identifiers related to the respective device group.

GxNet

GxNet language is used in labelers for both communication between the internal device components and for communication with other devices in the system group. Via EDP connection, communication with each individual device is possible.

IxNet

IxNet language is adapted to the requirements of industrial devices. Industrial devices do not use IxNet language internally, but solely as an interface language. *_connect.BRAIN* converts IxNet commands in ASCII commands for industrial devices that do not know the IxNet language.

18.2.1 Telegram structure

Commands and data are transmitted to a device as ASCII text in BxNet telegram format. There are two different formats available in *_connectServer* and/or *_connectControl*:

– Header and data separated (e.g. `Send()` method)
– Header and data together (e.g. `SendOne()` method)

Header (commands): A header can be composed of one or more commands. In case of more commands, these are separated by separators ("|").

Data (user data): The user data includes the data associated with the data description. They are also separated by separators ("|").

Header structure

Language area	1 character (`A` or `I`)
Command type	1 character (`!` or `?`)
BxNet command	1..n BxNet command

18.2.2 Coding of the data description

Data description is coded for read or write access. To achieve backward compatibility with the GxTools, there is an old and a new header format for data description. Assign the header format to be used to a device in *_connectConfig*, see page 46.

current: Gx/Ix-Net(A!, I!, A?, I?), for labelers and industrial devices
The 1st character of the header stands for the device, the 2nd character for the type of access:

`A!`	Write access for labelers
`A?`	Read access for labelers
`I!`	Write access for industrial devices
`I?`	Read access for industrial devices

old: Gx-Net (!,?), only for labelers (GxTools format)
The 1st character of the header defines the type of access:

`!`	Write access
`?`	Read access

18.2.3 BxNet data types

The BxNet language supports the following command file types:

Identification	Description	Number of parameters
`X`	Command without data	0 (no data element available)
`W`	Word, 16 bit with sign	1
`L`	Long, 32 bit with sign	1

Identification	Description	Number of parameters
D	Dimensional value	1 (3 subparameters)
V	Variable	0 (variable sequences are always terminates with the LX02 command.)
T	Text	1

18.2.4 Coding of the useful data

The user data are displayed in readable form:

– Numeric data in decimal notation
– Texts as ASCII character string

Exceptions:

– The first 32 characters of the ASCII character table (hexadecimal numeration <= $1F_{HEX}$) are control characters. They are represented in hexadecimal form. For identification, they are preceded by the escape character @.
Example: LF becomes @0A.
– The same is valid for the 127th character of the ASCII character table (hexadecimal numeration >= $7F_{HEX}$). These characters are treated as control characters. They are also represented in hexadecimal form and preceded by the escape character @.
– The characters "@" and "|" are treated as control characters because they can occur within a text and be used as escape characters or separators.
Example:
max.mustermann@firma.de becomes max.mustermann@40firma.de

18.2.5 Dimensionful data:

Dimensionful data consists of a unit, the number of decimal places and a related integer. These three values are separated by semicolon ";".

18

Design

Parameters	Description
1	Unit
2	Number of decimal places
3	Integer

18.2.6 Coding of prices

Coding of prices varies according to the set header format.

current: Gx/Ix-Net(A!, I!, A?, I?), for labelers and industrial devices
A price consists of the abbreviation for the local currency, the country-specific number of decimal places and the value of the price. The value is always coded as an integer. To determine the price, the value is multiplied by a power of ten of the decimal places.

Example:

Price	Coding
19,90 €	`EUR;-2;1990`
12,99 €	`USD;-2;1299`

old: Gx-Net (!,?), only for labelers (GxTools format)
A price consists of the country code and the price value. The value is always coded as an integer. The country code contains the currency and the country-specific number of decimal places. To determine the price, the value is multiplied by a power of ten of the decimal places.

Example:

Price	Coding
19,90 €	`6|1990`
12,99 $	`64|1299`

Country codes used:

– 6: EUR, 2 decimal places
– 64: USD, 2 decimal places

18.2.7 Coding of weights

A weight value consists of 3 parts:

– Weight unit (KG, LB, lb, OZ, %)
– Exponent for the decimal places (0, -1, -2, -3, -4)
– Weight value as integer

current: Gx/Ix-Net(A!, I!, A?, I?), for labelers and industrial devices
Example:

Weight	Coding
2,995 kg	`KG;-3;2995`
45,93 kg	`KG;-2;4593`

old: Gx-Net (!,?), only for labelers (GxTools format)
Example:

Weight	Coding
2.995 kg	`KG|-3|2995`
45.93 kg	`KG|-2|4593`

19 Program interfaces

19.1 _connectServer DCOM communication interface

The *_connectServer* DCOM communication interface provides functions for communication among *_connectServer* and client. The properties, methods and events are described below.

19.1.1 Methods

Close

It closes the connection to the connected device. If the `Open` method has not been carried out, `Close` reports an error.

Syntax and parameter

C++
`HRESULT Close ()`
C#
`int Close ()`

no parameters

CreateReceiveQueue

It generates a reception queue. Then one or more filters can be created for the reception queue using the `SetReceiveFilterQueue` method. Data can only be received after at least one filter has been created for the reception queue.

Syntax and parameter		

C++
`HRESULT CreateReceiveQueue ([out] BSTR *szQueueName)`
C#
`int CreateReceiveQueue (out String szQueueName)`		

Parameters	**Value**	**Description**
`szQueueName`	Character string	Queue name generated by *_connectServer*

DeleteReceiveQueue

It deletes a reception queue. If there is not any reception queue with the indicated name in the system, an error is triggered.

Syntax and parameter		

C++
`HRESULT DeleteReceiveQueue ([in] BSTR szQueueName)`
C#
`int DeleteReceiveQueue (string szQueueName)`		

Parameters	**Value**	**Description**
`szQueueName`	Character string.	Name of the queue to be deleted.

Error

It provides the error number, the system number and the error text.

Syntax and parameter		

C++
`HRESULT Error ([out] long *lErrNr, [out] long *lSystemNr, [out] BSTR *szErrTxt)`
C#
`void Error (out int lErrNr, out int lSystemNr, out string szErrTxt)`		

Parameters	**value**	**Description**
`lErrNr`	Numeric data	Error number
`lSystemNr`	Numeric data	Number of the system that has triggered the error.
`szErrTxt`	Character string	Error text in the set language

ErrorHeaderData

It provides the error number, the system number, the error text and the related data.

Syntax and parameter		

C++
`HRESULT ErrorHeaderData ([out] long *lErrNr, [out] long *lSystemNr, [out] BSTR *szErrTxt, [out] BSTR *szErrHeader, [out] BSTR *szErrData)`
C#
`void ErrorHeaderData (out int lErrNr, out int lSystemNr, out string szErrTxt, out string szErrHeader, out string szErrData)`		

Parameters	**value**	**Description**
`lErrNr`	Numeric data	Error number
`lSystemNr`	Numeric data	Number of the system that has triggered the error.
`szErrTxt`	Character string	Error text in the set language
`szErrHeader`	Character string	Header of the error data record. This value is only indicated when the device (e.g. GLP) transmits an error data record.
`szErrData`	Character string	Data portion of the error data record. This value is only indicated when the device (e.g. GLP) transmits an error data record.

GetBCSVersion

It provides the version name of *_connectServer* (BCS).

Syntax and parameter		

C++
`HRESULT GetBCSVersion ([out] BSTR *szVersion)`
C#
`int GetBCSVersion (out string szVersion)`		

Parameters	**Value**	**Description**
`szVersion`	Character string	*_connectServer* version name

GetCategory

It provides the number of the device family the open system is related to.

Syntax and parameter		

C++
`HRESULT GetCategory ([out] short *nCategory)`
C#
`int GetCategory (out short nCategory)`		

Parameters	**Value**	**Description**
`nCategory`	`1`: basic systems `2`: labeler / printer `3`: retail scales `4`: industrial devices `5`: external devices `6`: WinCWS `7`: industry special	Device family number

Open

It opens access to a device via *_connectServer*.

Syntax and parameter		

C++
`HRESULT Open ([in] BSTR szIdentUser, [in] BSTR szDeviceName, [in] short nTelegramType, [in] short nAccess, [in] short bLightLicenceEnable)`
C#
`int Open (string szIdentUser, string szDeviceName, short nTelegramType, short nAccess, short bLightLicenceEnable)`		

Parameters	**Value**	**Description**
`szIdentUser`	Any character string	User name for allocation of errors that occur.
`szDeviceName`	Name of an active device (character string)	Name of the system to be opened. This name is defined in *_connectConfig*. All active systems can be queried via the information interface.
`nTelegramType`	`0`: normal `1`: spontaneous telegrams	Specifies whether the device shall transmit spontaneous telegrams to the client. Spontaneous telegrams can only be sent to a client. Telegram duplication is not supported. If a client is opened for spontaneous telegrams, the attempt to open more clients for spontaneous telegrams causes an error message. However, the interface can be opened normally.
`nAccess`	`0`: multiple access `1`: single access	Device access type. In the event of single access, only one client communicates with the device, in the event of multiple access, more clients communicate with the device.
`bLightLicenceEnable`	`0`: full version `1`: light version (Bizerba software)	Licensing mechanism

Receive

It provides reception data (header and user data) and the transmission status. It requires the handle from the `Send` method.

If the `Open` method has not been carried out, `Receive` reports an error.

Syntax and parameter		

C++
`HRESULT Receive ([out] BSTR *szHeader, [out] BSTR *szData, [in] BSTR szHandle, [in] long lTimeout, [out] long *lStatus)`
C#
`int Receive (out string szHeader, out string szData, string szHandle, int lTimeout, out int lStatus)`		

Parameters	**Value**	**Description**
`szHeader`	Character string	Header data (data description)
`szData`	Character string	User data
`szHandle`	Character string	Handle that is returned from the `Send` method.
`lTimeout`	Numeric data	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

ReceiveAuthorizationRequest

It provides a license code which is used by the client to generate a response. The client inserts the calculated license code in the `SendAuthorizationResponse` method. This method is only used by the Bizerba software. If the `Open` method has not been carried out, `ReceiveAuthorizationRequest` reports an error.

Syntax and parameter		

C++
`HRESULT ReceiveAuthorizationRequest ([out] BSTR *szLizenzKey)`
C#
`int ReceiveAuthorizationRequest (ref string szLizenzKey)`		

Parameters	**Value**	**Description**
`szLizenzKey`	Character string	License code generated by *_connectServer*

ReceiveOne

It provides reception data (header and user data combined) and the transmission status. It requires the handle from the `Send` method.

If the `Open` method has not been carried out, `ReceiveOne` reports an error.

Syntax and parameter		

C++
`HRESULT ReceiveOne ([out] BSTR *szHeaderData, [in] BSTR szHandle, [in] long lTimeout, [out] long *lStatus)`
C#
`int ReceiveOne (out string szHeaderData, string szHandle, int lTimeout, out int lStatus)`		

Parameters	**Value**	**Description**
`szHeaderData`	Character string	Combination of header and user data
`szHandle`	Character string	Handle that is returned from the `Send` method.
`lTimeout`	Numeric data	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The transmission order remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

ReceiveOneWithoutAck

It receives spontaneous telegrams without direct acknowledgement. When using this method, the customer application can delay the transmission of the acknowledgement and transmit data (e.g. lock) to the labeler prior to transmitting the acknowledgement. The acknowledgement is separately transmitted using the `SendAcknowledge` method.

The `ReceiveOneWithoutAck` method can only be used for spontaneous data. The customer application has to acknowledge the data record with `SendAcknowledge` so that a GLP proceeds after a lock.

Syntax and parameter

C++

```
HRESULT ReceiveOneWithoutAck ([out] BSTR *szHeaderData, [in] BSTR szHandle,
[in] long lTimeout, [out] long *lStatus)
```

C#

```
int ReceiveOneWithoutAck (out string szHeaderData, string szHandle, int lTime-
out, out int lStatus)
```

Parameters	Value	Description
`szHeaderData`	Character string	Header data (data description)
`szHandle`	Character string	Handle for spontaneous data (DUSTBIN or created queue). Spontaneous data deriving from the device without related user queue is automatically allocated to a DUSTBIN queue. They can be picked up using the DUSTBIN handle.
`lTimeout`	Numeric data	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The transmission order remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

ReceiveWithoutAck

It receives spontaneous telegrams without direct acknowledgement. When using this method, the customer application can delay the transmission of the acknowledgement and transmit data (e.g. lock) to the labeler prior to transmitting the acknowledgement. The acknowledgement is separately transmitted using the `SendAcknowledge` method.

The `ReceiveWithoutAck` method can only be used for spontaneous data. The customer application has to acknowledge the data record with `SendAcknowledge` so that a GLP proceeds after a lock.

Syntax and parameter

C++

```
HRESULT ReceiveWithoutAck ([out] BSTR *szHeader, [out] BSTR *szData, [in] BSTR
szHandle, [in] long lTimeout, [out] long *lStatus)
```

C#

```
int ReceiveWithoutAck (out string szHeader, out string szData, string szHan-
dle, int lTimeout, out int lStatus)
```

Parameters	Value	Description
`szHeader`	Character string	Header data (data description)
`szData`	Character string	User data
`szHandle`	Character string	Handle for spontaneous data (DUSTBIN or created queue).
`lTimeout`	Numeric data	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The transmission order remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

Reset

It cancels the current send request. If a send request is not cancelled, it remains active until transmission is completed. Thus, it is impossible to start a new send request.

Use `Reset` to cancel the send request from the *_connectServer* transmission list. If the data record has already been sent to the device before `Reset` and the device continues responding after `Reset`, the response is identified as spontaneous data record and made available because the corresponding send request no longer exists.

Syntax and parameter		

C++
`HRESULT Reset ([in] BSTR szHandle)`
C#
`int Reset (string szHandle)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle that is returned from the `Send` method.

Send

It sends data to the connected evaluation unit. If the `Open` method has not been carried out, `Send` reports an error.

Syntax and parameter		

C++
`HRESULT Send ([in] BSTR szHeader, [in] BSTR szData, [out] BSTR *szHandle, [in] long lTimeout, [out] long *lStatus)`
C#
`int Send (string szHeader, string szData, out string szHandle, int lTimeout, out int lStatus)`		

Parameters	**Value**	**Description**
`szHeader`	Character string	Header data (data description)
`szData`	Character string	User data
`szHandle`	Character string	Handle generated by *_connect-Server*. It has to be specified in the `Receive` method to provide the client with the relevant data.
`lTimeout`	Numeric data	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

SendAcknowledge

It closes the `ReceiveWithoutAck` method with a positive acknowledgement.

Syntax and parameter		

C++
`HRESULT SendAcknowledge ([in] BSTR szHandle)`
C#
`int SendAcknowledge (string szHandle)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle used in the `ReceiveWithoutAck` method.

SendAcknowledgeNeg

It closes the `ReceiveWithoutAck` method with a negative acknowledgement. Transmits an error message to the device.

Syntax and parameter		

C++
`HRESULT SendAcknowledgeNeg ([in] BSTR szHandle, [in] long lError, [in] BSTR szErrTxt)`
C#
`int SendAcknowledgeNeg (string szHandle, int lError, string szErrTxt)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle used in the `ReceiveWithoutAck` method.
`lError`	Numeric data	Error number to be sent to the device
`szErrTxt`	Character string	Error text to be sent to the device

SendAuthorizationResponse

It compares the license code calculated by the client with the license code transmitted by *_connectServer* via `ReceiveAuthorizationRequest`. This method is only used by the Bizerba software.

In case of noncompliance, an error is reported.

If the `Open` method has not been carried out, `SendAuthorizationReponse` reports an error.

Syntax and parameter		

C++
`HRESULT SendAuthorizationResponse ([in] BSTR szLizenzKey)`
C#
`int SendAuthorizationResponse (string szLizenzKey)`		

Parameters	**Value**	**Description**
`szLizenzKey`	Character string	License code calculated by the client

SendCheck

It verifies whether data is available, the command has been completely processed or transmitted, and a positive or negative acknowledgement has been received.

Syntax and parameter		

C++
`HRESULT SendCheck ([in] BSTR szHandle, [in] long lTimeout, [out] long *lStatus)`
C#
`int SendCheck (string szHandle, int lTimeout, out int lStatus)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle that is returned from the `Send` method.
`lTimeout`	Numeric data	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

SendOne

It sends data to the connected evaluation unit. If the `Open` method has not been carried out, `Send` reports an error.

Syntax and parameter		

C++
`HRESULT SendOne ([in] BSTR szHeader Data, [out] BSTR *szHandle, [in] long lTimeout, [out] long *lStatus)`
C#
`int SendOne (string szHeader Data, out string szHandle, int lTimeout, out int lStatus)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle generated by *_connect-Server*. It has to be specified in the `Receive` method to provide the client with the relevant data.
`lTimeout`	Numeric data	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

SetReceiveQueueFilter

It sets a filter for reception data. The filter is required to receive data by means of customer-specific reception queues that have been created using the `CreateReceiveQueue` method. Without filters, spontaneous data is automatically assigned to the DUSTBIN reception queue.

Example for a GX: Use `CreateReceiveQueue` to create an individual queue for package data records starting with *PV05*. To allocate the package data records to a method, call up the `SetReceiveQueueFilter` method using the queue name and the `PV05` filter parameter. If data records, e.g. starting with *PV05* or *PV06*, are to be allocated to the same queue, call up the `SetReceiveQueueFilter` method two times. Specify the same queue both times and use the `PV05` filter parameter at one time and the `PV06` filter parameter the other time.

Syntax and parameter		

C++
`HRESULT SetReceiveQueueFilter ([in] BSTR szQueueName, [in] BSTR szFilter)`
C#
`int SetReceiveQueueFilter (string szQueueName, string szFilter)`		

Parameters	**Value**	**Description**
`szQueueName`	Character string	Queue name generated by *_connectServer* The `CreateReceiveQueue` method returns this name.
`szFilter`	Character string	Filter parameter, e.g. `PV05` for a GX.

IsUnicodeDevice

Syntax and parameter		

C++
`HRESULT IsUnicodeDevice([out] short* bIsUnicodeDevice)`
C#
`int IsUnicodeDevice(ref short bIsUnicodeDevice))`		

Parameter	**Value**	**Description**
`bIsUnicodeDevice`	`0`: Configured device is not a Unicode device `1`: Configured device is a Unicode device	Indicates if configured device is a code page based device or a Unicode device.

19.1.2 Events

DataArrival

It indicates reception of spontaneous telegrams and transmits the identification of the communication channel (handle) which can be used for receiving data via the `Receive`, `ReceiveWithoutAck`, `ReceiveOneWithoutAck` or `ReceiveOne` method.

Syntax and parameter		

C++
`long DataArrival(BSTR szQueueName)`
C#
`int DataArrival(string szQueueName)`		

Parameters	**Value**	**Description**
`szQueueName`	Character string	Identification of communication channel (handle) This handle has to be transferred in the reception method (e.g. `Receive`) in order to access data.

RemoteDataArrival

It answers a send request with a `?`, `A?` or `I?` header. Transmits the identification of the communication channel (handle) which can be used for receiving data via the `Receive` or `ReceiveOne` method.

The `ReceiveWithoutAck` and `ReceiveOneWithoutAck` methods are only to be used for receiving spontaneous data and must not be used in connection with the `RemoteDataArrival` event.

Syntax and parameter		

C++
`long RemoteDataArrival(BSTR szQueueName)`
C#
`int RemoteDataArrival(string szQueueName)`		

Parameters	**Value**	**Description**
`szQueueName`	Character string	Identification of communication channel (handle) This handle has to be transferred in the `Receive` or `ReceiveOne` method in order to access data.

19.2 _connectServer DCOM information interface

The *_connectServer DCOM* information interface supplies all information on *_connect.BRAIN* and on the devices that have been created and activated in *_connectConfig*. The available methods are described in the following paragraphs.

19.2.1 Methods

Error

It reports an error message after failure of a previous action. Even if the `Open` method has not been carried out, `Error` reports an error.

Syntax and parameter		

C++
`HRESULT Error ([out] long *lErrNr, [out] long *lSystemNr, [out] BSTR *szErrTxt)`
C#
`void Error (out int lErrNr, out int lSystemNr, out string szErrTxt)`		

Parameter	**Value**	**Description**
`lErrNr`	Number	Error number
`szErrTxt`	Character string	Error text in the set language

GetBCSVersion

It provides the version name of *_connectServer* (BCS). If the `Open` method has not been carried out, `GetBCSVersion` reports an error.

Syntax and parameter		

C++
`HRESULT GetBCSVersion ([out] BSTR *szVersion)`
C#
`int GetBCSVersion (out string szVersion)`		

Parameters	**Value**	**Description**
`szVersion`	Character string	*_connectServer* version name

GetDemo

It verifies whether *_connect.BRAIN* is used with license or in the developer mode.

Syntax and parameter		

C++
`HRESULT GetDemo ([out] BOOL *bDemo)`
C#
`int GetDemo (out int bDemo)`		

Parameters	**Value**	**Description**
`bDemo`	`0`: false, normal operation with licenses `1`: true, developer mode	Normal operation or developer mode

GetDevice

It returns all active systems defined in the configuration. If the `Open` method has not been carried out, `GetDevice` reports an error.

Syntax and parameter		

C++
`HRESULT GetDevice ([out] BSTR *szDeviceString, [out] long *lCount)`
C#
`int GetDevice (out string szDeviceString, out int lCount)`		

Parameter	**Value**	**Description**
`szDeviceString`	`SECT01` = section name `PARA02` = two related values follow `name` = device name `devCategory` = see table 1 `devType` = see table 2 `devicestate` = see table 3	Example: `szDeviceString=SECT01|SYSTEM1|PARA02|name|bizerbaScale1|PARA02|devCategory|2|PARA02|devType|19|PARA02|devicestate|0|` puts out: SYSTEM1 bizerbaScale1 2 19
`lCount`	Number	Number of active systems

Table 1: Possible values for `devCategory`
`1`: basic systems `2`: labeler / printer `3`: retail scales `4`: industrial devices `5`: external devices `7`: industry special

Table 2: Possible values for `devType`		
`0`: not defined	`15`: ITS	`38`: ITCS
`1`: GD	`16`: ITL	`39`: CWL Eco
`2`: GH	`17`: MCI	`40`: GLF
`3`: GV	`18`: MCE	`41`: GLM-E
`4`: GS	`19`: GLP	`42`: GLM-B
`5`: Scanner	`20`: Empty	`43`: GLM-P
`6`: Terminal	`21`: HW	`44`: GLM-L
`7`: ST	`22`: PSS	`45`: GLM-E Retail
`8`: ITU	`26`: NTScale	`46`: Bizerba Vision System (BVS)
`9`: ITC1	`30`: Addidata I/O card	`47`: CWP Neptune
`10`: ITC2	`34`: WM	`48`: CWD
`11`: EL	`35`: GLM-I	`49`: Addidata IO MSX E1516 Board
`12`: BT	`36`: CWM	`50`: iS50
`14`: ITE	`37`: CWE	

Table 3: Possible values for `devicestate`
`0`: active
`1`: inactive

GetDeviceDCOM

It returns the devices connected to a remote PC.

Syntax and parameter		

C++
`HRESULT GetDeviceDCOM ([out] BSTR *szDevice, [out] long *lCount, BSTR szDComPC)`
C#
`int GetDeviceDCOM (out string szDevice, out int lCount, string szDComPC)`		

Parameters	**Value**	**Description**
`szDeviceString`	see `GetDevice`	
`lCount`	see `GetDevice`	
`szDComPC`	Character string	Remote PC name

GetLicenseCountForSpecificLicense

This method is used to determine the number of available licenses of the specified license type.

Syntax and parameter		

C++
`HRESULT GetLicenseCountForSpecificLicense([in] long licenseType, [out] long *licenseCount)`
C#
`int GetLicenseCountForSpecificLicense(int licenseType, out int licenseCount)`		

Parameters	**Value**	**Description**
`licenseType`	License type: `1`: BCS_LICENSE_TYPE_DEVICE_CX `2`: BCS_LICENSE_TYPE_DEVICE_GX `3`: BCS_LICENSE_TYPE_DEVICE_IX `4`: BCS_LICENSE_TYPE_DEVICE_BRAIN `5`: BCS_LICENSE_TYPE_VS `6`: BCS_LICENSE_TYPE_INTERFACE_2FILE `7`: BCS_LICENSE_TYPE_INTERFACE_2DB `8`: BCS_LICENSE_TYPE_INTERFACE_2SAP `9`: BCS_LICENSE_TYPE_DEVICE_MX	
`licenseCount`	Number	Number of available licenses of the specified license type on the license server The value `9999` means that it is a bit license and not a counter license.

GetLicenseExpirationDateForProductVersion

This method checks the expiry date of time-limited licenses. After expiring, the license is no longer available and the function cannot be used in *_connect.BRAIN* any longer.

Syntax and parameter		

C++
`HRESULT GetLicenseExpirationDateForProductVersion([out] DATE *date)`
C#
`int GetLicenseExpirationDateForProductVersion(out system.DateTime date)`		

Parameters	**Value**	**Description**
`date`	Date	Date when license expires

GetLicenseExpirationDateForSpecificLicense

This method checks the expiry date of time-limited licenses. After expiring, the license is no longer available and the function cannot be used in *_connect.BRAIN* any longer.

Syntax and parameter		

C++
`HRESULT GetLicenseExpirationDateForSpecificLicense([in] long licenseType, [out] DATE *date)`
C#
`int GetLicenseExpirationDateForSpecificLicense(int licenseType, out system.DateTime date)`		

Parameters	**Value**	**Description**
`licenseType`	License type: 1: BCS_LICENSE_TYPE_DEVICE_CX 2: BCS_LICENSE_TYPE_DEVICE_GX 3: BCS_LICENSE_TYPE_DEVICE_IX 4: BCS_LICENSE_TYPE_DEVICE_BRAIN 5: BCS_LICENSE_TYPE_VS 6: BCS_LICENSE_TYPE_INTERFACE_2FILE 7: BCS_LICENSE_TYPE_INTERFACE_2DB 8: BCS_LICENSE_TYPE_INTERFACE_2SAP 9: BCS_LICENSE_TYPE_DEVICE_MX	
`date`	Date	Date when license expires

GetLicenseInformations

This method finds out information about the licensing system currently used, e.g. version number and license server name. The information is generated as xml structure.

19

Syntax and parameter		

C++
`HRESULT GetLicenseInformations([out] BSTR *xmlLicensingInformations)`
C#
`int GetLicenseInformations(out string xmlLicensingInformations)`		

Parameters	**Value**	**Description**
`xmlLicensingInformations`	Character string	XML data

GetLicenseNameForProductVersion

This method returns the license required by the installed *_connect.BRAIN*. The result can be used to verify whether the required license is available on the license server. If the license is missing, *_connect.BRAIN* cannot be used any longer.

Syntax and parameter		

C++
`HRESULT GetLicenseNameForProductVersion([out] BSTR *licenseProductVersion)`
C#
`int GetLicenseNameForProductVersion(out string licenseProductVersion)`		

Parameters	**Value**	**Description**
`licenseProductVersion`	Character string	Name of the required license, e.g. *IS_connect_V30*. The product cannot be executed without this license.

GetLicenseSystemName

The method identifies the licensing system currently used (up to now *BSP* only).

Syntax and parameter		

C++
`HRESULT GetLicenseSystemName([out] BSTR *licensingSystemName)`
C#
`int GetLicenseSystemName(out string licensingSystemName)`		

Parameters	**Value**	**Description**
`licensingSystemName`	Character string	Name of the current licensing system, `BSP` character string at the moment

GetSerialNumber

It returns the *_connectServer* serial number. If the `Open` method has not been carried out, `GetSerial Number` reports an error.

Syntax and parameter		

C++
`HRESULT GetSerialNumber ([out] BSTR *szSerialNumber)`
C#
`int GetSerialNumber (out string zSerialNumber)`		

Parameters	**Value**	**Description**
`szSerialNumber`	Character string	*_connectServer*serial number

Open

To allow access to the functions of the *_connect.BRAIN* DCOM interface, the interface has to be opened using the `Open` method. A user name is specified to facilitate trouble shooting.

Syntax and parameter		

C++
`HRESULT Open ([in] BSTR szIdentUser)`
C#
`int Open (string szIdentUser)`		

Parameters	**Value**	**Description**
`szIdentUser`	Any character string	User name for allocation of errors that occur.

19.3 _connectControl DCOM communication interface

19

The *_connectControl* DCOM communication interface uses the ActiveX control element BCC.OCX to exchange data among client and *_connectServer*. The properties, methods and events are described below.

19.3.1 Properties

Version

It supplies the BCC.OCX version ID code.

Syntax
C++
`BSTR Version`
C#
`string Version`

19.3.2 Methods

AboutBox

It shows the copyright and *_connectServer* information.

Syntax and parameter
C++
`void AboutBox()`
C#
`void AboutBox()`
no parameters

Close

It closes the connection to the connected device. If the `Open` method has not been carried out, `Close` reports an error.

Syntax and parameter
C++
`long Close ()`
C#
`int Close ()`
no parameters

CreateReceiveQueue

It generates a reception queue. Then one or more filters can be created for the reception queue using the `SetReceiveFilterQueue` method. Data can only be received after at least one filter has been created for the reception queue.

Syntax and parameter		

C++
`long CreateReceiveQueue (BSTR *szQueueName)`
C#
`int CreateReceiveQueue (ref string szQueueName)`		

Parameters	**Value**	**Description**
`szQueueName`	Character string	Queue name generated by *_connectServer*

DeleteReceiveQueue

It deletes a reception queue. If there is not any reception queue with the indicated name in the system, an error is triggered.

Syntax and parameter		

C++
`long DeleteReceiveQueue (BSTR szQueueName)`
C#
`int DeleteReceiveQueue (string szQueueName)`		

Parameters	**Value**	**Description**
`szQueueName`	Character string	Name of the queue to be deleted.

Error

It provides the error number, the system number and the error text.

Syntax and parameter		

C++
`void Error (long *nErrNr, long *nSystemNr, BSTR *szErrTxt)`
C#
`void Error (ref int nErrNr, ref int nSystemNr, ref string szErrTxt)`		

Parameters	**Value**	**Description**
`nErrNr`	Number	Error number
`nSystemNr`	Number	Number of the system that has triggered the error.
`szErrTxt`	Character string	Error text in the set language

ErrorHeaderData

It provides the error number, the system number, the error text and the related data.

Syntax and parameter		

C++
`long ErrorHeaderData (long *lErrNr, long *lSystemNr, BSTR *szErrTxt, BSTR *szErrHeader, BSTR *szErrData)`
C#
`void ErrorHeaderData (ref int lErrNr, ref int lSystemNr, ref string szErrTxt, ref string szErrHeader, ref string szErrData)`		

Parameters	**Value**	**Description**
`IErrNr`	Number	Error number
`ISystemNr`	Number	Number of the system that has triggered the error.
`szErrTxt`	Character string	Error text in the set language
`szErrHeader`	Character string	Header of the error data record. This value is only indicated when the device (e.g. GLP) transmits an error data record.
`szErrData`	Character string	Data portion of the error data record. This value is only indicated when the device (e.g. GLP) transmits an error data record.

GetBCSVersion

It provides the version name of *_connectServer* (BCS).

Syntax and parameter		

C++
`long GetBCSVersion (BSTR *szVersion)`
C#
`int GetBCSVersion (ref string szVersion)`		

Parameters	**Value**	**Description**
`szVersion`	Character string	*_connectServer* version name

GetCategory

It provides the number of the device family the open system is related to.

Syntax and parameter		

C++
`long GetCategory (short *nCategory)`
C#
`int GetCategory (ref short nCategory)`		

Parameters	**Value**	**Description**
`nCategory`	`1`: basic systems `2`: labeler / printer `3`: retail scales `4`: industrial devices `5`: external devices `6`: WinCWS `7`: industry special	Device family number

GetDemo

It verifies whether *_connect.BRAIN* is used with license or in the developer mode.

Syntax and parameter		

C++
`long GetDemo (short *nDemo)`
C#
`int GetDemo (ref short nDemo)`		

Parameters	**Value**	**Description**
`nDemo`	`0`: false, normal operation with licenses `1`: true, developer mode	User name for allocation of errors that occur Normal operation or developer mode

GetDevice

It returns all active systems defined in the configuration. If the `Open` method has not been carried out, `GetDevice` reports an error.

Syntax and parameter		

C++
`long GetDevice (BSTR *szDeviceString, long *lCount)`
C#
`int GetDevice (ref string szDeviceString, ref int lCount)`		

Parameter	**Value**	**Description**
`szDeviceString`	`SECT01` = section name `PARA02` = two related values follow `name` = device name `devCategory` = see table 1 `devType` = see table 2 `devicestate` = see table 3	Example: `szDeviceString=SECT01|SYSTEM1|PARA02|name|bizerbaScale1|PARA02|devCategory|2|PARA02|devType|19|PARA02|devicestate|0|` puts out: SYSTEM1 bizerbaScale1 2 19
`lCount`	Number	Number of active systems

Table 1: Possible values for `devCategory`
`1`: basic systems `2`: labeler / printer `3`: retail scales `4`: industrial devices `5`: external devices `7`: industry special

Table 2: Possible values for `devType`		
`0`: not defined	`15`: ITS	`38`: ITCS
`1`: GD	`16`: ITL	`39`: CWL Eco
`2`: GH	`17`: MCI	`40`: GLF
`3`: GV	`18`: MCE	`41`: GLM-E
`4`: GS	`19`: GLP	`42`: GLM-B
`5`: Scanner	`20`: Empty	`43`: GLM-P
`6`: Terminal	`21`: HW	`44`: GLM-L
`7`: ST	`22`: PSS	`45`: GLM-E Retail
`8`: ITU	`26`: NTScale	`46`: Bizerba Vision System (BVS)
`9`: ITC1	`30`: Addidata I/O card	`47`: CWP Neptune
`10`: ITC2	`34`: WM	`48`: CWD
`11`: EL	`35`: GLM-I	`49`: Addidata IO MSX E1516 Board
`12`: BT	`36`: CWM	`50`: iS50
`14`: ITE	`37`: CWE	

Table 3: Possible values for `devicestate`
`0`: active
`1`: inactive

GetDeviceDCOM

It returns the devices connected to a remote PC.

Syntax and parameter		

C++
`long GetDeviceDCOM (BSTR *szDevice, long *lCount, BSTR szDComPC)`
C#
`int GetDeviceDCOM (ref string szDevice, ref int lCount, string szDComPC)`		

Parameters	**Value**	**Description**
`szDeviceString`	see `GetDevice`	
`lCount`	see `GetDevice`	
`szDComPC`	Character string	Remote PC name

GetLicenseCountForSpecificLicense

This method is used to determine the number of available licenses of the specified license type.

Syntax and parameter		

C++
`long GetLicenseCountForSpecificLicense(long licenseType, long *licenseCount)`
C#
`int GetLicenseCountForSpecificLicense(int licenseType, ref int licenseCount)`		

Parameters	**Value**	**Description**
`licenseType`	License type: `1`: BCS_LICENSE_TYPE_DEVICE_CX `2`: BCS_LICENSE_TYPE_DEVICE_GX `3`: BCS_LICENSE_TYPE_DEVICE_IX `4`: BCS_LICENSE_TYPE_DEVICE_BRAIN `5`: BCS_LICENSE_TYPE_VS `6`: BCS_LICENSE_TYPE_INTERFACE_2FILE `7`: BCS_LICENSE_TYPE_INTERFACE_2DB `8`: BCS_LICENSE_TYPE_INTERFACE_2SAP `9`: BCS_LICENSE_TYPE_DEVICE_MX	
`licenseCount`	Number	Number of available licenses of the specified license type on the license server The value `9999` means that it is a bit license and not a counter license.

GetLicenseExpirationDateForProductVersion

This method checks the expiry date of time-limited licenses. After expiring, the license is no longer available and the function cannot be used in *_connect.BRAIN* any longer.

Syntax and parameter		

C++
`long GetLicenseExpirationDateForProductVersion(DATE *date)`
C#
`int GetLicenseExpirationDateForProductVersion(ref system.DateTime date)`		

Parameters	**Value**	**Description**
`date`	Date	Date when license expires

GetLicenseExpirationDateForSpecificLicense

This method checks the expiry date of time-limited licenses. After expiring, the license is no longer available and the function cannot be used in *_connect.BRAIN* any longer.

Syntax and parameter		

C++
`long GetLicenseExpirationDateForSpecificLicense(long licenseType, DATE *date)`
C#
`int GetLicenseExpirationDateForSpecificLicense(int licenseType, ref system.DateTime date)`		

Parameters	**Value**	**Description**
`licenseType`	License type: `1`: BCS_LICENSE_TYPE_DEVICE_CX `2`: BCS_LICENSE_TYPE_DEVICE_GX `3`: BCS_LICENSE_TYPE_DEVICE_IX `4`: BCS_LICENSE_TYPE_DEVICE_BRAIN `5`: BCS_LICENSE_TYPE_VS `6`: BCS_LICENSE_TYPE_INTERFACE_2FILE `7`: BCS_LICENSE_TYPE_INTERFACE_2DB `8`: BCS_LICENSE_TYPE_INTERFACE_2SAP `9`: BCS_LICENSE_TYPE_DEVICE_MX	
`date`	Date	Date when license expires

GetLicenseInformations

This method finds out information about the licensing system currently used, e.g. version number and license server name. The information is generated as xml structure.

Syntax and parameter		

C++
`long GetLicenseInformations(BSTR *xmlLicensingInformations)`
C#
`int GetLicenseInformations(ref string xmlLicensingInformations)`		

Parameters	**Value**	**Description**
`xmlLicensingInformations`	Character string	XML data

GetLicenseNameForProductVersion

This method returns the license required by the installed *_connect.BRAIN*. The result can be used to verify whether the required license is available on the license server. If the license is missing, *_connect.BRAIN* cannot be used any longer.

Syntax and parameter		

C++
`long GetLicenseNameForProductVersion(BSTR *licenseProductVersion)`
C#
`int GetLicenseNameForProductVersion(ref string licenseProductVersion)`		

Parameters	**Value**	**Description**
`licenseProductVersion`	Character string	Name of the required license, e.g. *IS_connect_V30*. The product cannot be executed without this license.

GetLicenseSystemName

The method identifies the licensing system currently used (up to now *BSP* only).

Syntax and parameter		

C++
`long GetLicenseSystemName(BSTR *licensingSystemName)`
C#
`int GetLicenseSystemName(ref string licensingSystemName)`		

Parameters	**Value**	**Description**
`licensingSystemName`	Character string	Name of the current licensing system, `BSP` character string at the moment

GetSerialNumber

It returns the *_connectServer* serial number. If the `Open` method has not been carried out, `GetSerial Number` reports an error.

Syntax and parameter		

C++
`long GetSerialNumber (BSTR *szSerialNumber)`
C#
`int GetSerialNumber (ref string zSerialNumber)`		

Parameters	**Value**	**Description**
`szSerialNumber`	Character string	*_connectServer*serial number

Open

It opens access to a device via *_connectServer*.

Syntax and parameter		

C++
`long Open (BSTR szIdentUser, BSTR szDeviceName, short nTelegramType, short nAccess, short bLightLicenceEnable)`
C#
`int Open (string szIdentUser, string szDeviceName, short nTelegramType, short nAccess, short bLightLicenceEnable)`		

Parameters	**Value**	**Description**
`szIdentUser`	Any character string	User name for allocation of errors that occur.
`szDeviceName`	Name of an active device (character string)	Name of the system to be opened. This name is defined in *_connectConfig*. All active systems can be queried via the information interface.
`nTelegramType`	`0`: normal `1`: spontaneous telegrams	Specifies whether the device shall transmit spontaneous telegrams to the client. Spontaneous telegrams can only be sent to a client. Telegram duplication is not supported. If a client is opened for spontaneous telegrams, the attempt to open more clients for spontaneous telegrams causes an error message. However, the interface can be opened normally.
`nAccess`	`0`: multiple access `1`: single access	Device access type. In the event of single access, only one client communicates with the device, in the event of multiple access, more clients communicate with the device.
`bLightLicenceEnable`	`0`: full version `1`: light version (Bizerba software)	Licensing mechanism

Receive

It provides reception data (header and user data) and the transmission status. It requires the handle from the `Send` method.

If the `Open` method has not been carried out, `Receive` reports an error.

Syntax and parameter		

C++
`long Receive (BSTR *szHeader, BSTR *szData, BSTR szHandle, long lTimeout, long *lStatus)`
C#
`int Receive (ref string szHeader, ref string szData, string szHandle, int lTimeout, ref int lStatus)`		

Parameters	**Value**	**Description**
`szHeader`	Character string	Header data (data description)
`szData`	Character string	User data
`szHandle`	Character string	Handle that is returned from the `Send` method.
`lTimeout`	Number	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

ReceiveAuthorizationRequest

It provides a license code which is used by the client to generate a response. The client inserts the calculated license code in the `SendAuthorizationResponse` method. This method is only used by the Bizerba software. If the `Open` method has not been carried out, `ReceiveAuthorizationRequest` reports an error.

Syntax and parameter		

C++
`long ReceiveAuthorizationRequest (BSTR *szLizenzKey)`
C#
`int ReceiveAuthorizationRequest (ref string szLizenzKey)`		

Parameters	**Value**	**Description**
`szLizenzKey`	Character string	License code generated by *_connectServer*

ReceiveOne

It provides reception data (header and user data combined) and the transmission status. It requires the handle from the `Send` method.

If the `Open` method has not been carried out, `ReceiveOne` reports an error.

Syntax and parameter		

C++
`long ReceiveOne (BSTR *szHeaderData, BSTR szHandle, long lTimeout, long *lStatus)`
C#
`int ReceiveOne (ref string szHeaderData, string szHandle, int lTimeout, ref int lStatus)`		

Parameters	**Value**	**Description**
`szHeaderData`	Character string	Combination of header and user data
`szHandle`	Character string	Handle that is returned from the `Send` method.
`lTimeout`	Number	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

ReceiveOneWithoutAck

It receives spontaneous telegrams without direct acknowledgement. When using this method, the customer application can delay the transmission of the acknowledgement and transmit data (e.g. lock) to the labeler prior to transmitting the acknowledgement. The acknowledgement is separately transmitted using the `SendAcknowledge` method.

The `ReceiveOneWithoutAck` method can only be used for spontaneous data. The customer application has to acknowledge the data record with `SendAcknowledge` so that a GLP proceeds after a lock.

Syntax and parameter

C++

```
long ReceiveOneWithoutAck (BSTR *szHeaderData, BSTR szHandle, long lTimeout,
long *lStatus)
```

C#

```
int ReceiveOneWithoutAck (ref string szHeaderData, string szHandle, int lTime-
out, ref int lStatus)
```

Parameters	Value	Description
`szHeaderData`	Character string	Header data (data description)
`szHandle`	Character string	Handle for spontaneous data (DUSTBIN or created queue). Spontaneous data deriving from the device without related user queue is automatically allocated to a DUSTBIN queue. They can be picked up using the DUSTBIN handle.
`lTimeout`	Number	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

ReceiveWithoutAck

It receives spontaneous telegrams without direct acknowledgement. When using this method, the customer application can delay the transmission of the acknowledgement and transmit data (e.g. lock) to the labeler prior to transmitting the acknowledgement. The acknowledgement is separately transmitted using the `SendAcknowledge` method.

The `ReceiveWithoutAck` method can only be used for spontaneous data. The customer application has to acknowledge the data record with `SendAcknowledge` so that a GLP proceeds after a lock.

Syntax and parameter

C++

```
long ReceiveWithoutAck (BSTR *szHeader, BSTR *szData, BSTR szHandle, long lTimeout, long *lStatus)
```

C#

```
int ReceiveWithoutAck (ref string szHeader, ref string szData, string szHandle, int lTimeout, ref int lStatus)
```

Parameters	Value	Description
`szHeader`	Character string	Header data (data description)
`szData`	Character string	User data
`szHandle`	Character string	Handle for spontaneous data (DUSTBIN or created queue).
`lTimeout`	Number	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

Reset

It cancels the current send request. If a send request is not cancelled, it remains active until transmission is completed. Thus, it is impossible to start a new send request.

Use `Reset` to cancel the send request from the *_connectServer* transmission list. If the data record has already been sent to the device before `Reset` and the device continues responding after `Reset`, the response is identified as spontaneous data record and made available because the corresponding send request no longer exists.

Syntax and parameter		

C++
`long Reset (BSTR szHandle)`
C#
`int Reset (string szHandle)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle that is returned from the `Send` method.

Send

It sends data to the connected evaluation unit. If the `Open` method has not been carried out, `Send` reports an error.

Syntax and parameter		

C++
`long Send (BSTR szHeader, BSTR szData, BSTR *szHandle, long lTimeout, long *lStatus)`
C#
`int Send (string szHeader, string szData, ref string szHandle, int lTimeout, ref int lStatus)`		

Parameters	**Value**	**Description**
`szHeader`	Character string	Header data (data description)
`szData`	Character string	User data
`szHandle`	Character string	Handle generated by *_connect-Server*. It has to be specified in the `Receive` method to provide the client with the relevant data.
`lTimeout`	Number	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

SendAcknowledge

It closes the `ReceiveWithoutAck` method with a positive acknowledgement.

Syntax and parameter		

C++
`long SendAcknowledge (BSTR szHandle)`
C#
`int SendAcknowledge (string szHandle)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle used in the `ReceiveWithoutAck` method.

SendAcknowledgeNeg

It closes the `ReceiveWithoutAck` method with a negative acknowledgement. Transmits an error message to the device.

Syntax and parameter		

C++
`long SendAcknowledgeNeg (BSTR szHandle, long lError, BSTR szErrTxt)`
C#
`int SendAcknowledgeNeg (string szHandle, int lError, string szErrTxt)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle used in the `ReceiveWithoutAck` method.
`lError`	Number	Error number to be sent to the device
`szErrTxt`	Character string	Error text to be sent to the device

SendAuthorizationResponse

It compares the license code calculated by the client with the license code transmitted by *_connectServer* via `ReceiveAuthorizationRequest`. This method is only used by the Bizerba software.

In case of noncompliance, an error is reported.

If the `Open` method has not been carried out, `SendAuthorizationReponse` reports an error.

Syntax and parameter		

C++
`long SendAuthorizationResponse (BSTR szLizenzKey)`
C#
`int SendAuthorizationResponse (string szLizenzKey)`		

Parameters	**Value**	**Description**
`szLizenzKey`	Character string	License code calculated by the client

SendCheck

It verifies whether data is available, the command has been completely processed or transmitted, and a positive or negative acknowledgement has been received.

Syntax and parameter		

C++
`long SendCheck (BSTR szHandle, long lTimeout, long *lStatus)`
C#
`int SendCheck (string szHandle, int lTimeout, ref int lStatus)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle that is returned from the `Send` method.
`lTimeout`	Number	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

SendOne

It sends data to the connected evaluation unit. If the `Open` method has not been carried out, `Send` reports an error.

Syntax and parameter		

C++
`long SendOne (BSTR szHeaderData, BSTR *szHandle, long lTimeout, long *lStatus)`
C#
`int SendOne (string szHeaderData, ref string szHandle, int lTimeout, ref int lStatus)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Handle generated by *_connect-Server*. It has to be specified in the `Receive` method to provide the client with the relevant data.
`lTimeout`	Number	Period of time during which the method waits for an answer from the device. After the time elapsed, the function will be exited with status 1 (timeout). The send request remains in the transmission list.
`lStatus`	`0`: function OK `1`: timeout `2`: more data available	Transmission status

SetReceiveQueueFilter

It sets a filter for reception data. The filter is required to receive data by means of customer-specific reception queues that have been created using the `CreateReceiveQueue` method. Without filters, spontaneous data is automatically assigned to the DUSTBIN reception queue.

Example for a GX: Use `CreateReceiveQueue` to create an individual queue for package data records starting with *PV05*. To allocate the package data records to the method, call up the `SetReceiveQueueFilter` method with the queue name and the `PV05` filter parameter. If data records, e.g. starting with *PV05* or *PV06*, are to be allocated to the same queue, call up the `SetReceiveQueueFilter` method two times. Specify the same queue both times and use the `PV05` filter parameter at one time and the `PV06` filter parameter the other time.

Syntax and parameter		

C++
`long SetReceiveQueueFilter (BSTR szQueueName, BSTR szFilter)`
C#
`int SetReceiveQueueFilter (string szQueueName, string szFilter)`		

Parameters	**Value**	**Description**
`szQueueName`	Character string	Queue name generated by *_connectServer* The `CreateReceiveQueue` method returns this name.
`szFilter`	Character string	Filter parameter, e.g. `PV05` for a GX.

19.3.3 Events

BCCDataArrival

It indicates reception of spontaneous telegrams and transmits the identification of the communication channel (handle) which can be used for receiving data via the `Receive`, `ReceiveWithoutAck`, `ReceiveOneWithoutAck` or `ReceiveOne` method.

Syntax and parameter		

C++
`void BCCDataArrival(BSTR szHandle)`
C++
`void BCCDataArrival(string szHandle)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Identification of communication channel (handle)

BCCRemoteDataArrival

It answers a send request with a `?`, `A?` or `I?` header. Transmits the identification of the communication channel (handle) which can be used for receiving data via the `Receive`, `ReceiveWithoutAck`, `ReceiveOneWithoutAck` or `ReceiveOne` method.

Syntax and parameter		

C++
`void BCCRemoteDataArrival(BSTR szHandle)`
C#
`void BCCRemoteDataArrival(string szHandle)`		

Parameters	**Value**	**Description**
`szHandle`	Character string	Identification of communication channel (handle)

BCCError

It indicates the failure of a previous action. Further details are displayed by the error code and text.

Syntax and parameter		

C++
`void BCCError(long lErrCode, BSTR szErrTxt)`
C#
`void BCCError(int lErrCode, string szErrTxt)`		

Parameters	**Value**	**Description**
`lErrCode`	Number	Error code
`szErrTxt`	Character string	Error text

19.3.4 IsUnicodeDevice

Syntax and parameter		

C++
`Long IsUnicodeDevice(short* bIsUnicodeDevice)`
C#
`int IsUnicodeDevice(ref short bIsUnicodeDevice)`		

Parameter	**Value**	**Description**
`bIsUnicodeDevice`	`0`: Configured device is not a Unicode device `1`: Configured device is a Unicode device	Indicates if configured device is a code page based device or a Unicode device.

19.4 BctFunctions

The BctFunctions.dll is a COM-Dll that makes available auxiliary functions for handling Bx telegrams. It includes conversion functions for handling dimensionful values and UTF-8/ Unicode texts as well as parse functions for Bx telegrams in different formats.

19.4.1 Conversion function

ConvertBxNetToWeight

Conversion function that breaks down a Bx dimension value in single components.

Syntax and parameter	

C++
`HRESULT ConvertBxNetToWeight([in] BSTR szWeightData, [in] BSTR szWeightDelimiter, [out] BSTR *szWeight, [out] BSTR *szWeightUnit);`
C#
`int ConvertBxNetToWeight(string szWeightData, string szWeightDelimiter, out string szWeight, out string szWeightUnit);`	

Parameters	**Description**
`szWeightData`	Dimension value in Bx format Example: "kg;-3;1234"
`szWeightDelimiter`	Decimal separator to be used
`szWeight`	The weight is returned as string with the specified decimal separator.
`szWeightUnit`	It is used to return the Bx dimension value unit.

Example (C#)

```
string weight, weightUnit;
tools.ConvertBxNetToWeight("kg;-3;1234", ".", out weight, out weightUnit);
Console.WriteLine(weight + " " + weightUnit); // "1.234 kg"
```

ConvertWeightToBxNet

Conversion function that creates a Bx dimension value from weight, decimal separator and weight unit.

Syntax and parameter	

C++
`HRESULT ConvertWeightToBxNet([in] BSTR szWeight, [in] BSTR szWeightDelimiter, [in] BSTR szWeightUnit, [out] BSTR *szWeightData);`
C#
`int ConvertWeightToBxNet(string szWeight, string szWeightDelimiter, string szWeightUnit, out string szWeightData);`	

Parameters	**Description**
`szWeight`	Weight as string with decimal separator Example: "1.234 kg".
`szWeightDelimiter`	Decimal separator used in `szWeight`. Example: ","
`szWeightUnit`	Weight unit used in `szWeight`. Example: "kg"
`szWeightData`	Dimension values are returned in Bx format. Example: "kg;-3;1234"

Example (C#)

```
string weight.Data;
tools.ConvertWeightToBxNet("1.234", ".", "kg", out weightData);
Console.WriteLine(weightData); // "kg;-3;1234"
```

ConvertTextUnicodeToBxNetUTF8

Function that converts a Unicode string in UTF-8. Characters that cannot be represented in ASCII format are coded according to Bx rules.

Syntax and parameter	

C++
`HRESULT ConvertTextUnicodeToBxNetUTF8([in] BSTR szUnicodeText, [out] BSTR *szBxNetUTF8Text);`
C#
`int ConvertTextUnicodeToBxNetUTF8(string szUnicodeText, out string szBxNetUTF8Text);`	

Parameters	**Description**
`szUnicodeText`	Unicode string
`szBxNetUTF8Text`	An entered string is returned as UTF-8 text that is coded according to Bx rules for texts.

Example (C#)

```
string utf8;
tools.ConvertTextUnicodeToBxNetUTF8("fünf €", out utf8);
Console.WriteLine(utf8); // "f@C3@BCnf @E2@82@AC"
```

ConvertTextBxNetUTF8ToUnicode

Function that converts a UTF-8 string in Unicode. The input string must be coded according to the Bx rules.

Syntax and parameter	

C++
`HRESULT ConvertTextBxNetUTF8ToUnicode([in] BSTR szBxNetUTF8Text, [out] BSTR *szUnicodeText);`
C#
`int ConvertTextBxNetUTF8ToUnicode(string szBxNetUTF8Text, out string szUnicodeText);`	

Parameters	**Description**
`szBxNetUTF8Text`	UTF-8 string coded according to Bx rules for texts
`szUnicodeText`	Return of decoded Unicode string

Example (C#)

```
string unicode;
tools.ConvertTextBxNetUTF8ToUnicode("f@C3@BCnf @E2@82@AC", out unicode);
Console.WriteLine(unicode); // "fünf €"
```

ConvertTextUnicodeToBxNetUnicode

Conversion function converting a Unicode string to a BxNet Unicode string allowing the string to be sent to a Unicode device. Characters that cannot be represented in ASCII format are coded according to Bx rules.

Syntax and parameter	

C++
`HRESULT ConvertTextUnicodeToBxNetUnicode ([in] BSTR szUnicodeText, [out] BSTR* szBxNetUnicodeText);`
C#
`int ConvertTextUnicodeToBxNetUnicode (string szUnicodeText, out string szBxNetUnicodeText);`	

Parameter	**Description**
`szUnicodeText`	Unicode string
`szBxNetUnicodeText`	Returns an entered string as Unicode text for the devices which is coded based on Bx text rules.

Example (C#)

```
string unicodeForDevice;
tools.ConvertTextUnicodeToBxNetUnicode ("hello world, with characters € and @",
out unicodeForDevice);
Console.WriteLine(unicodeForDevice); // "hello world, with characters € and @40"
```

ConvertTextBxNetUnicodeToUnicode

Conversion function converting a device Unicode string to a Unicode string. The input string must be coded according to the Bx rules.

Syntax and parameter	

C++
`HRESULT ConvertTextBxNetUnicodeToUnicode ([in] BSTR szBxNetUnicodeText, [out] BSTR* szUnicodeText);`
C#
`int ConvertTextBxNetUnicodeToUnicode (string szBxNetUnicodeText, out string szUnicodeText);`	

Parameter	**Description**
`szBxNetUnicodeText`	Unicode text of a Unicode device
`szUnicodeText`	Returns decoded Unicode string

Example (C#)

```
string unicode;
tools.ConvertTextBxNetUnicodeToUnicode ("hello world, with characters € and
@40", out unicode);
Console.WriteLine(unicode); // " hello world, with characters € and @"
```

19.4.2 Parse functions

ParseTelegram

Parse function used to break down Bx telegrams (header and data separated) in single components.

Syntax and parameter	

C++
`HRESULT ParseTelegram([in] BSTR szHeader, [in] BSTR szData, [out] VARIANT *pHeaders, [out] VARIANT *pDatas);`
C#
`int ParseTelegram(string szHeader, string szData, out object pHeaders, out object pDatas);`	

Parameters	**Description**
`szHeader`	Bx header
`szData`	Bx data string suitable for Bx header
`pHeaders`	Returns single command as array.
`pDatas`	Returns single data values as array.

Example (C#)

```
object headers, datas;
string[] headersArray, datasArray;
tools.ParseTelegram("I!LV01|RX01|GT08|LX02", "Scale1", out headers, out datas);
headersArray = (string[])headers; // "LV01", "RX01", "GT08", "LX02"
datasArray = (string[])datas; // "", "", "Scale1", ""
```

ParseTelegramEx

Parse function used to break down Bx telegrams (header and data combined) in single components.

Syntax and parameter	

C++
`HRESULT ParseTelegramEx([in] BSTR szHeaderData, [out] VARIANT *pHeaders, [out] VARIANT *pDatas);`
C#
`int ParseTelegramEx(string szHeaderData, out object pHeaders, out object pDatas);`	

Parameters	**Description**
`szHeaderData`	Bx telegram, header and data combined
`pHeaders`	Returns single command as array.
`pDatas`	Returns single data values as array.

Example (C#)

```
object headers, datas;
string[] headersArray, datasArray;
tools.ParseTelegramEx("I!LV01|RX01|GT08|Scale1|LX02", out headers, out datas);
headersArray = (string[])headers; // "LV01", "RX01", "GT08", "LX02"
datasArray = (string[])datas; // "", "", "Scale1", ""
```

ParseMemocardTelegram

Parse function used to divide memory card telegrams deriving from Gx (header and data separated) in single telegrams Gx (header and data separated).

Syntax and parameter	

C++
`HRESULT ParseMemocardTelegram([in] BSTR szHeader, [in] BSTR szData, [out] VARIANT *pHeaders, [out] VARIANT *pDatas);`
C#
`int ParseMemocardTelegram(string szHeader, string szData, out object pHeaders, out object pDatas);`	

Parameters	**Description**
`szHeader`	Gx memory card header
`szData`	Gx memory card data string suitable for header
`pHeaders`	Returns single header telegrams deriving from the memory card as array.
`pDatas`	Returns single data telegrams deriving from the memory card as array.

Example (C#)

```
object headers, datas;
string[] headersArray, datasArray;
tools.ParseMemocardTelegram("A!MV07|GT01|LV01|RX01|GT08|LX02|LX02", "test|
Scale1", out headers, out datas);
headersArray = (string[])headers; // "A!GT01", "A!LV01|RX01|GT08|LX02"
datasArray = (string[])datas; // "test", "Scale1"
```

ParseMemocardTelegramEx

Parse function used to divide memory card telegrams deriving from Gx (header and data combined) in single telegrams Gx (header and data separated).

Syntax and parameter	

C++
`HRESULT ParseMemocardTelegramEx([in] BSTR szHeaderData, [out] VARIANT *pHeaders, [out] VARIANT *pDatas);`
C#
`int ParseMemocardTelegramEx(string szHeaderData, out object pHeaders, out object pDatas);`	

Parameters	**Description**
`szHeaderData`	Gx memory card telegram, header and data combined
`pHeaders`	Returns single header telegrams deriving from the memory card as array.
`pDatas`	Returns single data telegrams deriving from the memory card as array.

Example (C#)

```
object headers, datas;
string[] headersArray, datasArray;
tools.ParseMemocardTelegramEx("A!MV07|GT01|test|LV01|RX01|GT08|Scale1|LX02|
LX02", out headers, out datas);
headersArray = (string[])headers; // "A!GT01", "A!LV01|RX01|GT08|LX02"
datasArray = (string[])datas; // "test", "Scale1"
```

ParseMemocardTelegramEx2

Parse function used to divide memory card telegrams deriving from Gx (header and data combined) in single telegrams Gx (header and data combined).

Syntax and parameter	

C++
`HRESULT ParseMemocardTelegramEx2([in] BSTR szHeaderData, [out] VARIANT *pHeaderData);`
C#
`int ParseMemocardTelegramEx2(string szHeaderData, out object pHeaderData);`	

Parameters	**Description**
`szHeaderData`	Gx memory card telegram, header and data combined
`pHeaderData`	Returns single telegrams with header and data combined, as array.

Example (C#)

```
object headerdata;
string[] headerDataArray;
tools.ParseMemocardTelegramEx2("A!MV07|GT01|test|LV01|RX01|GT08|Scale1|LX02|
LX02", out headerdata);
headerDataArray = (string[])headerdata; // "A!GT01|test", "A!LV01|RX01|GT08|
Scale1|LX02"
```