Rexroth

ScreenManager 04VRS

Operating and Programming Guide

SYSTEM200

DOK-SUPPL*-SCM*PROG*V4-AW01-EN-P

Rexroth Indramat

Title ScreenManager 04VRS

Type of Documentation Operating and Programming Guide

Document Typecode DOK-SUPPL*-SCM*PROG*V4-AW01-EN-P

Internal File Reference Document Number, 120-2100-B340-01/EN

Purpose of Documentation This documentation

- serves as installation instruction,
- describes the use of the project planning tool,
- describes the system library.

Record of Revisions

Description	Release Date	Notes
120-2100-B340-01/EN	11/01	First issue

Published by Rexroth Indramat GmbH
Bgm.-Dr.-Nebel-Str. 2 • D-97816 Lohr a. Main

Telephone +49 (0)93 52/40-0 • Tx 68 94 21 • Fax +49 (0)93 52/40-48 85

http://www.boschrexroth.de/

Dept. BRC/EP7 (KP/JI)

Dept. BRC/EP (NH)

Note This document has been printed on chlorine-free bleached paper.

Contents

1 Introduction

1.1 Purpose of this Documentation

This documentation describes the operating and handling of the ScreenManager PC software.

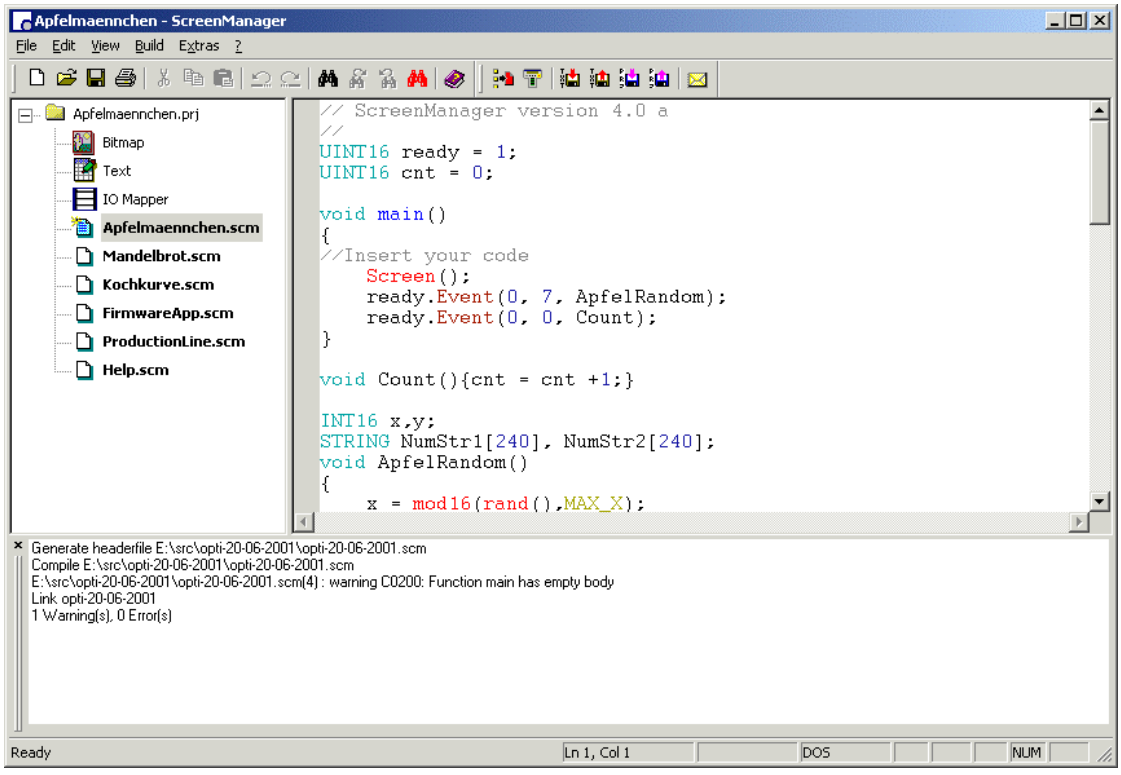
1.2 Further Documentation

Device/accessories	Documentation type code	Remark
BTV04 – Project planning manual	DOK-SUPPL*-BTV04.2****	Hardware descriptions with technical data, outline dimensions, connections, program download with Dolfi, accessories, cables, application examples for Indramat controls, etc.
BTV05 – Project planning manual	DOK-SUPPL*-BTV05.2****	
BTV06 – Project planning manual	DOK-SUPPL*-BTV06.1****	
BTC06 – Project planning manual	DOK-SUPPL*-BTC06******	
ScreenManager	DOK-SUPPL*-SCM*PROG***	Description of the PC programming interface ScreenManager with installation, operation and explication of the programming language

Fig. 1-1: Further documentation

1.3 Brief Description

ScreenManager is a development environment to generate applications for Rexroth Indramat miniature control panels.



SCM4_Anwendungsfenster.bmp

Fig. 1-2: ScreenManager – application window

ScreenManager 04VRS performs the following tasks:

- Management of SCM source files
- Support of multilingual text and image resources
- IO mapper functionality
- Starting of compilation and download to the miniature control panel

2 Important Directions for Use

2.1 Appropriate Use

Introduction

Rexroth Indramat products represent state-of-the-art developments and manufacturing. They are tested prior to delivery to ensure operating safety and reliability.

The products may only be used in the manner that is defined as appropriate. If they are used in an inappropriate manner, then situations can develop that may lead to property damage or injury to personnel.

Note: Rexroth Indramat, as manufacturer, is not liable for any damages resulting from inappropriate use. In such cases, the guarantee and the right to payment of damages resulting from inappropriate use are forfeited. The user alone carries all responsibility of the risks.

Before using Rexroth Indramat products, make sure that all the pre-requisites for appropriate use of the products are satisfied:

- Personnel that in any way, shape or form uses our products must first read and understand the relevant safety instructions and be familiar with appropriate use.
- If the product takes the form of hardware, then they must remain in their original state, in other words, no structural changes are permitted. It is not permitted to decompile software products or alter source codes.
- Do not mount damaged or faulty products or use them in operation.
- Make sure that the products have been installed in the manner described in the relevant documentation.

Areas of Use and Application

The system ScreenManager is a development environment to generate applications for Rexroth Indramat miniature control panels. The system ScreenManager performs the following tasks:

- management of SCM source files
- support of multilingual text and image files
- IO mapper functionality
- starting of compilation and download to the miniature control panel.

Note: The system ScreenManager may only be used with the accessories and parts specified in this document. If a component has not been specifically named, then it may not be either mounted or connected. The same applies to cables and lines.

Operation is only permitted in the specified configurations and combinations of components using the software and firmware as specified in the relevant function descriptions.

2.2 Inappropriate Use

Using the system ScreenManager outside of the above-referenced areas of application or under operating conditions other than described in the document and the technical data specified is defined as “inappropriate use".

The system ScreenManager may not be used if

- they are subject to operating conditions that do not meet the above specified ambient conditions. or if
- Rexroth Indramat has not specifically released them for that intended purpose. Please note the specifications outlined in the general Safety Instructions!

2.3 Delivery Stipulations for Computer Programs

The copyrights, present and future commercial proprietary rights of all kinds, as well as all the rights of exploitation to delivered computer programs -- in equipment or separate from it -- belong exclusively to the Supplier.

A computer program may only be used in one single piece of equipment. Exceptions are commissioning software, which are marked with the designation -COPY at the end. These can be copied freely within the context of regular product usage by the customer.

Every act exceeding the minimum use outlined in the proprietary rights requires the consent of the Supplier. If a computer program delivered by the Supplier is not protected by proprietary rights, then the minimum use stated in the proprietary rights laws is declared as agreed upon.

If the Orderer transfers a computer program then he must completely surrender the program carrier and all copies in their entirety to the Acquiring Party, or these must be erased. A limitation of use corresponding to these stipulations (1 through 6) must be agreed upon with the Acquiring Party.

The Supplier will eliminate any fault in the computer program either by a circumvention of the fault, which is agreeable to the Orderer, or by delivering a new program.

All documents and information needed to reconstruct a fault must accompany the notification of a fault in the computer program.

Otherwise, the general delivery stipulations outlined by INDRAMAT apply.

3 Safety Instructions for Electric Drives and Controls

3.1 Introduction

Read these instructions before the initial startup of the equipment in order to eliminate the risk of bodily harm or material damage. Follow these safety instructions at all times.

Do not attempt to install or start up this equipment without first reading all documentation provided with the product. Read and understand these safety instructions and all user documentation of the equipment prior to working with the equipment at any time. If you do not have the user documentation for your equipment, contact your local Rexroth Indramat representative to send this documentation immediately to the person or persons responsible for the safe operation of this equipment.

If the equipment is resold, rented or transferred or passed on to others, then these safety instructions must be delivered with the equipment.

WARNING

Improper use of this equipment, failure to follow the safety instructions in this document or tampering with the product, including disabling of safety devices, may result in material damage, bodily harm, electric shock or even death!

3.2 Explanations

The safety instructions describe the following degrees of hazard seriousness in compliance with ANSI Z535. The degree of hazard seriousness informs about the consequences resulting from non-compliance with the safety instructions.

Warning symbol with signal word	Degree of hazard seriousness according to ANSI
 **DANGER**	Death or severe bodily harm will occur.
 **WARNING**	Death or severe bodily harm may occur.
 **CAUTION**	Bodily harm or material damage may occur.

Fig. 3-1: Hazard classification (according to ANSI Z535)

3.3 Hazards by Improper Use

DANGER

High voltage and high discharge current! Danger to life or severe bodily harm by electric shock !

DANGER

Dangerous movements! Danger to life, severe bodily harm or material damage by unintentional motor movements!

WARNING

High electrical voltage due to wrong connections! Danger to life or bodily harm by electric shock!

WARNING

Health hazard for persons with heart pacemakers, metal implants and hearing aids in proximity to electrical equipment!

CAUTION

Surface of machine housing could be extremely hot! Danger of injury! Danger of burns!

CAUTION

Risk of injury due to improper handling! Bodily harm caused by crushing, shearing, cutting and mechanical shock or incorrect handling of pressurized systems!

CAUTION

Risk of injury due to incorrect handling of batteries!

3.4 General Information

- Rexroth Indramat GmbH is not liable for damages resulting from failure to observe the warnings provided in this documentation.
- Read the operating, maintenance and safety instructions in your language before starting up the machine. If you find that you can not completely understand the documentation for your product, please ask your supplier to clarify.
- Proper and correct transport, storage, assembly and installation as well as care in operation and maintenance are prerequisites for optimal and safe operation of this equipment.
- Only persons who are trained and qualified for the use operation of the equipment may work on this equipment or within its proximity. The persons are qualified if they have sufficient knowledge of the assembly, installation and operation of the equipment as well as an understanding of all warnings and precautionary measures noted in these instructions.
  Furthermore, they must be trained, instructed and qualified to switch electrical circuits and equipment on and off in accordance with technical safety regulations, to ground them and to mark them according to the requirements of safe work practices. They must have adequate safety equipment and be trained in first aid.
- Only use spare parts and accessories approved by the manufacturer.
- Follow all safety regulations and requirements for the specific application as practiced in the country of use.
- The equipment is designed for installation in industrial machinery.
- The ambient conditions given in the product documentation must be observed.
- Use only safety features and applications that are clearly and explicitly approved in the Project Planning Manual.
  For example, the following areas of use are not permitted: construction cranes, elevators used for people or freight, devices and vehicles to transport people, medical applications, refinery plants, transport of hazardous goods, nuclear applications, applications sensitive to high frequency, mining, food processing, control of protection equipment (also in a machine).
- The information given in this documentation with regard to the use of the delivered components contains only examples of applications and suggestions.

  The machine and installation manufacturer must
  - make sure that the delivered components are suited for his individual application and check the information given in this documentation with regard to the use of the components,
  - make sure that his application complies with the applicable safety regulations and standards and carry out the required measures, modifications and complements.
- Startup of the delivered components is only permitted once it is sure that the machine or installation in which they are installed complies with the national regulations, safety specifications and standards of the application.

- Operation is only permitted if the national EMC regulations for the application are met.
  The instructions for installation in accordance with EMC requirements can be found in the documentation "EMC in Drive and Control Systems."
  The machine or installation manufacturer is responsible for compliance with the limiting values as prescribed in the national regulations.
- Technical data, connections and operational conditions are specified in the product documentation and must be followed at all times.

3.5 Protection Against Contact with Electrical Parts

Note: This section refers to equipment and drive components with voltages above 50 Volts.

Touching live parts with voltages of 50 Volts and more with bare hands or conductive tools or touching ungrounded housings can be dangerous and cause electric shock . In order to operate electrical equipment, certain parts must unavoidably have dangerous voltages applied to them.

DANGER

High electrical voltage! Danger to life, severe bodily harm by electric shock!

⇒ Only those trained and qualified to work with or on electrical equipment are permitted to operate, maintain or repair this equipment.

⇒ Follow general construction and safety regulations when working on high voltage installations.

⇒ Before switching on power the ground wire must be permanently connected to all electrical units according to the connection diagram.

⇒ Do not operate electrical equipment at any time , even for brief measurements or tests, if the ground wire is not permanently connected to the points of the components provided for this purpose.

⇒ Before working with electrical parts with voltage higher than 50 V, the equipment must be disconnected from the mains voltage or power supply. Make sure the equipment cannot be switched on again unintended.

⇒ The following should be observed with electrical drive and filter components:
Wait five (5) minutes after switching off power to allow capacitors to discharge before beginning to work. Measure the voltage on the capacitors before beginning to work to make sure that the equipment is safe to touch.

⇒ Never touch the electrical connection points of a component while power is turned on.

⇒ Install the covers and guards provided with the equipment properly before switching the equipment on. Prevent contact with live parts at any time.

⇒ A residual-current-operated protective device (RCD) must not be used on electric drives! Indirect contact must be prevented by other means, for example, by an overcurrent protective device.

⇒ Electrical Components with exposed live parts and uncovered high voltage terminals must be installed in a protective housing, for example in a control cabinet.

To be observed with electrical drive and filter components:

DANGER

High electrical voltage on the housing!
High leakage current! Danger to life, danger of injury by electric shock!

⇒ Connect the electrical equipment, the housings of all electrical units and motors permanently with the safety conductor at the ground points before power is switched on. Look at the connection diagram. This is even necessary for brief tests.

⇒ Connect the safety conductor of the electrical equipment always permanently and firmly to the supply mains. Leakage current exceeds 3.5 mA in normal operation.

⇒ Use a copper conductor with at least 10 mm² cross section over its entire course for this safety conductor connection!

⇒ Prior to startups, even for brief tests, always connect the protective conductor or connect with ground wire. Otherwise, high voltages can occur on the housing that lead to electric shock.

3.6 Protection Against Electric Shock by Protective Low Voltage (PELV)

All connections and terminals with voltages between 0 and 50 Volts on Rexroth Indramat products are protective low voltages designed in accordance with international standards on electrical safety.

WARNING

High electrical voltage due to wrong connections! Danger to life, bodily harm by electric shock !

⇒ Only connect equipment, electrical components and cables of the protective low voltage type (PELV = Protective Extra Low Voltage) to all terminals and clamps with voltages of 0 to 50 Volts.

⇒ Only electrical circuits may be connected which are safely isolated against high voltage circuits. Safe isolation is achieved, for example, with an isolating transformer, an opto-electronic coupler or when battery-operated.

3.7 Protection Against Dangerous Movements

Dangerous movements can be caused by faulty control of the connected motors. Some common examples are:

- improper or wrong wiring of cable connections
- incorrect operation of the equipment components
- malfunction of sensors, encoders and monitoring device
- defective components
- incorrect projecting
- software or firmware errors

Dangerous movements can occur immediately after equipment is switched on or even after an unspecified time of trouble-free operation.

The monitoring in the drive components will normally be sufficient to avoid faulty operation in the connected drives. Regarding personal safety, especially the danger of bodily injury and material damage, this alone cannot be relied upon to ensure complete safety. Until the integrated monitoring functions become effective, it must be assumed in any case that faulty drive movements will occur. The extent of faulty drive movements depends upon the type of control and the state of operation.

DANGER

Dangerous movements! Danger to life, risk of injury, severe bodily harm or material damage!

⇒ Ensure personal safety by means of qualified and tested higher-level monitoring device or measures integrated in the installation. Unintended machine motion is possible if monitoring device are disabled, bypassed or not activated.

⇒ Pay attention to unintended machine motion or other malfunction in any mode of operation.

⇒ Keep free and clear of the machine's range of motion and moving parts. Possible measures to prevent people from accidentally entering the machine's range of movement:

- use safety fences
- use safety guards
- use protective coverings
- install light curtains or light barriers

⇒ Fences and coverings must be strong enough to resist maximum possible momentum, especially if break off parts can fly into the environment.

⇒ Mount the emergency stop switch in the immediate reach of the operator. Verify that the emergency stop works before startup. Don't operate the machine if the emergency stop is not working.

⇒ Isolate the drive power connection by means of an emergency stop circuit or use a starting lockout to prevent unintentional start.

⇒ Make sure that the drives are brought to a safe standstill before accessing or entering the danger zone. Safe standstill can be achieved by switching off the power supply contactor or by safe mechanical locking of moving parts.

⇒ Secure vertical axes against falling or dropping after switching off the motor power by, for example:
- mechanically securing the vertical axes
- adding an external braking/ arrester/ clamping mechanism
- ensuring sufficient equilibration of the vertical axes

The standard equipment motor brake or an external brake controlled directly by the drive controller are not sufficient to guarantee personal safety!

⇒ Disconnect electrical power to the equipment using a master switch and secure the switch against reconnection for:
- maintenance and repair work
- cleaning of equipment
- long periods of discontinued equipment use

⇒ Prevent the operation of high-frequency, remote control and radio equipment near electronics circuits and supply leads. If the use of such equipment cannot be avoided, verify the system and the installation for possible malfunctions in all possible positions of normal use before initial startup. If necessary, perform a special electromagnetic compatibility (EMC) test on the installation.

3.8 Protection Against Magnetic and Electromagnetic Fields During Operation and Mounting

Magnetic and electromagnetic fields generated near current-carrying conductors and permanent magnets in motors represent a serious health hazard to persons with heart pacemakers, metal implants and hearing aids.

WARNING

Health hazard for persons with heart pacemakers, metal implants and hearing aids in proximity to electrical equipment!

⇒ Persons with heart pacemakers, hearing aids and metal implants are not permitted to enter following areas:
- Areas in which electrical equipment and parts are mounted, being operated or started up.
- Areas in which parts of motors with permanent magnets are being stored, operated, repaired or mounted.

⇒ If it is necessary for a person with a heart pacemaker to enter such an area, then a doctor must be consulted prior to doing so. Heart pacemakers that are already implanted or will be implanted in the future, have a considerable variation in their electrical noise immunity. Therefore there are no rules with general validity.

⇒ Persons with hearing aids, metal implants or metal pieces must consult a doctor before they enter the areas described above. Otherwise, health hazards will occur.

3.9 Protection Against Contact with Hot Parts

CAUTION

Housing surfaces could be extremely hot! Danger of injury! Danger of burns!

⇒ Do not touch housing surfaces near sources of heat! Danger of burns!

⇒ After switching the equipment off, wait at least ten (10) minutes to allow it to cool down before touching it.

⇒ Do not touch hot parts of the equipment, such as housings with integrated heat sinks and resistors. Danger of burns!

3.10 Protection During Handling and Mounting

Under certain conditions, incorrect handling and mounting of parts and components may cause injuries.

CAUTION

Risk of injury by incorrect handling! Bodily harm caused by crushing, shearing, cutting and mechanical shock!

⇒ Observe general installation and safety instructions with regard to handling and mounting.

⇒ Use appropriate mounting and transport equipment.

⇒ Take precautions to avoid pinching and crushing.

⇒ Use only appropriate tools. If specified by the product documentation, special tools must be used.

⇒ Use lifting devices and tools correctly and safely.

⇒ For safe protection wear appropriate protective clothing, e.g. safety glasses, safety shoes and safety gloves.

⇒ Never stand under suspended loads.

⇒ Clean up liquids from the floor immediately to prevent slipping.

3.11 Battery Safety

Batteries contain reactive chemicals in a solid housing. Inappropriate handling may result in injuries or material damage.

CAUTION

Risk of injury by incorrect handling!

⇒ Do not attempt to reactivate discharged batteries by heating or other methods (danger of explosion and cauterization).

⇒ Never charge non chargeable batteries (danger of leakage and explosion).

⇒ Never throw batteries into a fire.

⇒ Do not dismantle batteries.

⇒ Do not damage electrical components installed in the equipment.

Note: Be aware of environmental protection and disposal! The batteries contained in the product should be considered as hazardous material for land, air and sea transport in the sense of the legal requirements (danger of explosion). Dispose batteries separately from other waste. Observe the legal requirements in the country of installation.

3.12 Protection Against Pressurized Systems

Certain motors and drive controllers, corresponding to the information in the respective Project Planning Manual, must be provided with pressurized media, such as compressed air, hydraulic oil, cooling fluid and cooling lubricant supplied by external systems. Incorrect handling of the supply and connections of pressurized systems can lead to injuries or accidents. In these cases, improper handling of external supply systems, supply lines or connections can cause injuries or material damage.

CAUTION

Danger of injury by incorrect handling of pressurized systems !

Do not attempt to disassemble, to open or to cut a pressurized system (danger of explosion).

Observe the operation instructions of the respective manufacturer.

Before disassembling pressurized systems, release pressure and drain off the fluid or gas.

Use suitable protective clothing (for example safety glasses, safety shoes and safety gloves)

Remove any fluid that has leaked out onto the floor immediately.

Note: Environmental protection and disposal! The media used in the operation of the pressurized system equipment may not be environmentally compatible. Media that are damaging the environment must be disposed separately from normal waste. Observe the legal requirements in the country of installation.

4 Installation

4.1 Installing the ScreenManager Software

The SETUP.EXE program starts the software installation.

First, the setup program installs the latest version of the INDRAMAT FLASHTOOL "Dolfi". This program is required for transferring the configured data into the small operator input units. Another configuration of this software is usually shown in the hardware documentation of the BTV04/05/06 or BTC06 unit.

The installation of the ScreenManager software is performed in the second part of the setup.

4.2 Important Notes for the Users of Version 1

The compiled programs of version 1 execute under the runtime versions 2 and 3. Since the virtual machines have been enhanced in many points, compiled programs of version 3 require at least a runtime version of version 3; they are not executable under an older runtime version. A corresponding note is issued on the control panel.

Note: Both versions of the ScreenManager can be installed in parallel on the same computer. Leaving version 1 in the computer is therefore recommended.

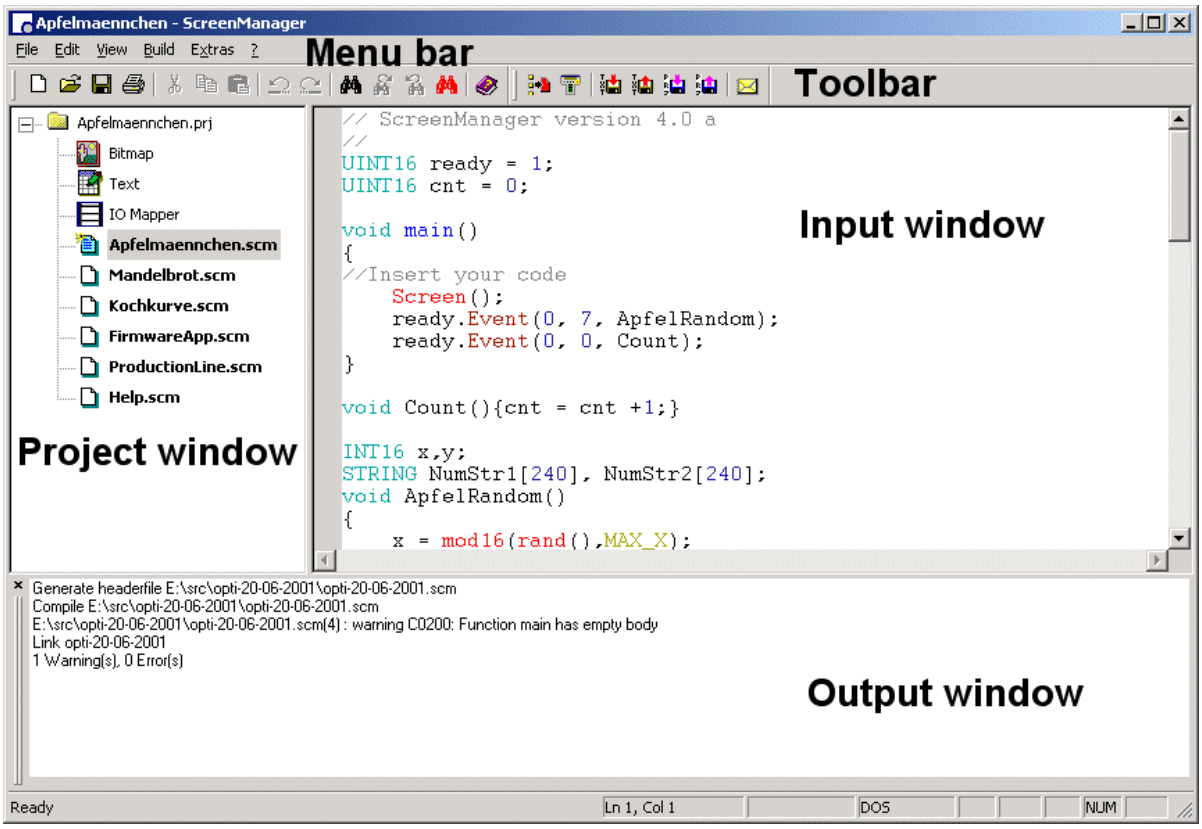
A project that ware created using version 1 can always be opened and compiled using ScreenManager version 3. As long as the new syntax features are not used, the reverse way functions too. All settings from the project file are accepted. Errors are not to be anticipated if programming is synthetically correct. With syntax errors, the version 1 compiler was less stringent than the version 3 compiler. Standard errors that are unveiled when old projects are ported include: Forgotten semicolons and parentheses, incorrect constant declarations, and missing variable declarations.

Note: Only values with decimal point are accepted as floating point constants (FLOAT). Examples 0.0 3.1415 1.7e-23

5 ScreenManager – Development Environment

5.1 Program Structure

The structure of the program ScreenManager is illustrated in the following figure:



SCM4_Anwendungsfenster_.bmp

Fig. 5-1: ScreenManager 4

The menu bar provides, as usual in Windows programs, most of the function blocks of the ScreenManager.

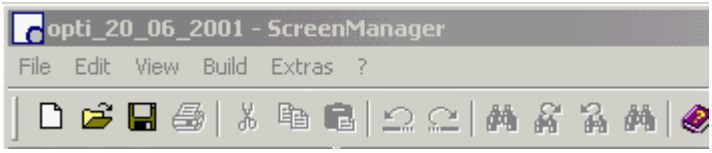
The toolbar affords the opportunity to activate frequently needed functionalities directly. The toolbar symbols are linked with selected menu items and provide therefore a faster access to this menu items. The corresponding toolbar symbols are inserted at the respective left side of the menu item, so that it is possible to recognize the correlation between toolbar and menu.

The project window allows to navigate in your current project and gives a survey of the involved files. Within the project window you can add files to the project or also remove them from the project. Use the project window to select the files to be edited.

The input window allows to edit your source files, to define the project properties, to determine and modify the resources as well as to specify the IO mapper file.

The output window indicates the messages while creating and downloading the project. Furthermore, it is possible to reach errors directly in the source text.

5.2 Main Menu Bar



SCM4_Hauptmenu.bmp

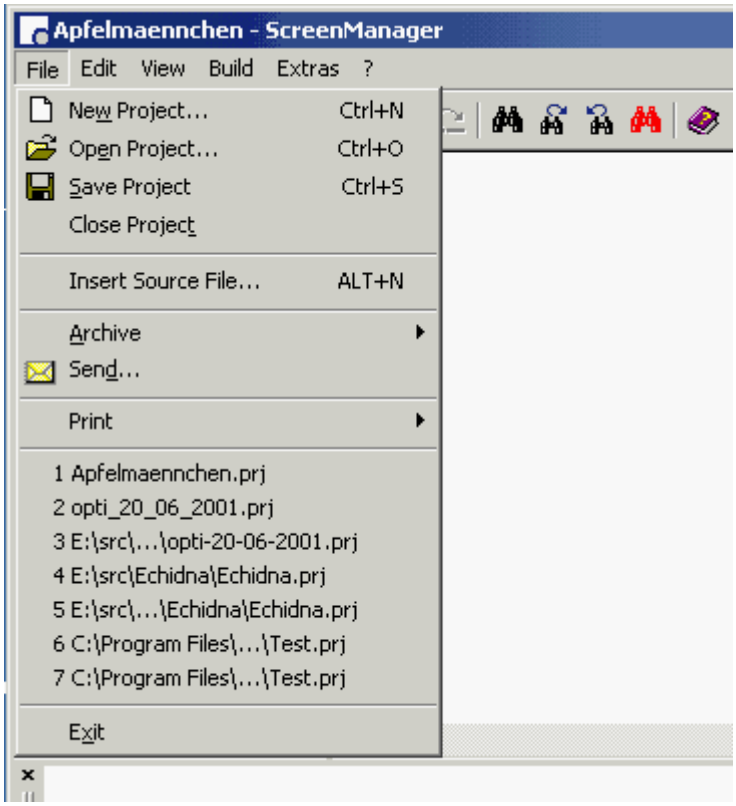
Fig. 5-2: Main menu

The ScreenManager menu bar shows - similar to all other Windows programs - the following menu items:

- **File**
- **Edit**
- **View**
- **Build**
- **Extras**
- **? Help**

The individual menu items have several submenu items, which are indicated in gray, when they are inactive, e.g. if they are not useful for the moment or are not relevant for the user.

5.3 File



SCM4_Dateimenu.bmp

Fig. 5-3: Menu "File"

The menu **"File"** combines all file-related operations.

It covers the following command groups:

- **New Project**: newcreating of a ScreenManager project / **Open Project**: open an existing ScreenManager project / **Close Project**: close a ScreenManager project
- **Save Project**: Save the whole project including all associated files
- **Insert Source File** to insert source files in the ScreenManager project
- **Archive** with export / import of the ScreenManager project in packed file format or export / import of the text resources as Excel-compatible text file.
- **Send** the current project via e-mail
- **Print** the current source text and printer settings
- **Exit** of ScreenManager

New Project

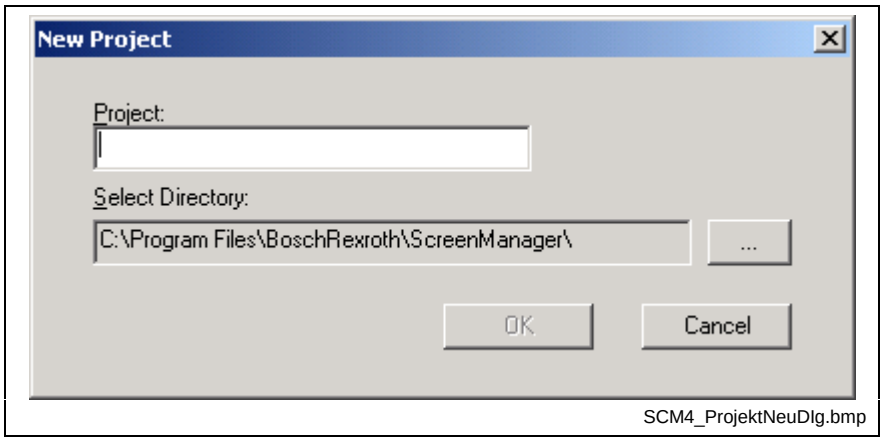


Fig. 5-4: Menu item "File / New Project" – Build a new project

The menu item **"File / New Project" (<Ctrl>+<N>)** allows to build new ScreenManager projects. If this menu item is selected, the dialogue **"New Project"** is activated. You are asked to enter the name and the target folder of the new project. Fill in the project name in the input field **"Project"**. Then, select the target folder in the input field **"Select Directory"** which is called by activating the ...-button.

Open Project

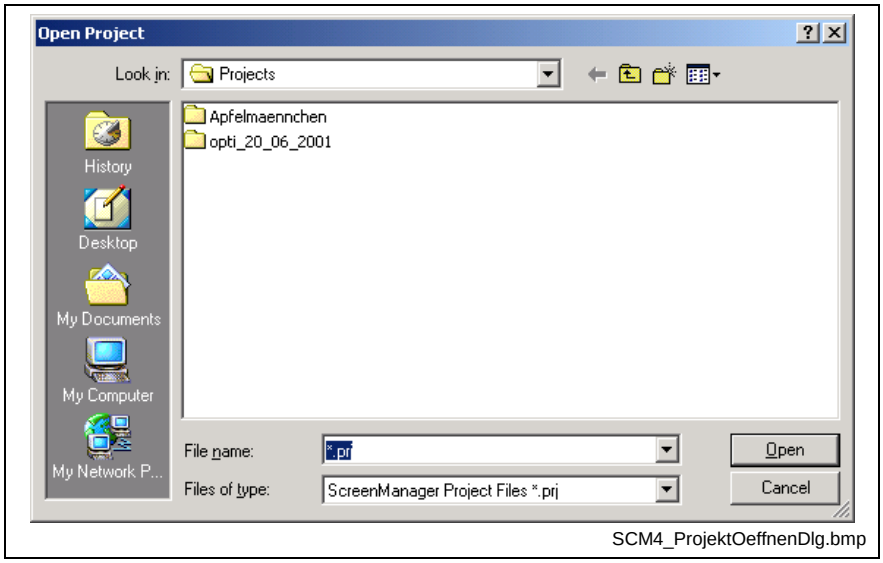


Fig. 5-5: Menu item "File / Open Project"

Menu item **"File / Open Project" (<Crtl>+<O>)** activates the dialogue "Open Project". You are asked to select an already existing ScreenManager project that you intend to edit.

Save Project

By using the menu item **Save Project (<Ctrl>+<S>)** all files belonging to the project are stored if the project has changed since the last storage.

A project which has not been stored since the last modification is marked with a "*****" behind the file name in the window header.

Close Project

The existing project is closed. In case of modifications you are asked if you want to save them.

Insert Source File

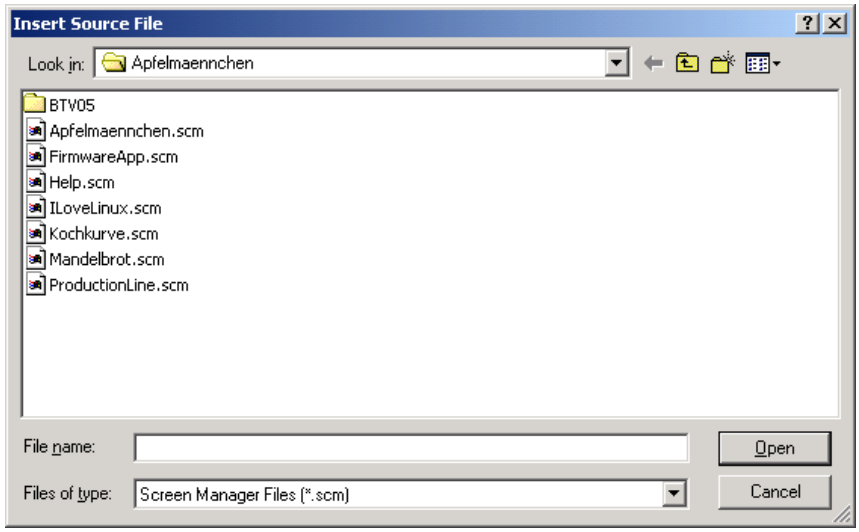


Fig. 5-6: Menu item "Insert Source File"

After the selection of menu item **"Insert Source File" (<ALT>+<N>)** you are asked to enter a SCM source file. If the source file already exists, it is integrated in the current project. If the file does not exist yet, a new source file is created and integrated in the project.

Archive

The programming system provides not only an archive functionality for the whole ScreenManager project but also the possibility to import and export the text resources.

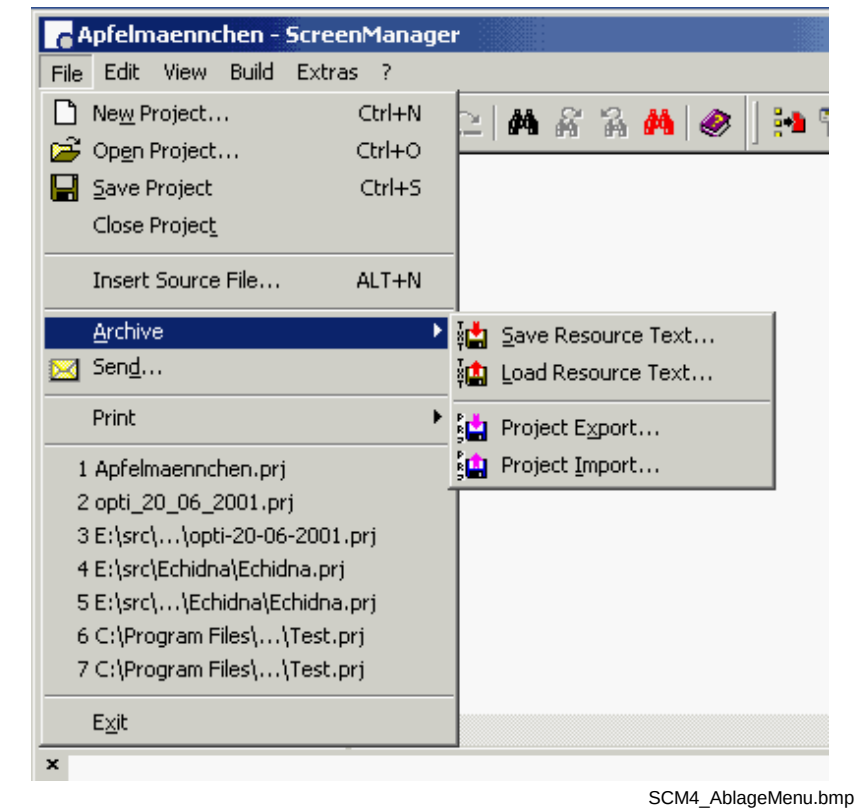
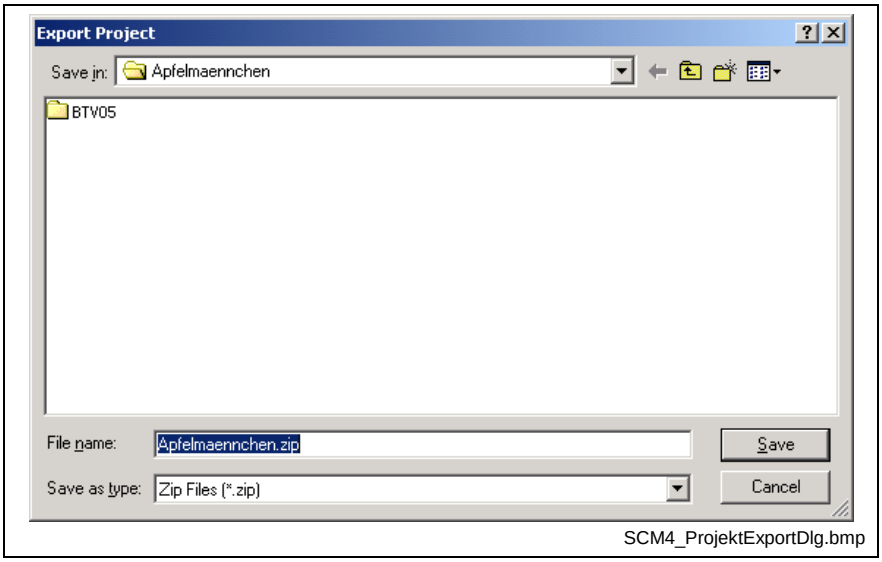


Fig. 5-7: Menu item "Archive"

Project Export

All files of the project including the used image resources are combined and archived in a packed file format (ZIP-compatible). You have to enter the name of the project archive in the file dialogue.

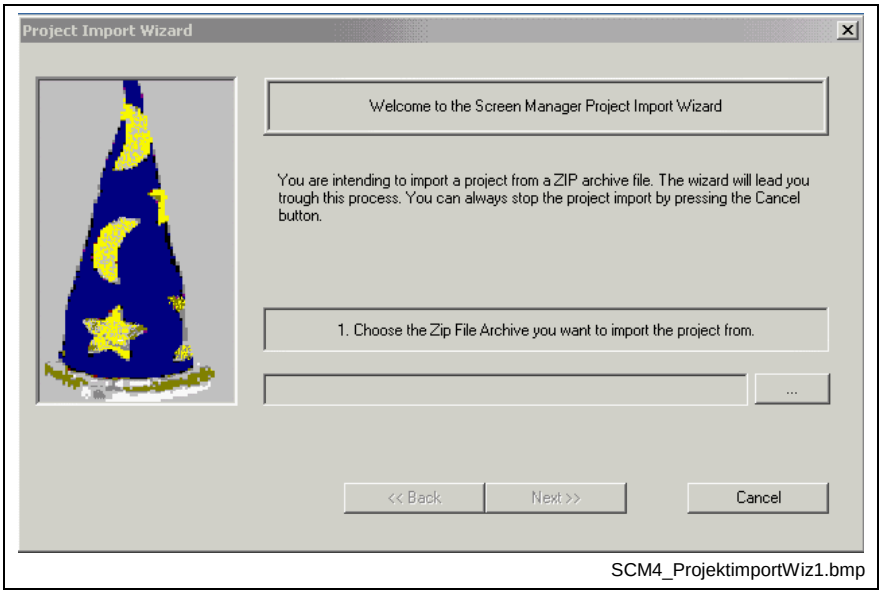


SCM4_ProjektExportDlg.bmp

Fig. 5-8: Menu item "Project Export"

Project Import

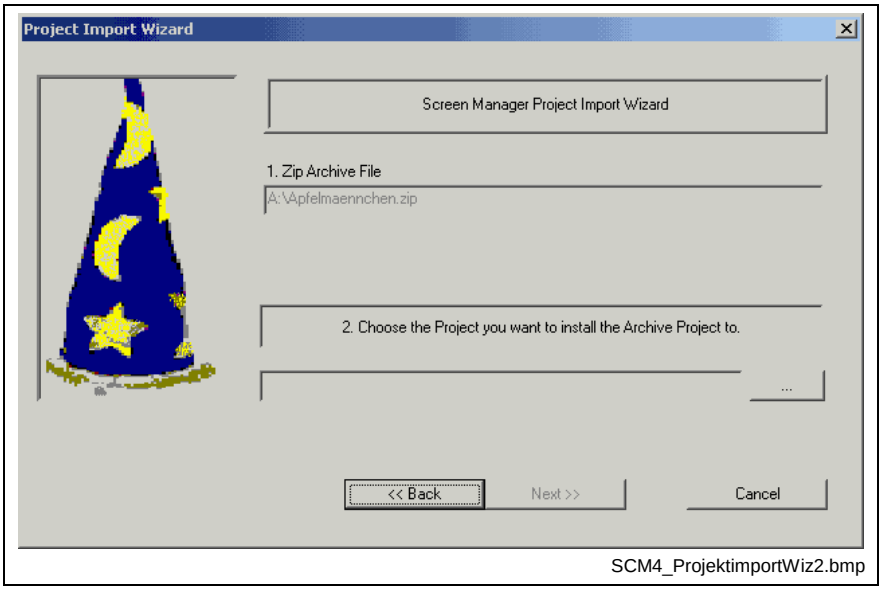
The "Project Import Wizard" facilitates the import of projects. The project import takes place in several steps. Each step is supported by an user dialogue which also stores the preceding steps. With the button "Back" you can change at any time to the previous step.



SCM4_ProjektimportWiz1.bmp

Fig. 5-9: Project Import – First step

The first step is to enter the archive file from which the project shall be imported. For this purpose, activate the ...-button. A file selection dialogue is opened. Select the desired project archive (ZIP-file).



SCM4_ProjektimportWiz2.bmp

Fig. 5-10: Project Import – second step

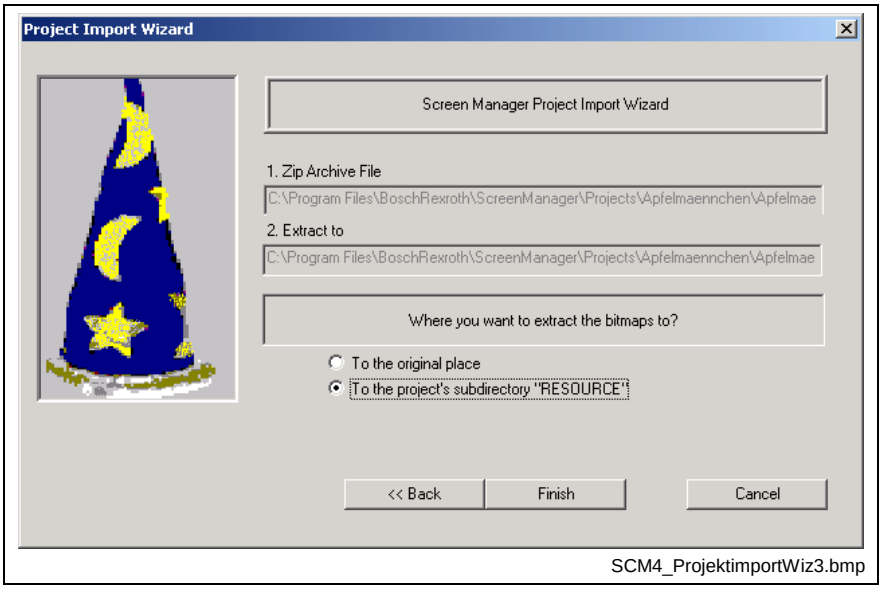
The second step is to enter the target directory of the project import. Furthermore, you have the possibility to change the name of the project.

Note: It is also possible to use the functionality "Project Import" to save existing projects in another directory structure and/or to change the name of the project. To do so, export the project and select another directory name or another target directory during the import.

Activate the  -button and enter a project name and a target directory in the appearing dialogue. Proceed as if you build a **New Project** <Ctrl>+<N>.

The names stored in the archive as standard are given as project name and directory.

Note: The storage procedure is **not** able to save drive information, i. e. drive letters are lost during storage. The import drive is the current drive.



SCM4_ProjektimportWiz3.bmp

Fig. 5-11: Project Import – third step

Select if the bitmap image resources shall be imported in one of the directories of the archive (decentralized archive), or if you want to import all bitmaps in a separate directory within your project directory.

Note: In the case you import the project from another computer you should select the option **"To the project's subdirectory RESOURCE"**.

Send

The menu item **"Send"** allows to send the current project via e-mail. A file dialogue is opened. You are asked to enter a file name. An archive with this file name is created (see **Project Export**). This archive is integrated as attachment with your standard e-mail program. You have to enter the address of the recipient and you can add a message.

Print

Print <Ctrl>+<P>

Only the printing of source files is supported.

The printing process itself is initiated with the window as for all Windows programs. You are asked to select the target printer and you can change the properties of the printer.

Print Preview

The print preview allows to verify the number and formatting of the pages to be printed.

SCM4_DruckvorschauDlg.bmp

Fig. 5-12: Print Preview

Page Setup

As for all Windows programs printer-dependent possibilities to set-up printed pages are offered under this menu item.

Note: Format restrictions are given by printer or printer driver.

Print Setup

The printer settings can be changed with the device-specific dialogue.

Exit

This menu item closes the program. Open and changed files are stored after confirmation of a security prompt.

5.4 Edit

The menu item **"Edit"** comprises the group **"Undo"**, the group of block commands and the group **"Find / Replace"**.

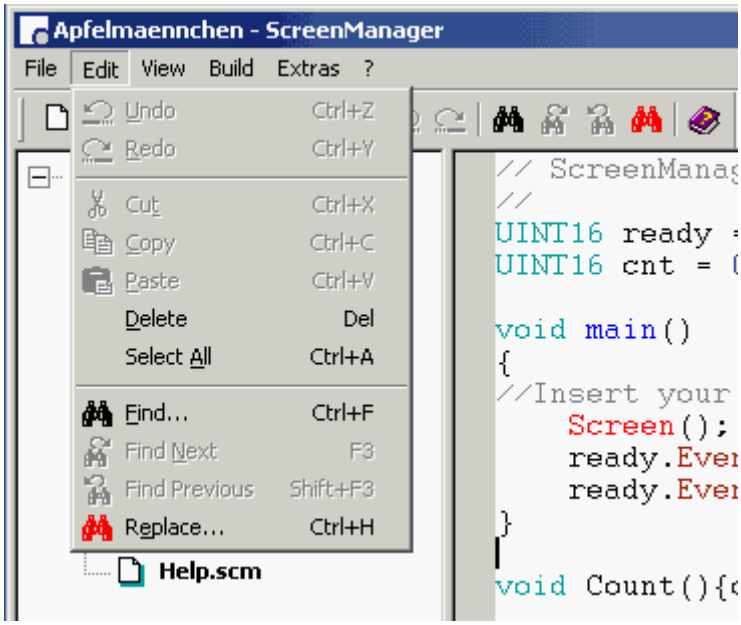


Fig. 5-13: Menu "Edit"

Group: **Undo**:

- Undo <Ctrl>+<Z>
- Redo <Ctrl>+<Y>

Group: **block commands**

- Cut <Ctrl>+<X>
- Copy <Ctrl>+<C>
- Paste <Ctrl>+<V>
- Delete <Del>
- Select All <Ctrl>+<A>

Group: **Find / Replace**

- Find <Ctrl>+<F>
- Find Next <F3>
- Find Previous <Shift>+<F3>
- Replace <Ctrl>+<H>

Undo <Ctrl>+<Z>

With this command, you can cancel accidentally made changes in the source text editor.

Redo <Ctrl>+<Y>

Changes of the menu item **Undo** <Ctrl>+<Z> are restored.

Cut <Ctrl>+<X>

is a standard Windows command. With this command, the marked text passage / block is removed and stored in the clipboard (see also **Copy** <Ctrl>+<C>).

Copy <Ctrl>+<C>

is a standard Windows command. The marked text passage / block is stored in the clipboard and is retained for the copy function (see also **Cut** <Ctrl>+<X>).

Paste <Ctrl>+<V>

is a standard Windows command. The text which was stored in the clipboard by the **Copy** <Ctrl>+<C> command, is taken over into the current editor.

Delete <Del>

is a standard Windows command to delete the marked text passage / block.

Select All <Ctrl>+<A>

is a standard Windows command to mark the whole file.

Find <Ctrl>+<F>

The find function is a standard Windows command. Enter the text to be searched for in the **"Find what"** field and click on button **"Find next"**. The cursor stops on the search criterion.

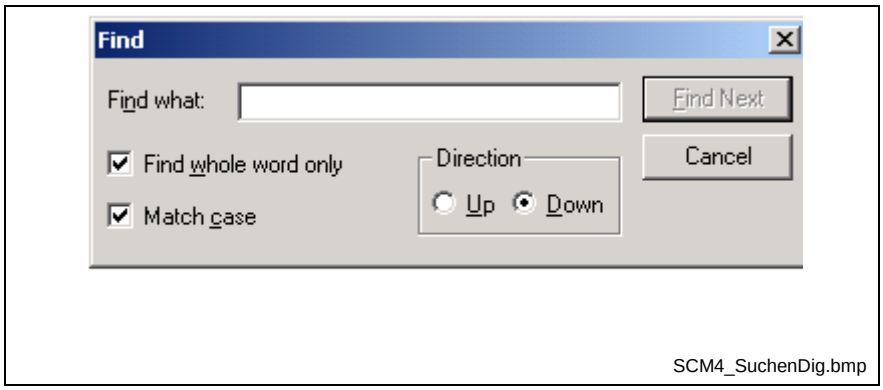


Fig. 5-14: Menu item "Find"

The search function can be restricted to

- **Find whole word only** and
- **Match case**.

Furthermore, the search direction can be defined.

A started search process can be continued with button **Find Next** <F3>.

Find Next <F3>

The standard Windows command continues the earlier begun search for a given text in direction document end.

Find Previous <Shift>+<F3>

The standard Windows command continues the earlier begun search for a given text in direction document beginning.

Replace <Ctrl>+<H>

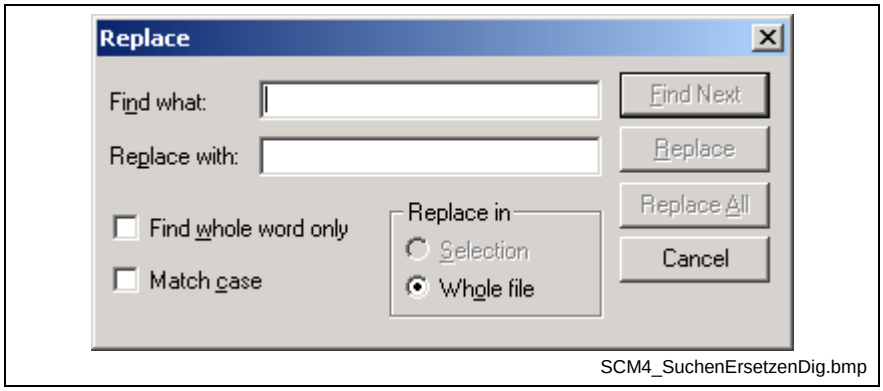


Fig. 5-15: Menu item "Find what / Replace with"

The standard Windows command searches for a given term. The cursor stops on the search criterion. The found term is replaced when you press the button **"Replace"**. With the button **"Replace All"** the searched term is automatically replaced at any point where it occurs.

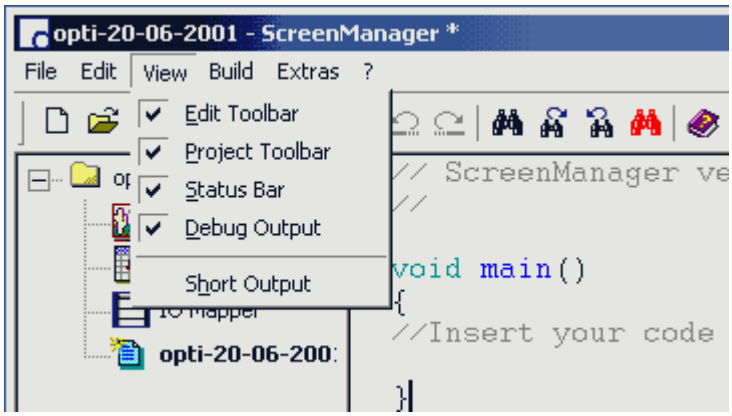
The search function can be restricted to

- **Find whole word only** and
- **Match case**.

The area to be searched through can be defined; either

- search in **"Whole file"** or
- only in the marked block with the option **"Selection"**.

5.5 View



SCM4_AnsichtMenu.bmp

Fig. 5-16: Menu "View"

The menu **"View"** allows to activate / deactivate certain operating elements of the ScreenManager. Click on the respective menu item to activate / deactivate it. An operating element is active when the menu item is provided with a hitch.

Edit Toolbar

The Edit Toolbar contains operating elements to administrate the source texts.

Project Toolbar

Archive functionality and build / download of projects.

Status Bar

Status bar.

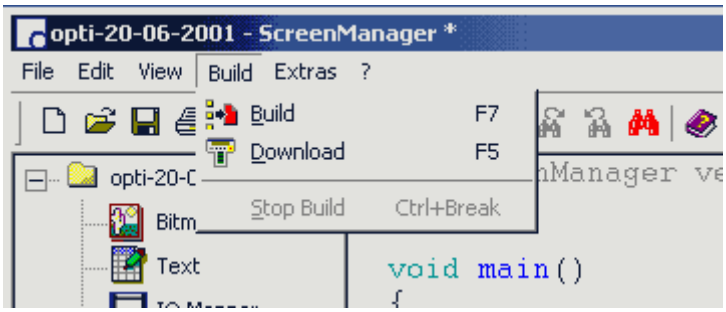
Debug Output

Output messages of the compiler, the linker and the download program while building a project.

Short Output

Error messages are exclusively indicated in the output window.

5.6 Build



SCM4_MenuErstellen.bmp

Fig. 5-17: Menu "Build"

Build

The project is compiled and linked. But it is ***not*** transferred to the miniature control panel.

Note: Use this function to verify the syntactical correctness of your project.

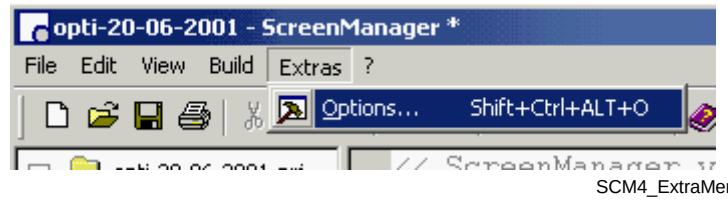
Download

The project is built and transferred to the operating device with the download program `DOLFI`.

Stop Build

The creation of the project is stopped.

5.7 Extras



SCM4_ExtraMenu.bmp

Fig. 5-18: Menu "Extras"

The basic settings of the ScreenManager are administrated with menu **"Extras"**.

Options

The dialogue window allows to define project standard settings. While creating a new project these settings are automatically entered in the project file. However, you can change these settings project-specific for every project (see **Project Properties**).

The setting **"Target Device"** indicates the miniature control panel for that the project shall be build. The button Mtc/Isp determines if a *.btv-file necessary for the PLC communication is build.

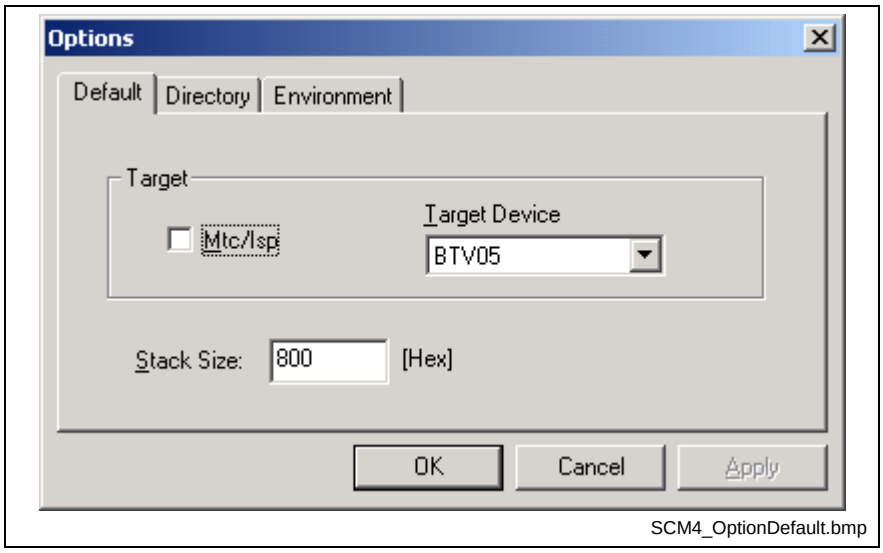


Fig. 5-19: Default settings

The option **"Directory"** defines the path for the DOLFI program and the *.btv-file (necessary for the MTS connection), which must fit to the existing MT-CNC installation.

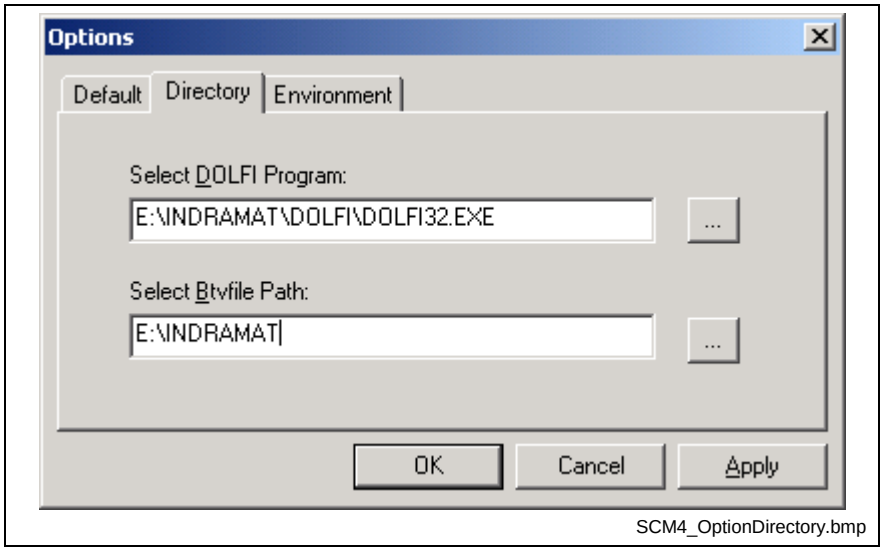


Fig. 5-20: Directory settings

To select another path press the ...-button.

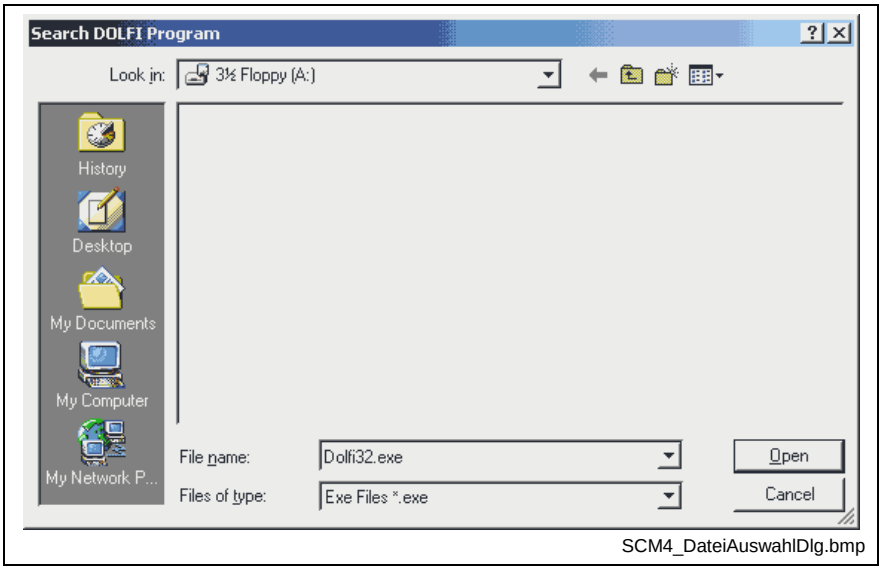


Fig. 5-21: Environment

The option **"Environment"** allows to switch over to the expert mode.

5.8 ? Help

Fig. 5-22: Menu "? Help"

Help <F1>

The help is opened with F1.

About ScreenManager

This submenu item contains detailed information about the version with date and time; this allows unequivocal identification of the used version.

F Keys and their Alt / Ctrl / shift Combinations

	Key	<Shift>+Key	<Ctrl>+Key	<Alt>+Key
F1	General help on cursor position			
F3	Repeat search	Repeat search		
F4			Close the active window	Close the programming system; prompt if changed files shall be stored
F5	Build and download the current project			
F6			Switch over to the next window	
F7	Build the current project			
F10		Go to local menu (PopUp menu)		

Fig. 5-23: List with F key combinations

Key Combinations

<Alt>

Goes to the first menu item (file) of the main menu.

<Ctrl>+<A>

Marks the whole file content.

<Ctrl>+<C>

Block command: Copies the marked blocks in the Windows clipboard.

<Ctrl>+<F>

Searches for a character string.

<Ctrl>+<H>

Replaces a character string by another string.

<Ctrl>+<N>

Builds a new project.

<ALT>+<N>

Inserts a new source file.

<Ctrl>+<O>

Opens an existing project.

<Ctrl>+<P>

Prints the editor contents.

<Ctrl>+<S>

Saves the current project.

<Ctrl>+<V>

Block command: Inserts a block from the Windows clipboard.

<Ctrl>+<X>

Block command: Cuts out the marked block and saves it into the Windows clipboard.

<Ctrl>+<Y>

Cancels accidentally made changes.

<Ctrl>+<Z>

Cancels an editor action.

<Del>

Deletes the marked block or a source file out of the project.

<Ctrl>+<Shift>+<Alt>+<O>

Changes the standard settings of the projects.

5.9 The Project Window

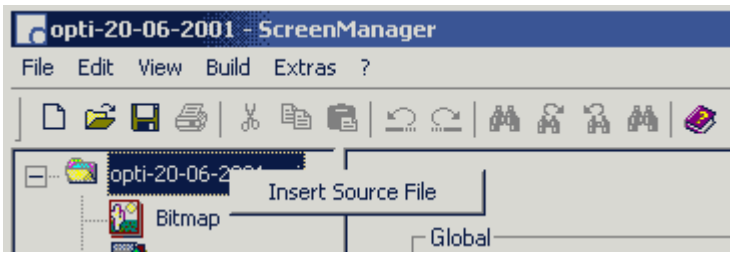
Fig. 5-24: Project window

The project window determines the active data structure, that can be modified in the input window. The following data structure can appear:

- **Project Properties**
- **Resource Editor** – the Bitmap Window
- **Resource Editor** – the Text Window
- **IO Mapper** Window
- **Source Text Editor**

You can move within the project tree with the help of the cursor keys. Thereby, the currently selected data structure is indicated in the editor window. Press the **<ENTER> key** to change to the editor window.

Insert Source File

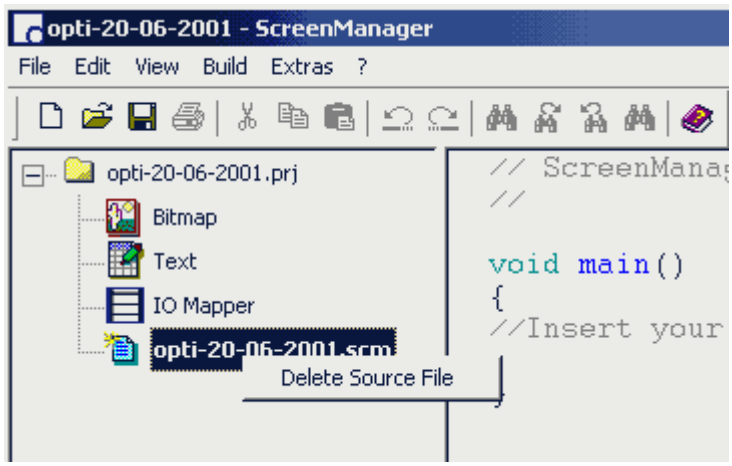


SCM4_ProjektMnuDateiEinfügen.bmp

Fig. 5-25: Insert source file in project

Select the project properties in the project window (root knot in the tree illustration) and open the context menu. Select **"Insert Source File"**. A file dialogue opens which allows to select a source file. This source file is integrated in the project.

Delete Source File



SCM4_ProjektMnuDateiLöschen.bmp

Fig. 5-26: Delete source file from project

Activate the source file to be deleted and open the context menu. Select **"Delete Source File"**. The file is deleted after security prompt.

Note: The file is not deleted on the storage medium, so that the file can be integrated in the project.

5.10 The Input Window

Project Properties

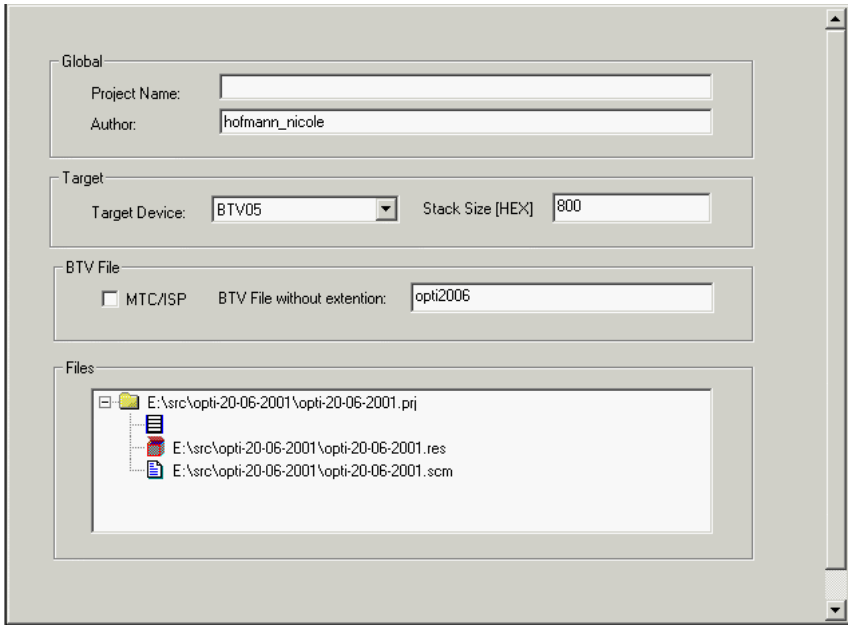


Fig. 5-27: Project properties

With the project properties window you determine the basic properties of the project.

- **General information:** Enter project name and author of the project.
- **Target device:** Select the desired target device from the list and determine the required **stack seize**. Enter the stack seize as hexadecimal number.
- **BTV-file:** Enter the file name, under which your project shall be available on the miniature control panel (BTV). The file name must not increase eight characters. Select **MTC/ISP** if you need the communication with the PLC.
- **Files:** The tree structure gives an overview, which files are integrated in a project.

Resource Editor – General Operation

The resource editors to administrate multilingual texts and bitmaps are similar structured. With the resource editor non-ambiguous identifiers are allocated to the corresponding resources – either text or bitmap. The editor is organized in tabular form. The columns are allocated to the respective languages, the lines administrate the identifiers.

Insert, Delete and Rename Languages

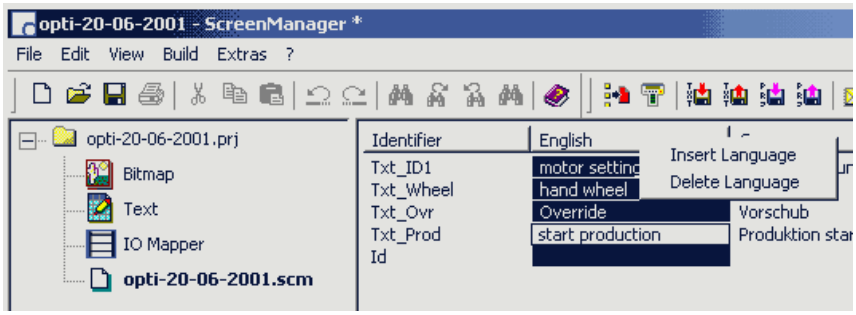


Fig. 5-28: Insert Language, Delete Language

Click with the right mouse button on a language in the table header. A context menu opens which allows to select **"Delete Language"** or **"Insert Language"**. The selected language column is marked.

Delete Language
Select menu item **Delete Language**. After confirming the safety prompt, the language is deleted.

Note: If you delete one language all entries of this language – either text or bitmaps – are deleted.

Insert Language
Select menu item **Insert Language**. Right beside the selected language an additional language is inserted. At first, the new created language column receives the designation **Generic LanguageX.**

Rename Language
Click with the left mouse button on the language to be renamed.

Fig. 5-29: Rename Language

A dialogue opens which allows to enter the new language designation.

The Editing Field

The resource editors contain a framed input field. If the editing field is indicated in gray the editor is in the display mode. If the editing field is indicated in white the editor is in the editing mode. The two modes offer different actions.

Display Mode

Press

- **<Enter>** to change to the editing mode.
- **<←>,<↑>,<→>,<↓>** to move the input field.
- **<TAB>** to move the input field further to the right; if the input field is situated at the end of the line, it is moved to the next line.
- **<Shift+<TAB>** to move the input field further to the left; if the input field is situated at the beginning of a line, it is moved to the preceding line.
- **<Page Up>** to move the input field to the first line.
- **<Page Down>** to move the input field to the last line.

Editing Mode

Press

- **<ESC>** to change to the display mode. Inputs are deleted.
- **<Enter>** to change to the display mode. Input is stored.
- **<←><→>** to move within the input field.
- **<TAB>** to move the input field further to the right; the input is stored; if the input field is situated at the end of the line, the input field is moved the next line.
- **<Shift>+<TAB>** to move the input field further to the left; the input is stored.

Resource Editor – the Bitmap Window

The bitmap window is your input template for the administration of multilingual bitmaps. You can allocate bitmaps to non-ambiguous identifiers. The bitmap, whose language is just active, is indicated in the program. This means that you can change the bitmaps at run time. Therefore, every bitmap is allocated to a language and a non-ambiguous identifier.

The bitmap administration is structured in tabular form. Every existing language is allocated to a column. The lines are allocated to the IDs.

At the bottom of the bitmap window the selected bitmap is illustrated.

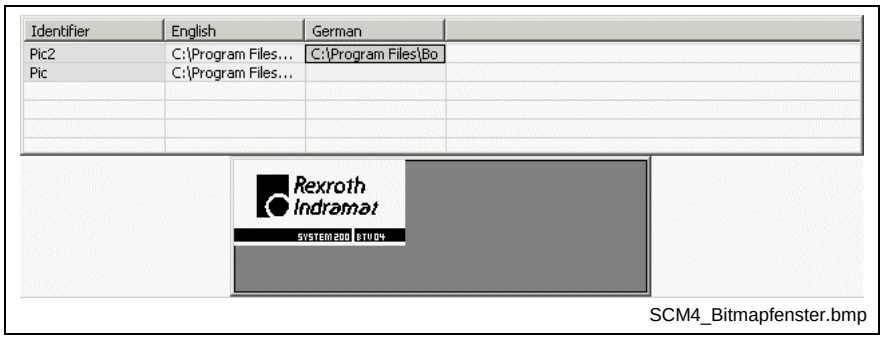


Fig. 5-30: Resource editor – the bitmap window

Insert Identifiers and Bitmaps

Click on the unused white part of the table or move the input field in input mode behind the last cell (**<Tab>** key). In the column **Identifier** a new editing line is opened.

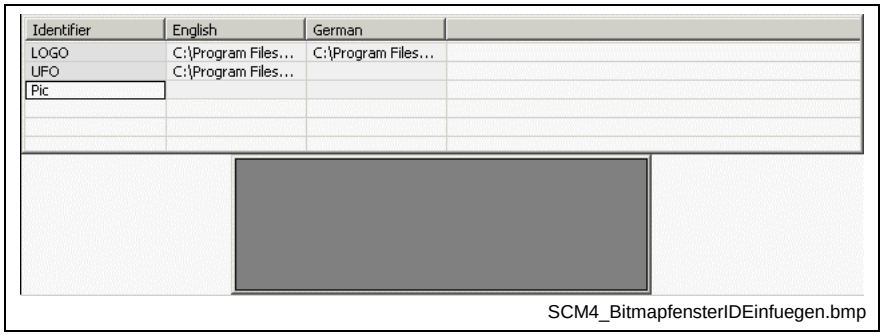
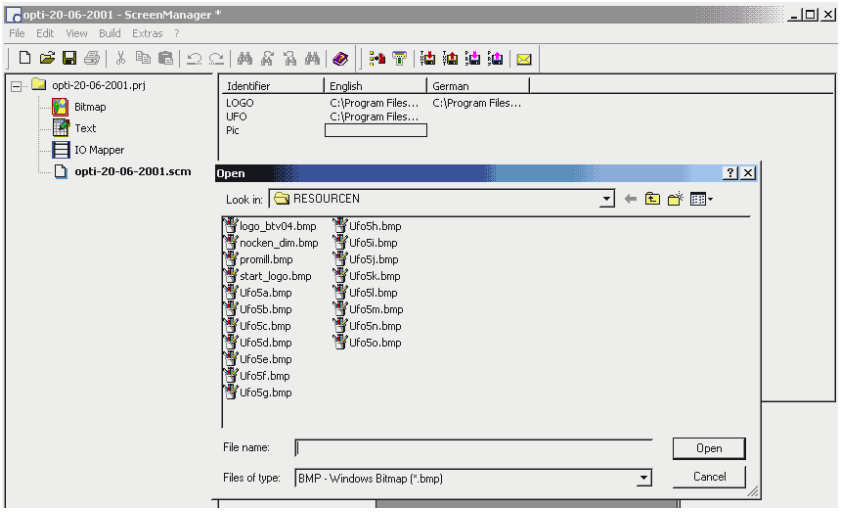


Fig. 5-31: Insert identifier

Replace the entry **Pic** with your identifier. Your entry is accepted by pressing **<Enter>**.

Note: Use non-ambiguous identifiers. If an identifier is entered twice, the preceding entry is overwritten. It is possible that references to bitmaps get lost.

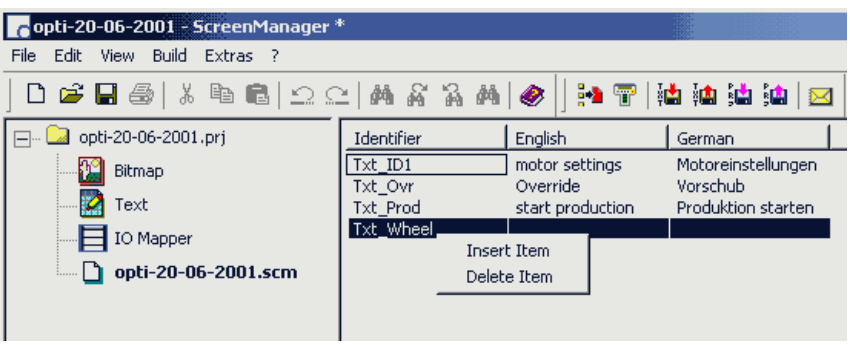
Move the gray marked input field with the cursor keys on the language position with which you want to link a bitmap. Confirm with **<Enter>**. A dialogue is opened to select your bitmap.



SCM4_BimapfensterBmpEinfuegen.bmp

Fig. 5-32: Insert bitmap

Delete Identifiers



SCM4_RessourceIDLöschen.bmp

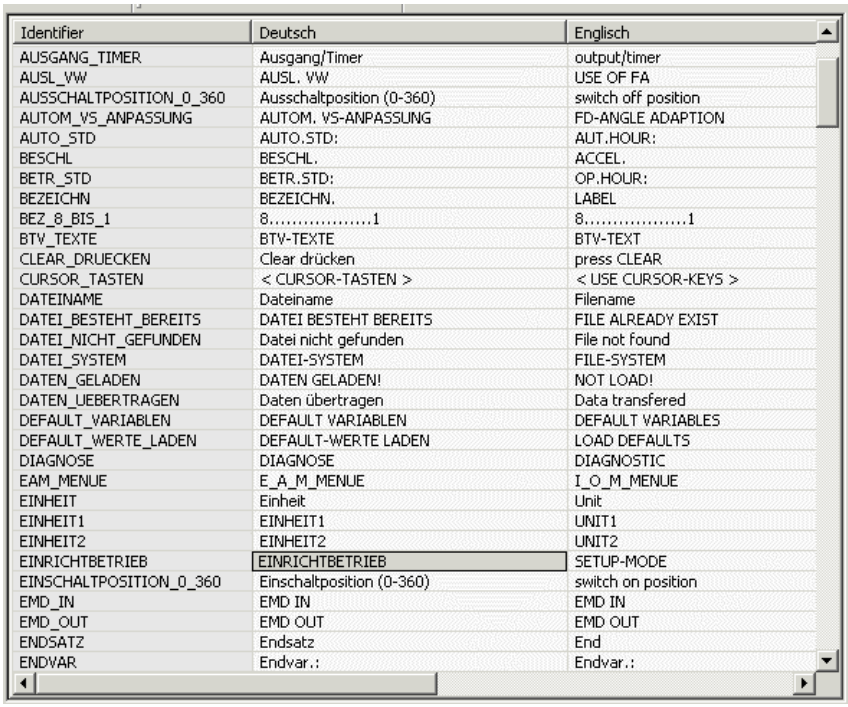
Fig. 5-33: Delete Item

Click with the right mouse button on the identifier that you intend to delete. Consider that the input field is not in the column **Identifier.** A context menu opens. Select **Delete Item**.

Resource Editor – the Text Window

The text window is your input template to administrate multilingual texts. You can allocate texts to non-ambiguous identifiers. The text, whose language is just active, is indicated in the program. This means that you can change texts at run time. Therefore, a language and a non-ambiguous identifier is allocated to every text.

The text administration is structured in tabular form. Every existing language is allocated to a column. The lines are allocated to the IDs.



Identifier	Deutsch	Englisch
AUSGANG_TIMER	Ausgang/Timer	output/timer
AUSL_VW	AUSL. VW	USE OF FA
AUSSCHALTPOSITION_0_360	Ausschaltposition (0-360)	switch off position
AUTOM_VS_ANPASSUNG	AUTOM. VS-ANPASSUNG	FD-ANGLE ADAPTION
AUTO_STD	AUTO.STD:	AUT.HOUR:
BESCHL	BESCHL.	ACCEL.
BETR_STD	BETR.STD:	OP.HOUR:
BEZEICHN	BEZEICHN.	LABEL
BEZ_8_BIS_1	8.....1	8.....1
BTV_TEXTE	BTV-TEXTE	BTV-TEXT
CLEAR_DRUECKEN	Clear drücken	press CLEAR
CURSOR_TASTEN	< CURSOR-TASTEN >	< USE CURSOR-KEYS >
DATEINAME	Dateiname	Filename
DATEI_BESTEHT_BEREITS	DATEI BESTEHT BEREITS	FILE ALREADY EXIST
DATEI_NICHT_GEFUNDEN	Datei nicht gefunden	File not found
DATEI_SYSTEM	DATEI-SYSTEM	FILE-SYSTEM
DATEN_GELADEN	DATEN GELADEN!	NOT LOAD!
DATEN_UEBERTRAGEN	Daten übertragen	Data transfered
DEFAULT_VARIABLEN	DEFAULT VARIABLEN	DEFAULT VARIABLES
DEFAULT_WERTE_LADEN	DEFAULT-WERTE LADEN	LOAD DEFAULTS
DIAGNOSE	DIAGNOSE	DIAGNOSTIC
EAM_MENUE	E_A_M_MENUE	I_O_M_MENUE
EINHEIT	Einheit	Unit
EINHEIT1	EINHEIT1	UNIT1
EINHEIT2	EINHEIT2	UNIT2
EINRICHTBETRIEB	EINRICHTBETRIEB	SETUP-MODE
EINSCHALTPOSITION_0_360	Einschaltposition (0-360)	switch on position
EMD_IN	EMD IN	EMD IN
EMD_OUT	EMD OUT	EMD OUT
ENDSATZ	Endsatz	End
ENDVAR	Endvar.:	Endvar.:

SCM4_Textfenster.bmp

Fig. 5-34: Text window

Insert Identifiers and Texts

Click on the unused white part of the table or move the input filed in input mode behind the last cell (**<TAB>** key). In the column **Identifier** a new editing line is opened.

SCM4_TextfensterIDEinfügen.bmp

Fig. 5-35: Insert identifier

Replace the entry **Id** with your identifier. Your input is accepted by pressing **<Enter>**.

Note: Use non-ambiguous identifiers. If an identifier is entered twice, the preceding entry is overwritten. It is possible that references to bitmaps get lost.

Move the gray marked input field with the cursor keys on the language position where you want to enter a text. Confirm with **<Enter>**. The input field changes to the edit mode and you can modify the text. Confirm by pressing the **<Enter>** or **<TAB>** key.

Delete Identifiers

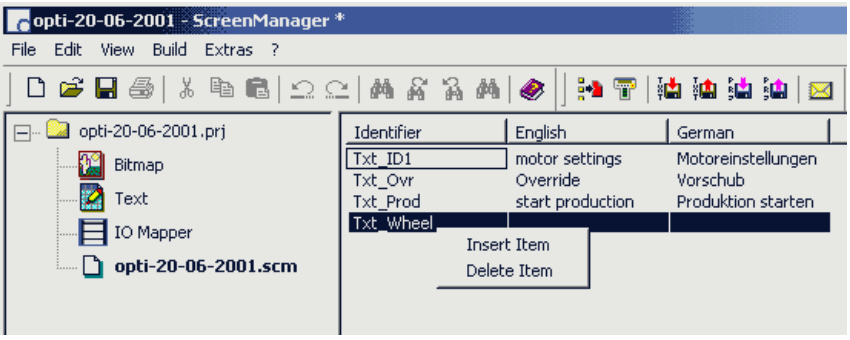


Fig. 5-36: Delete Item

Click with the right mouse button on the identifier that you want to delete. Consider that the input field is not in the column **Identifier**. The context menu opens. Select **Delete Item**.

IO Mapper Window

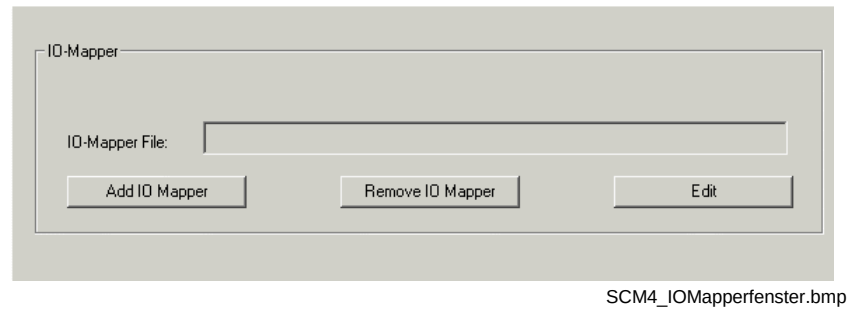


Fig. 5-37: IO Mapper window

Insert an IO mapper file in your project in which you press the button **Add IO Mapper**. You are asked via a file dialogue to enter a file name. You can delete the entered file with button **Delete IO Mapper**.

By activating the button **Edit IO Mapper** the external program `IOMapper` is called.

Source Text Editor

```
// SCM Version 3
//#define text_zeile5_0 "HILFE-MENUE"
//#define text_zeile5_1 "Warten auf VW"
//#define text_zeile5_2 "Warten auf Presse"
//#define text_zeile5_3 "Warten WKZ Quittung"
//#define text_zeile5_leer "Keine Meldung"

UINT16 bbb;
STRING ss[4] = "0000";
STRING strStatus63[50]="STATUS 63";

extern STRING streamZeile;
extern STRING streamZeile2;

extern STRING eam;
extern STRING typ;
extern STRING nummer;
extern STRING eam2;
extern STRING typ2;
extern STRING nummer2;


UINT16 ui16_zeile1=0;
UINT16 ui16_zeile2=0;
UINT16 ui16_zeile3=0;
UINT16 ui16_zeile4=0;
UINT16 ui16_zeile5=0;

extern UINT16 usNcTask 1;          // Variable NC-Linenumber
```

SCM4_Quelltextfenster.bmp

Fig. 5-38: Source text window

The source text editor allows to create and edit your SCM source files. Thereby, the key words are automatically highlighted.

The input of the source text is executed in the same way as for other text editors (e.g. `Notepad`).

Special Keys

With the editor you can set up to ten bookmarks.

- **<Ctrl>+digit** Set bookmark no. digit
- **<ALT>+digit** Go to bookmark no. digit

5.11 The Output Window

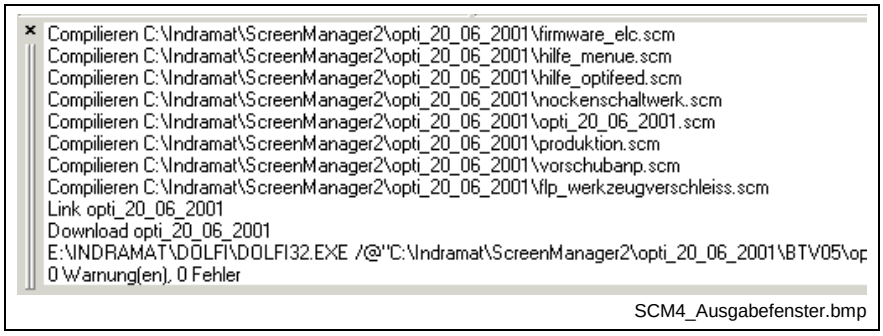


Fig. 5-39: Output window

The output window shows the actions of the programming environment. Furthermore, errors during the creation of a project are indicated. With double click on the corresponding error the respective file is loaded in the source text editor and the cursor is moved to the error line.

6 Brief Introduction into Programming

6.1 Is this Really C?

Brief Introduction

The ScreenManager programming language is based on ANSI-C. However, the syntax required for creating simple screens is limited to a minimum. Having passed a C training course is not necessary. Studying the Kernighan/Ritchie (the Reference Manual for C Programmers: Brian W. Kernighan and Dennis M. Ritchie – C Programming Language, 2nd Edition, ISBN: 0131103709) is not necessary either. However, it is recommended as a reference manual.

You usually start with a screen example which you copy and modify to create your own screens.

The following chapter superficially discusses all language elements – more detailed descriptions can be found in the following chapters.

Variable Declaration, Variable Types

All variables must be declared before they are used. This is possible either outside of the functions at any location in the text. This is how global variables are produced. A global variable may be used at any point in the program. They are the sole variables that are permitted for read, bind, edit, display and event commands. These functions will be explained in detail a few pages further down.

If you merely need a variable within a function to store intermediate results, for example, you may declare this variable locally at the beginning of the function. In this case, the variable is only valid within this function; the same name may be used for another local declaration in another function. Local variables only occupy memory as long as the function is processed.

Typical global variable declarations:

```
FLOAT  a = 0.0;          // Floating point number
STRING b[10] = "Hallo"   // Text variable of length 10
UINT16 c = 33;           // 16 bit unsigned integer number
INT32   d = 0xFFFF;        // 32 bit integer number initialized to
65535
```

Typical local variable declaration:

```
void MyTest()
{
INT16 i = 0;           // 16 bit integer number
...here follows the program code...
}
```

Note: In contrast to uninitialized global variables (which always contain the value 0 when the program is started), the start value of a local variable depends on the random actual state of the stack. To avoid unnecessary troubleshooting, local variables should therefore always be initialized.

Designation	Size in bytes	Range
Void	0	-
BOOL	1	TRUE, FALSE
INT8	1	-127...127
UINT8	1	0...255
INT16	2	-32768...32767
UINT16	2	0...65535
INT32	4	-2147483648...2147483647
UINT32	4	0...4294967295
FLOAT32	4	± 3.4e-38...3.4e38
FLOAT64	8	± 1.18e-308...3.4e308
STRING[1...254]	Defined length + 2	ASCII

Fig. 6-1: ScreenManager variable types

Calling Subfunctions

At any time, you may define your own functions that can be employed at any point be invoking the name. The control panel executes the function and returns to the point where the subfunction was invoked.

Example:

```
void ShowText()
{
  Text(0,23,110,8,"F2-Integer");
  Text(0,33,110,8,"F3-Global Float");
  Text(0,43,120,8,"F4-Global Integer");
  Text(160,99,0,0,"OVERRIDE:");
}

void MyScreen();
{
  Screen();
  TexteAnzeigen(); // TexteAnzeigen is invoked here
...
```

Parameters

A function can have parameter values assigned in the parentheses. Copies of these parameters are transferred to the function and may be used in it as local variables. A value returned by the function is a "function value". A function that shall not provide a value have the ‚void' type assigned.

Example:

```
UINT16 SumPlusOne(UINT16 a, UINT16 b)
{
  a = a +  + 1;
  return a;
}

void MyTest();
{
  x = SumPlusOne(y,5);
...
```

While the global variable y is not modified here, x has the value y + 6 assigned.

Arrays

One-dimensional arrays can be produced from any variable type - except from strings. The declaration of these arrays is as follows:

```
UINT32 myArray[10] = {1,2,3,4,5,6,7,8,9,10};
```

Initialization with start values is an option.

In contrast to ANSI-C, the possibilities of declaring variables stop at this point. There are no pointers (most common error source in C programs, that can virtually not be monitored during runtime), no structures and no multi-dimensional arrays.

An array can be transferred as a parameter. This permits indirect working with pointers:

```
void SetArray(UINT32 a[])
{
   UINT32 c = 0;
   for (c = 0; c < 5; c = c + 1)
   {
      a[c] = c + 32;
   }
}
```

At runtime, the range of an array is monitored by the runtime program. Any access to an area outside the defined boundaries would lead to a runtime error. With the example above this would happen if the SetArray function were invoked with an array of less than five elements.

Defines

Defines can be used for replacing frequently required expressions by a designator. Constants (such as static text) should not be placed within a program, but at its beginning. When they are employed expediently, defines contribute to transparency of the code – who knows by heart that 36862 is the decimal representation of the ident of the Sercos parameter P-0-4094? A define works like a text replacement. Prior to compilation, the specified designator is - invisibly to the programmer - replaced with the text behind it. In contrast to variables, defines do not require any RAM in the system since they are resolved before the compilation.

Examples:

```
#define P_0_4094 36862
#define TXT_D_1_SCHLEPP_MENUE     "POSITION LAG:"
#define TXT_D_2a_SCHLEPP_MENUE    "ABSOLUTE POSITION:"
#define KEY_PGUP     19
```

Expressions

Any assignment in the form of an equation is considered as an expression. The simplest case is a = 3; or, slightly more complicated, a = b * (2 * MeineFunktion(8*c, 3));

The invocation of an individual function is also considered as an expression: Screen();

Each expression ends with a semicolon; a line may contain several expressions.

With an assignment, only identical types are accepted at the two sides of the equal sign. If the types are different, the assigned type must explicitly be converted. The conversion follows ANSI C – the type designator is placed in brackets before the variable that is to be converted. Example:

```
UINT16 A;
UINT8 b = 7;
A = (UINT16) b + 8;
```

Or, when a float value is converted into an integer value – using the ScreenManager-specific conversion methods. The variable is followed by a point, the type designator in small letters is followed by ,()'. These methods are further-reaching than the C specification and allow, for example, any conversion between floating point and integer figures and BOOLEANs:

```
UINT16 A;
FLOAT32 b = 7.88;
A = b.unit16() + 8;
```

Structuring Language Elements

A program only becomes flexible if it is able to conditionally respond to external influences. This is done by conditional jumps. With the ScreenManager, the following constructs are responsible for it:

if()... [else]...

The simplest case – the ‚if'-‚then' instruction. Example:

```
if (Amplitude > 126)
  {
     Amplitude = 126;
  }
```

This delimits Amplitude to 126.

Optionally, an additional ‚else' branch can be added:

```
if (a > 240)
{
  Box(0,10,240,5,1);
}
else
{
  Box(0,10,a,5,1);
}
```

while()

While has the same method of operation as ‚if'. The difference is in that the subsequent block is repeated until the condition is no longer true. The loop is skipped immediately if the condition is not fulfilled right at the beginning (same behavior as ‚if').

```
while ((c < 10) && (a[c] != 0))
{
  c = c + 1;
}
```

This code fragment looks for the first zero in the array ‚a', up to the tenth element.

do...while()

This is the inverse notation of a ‚while' loop. In contrast to the latter, this loop is executed at least once since the condition is checked at the end of the loop.

```
do {
  c = c + 1;
}while ((c < 10) && (a[c] != 0))
```

for(;;)

For is the representation of a count loop. The method of operation is the same as in ‚while', the only difference is that the parentheses may contain an assignment and the increment of the count variable.

```
for (c = 0; c < 5; c = c + 1)
   {
      a[c] = 0;
   }
```

Here, the first five elements of the array a are set to zero.

Note: Notes for C programmer: Constructs like a++; b=()?:()=...etc. are not supported. Likewise, braces are mandatory for if-else, do-while, while, for and case, even if they are only followed by an expression.

switch() case:... break;

Case permits several selections to be made on the basis of one variable. This replaces many successive ‚if' instructions.

```
switch(t)
{
  case 1:
  {
    Text(1,120,0,0,"one");
    break;
  }
  case 50:
  {
    Text(10,120,0,0,"fifty");
    break;
  }
  default:
  {
   Text(5,120,0,0,"*");
  }
}
```

Note: A case element must be a constant, not an arithmetic expression or variable.

Although the syntax is C-compliant, the required braces make it differ slightly from the usual C syntax.

Conditional Compilation #ifdef #else #endif

These instructions enable the same code conditionally to be compiled differently, depending on the existence of certain defines. These conditions are interpreted before compilation, not during program execution.

```
#define SmallDiagnostics 1

void MainMenue()
{
  Screen();
  MenuItem(10,10,0,0,"Ausgabe", Output);
  MenuItem(10,20,0,0,"Eingabe", Input);
#ifdef SmallDiagnostics
  MenuItem(10,30,0,0,"Diagnostics", SmallDiagnostics);
#else
  MenuItem(10,30,0,0,"Diagnostics", ExtendedDiagnostics);
#endif
}
```

If the define SmallDiagnostics exists in this example, the part that follows the #ifdef will be compiled. If, for example, the define is transformed into a comment:

```
// #define SmallDiagnostics 1
```

the compiler only translates what follows the #else.

Predefined Symbols for Hardware-independent Programming

Depending on the selected hardware, the compiler provides the following #defines which have already been defined in the Include File system as follows:

```
#ifdef BTV04
#define MAX_X (128)
#define MAX_Y (64)
#endif

#ifdef BTV05
#define MAX_X (256)
#define MAX_Y (64)
#endif

#ifdef BTV06
#define MAX_X (240)
#define MAX_Y (128)
#endif

#ifdef BTC06
#define MAX_X (240)
#define MAX_Y (128)
#endif
```

This permits hardware-independent screens to be created if these defines are used for the positions on the screen:

```
void CLCDiag1()
{
  Screen();
  Text(0,0,MAX_X,0,_headline);
  Diagnose1.BindCLC ( 6, "CP", 1, 122);
  Diagnose1.Display(0,10,MAX_X,16,0,0);
  Key(KEY_F6,1,ScreenBack);
  Text(0,MAX_Y-8,MAX_X,8,"F6-back");
}
```

6.2 Standard Functions

The sytem ScreenManager provides most of the standard functions known from C.

Character Functions

Character Classification

Functions for the classification of characters demand of characters as argument the character to be classified (UINT8) and provide a logical value (TRUE or FALSE).

Function	Description
`BOOL isalnum(UINT8 c)`	TRUE, if `c` represents an alphanumerical character (digit or letter)
`BOOL isalpha(UINT8 c)`	TRUE, if `c` represents a letter
`BOOL iscntrl(UINT8 c)`	TRUE, if `c` represents a control character
`BOOL isdigit(UINT8 c)`	TRUE, if `c` represents a digit
`BOOL isgraph(UINT8 c)`	TRUE, if `c` represents a printable character, except blanks
`BOOL islower(UINT8 c)`	TRUE, if `c` represents small letters
`BOOL isprint(UNIT8 c)`	TRUE, if `c` represents a printable character or blank
`BOOL ispunct(UINT8 c)`	TRUE, if `c` represents a printable special character
`BOOL isspace(UINT8 c)`	TRUE, if `c` represents empty space (i.e. blanks or one of the following signs: `\f, \n, \r, \t, \v)`
`BOOL isupper(UINT8 c)`	TRUE, if `c` represents capital letters
`int isxdigit(int c)`	TRUE, if `c` represents a hexadecimal digit

Fig. 6-2: Functions to classify characters

How to Convert Characters

This functions allow to convert capital into small letters and vice versa.

Function	Description
`UINT8 tolower(UINT8 c)`	Delivers (as far as possible) the representation of the argument value `c` converted in the corresponding small letter.
`UINT8 toupper(UINT8 c)`	Delivers (as far as possible) the representation of the argument value `c` converted in the corresponding capital letter.

Fig. 6-3: Functions to convert characters

How to Convert Strings

The system also provides functions to convert strings into numbers.

Function	Description
FLOAT64 atof(STRING s)	Returns the corresponding float number, if string s represents a numerical value.
INT16 atoi(STRING s)	Returns the corresponding INT16 number, if the string s represents a numerical value.
INT32 atol(STRING s)	Returns the corresponding INT32 number, if the string s represents a numerical value.

Fig. 6-4: Functions to convert strings

String Functions

Function	Description
`void strncpy(STRING a, STRING b, UINT8 n)`	Copies `n` characters of string b in string a.
`void strncat(STRING a, STRING b, UINT8 n)`	Attaches n characters of string b to string a.
`INT8 strcmp(STRING a, STRING b)`	Lexigrafic comparison of string a and b. <0 a<b =0 a=b >0 a>b
`INT8 strncmp(STRING a, STRING b, UINT8 n)`	Lexigrafic comparison of the first n character of string a and b. <0 a<b =0 a=b >0 a>b
`UINT8 strchr(STRING a, STRING b)`	Finds the first appearance of character b in string a and returns its position; return value is 255, if character b was not found in string a.
`UINT8 strrchr(STRING a, STRING b)`	Finds the first appearance of character b in string a and returns its position; return value is 255, if string b was not found in string a.
`UINT8 strcspn(STRING a, STRING b)`	Delivers the number of characters at the beginning of a, which don't appear in b.
`UINT8 strpbrk(STRING a, STRING b, UINT8 n)`	Delivers the position in a, at which any character of b appears for the first time; return value is 255, if no character of b was found in a.
`UINT8 strstr(STRING a, STRING b)`	Finds the first appearance of string b in string a and returns its position; return value is 255, if b doesn't appear in string a.
`UINT8 strlen(STRING a)`	Returns the length of the strings.
`UINT8 strtok(STRING a, STRING b)`	Searches a for the first character string, which is restricted by characters of b, and returns its position; return value is 255, if no character string of a is restricted by b.

Fig. 6-5: String functions

Mathematic Functions

Basically, the mathematic functionality corresponds to the math.library known from C.

Function	Description
FLOAT64 sin(FLOAT64 x)	Calculates the sine of argument x, whereby x is indicated in radiant.
FLOAT64 cos(FLOAT64 x)	Calculates the cosine of argument x, whereby x is indicated in radiant.
FLOAT64 tan(FLOAT64 x)	Calculates the tangent of argument x, whereby x is indicated in radiant.
FLOAT64 asin(FLOAT64 x)	Calculates the arc sine of argument x; the result is indicated in radiant.
FLOAT64 acos(FLOAT64 x)	Calculates the arc cosine of argument x; the result is indicated in radiant.
FLOAT64 atan(FLOAT64 x)	Calculates the arc tangent of argument x; the result is indicated in radiant [-Pi/2, Pi/2].
FLOAT64 atan2(FLOAT64 y, FLOAT64 x)	Calculates the arc tangent of quotient y/x; the result is indicated in radiant [-Pi, Pi].
FLOAT64 sinh(FLOAT64 x)	Calculates the hyperbolic sine of argument x, whereby x is indicated in radiant.
FLOAT64 cosh(FLOAT64 x)	Calculates the hyperbolic cosine of argument x, whereby x is indicated in radiant.
FLOAT64 tanh(FLOAT64 x)	Calculates the hyperbolic tangent of argument x, whereby x is indicated in radiant.
FLOAT64 exp(FLOAT64 x)	Calculates the exponential function e^x.
FLOAT64 log(FLOAT64 x)	Calculates the natural logarithm of argument x.
FLOAT64 log10(FLOAT64 x)	Calculates the decade logarithm of argument x.
FLOAT64 pow(FLOAT64 x, FLOAT64 y)	Calculates the value x power y.
FLOAT64 sqrt(FLOAT64 x)	Calculates the square root of argument x.
FLOAT64 ceil(FLOAT64 x)	Calculates the lowest integer, which is not less than x.
FLOAT64 floor(FLOAT64 x)	Calculates the largest integer, which is not greater than x.
FLOAT64 fabs(FLOAT64 x)	Calculates the amount of x.
FLOAT64 ldexp(FLOAT64 x, INT32 n)	Calculates x^n.
FLOAT64 fmod(FLOAT64 x, FLOAT64 y)	Calculates the floating point rest of division x/y.

Fig. 6-6: Mathematic function

6.3 First Programming Steps

This chapter describes the major points of ScreenManager programming. All features of the system are discussed briefly and explained to the extent that is necessary for creating own applications. In the directory Projects\Examples, the ScreenManager installation is followed by accompanying examples, beginning with the first screen up to the utilization of the Flash file system. Going through the entire chapter is recommended.

The detailed description of the individual commands can be found in the rear part of the book that is mainly intended to be used as a reference manual.

The Screen() Command

This command is used for clearing the screen, removing all existing cyclic inputs and outputs, and canceling cyclic links to control variables. If the command is used within a script, that script becomes a SCREEN-generating script. Below, this is simply referred to as SCREEN.

Outputs and Graphics Commands

The following commands are used for static outputs on the screen. They are not updated during runtime. Thus, they remain on the screen in the form they were output initially.

Text() Command

The text command outputs one line of a static text.

TextAttr() Command

This command sets the attributes of all subsequent text edit and display outputs. It enables proportional font, inverted, bold or italic representation to be selected or to specify the text alignment (left-justified, right-justified, centered).

```
void main()
{
        TextAttr(1,1,0,0,1);
        Text(0,2,128,8,"Hallo World");

        TextAttr(1,0,1,0,1);
        Text(0,15,128,8,"Hallo World");

        TextAttr(1,0,0,1,2);
        Text(0,30,128,8,"Hallo World");

        TextAttr(1,0,0,0,3);
        Text(0,45,128,8,"Hallo World");
}
```

PutPixel()

This function outputs a single pixel at the specified X-Y position.

Line()

This command permits a line to be drawn between the specified start and end position. The line can be output either as a continuous, broken, or white (to delete a line) line.

Box()

The box command draws a rectangular box on the screen. This box is either solid black, a white area with a black frame, or completely white (deleting).

Circle()

Circle permits a circle to be drawn on the screen by specifying the center and the radius. The circle is drawn either in a continuous line, a broken line, or a white line. Filling in the surface is not possible. The circle command extensively used the computer resources, and should not be employed in a refresh script.

```
void main()
{
        Grafic_Commands();
}

void Grafic_Commands()
{
        Screen();
        Circle(64,32,20,1);    // Head
        Circle(54,27,3,1);     // Eyes
        Circle(74,27,3,1);

        Line(64,32,64,39,1);   // Nose

        Line(64,12,54,5,1);    // Hair
        Line(64,12,64,2,1);
        Line(64,12,74,5,1);

        PutPixel(50,34,1);     // Speckles
        PutPixel(52,38,1);
        PutPixel(54,36,1);
        PutPixel(56,40,1);
        PutPixel(70,34,1);
        PutPixel(72,38,1);
        PutPixel(74,36,1);
        PutPixel(76,40,1);

        Box(59,46,12,3,1);     // Mouth
}
```

ScreenUpdate()

The ScreenManager never updates the screen contents before a script has completely been processed. This avoids, for example, a flickering screen when overlapping areas are output cyclically. However, if, for example, a progress bar shall be output at a routine that takes some time to complete, this function can be used for forcing an advanced screen update during script runtime.

Invoking other Screens

To be able to invoke other screens from a given screen, buttons or menu entries of that screen must be interconnected with the screens that shall be linked. This is done using the following commands:

Key() Command

This function is used for linking a script function with either pressing or releasing a key. The operating system invokes the associated script function when this key is pressed or released. When the key assignment is changed using the key command, the original function of that key is no longer active (this can be relevant, for example, with number or cursor keys that have fixed assignments for editing values).

Special variant Key(0,..,..): With this exception of the command (the 0 is assigned as the key), the assigned script is executed whenever a key is actuated. The original key function is retained. Besides other purposes, this command is used for transferring each key to a connected controller.

```
void main()
{
        Pic_1();
}

void Pic_1()
{
        TextAttr(1,0,0,0,1);
        Text(0,20,128,8,"Pic 1");
        Text(0,20,128,8,"F1 --> Pic_2");

        Key(KEY_F1,KEY_PRESSED,Pic_2);
}

void Pic_2()
{
        TextAttr(1,0,0,0,1);
        Text(0,20,128,8,"Pic 2");
        Text(0,20,128,8,"F2 --> Pic_1");

        Key(KEY_F2,KEY_PRESSED,Pic_1);
}
```

MenuItem() Command

This command writes a text on the screen that can be moved to using the cursor (like in a Windows menu). The assigned script is invoked when the OK button is pressed. This command can be used for pretty easily creating menus with the ScreenManager.

```
void main()
{
        Programm1();
}

void Programm1()
{
    Screen();
    TextAttr(1,1,0,0,1);
    Text(0,0,128,8,"MenuItem-Befehl");
    TextAttr(1,0,0,0,0);
```

```
    MenuItem(10,12,100,0,"Menu 1", Menu1);
    MenuItem(10,22,100,0,"Menu 2", Menu2);
    MenuItem(10,32,100,0,"Menu 3", Menu3);
}

void Menu1()
{
        Screen();
        TextAttr(1,1,0,0,1);
        Text(0,0,128,8,"Menu 1");
}

void Menu2()
{
        Screen();
        TextAttr(1,1,0,0,1);
        Text(0,0,128,8,"Menu 2");
}

void Menu3()
{
        Screen();
        TextAttr(1,1,0,0,1);
        Text(0,0,128,8,"Menu 3");
}
```

Automatic ESC Function / ScreenBack() Command

The default assignment of the ESC key is such that the original values of the input fields are restored (if they had been modified) when the key is hit for the first time. Hitting the key again invokes the previous screen. The ScreenManager operating system retains the last 10 activated screens for this purpose. If the ESC key is overloaded by an own function, or if this return jump function is required elsewhere in the application, it can be invoked at any time with the ScreenBack command. This command opens the previous window.

Note: To be able to invoke a previous screen, ScreenManager retains the function in which the "Screen()" invocation was made. If a screen is produced by an invocation sequence of several functions, a return to the screen via ESC only invokes the first function that contained the word screen.

ParameterScreen()

This system call permits the fixed programmed parameter screens to be invoked at any time from inside the application. It is recommended to always assign them to a key (e.g. Shift - Help) in order to make the parameter menus and, consequently, the interface diagnosis to be available when the application is running.

6.4 Dynamic Inputs and Outputs on the Display

Display()

This call is used for representing the transferred variable cyclically at the specified point on the screen. The operating system automatically performs the cyclic updating after the current script has completely been processed. Display fields are only removed by the subsequent Screen command (i.e. when a new image is built up). Due to the cyclic updating, the Display command may only be used in conjunction with global variables that are still valid after the script has been processed. The output is event-controlled; it only takes place when a value changes. Thus it takes significantly less time as, for example, the cyclic output of a value in a refresh script.

Edit()

The Edit command has the same function as the Display command. Here, too, variables are updated cyclically on the screen whenever they change. Furthermore, the cursor can be moved into the field in order to edit the values. Modified values are displayed in italics and are no longer updated.

EditList()

DisplayList and EditList permit numbers to be edited and displayed in textual form. Beginning at zero, the numbers have a specific text assigned that is copied from the transferred string.

Example: "red gray blue" as the transferred string, and an individual word length of four results in the display of "red" at zero, "gray" at one, and "blue" at two. In return, the right-left cursor keys can be used during editing for toggling the text between red/gray/blue, and the specified variable obtains the assigned ordinal numeric value.

EditFilter()

This editing command basically works like Edit(). In addition, it permits a format string to be transferred that defines a specified character quantity for each cursor position. This input method is required, for example, in conjunction with editable segments of CLM program blocks.

EditFieldOptions()

The cursor is usually located on the input first that was initialized first. This command permits any other field to be defined as the start field.

Another selectable option is that the entered value is accepted immediately after the field has been exited, not (as usual) after the OK button has been pressed. This is particularly expedient if the screen shall be restored through an event that is related to this variable.

```
STRING st1[10]="+123";
STRING st2[10]="0110";
STRING st3[10]="09600";
STRING st4[10]="";

void main()

{
        Examples();
}

void Examples()
{
        Screen();
        TextAttr(1,0,0,0,3);
        Text(50,5,0,0,"all characters");
        Text(50,15,0,0,"only 0 and 1");
        Text(50,25,0,0,"<- / -> used");
        st1.Edit(2,5,40,0,4,0);
        st2.EditFilter(2,15,40,0,"BBBB");
        // set directly beneath the required edit field
        EditFieldOptions(0,1,0);
        st3.EditList(10,25,40,0,5,0,"096001920038400");
}
```

RefreshScript()

This function is used for logging on a routine with the operating system that is used for cyclically updating the screen. Depending on the system loading, this script is invoked every 100 to 250 ms. Only static screen outputs are permitted in this script.

Caution: Using Display, Edit, or Bind commands leads to a cyclic production of an increasing number of fields on top of each other. After a short time the system stops and an error message is issued. These commands may never be used in a Refresh Script. It should only be used for calculations, comparisons and static outputs.

It is intended to represent moving graphical images on the screen or to cyclically perform certain calculation or monitoring tasks.

ScrollEvent()

When the upper or lower screen margin is reached with the cursor, the ScrollEvent command permits the transferred scripts to be invoked. Upward or downward scrolling, these scripts can build up the image again. This is necessary, for example, for being able to represent larger lists on the screen.

6.5 Linking Control Variables

Single Read Process Read...()

The Read... commands read a variable from any connected controller. Since reading is performed in the background after the script has been processed, the read value is not immediately available in the script. This prevents that the script is blocked (= waiting) by communication. As far as possible, the ScreenManager operating system converts the variable type in the controller into the transferred type. (If waiting is necessary, there is a waiting function that does it explicitly).

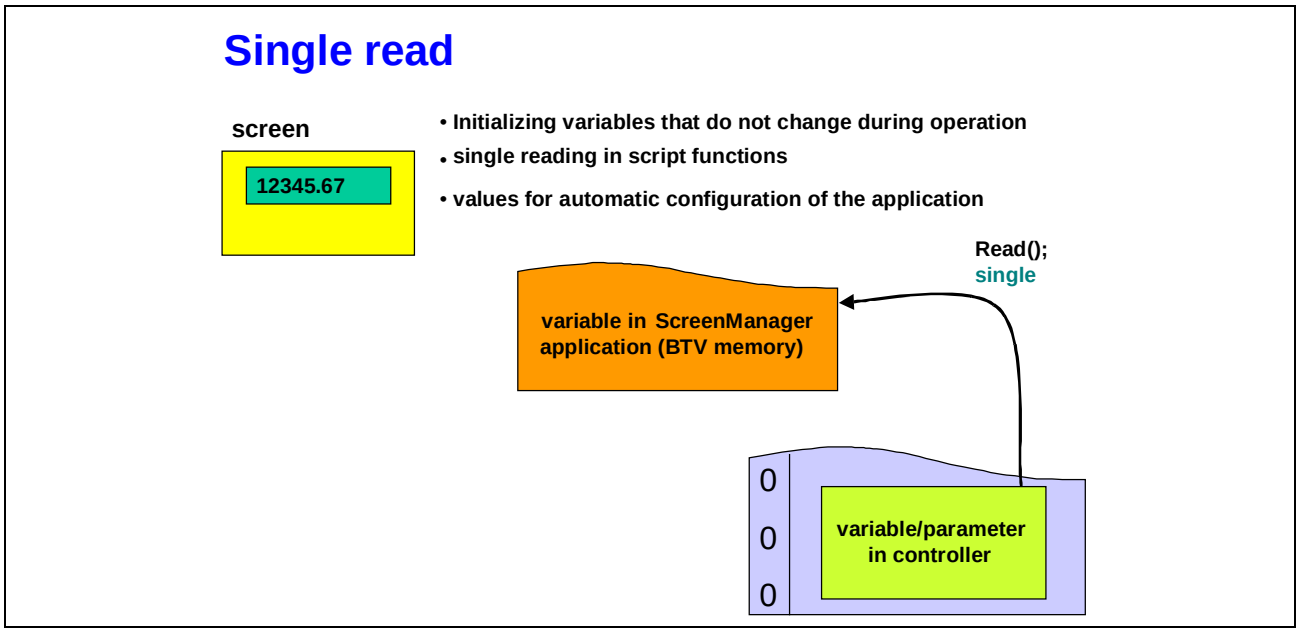


Fig. 6-7: Single read process

Single Write Process Write...()

Complement to Read. Here, too, writing is performed asynchronously after the script has been processed.

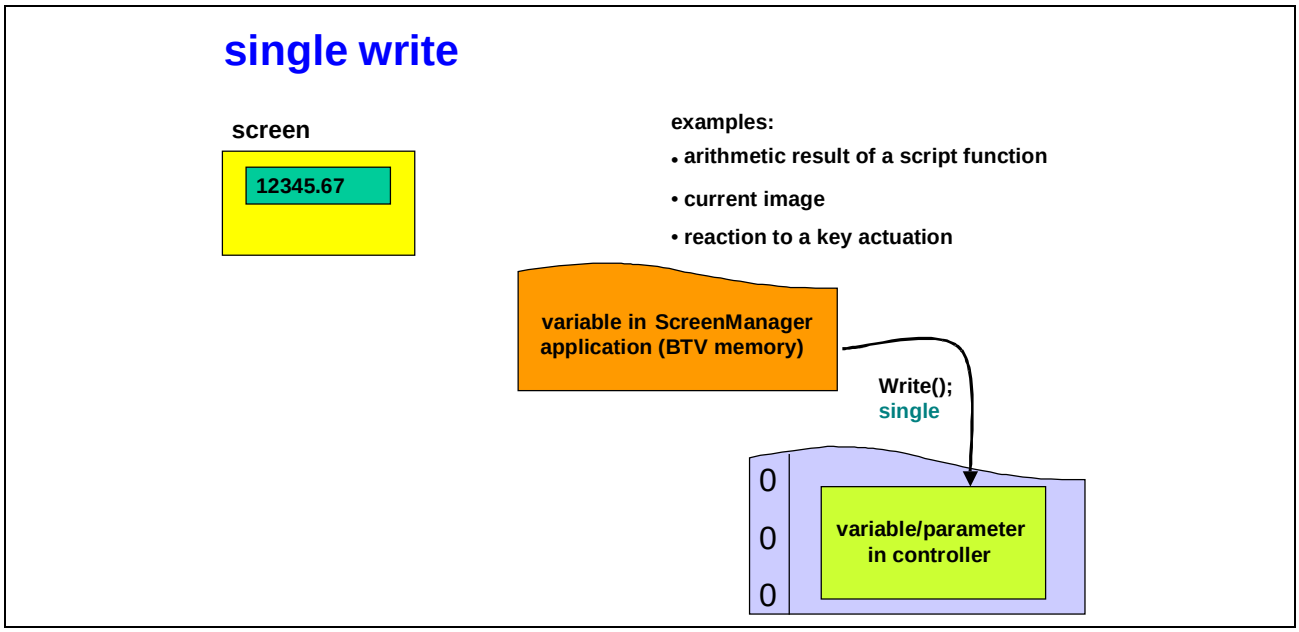


Fig. 6-8: Single write process

Dynamic Linking Bind...()

Same method of operation as Read; but the controller reads the value cyclically. Cyclic binding is cleared by a new Screen invocation.

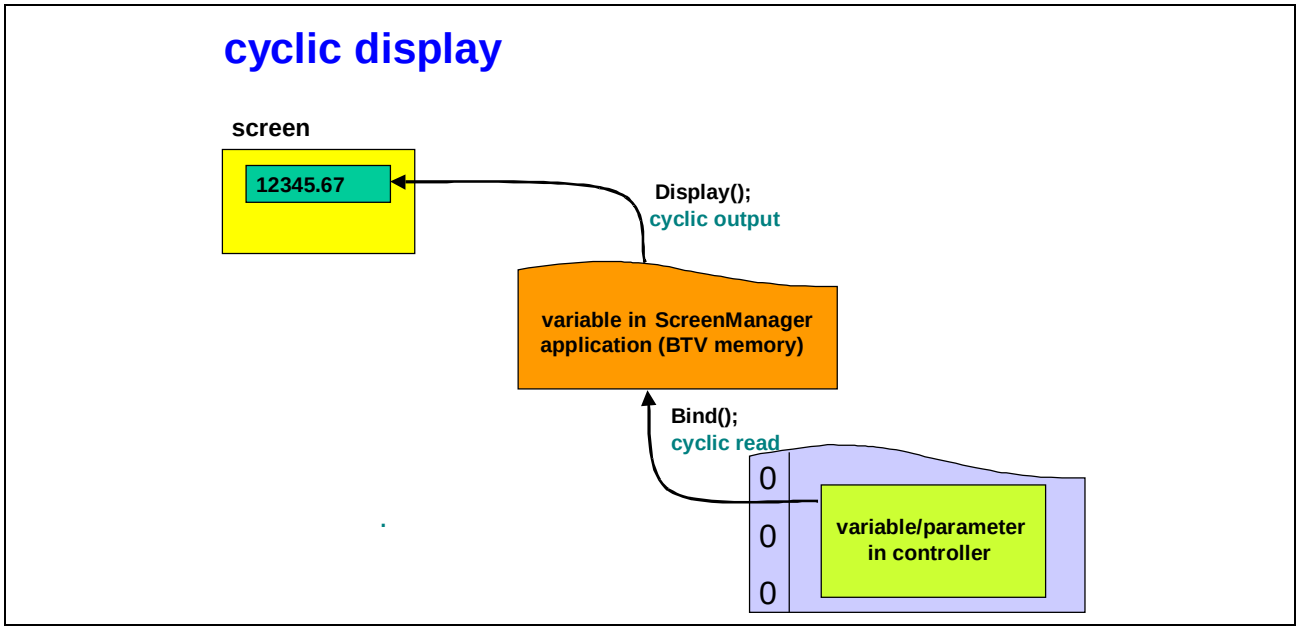


Fig. 6-9: Cyclic display

Forced Synchronous Writing and Reading: Wait...()

To avoid asynchronous writing and reading in critical program segments, this command can be used for waiting for the completion of all writing and/or reading processes of all pending serial buffers. Once this command has been terminated, all pending write and read instructions have been executed. Thus, the values are already available in the very same script run. The command should not be used in a Refresh Script; this would unnecessarily slow down the system.

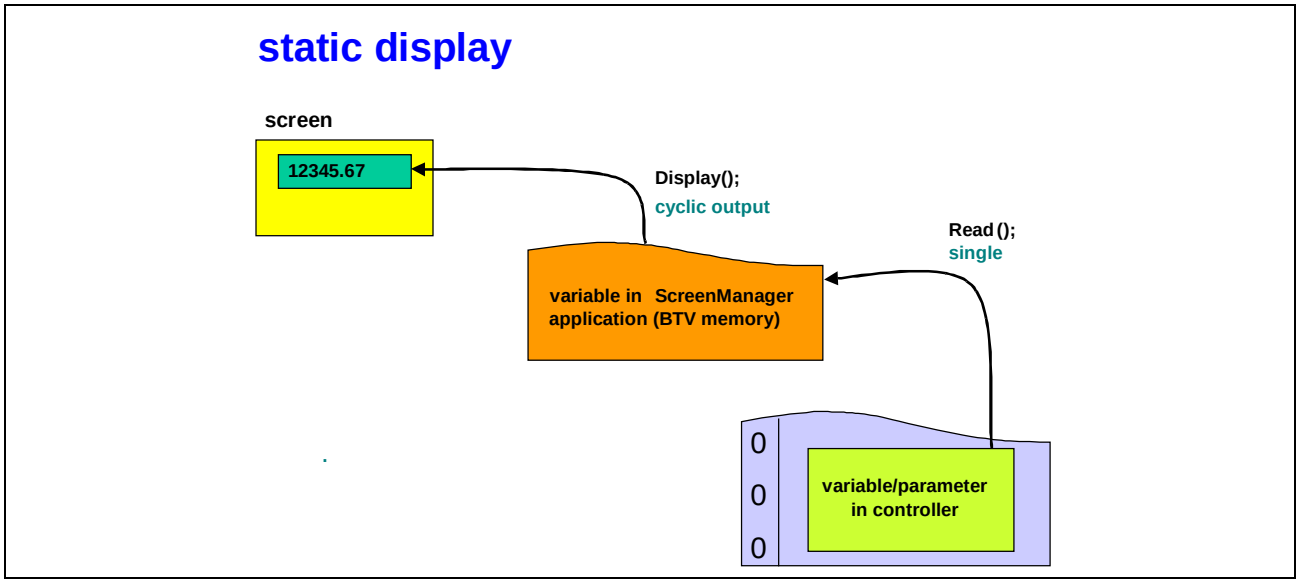


Fig. 6-10: Static displays without Wait...()

6.6 Editing Linked Variables

Editing Values on the Screen

If variables that were linked using Bind are edited on the screen, pressing the OK button leads to an automatic transfer of the modified value to the controller. An explicit write command is not necessary.

```
UINT16 ui16_1=150;
UINT16 ui16_2=0;
UINT16 ui16_3=0;
UINT16 ui16_4=0;

void main()
{
        ConnectDKC();
        Example _1();
}

void Example_1()
{
        Screen();
        ui16_1.WriteDKC(1,36774,7,0);
        ui16_2.ReadDKC(1,36774,7,0);
        ui16_3.BindDKC(1,36774,7,0);
        ui16_4.BindDKC(1,36774,7,0);

        ui16_1.Edit(10,10,0,8,0,0);
```

```
        ui16_2.Display(10,30,0,8,0,0);

        ui16_3.Display(10,50,0,8,0,0);
        ui16_4.Edit(80,50,0,8,0,0);

        TextAttr(1,0,0,0,3);
        Text(50,10,35,8,"Write");
        Text(50,30,35,8,"Read");
        Text(50,50,35,8,"Bind");
}
```

Function of OK & ESC Keys during the Editing Process

All values changed on a screen remain on the screen in italics as long as the OK button has not been pressed. Pressing the OK button transfers all modified values.

Using ESC to Cancel Editing

If values have already been modified, pressing the ESC key does not take you back to the previous screen. In this case, the modifications are discarded and the old values are restored. Pressing the ESC key again would branch to the previous image.

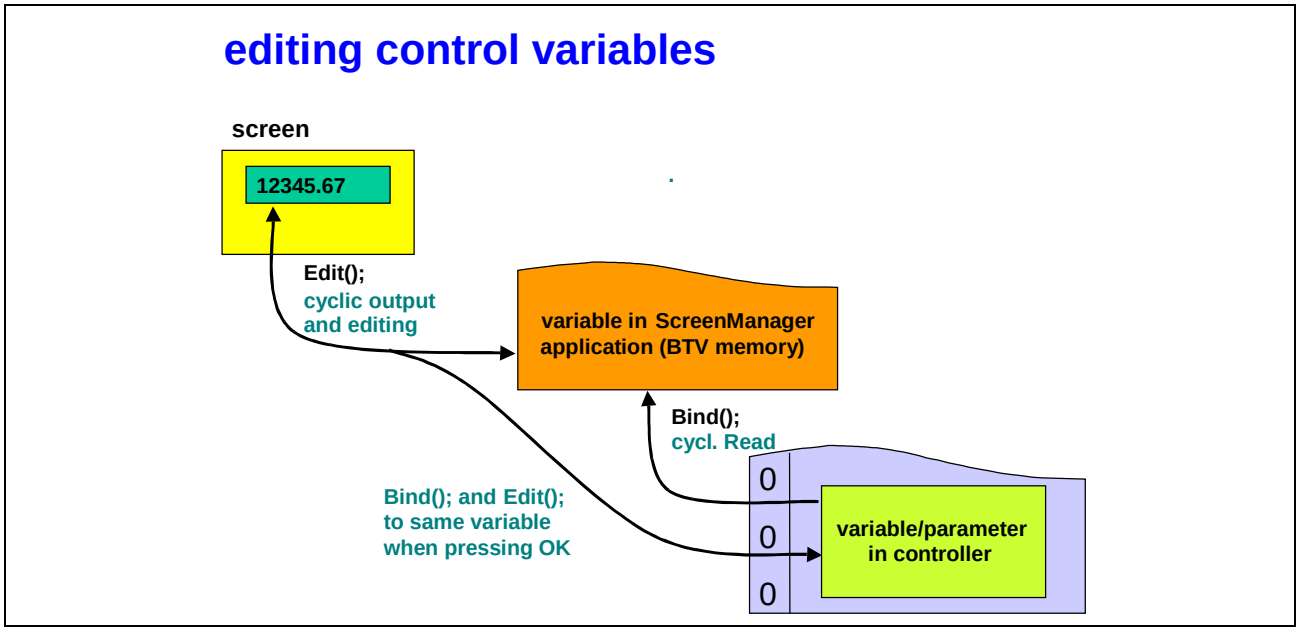


Fig. 6-11: Editing the control variables

Special Variants of the Key(KEY_OK...) Command

If the OK button is overloaded with a script, handling is slightly different than it is for the remaining keys. To be able to completely simulate the functions that are contained in the system and to enhance a range check, the executional sequence in the system is as follows:

The modified values are temporarily stored in the ScreenManager variables. Thus, they are available within this OK script. The modifications are discarded if EditOK is not invoked. Caution: The acceptance of the values is final if there is a new screen invocation in this OK script; but there is no transfer to the controller.

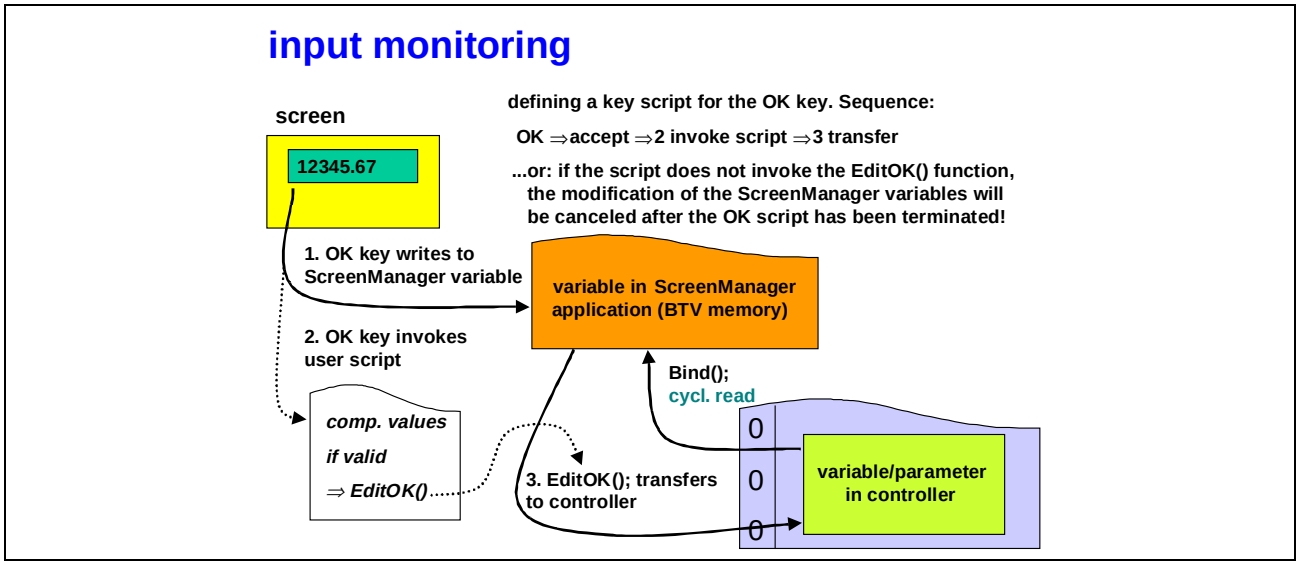


Fig. 6-12: Input monitoring

EditOK() Function

This function transfers the modified values to the controller and ensures that the values are permanently accepted in the ScreenManager program.

```
INT16 i16_1=0;

void main()
{
        ConnectDKC();
        Beispiel_1();
}

void Beispiel_1()
{
        Screen();
        TextAttr(1,1,0,0,1);
        Text(0,2,128,8,"Position block");
        TextAttr(1,0,0,0,1);
        Text(0,20,128,8,"Range +/-9999");
        i16_1.BindDKC(1,36774,7,0);
        i16_1.EditFilter(40,32,38,8,"911111");

        Key(KEY_OK,KEY_PRESSED,Check);
}
void Check()
{
        if ((i16_1 >= -9999) && (i16_1 <= +9999))
```

```
        {
                EditOK();
        }
}
```

EditESC() Function

In the default, this function is assigned to the ESC key. It can be used for restoring the old values in the OK script (if input errors are detected, for example).

```
INT16 i16_1=0;

void main()
{
        ConnectDKC();
        Beispiel_1();
}

void Beispiel_1()
{
        Screen();
        TextAttr(1,1,0,0,1);
        Text(0,2,128,8,"Position block");
        TextAttr(1,0,0,0,1);
        Text(0,20,128,8,"Range +/-9999");
        i16_1.BindDKC(1,36774,7,0);
        i16_1.EditFilter(40,32,38,8,"911111");
        Key(KEY_OK,KEY_PRESSED,Check);
}

void Check()
{
        if ((i16_1 >= -9999) && (i16_1 <= +9999))
        {
                EditOK();
        }
        else
        {
                EditESC(); // do not omit! ERROR!!!
                Warnung();
        }
}

void Warnung()
{
        Screen();
        TextAttr(1,0,0,0,1);
        Text(0,20,128,0,"illegal value");
}
```

What Happens if you Omit EditOK() ?

As soon as the key script is exited, ScreenManager sets the variable values back to the values they had before they were changed by the user. Nothing is transferred to the controllers.

What Happens when the Screen is changed in the OK Script ?

The ScreenManager responds as expected if EditOK() has been invoked beforehand. The new entries are accepted and transferred to the controller. Next, the image is changed.

If EditOK() is not invoked before the image is changed, the new values are retained in the ScreenManager, but there will be no transfer to the controller.

6.7 The Event() Function

Outputting Warnings

The Event function permits unsigned 16-bit variables to be monitored cyclically for changes/conditions. If the condition is true, the transferred script is invoked once (positive edge). In the simplest case, this is done to display warnings that indicate positive or negative alarms on the screen.

Event-driven Screen Change

These event scripts can of course invoke a new screen that will be opened when the condition is fulfilled.

New Screen Setup by Variable Input

A special version is a screen that overwrites itself when a certain event occurs that, for example, requires another variable to be bound.

EditSetFieldOptions()

This command makes it possible to accept the variables in certain edit fields immediately when the field is exited, not when the OK button is pressed. A screen programmed in this way would then be restored if, for example, a parameter number is changed.

Event-driven Screen Update

Instead of using a refresh script using an event wherever possible is significantly more efficient. This only happens when a value changes, not periodically. Updating a graphical image, for example, is quicker and puts less load on the system.

```
UINT16 ui16_1=10;

void main()
{
        ui16_1.Event(10,2,Above);
        ui16_1.Event(10,3,Below);
        Example_1();
}

void Example_1()
{
        Screen();
        TextAttr(1,1,0,0,1);
        Text(0,2,128,8,"Event example");
        TextAttr(1,0,0,0,1);
        Text(0,15,128,8,"Enter a number");
        Text(0,25,128,8,"between 0 and 20:");
        ui16_1.Edit(40,38,0,0,0,0);
}

void Above()
{
        Screen();
        TextAttr(1,0,0,0,1);
        Text(0,15,128,8,"number is > 10");
        Key(0,1,Example_1);
}

void Below()
{
        Screen();
        TextAttr(1,0,0,0,1);
        Text(0,15,128,8,"number is < 10");
        Key(0,1,Example_1);
}
```

6.8 The List Function

What is a Handle ?

A handle is an unsigned 16-bit value (type UINT16) that is transferred from the system to the application. It identifies a list or a flash file.

A handle must always be given by the system and be stored in a variable. Only this variable may be used for accessing the open memory object. The value of a handle is only of interest for the operating system. It cannot be guarantees that, for example, the first requested handle is ‚1' and the next one is ‚2'. The values can be absolutely arbitrary.

65535 (hexadecimal 0xffff) is the only known value a handle can't assume. This value stands for an invalid handle. You can obtain the value if you try to open a non-existent file, for example. This value is predefined by the system as INVALID HANDLE. Comparing the handle with not equal INVALID HANDLE (0xffff) shows whether or not the specified file has been located.

Note: INVALID_HANDLE is only defined by the system starting from version 4. Concerning older projects it is possible that INVALID_HANDLE itself is specified as define. This causes an error during compiling. Delete the define-line, in which the INVALID_HANDLE was specified, from this projects.

GetFreeList()

This function is used for requesting a new empty list from the system. The return value is a handle. This handle must be stored and be used for any access to the list.

WriteList()

This function permits variables to be written to any XY position of the list. The list can be compared to an Excel spreadsheet that can store a value in each field. Each field can accommodate a different data type. For storing, the system automatically selects the type that requires least storage space.

ReadList()

This function reads back values from any list position. Zero or a blank string is returned if the list position was previously unassigned.

InsertList()

This function inserts an element at the transferred position. The remaining elements of the line are shifted to the right.

EraseElementList()

This function deletes the specified element and pulls the elements that are further to the right to the empty location.

CloseList()

This function is used for returning a handle to the operating system. In turn, the operating system relinquishes the memory space that was occupied by the list. Any subsequent access using the same handle leads either to a runtime error or to an incorrectly assigned list, and must therefore be avoided. Ensured that you do not forget CloseList. Only up to 50 handles of a given type can be open at any one time in the system.

FreeMemoryList()

The function returns the free RAM space that is available in the system. If several large lists are used, the application should ensure that there is no memory overflow. In this case, the program would be aborted with a runtime error.

6.9 The Flash File System

Organization of the File System

Formatting

Formatting clears the entire storage space that is available to the file system.

Compression, Upload and Download using Dolfi

The following sequences are started:

- Deleted files are removed (compressing)

If the free memory space is still less than 80 Kbytes, the following files are removed until that value has been reached:

- Deleting the oldest temp files
- Deleting files (user) of old applications (Application-CRC different than the actual one)

Next:

- Writing a header that is valid for Dolfi Upload.
- Calculating the checksum that is valid for the currently existing file system.

Note: It is only immediately after this procedure has been performed that the file system can successfully be saved using Dolfi. If the file system has been modified after the compression process (new files are created or existing ones deleted), an upload using Dolfi is still possible, but the data loaded into the PC are useless. Saving the data in that window immediately after compression is the safest possibility of a correct file upload.

Behavior during a Power Loss

The complete flash content is turned over twice during the compression process (the process can take up to one minute). Twice, because this always keeps an undestroyed set of data in the flash in the event of a sudden power loss. When the operating voltage returns, the compression process is resumed at the interruption point. When a flash is newly written, the old one remains marked valid until writing the new one has been completed. Thus, it can almost be excluded that (as it is possible in a PC) files are lost by a sudden power loss.

FlashStoreList()

This command stores a list that exists in the memory in a file, specifying a file name and a file extension. Using the next command, this file can again be opened as a list for reading.

FlashOpenList()

This function returns a handle to a list that is stored in the flash. This list is identified by its file name and file name extension. Writing to this list is no longer possible.

CloseList()

As described above, handles to a flash list must be relinquished after they have been used. There is a limit of 50 free handles for flash lists.

EditFileName()

This function opens a modified form of the file manager that permits a file name to be selected for files with the specified extension. This enables the user to select a file from the existing files.

Flash Lifetime Warning

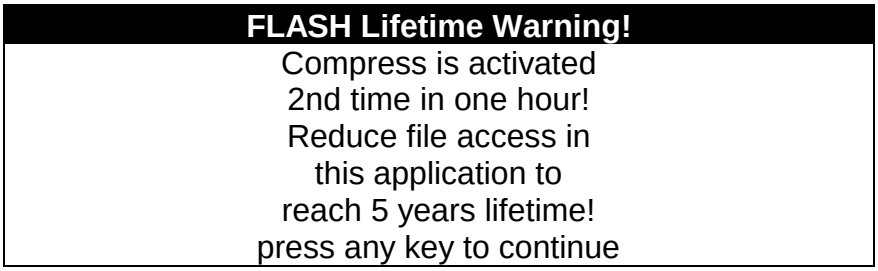


Fig. 6-13: FLASH Lifetime Warning window

This warning appears if the data amount written by the application into the FLASH file system is such that the compression process is activated twice within one hour. This process physically clears the free space that was produced by deleting files, and makes it available to the file system. The FLASH has a limited lifetime of 50,000 compression cycles. (The employed flash permits at least 100,000 cycles. As a protection against sudden power losses, each page is written twice during the compression process. This ensures that there is always one valid file system.) To ensure operability for five years, this process may only be performed once every hour during this time.

The window is merely for information. It must be acknowledged by the user. Normal program execution remains interrupted until the window is acknowledged. This prevents reliably that a unit accidentally destroys the FLASH by writing to it too frequently. It must be ensured that there are no regular write access procedures in the application that would trigger off this mechanism.

Example: Storing Configuration Data

```
#define INVALID_HANDLE 0xffff

UINT16 MyList = INVALID_HANDLE;     // List handle

UINT16 Language        = 0;
UINT16 Axis           = 0;
STRING Station[20]     = "System 1";

void WriteSetup()
{
    UINT16 MyList;     // List handle

        MyList = GetFreeList();
        Language.WriteList(MyList,0,0);
        Axis.WriteList(MyList,1,0);
        Station.WriteList(MyList,2,0);
        FlashStoreList(MyList,"My","Setup");
        CloseList(MyList);
}

void ReadSetup()
{
    UINT16 MyList;     // List handle
        MyList = FlashOpenList("My","Setup");
        if (MyList != INVALID_HANDLE)
        {
               Language.ReadList(MyList,0,0);
               Axis.ReadList(MyList,1,0);
```

```
                Station.ReadList(MyList,2,0);
                CloseList(MyList);
        }
        else
        {
                WriteSetup();
        }
}
void Save()
{
        EditOK();
        WriteSetup();
}

void MySetupScreen()
{
        Screen();
        TextAttr(1,1,0,0,1);
        Text(0,0,MAX_X,10,"Setup settings");
        TextAttr(1,0,0,0,0);
        Text(10,15,0,0,"Axis");
        Text(10,25,0,0,"Language");
        Text(10,35,0,0,"Name");
        TextAttr(0,0,0,0,0);
        Axis.Edit(60,15,0,0,0,0);
        Language.EditList(60,25,0,0,7,1,"Germanenglish");
        Station.Edit(60,35,0,0,0,0);
        Key(KEY_OK,KEY_PRESSED,Save);
}

void MainMenu()
{
        Screen();
        TextAttr(1,1,0,0,1);
        Text(0,0,MAX_X,10,Station);
        TextAttr(1,0,0,0,0);
        Text(10,MAX_Y-10,0,0,"F1 - Setup");
        Key(KEY_F1,KEY_PRESSED,MySetupScreen);
}

void main()
{
        ReadSetup();
        MainMenu();
}
```

Example: Storing and Loading CLM Programs

This typical program can be found on the ScreenManager data carrier.

6.10 I/O Access

The I/O Registers

The control panels possess 100 word registers. The first 20 are firmly assigned to the hardware inputs and outputs of the control panels. The remaining ones can be used as required.

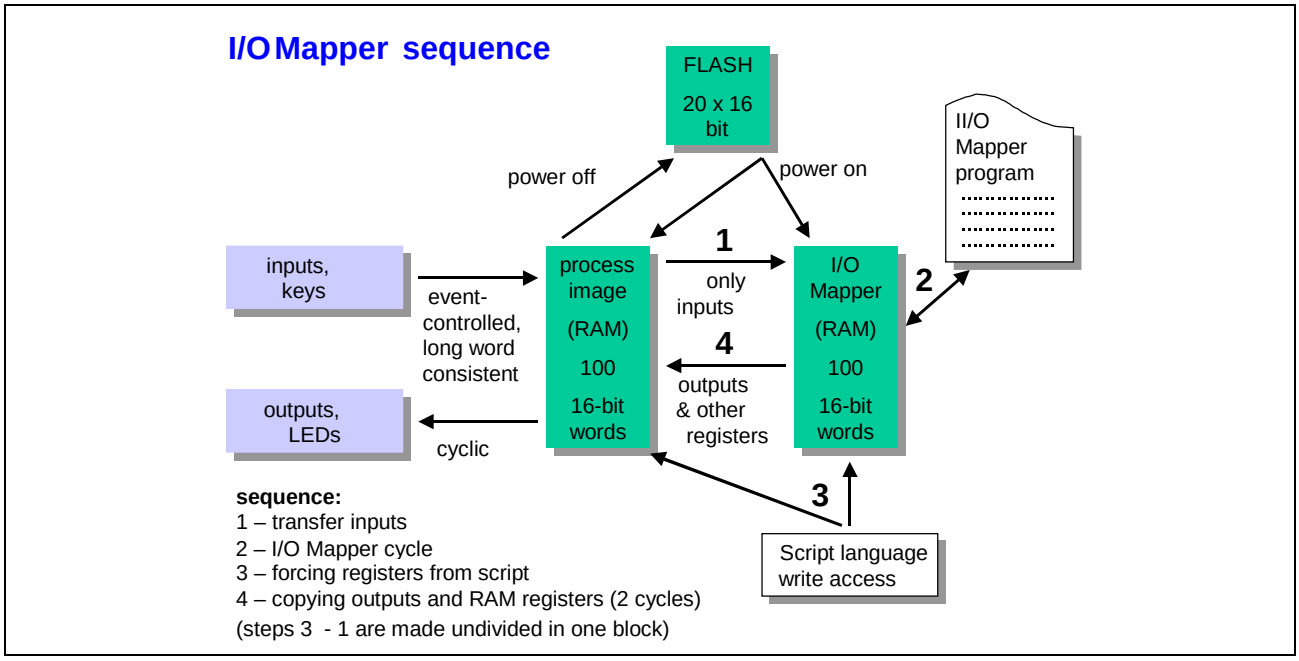


Fig. 6-14: I/O mapper sequence

Direct I/O Access to I/O Registers

From the script language, the following commands can be used for reaching the register area cycle-synchronously with the I/O mapper:

GetIO(), SetIO()

Access to individual bits of the I/O area.

GetIORegister()

Reading an entire 16-bit register word.

SetIORegister()

Writing an entire 16-bit register word.

BindIORegister()

Cyclic binding of a ScreenManager variable to a 16-bit register word.

The I/O Mapper

The I/O mapper is a machine program that runs in parallel to the ScreenManager application. It has a cycle time of 3 ms (with many interconnections, this time is extended in steps of one ms). This program processes interconnections that can be programmed with a graphical desktop in the ScreenManager. The I/O mapper runs in a non-maskable interrupt, and is not even interrupted by system fault messages.

Combination of Script and I/O Mapper

The following figure shows a possibility of combining sequences of the interconnection logic and script functions.

set register 70-0 as result of an interconnection

initialization in the ScreenManager main program:

```
...
   x.BindIORegister(70);  // read I/O register cyclically
   x.Event(1,1,MyScript); // event upon edge at 70-0
...
```

script function starts up...

event script starts up once upon each positive edge:

```
void MyScript()
{
  ...any script function ...
  ...serial read, write,
  ...compute, open screens   etc...
  SetIO(70,1,1);  // ready message to I/O Mapper
  SetIO(70,1,0);  // by an edge at register 70-1
}
```

...script finished

Fig. 6-15:Combination of I/O mapper programming and script access

I/O Access and SIS Real-time with MTC/ISP or ELC

Establishing a real-time connection to one of the above-mentioned controllers immediately terminates the execution of the I/O mapper. Instead, an I/O mapper is started that is invariably programmed in the firmware, and ensures that always the same keys and terminals are interconnected with the corresponding bits in the controller. The original I/O mapper is no longer started, even if communication with the corresponding controller is terminated.

6.11 Links with Controllers – Special features

The commands specified for the individual controllers go beyond the Read...() Write...() and Bind...() that are supported for all; they represent peculiarities for the related units.

MTS/ISP - ConnectPLC()

As described above, the MTC/ISP establishes the real-time connection to the control panel without any contribution of the application. In order be able to access variables and parameters of the controller, the ConnectPLC() command must be invoked at the beginning of the application. This command shows "waiting for PLC..." on the display until the controller has reacted. Next, uploading MiniMap and ProVi data is started. If communication to the ISP is interrupted, the control panel always performs a restart of the application.

Flash Cache of MiniMap and ProVi

If the control panel recognizes that the data of this controller has already been loaded and is available in the flash memory, the stored files are accessed and the application is started without any delay.

ConnectCLC()

The call must be executed once before communication with a VisualMotion controller is started. This does not delay the program execution.

JogCLC()

This command interconnects a register bit of the control panel with a register bit of the VisualMotion controller. The link is protected by a checksum and is monitored by a time-out time in the controller. The normal communication with the controller is continued at a reduced rate. Only one bit can actively be transferred at any one time. Any bit that becomes active afterwards will not be taken into account.

ConnectCLM()

The call must be executed once before communication with a CLM/DLC controller is started. This does not delay the program execution.

JogCLM()

This command interconnects a register bit of the control panel with a control register of the DLC/CLM. The link is protected by a checksum and is monitored by a time-out time in the controller. The normal communication with the controller is continued at a reduced rate. Only one bit can actively be transferred at any one time. Any bit that becomes active afterwards will not be taken into account.

ConnectDKC()

The call must be executed once before communication with an Ecodrive3 is started. This does not delay the program execution.

ConnectELC()

This call initializes the communication to an ELC. The address of the controller on the bus to which real-time communication is required is specified as parameter. Communication is immediately established and maintained even if the application or the PC accesses a different controller via the second port. The transfer of the address zero terminates real-time communication or starts data communication without real-time transfer.

ELC Format Lists for NC Command and Parameter Syntax

The InitProgListELC() command starts the upload of the format lists for NC blocks and parameters from the specified controller address. Like with MTC/ISP, they are stored in a temporary file in the flash and are used as required without a new upload. If the program or parameter editor of the control panel is used for different controller versions on a bus, the correct access of the control panel to the controller requires this command to be executed once after each controller selection.

6.12 Own Error Handling Routines

Which Errors Can Be Trapped ?

Serial error messages from the connected controller and time-outs of a controller that does not respond can be trapped. These errors usually lead to an acknowledgeable window that can be replaced with an own one. Alternatively, a way can be found that permits certain error messages (that can sometime be waited for) to be interpreted and correctly be processed in the application. Unknown error messages can easily be transferred to the fault routine that exists in the runtime system - they are simply not acknowledged.

OnErrorCall()

Function for the definition of an own error script. Invoked in main, it is globally or locally valid for any screen.

ExceptionWindow()

Freezes the current screen image. Any information can be output in the form of a window, for example. Only one screen can be saved at any one time. ExceptionWindow may therefore not be invoked repeatedly without being deactivated with CloseWindow in between.

GetError()

Provides the error number and - if applicable - an error text from the controller.

GetErrorTelegrams()

Permits the serial message frames that caused the error to be read completely.

ClearError()

Clears the error. The message frame that caused the error is either discarded or repeated. This can be specified by a parameter.

WaitKey()

Never use this command in normal scripts. With the exception of the I/O mapper and real-time communication, the entire system is stopped until the user hits a key. This is necessary in the event of a serial fault since the system does not know anyway what it shall do without the expected response from the controller.

CloseWindow()

This restores the previously saved screen contents. Program execution is resumed at the point where it was interrupted.

7 Language Definition

7.1 Identifiers

All identifiers are case-sensitive. An identifier must begin with a letter. It may contain numbers and the special character '_.' Label size is unlimited but only the first 20 characters are significant.

7.2 Compiler Instructions

#define

Syntax: #define *identifier text*

All occurrences of 'label' are replaced with 'text' before compiling the sources. To simplify maintenance, this command should be used for string definitions (if multi-lingual) and serial control definitions.

Various constants have already predefined by the user interface. This concerns all key codes, text and graphics command features, and the width and height of the selected control panel type:

#define MAX_X (...value of the selected control panel...)

#define MAX_Y (...."....)

#ifdef

#ifdef *(defined identifier)*

...compiled if 'identifier is defined (to any value)

#else

...is compiled if 'identifier is not defined

#endif

The user interface predefines a symbol of the selected control panel that can be used for creating hardware-independent programs. Currently, these are: BTV04, BTV05, BTV06 or BTC06

#include

Syntax: #include "*example.scm*"

Includes the example.scm file at the current position.

7.3 Variable Definition

Supported Types

Keyword	Size in bytes	Range
void	0	-
PTR	4	Function pointer – is only used for system calls. This type cannot be used as a parameter in a separate function.
BOOL	1	true, false
INT8	1	-127...127
UINT8	1	0...255
INT16	2	-32768...32767
UINT16	2	0...65535
INT32	4	-2147483648...2147483647
UINT32	4	0...4294967295
Float	4	± 3.4e-38...3.4e38
FLOAT64	4	± 1.18e-308...3.4e308
STRING[1–254]	defined size + 2	ASCII

Fig. 7-1: Supported variable types

Declaration of Variables and Constants

Variables defined at the beginning of the program, outside of function blocks, are global variables. They can be used in any function. Local variables are defined within functions before the first command.

Caution: Using the same variable is used in different event scripts may cause unpredictable results. However, one running event script does not interrupt another; the variables are consistent for each call.

Numerical Declarations

Definition of an integer variable, name: 'Position' without initial value. As a global variable, it has the initialized value 0; as a local variable, it has a **random value**!

```
UINT16 Position;
```

Definition of a variable, name: 'Line' with the initialized value 10

```
INT8 Line = 10;          // decimal
INT8 Line = 012;         // octal (base 8)
INT8 Line = 0x0a;        // hexadecimal
```

Definition of a float variable, name: 'Temp' with the initialized value 315.23

```
FLOAT Temp = 315.23;          // decimal
FLOAT Temp = 3.1523e2;        // exponential notation
```

The decimal point is mandatory. A declaration such as FLOAT a = 0; will be rejected. The correct notation is FLOAT a = 0.0;

Boolean Declarations

Usually, only TRUE or FALSE are supported as valid values of parameters and variables of the Boolean type.

```
BOOL b = TRUE;
BOOL c = FALSE;
```

To maintain compatibility with existing projects, 0 and 1 may be used in assignments and as parameters, but not for initialization.

Arrays

All variable types but strings can be used for declaring one-dimensional arrays. The index parameter of an array must be of the UINT16 type. The maximum number of elements is restricted to 65535.

```
FLOAT64 f[100];
INT8 i[5] = {88,77,66,55,44};
```

Note: The list functions can be used if texts shall be stored in an array structure or if two-dimensional arrays are needed. List variables are dynamically managed by the system, store string variables in a compressed way (only in the required length), and - as opposed to arrays - only consume memory space as long as they are required.

Typical String Definitions

Definition of an empty string variable, name: 'Message', maximum length 254 characters:

STRING Message[254];

Definition of a string of 30 characters and the initialized value "Hello"

STRING[30] Message = "Hello";

Using string constants:

Text (10, 10, 40, 8, "Hello");

Assigning a Value to a String, Copying Strings

'=' cannot be used for string assignments. The 'StrCopy' command must be used instead.

StrCopy (Message, "This is the first message");

(in compatibility with C, the strcpy() command is supported too).

7.4 Syntax of the PLC Variable Names

The definition of a PLC variable may be done similar to a string declaration:

"MY_PLC_Var"

The PLC programming environment needs a *.btv-file to generate the MiniMap file for serial variable access. To generate this file, all strings that contain PLC variable names must be defined as follows:

_PLC("MY_PLC_Var")

This declaration behaves like a normal string and may be used wherever a string declaration with "..." is permitted.

7.5 Multi-Lingual String Definition

Multi-lingual strings can be defined in a resource editor that is a part of the user interface. Identifiers with texts in several languages can be defined there. In the ScreenManager program, the identifier can be used instead of a string variable.

The language can be changed at any time. This can be done, for example, using a user-programmed menu or according to the result of an equation in a control variable in an event script. The first language definition will always be used if the selected language does not exist in the definition.

Using a multi-lingual string:

In a function call:

```
Text (10, 10, 40, 8, MyMultiLanguageText);
```

In an assignment:

```
{
  STRING s[30];
  StrCopy(s, MyMultiLanguageText);
```

The text in 'MyMultiLanguageText' changes after the language has been changed. Any assignments or outputs made using this file must then be repeated. This is not usually necessary with text output since the manual language selection is normally connected with an image change and, consequently, with a new image setup.

7.6 Language Structure

Operators

Character	Meaning	Types
=	Assign	Numerical
+	Plus	Numerical
-	Minus	Numerical
*	Multiply	Numerical
/	Divide	Numerical
%	modulo	Numerical
>>	Shift right bit by bit	Ordinal
<<	Shift left bit by bit	Ordinal
~	Invert bit by bit	Ordinal
^	XOR bit by bit	Ordinal
&	Binary AND	Ordinal
|	Binary OR	Ordinal
!	Logic NOT	BOOL
||	Logic OR	BOOL
&&	Logic AND	BOOL
==	Comparison for "equal to"	Ordinal
!=	Not equal to	Ordinal
>	Greater than	Numerical
>=	Greater than or equal to	Numerical
<	Less than	Numerical
<=	Less than or equal to	Numerical

Fig. 7-2: Operators

Equations

Syntax of equations:

a = b + c;

b = d * e;

a = b * (7 + c);

bb = (a >= c); // here, bb is of BOOL type

All variables must be numerical constants or of the same type.

a = MyFunction(b);

'a' and 'MyFunction' must be of the same type. 'b' must be of the type used in the declaration of 'MyFunction'. Expressions of the following form are permitted:

a = MyFunction (b * (1123.4 + c));

Type Conversion

Ordinal types (i.e. integer variables, no FLOATs) can mutually be converted according to the following syntax:

UINT8 a;

UINT16 b;

UINT32 c;

FLOAT f;

a = (UINT8) b + (UINT8) c; // here, 8 bits are used for adding up

or

a = (UINT8) (b + (UINT16) c); // here, 16 bits are used for adding up

or

a = (UINT8) ((UINT32) b + c); // here, 32 bits are used for adding up

There will be a binary truncation of any overflow of types of a smaller bit width. Thus, the conversion result is not correct.

Converting ordinal types into FLOAT values is done in the same way.

f = (FLOAT) a;

Converting FLOATs into ordinal types is not permitted in ANSI-C. Thus, conversion in ScreenManager is only possible using the conversion methods:

a = f.uint8();

b = f.uint16() + b * 2;

Definition of Functions

TYPE LABEL_1 (*TYPE LABEL_2, TYPE LABEL_3*)

{

...

}

return a *constant value* or an *identifier*.

A function definition within a function is not allowed. If a value is not returned, the (empty) 'void' type must be used.

A function is called by the function name, including the comma-separated parameters. Example: `A = Test(b,c);`

Execution is stack-based. Recursive calls are permitted. The function always returns a copy of the variable. A function that is linked with an event or a menu can be invoked by the operating system. A function of the 'void' type cannot return a value. All the other functions must return a value.

The 'return' key word is used for returning a value:

```
UINT8 Sum(UINT8 a, UINT8 b)
{
      a = a + b;
      return a;
}
```

Arrays as Parameters

An array is transferred as an open array. This is the only possibility of implementing constructs that are similar to pointers. In contrast to normal parameters, a pointer is transferred when an array is transferred. When the content of an array element is changed within a function, its value also changes outside the function.

```
void MyFunction (INT32 a[], UINT16 Max)
{
      UINT16 i;
      for (i = 0; i < Max; i = i + 1)
      {
            a[i] = 0;
      }
}
```

The system monitors the array size. Any access to an element outside the declared area would lead to a runtime error.

Structure Element

if

```
if (BOOL variable or comparison expression)
{
...
}
else
{
...
}
```

The "else" construct is optional.

while

```
while (comparison or ordinal variable)
{
...
}
```

do...while()

This is the inverse notation of a ‚while' loop. In contrast to the latter, this loop is executed at least once since the condition is checked at the end of the loop.

```
do {
  c = c + 1;
}while ((c < 10) && (a[c] != 0))
```

Note: Both commands must always be followed by parentheses even if these contain only an invocation. A 'while' loop can block the actual task, and should only be used for menus in which the expected number of loops is known.

for

```
for (assignment; (comparison or ordinal variable); instruction)
{
...
}

for (i = 0; (i < 4); i = i + 1)
{
...these instructions are executed four times
}
```

Note: ScreenManager does not support the i++ syntax that is possible in ANSI C.

switch() case:

```
switch(a)
{
      case 1: { ... instruction(s)... break;}
      case 42: { ... instruction(s)... break;}
      case 4711: { ... instruction(s)... break;}
      default: { ... instruction(s)... break;}
}
```

Note: The braces before and after each ‚case' block are mandatory in ScreenManager. In ANSI C, they are optional.

7.7 Object Extension

The ScreenManager programming language uses an object extension for type-dependent variable operations that are used for I/O-binding, output or conversion commands. The object methods are limited to the system commands. The reaction of the system function changes as the employed type changes. Each variable has the following standard methods:

Method	Description
BindPLC (STRING Identifier)	Links a variable to the PLC
ReadPLC (STRING Identifier)	Initializes individual read commands to the PLC
WritePLC (ISTRING Identifier)	Initializes individual write commands to the PLC
Write (INT16 y, INT16 y, INT16 Width, INT16 Conv)	Initializes single writing to the display unit
Display (INT16 y, INT16 y, INT16 Width, INT16 Conv)	Continuous display on the display unit
Edit (INT16 y, INT16 y, INT16 Width, INT16 Conv)	Editing field on the display unit
...additional methods within this document...	

Fig. 7-3: I/O handling

Method	Description
void string(STRING s, STRING format)	Converts variable into string, result in s
FLOAT float()	Converts numerical types into FLOAT
INT8 int8()	Converts numerical types into INT8
INT16 int16()	Converts numerical types into INT16
INT32 int32()	Converts numerical types into INT32
UINT8 int8()	Converts numerical types into UINT8
UINT16 int16()	Converts numerical types into UINT16
UINT32 int32()	Converts numerical types into UINT32
BOOL bool()	Converts numerical types into BOOT

Fig. 7-4: Access and conversion functions

If the converted value exceeds the valid range of the destination type, the result rolls over. Example: Conversion of 258 to a UINT8 results in a 2.

Syntax - Typical Object Extensions

Continuously Displaying a PLC Variant

```
...
INT16 Time;
...
  Time.BindPLC ("SysFB.Clock");
  Time.Display (70,0,0,0,5,0);
...
```

Outputting a PLC Variable

```
...
FLOAT Size;
...
  Size.BindPLC("PartSize");
  Size.Edit(30,10,0,0,6,2);
...
```

A complete explanation of the system commands and the meanings of the parameters can be found in the Chapter 'System Library'.

Differences to ANSI-C

Restrictions:

- Not permitted: `#if(HARDWARE==BTV04)`, x++, a =?():...
- There are no typedef, union, struct
- Arrays are restricted to one dimension

Features to increase the operational reliability of the created applications:

- Range monitoring or arrays during runtime
- There are no pointers (main source of errors in the C programs. Alternatively, open arrays can be transferred as parameters. The function is the same, but it is safely monitored by the range check function.)
- Enclosing instruction blocks in braces '{}' is mandatory.
- Stringent type check of the compiler (no automatic conversion)

Extensions:

- Object-oriented method calls
- Multi-lingual string declarations
- Multi-lingual bitmap declarations
- External resource management (version 4 or later)
- Extracting ISP variable names for MiniMap file generation

8 System Library

8.1 Static Display Commands

Text();

Basic output command for strings. This command writes the transmitted string to the screen once. The output may be overwritten by the successive commands. Normal usage of Text() is to print out the static text during the creation of a screen or to make some special outputs in a screen refresh script (using an 'if'-condition, for example).

Note: The command does not support multi-line output with automatic word wrapping. This feature is only available in conjunction with the Display() command.

```
void Text(INT16 X, INT16 Y, INT16 Width, INT16 Height,
STRING s);
```

Parameter	Description
X	Left margin of the text window
Y	Upper margin of the text window
Width	Width of the text window in pixels; strings that do not fit in are truncated 0-default: Length of the transferred string
Height	Height of the text window in pixels 0-default: Height of the transferred string
s	String to be displayed, this can be a constant: "Hello World"', a multi-lingual string or a string variable

Fig. 8-1: Text() parameters

Box();

Drawing command for a rectangle. This command can be used for creating simple graphics. The graphics can be animated if they are used in the refresh script of a screen with control-bound variables. Tip: In conjunction with a width or height of 1, this command may be used for swiftly draw orthogonal lines.

```
void Box(INT16 X, INT16 Y, INT16 Width, INT16 Height,
UINT8 Fill);
```

Parameter	Description
X, Y	Left-hand upper point of the rectangle
Width, Height	Size of the rectangle in pixels
Fill	Fill mode: 0-empty with frame 1-bold (black) 2-empty

Fig. 8-2: Box() parameters

PutPixel();

Drawing command for one pixel. The graphics can be animated if the command is used in the refresh script of a screen with control-bound variables.

```
void PutPixel(INT16 X, INT16 Y, UINT8 Fill);
```

Parameter	Description
X, Y	Coordinates of the pixel
Fill	Color: 0-white 1-black

Fig. 8-3: PutPixel() parameters

Line();

Drawing command for a line. The graphics can be animated if the command is used in the refresh script of a screen with control-bound variables.

```
void Line(INT16 X, INT16 Y, INT16 X2, INT16 Y2, UINT8
Fill);
```

Parameter	Description
X, Y	Start point of the line
X2, Y2	End point of the line
Fill	Line mode: 0-white 1-black 2-broken line

Fig. 8-4: Line() parameters

Circle();

Draw command for a circle where center and radius are specified. The graphics can be animated if the command is used in the refresh script of a screen with control-bound variables.

Note: This command takes relatively much computing time. In refresh scripts it should be used very sparingly (if possible not at all).

```
void Circle(INT16 X, INT16 Y, INT16 r, UINT8 Fill);
```

Parameter	Description
X, Y	Circle center
r	Radius in pixels
Fill	Line mode: 0-white 1-black 2-broken line

Fig. 8-5: Circle() parameters

Bitmap()

Output command for static bitmaps. This command writes the bitmap to the screen once. The output may be overwritten by the successive commands. Using the mode parameter, this command may also be used for creating special functions with overlapping bitmaps in the background and foreground.

```
void Bitmap(INT16 X, INT16 Y, INT16 Width, INT16 Height, INT16 Xofs, INT16 Yofs, BMP Bitmap, UINT8 Mode);
```

Parameter	Description
X	Left-hand margin of the bitmap window = X-position
Y	Upper margin of the bitmap window = Y-position
Width, Height	Size of the bitmap window in pixels. Bitmaps that do not fit in are truncated. A bitmap that is smaller than the designated area appears in the upper left position. The remaining area is white. 0-default: Size of the transferred bitmap.
Xofs, Yofs	Offset within the bitmap. This is the upper left point of the bitmap that is displayed at the X,Y position.
Bitmap	Identifier of the bitmap declared in the resource editor (optionally multi-lingual)
Mode	Bitmap copy mode: 0 – Bitmap is drawn in overwriting mode 1 – ORed interconnection with the current screen contents; corresponds to a transparent representation (see also Windows Paint) 2 – XOR interconnection with the current screen contents. Using the bitmap as a mask, the current background is inverted.

Fig. 8-6: Bitmap() Parameter

BitmapWidth()

Determines the width of the transferred bitmap in pixels.

```
INT16 BitmapWidth(BMP Bitmap);
```

BitmapHeight()

Determines the height of the transferred bitmap in pixels.

```
INT16 BitmapHeight(BMP Bitmap);
```

ScreenUpdate()

The ScreenManager never updates the screen contents before a script has completely been processed. This avoids, for example, a flickering screen when overlapping areas are output cyclically. If, for example, a progress bar shall be output for a routine that takes some time to complete, this function can be used for forcing an advanced screen update during script runtime.

```
void ScreenUpdate();
```

8.2 Screen Commands

Screen();

This command is used for clearing the screen, removing all existing cyclic inputs and outputs, and canceling cyclic links to control variables. If the command is used within a script, that script becomes a SCREEN-generating script. Below, this is simply referred to as SCREEN.

This command initiates the following actions in the system:

1. Clearing the screen.
2. Clearing all editing and display fields of the previous screen. The Edit() and Display() call made before the first Screen() call remain active.
3. Clearing the variable links to the controller that were established in the last screen. Global links that were created before the first invocation of Screen() remain.
4. Resetting font and text attributes to the default values.
5. Deactivating an active refresh script.
6. Clearing local events and timer events.
7. Clearing the local key definitions.
8. Saving the currently executing script function as the current screen refresh script. The saved function is invoked for the screen when the user presses the ESC key to jump backwards.

```
void Screen();
```

The Screen() command does not possess any parameters.

Caution: This command may only be used within script functions without parameters. Otherwise, ESC (= ScreenBack) would lead to a script execution error.

Key();

This command interconnects a key and a script function. The simplest example would be binding of a function key to a new script. That is how the whole screen system works.

Every key can have up to two script functions assigned, one for pressing and one for releasing the key. The definition remains global if Key() is used before the first Screen() command. All global key bindings may be redefined in a screen. When the key is pressed, only the last function connected with it is executed. The global connection remains active outside the screen or within another screen, where that key is not assigned. The original function of the key will no longer be available if the key functions that have firmly been encoded as numbers or edit/cursor commands are redefined.

A maximum of 50 key assignments can be active at the same time.

```
void Key(UINT8 Key, UINT8 UpDown, PTR ScriptFunction);
```

Parameter	Description
Key	Key number or 0 for 'any key script'
UpDown	0-release key 1-press key

ScriptFunction	Name of the script function. This function need not have any parameters and need not return a value. Typical definition:
	void MyScript()
	{
	}

Fig. 8-7: Key() parameters

Script for Any Key

If zero is used as a key number, the script function is called with any key in conjunction with the normal key function.

In this special case, the normal key function is not overloaded. Each key function can have several scripts assigned. All these scripts are called one after another, and are not overloaded.

GetKey();

The GetKey() function returns the value of the key that is currently pressed. It may be used in key script functions to inquire for the actual key. The return value is zero if a key is not pressed.

```
UINT8 GetKey();
```

GetKey() does not possess any parameters.

Example of a program that transfers all keys to the PLC:

```
void MyClick()
{
   UINT8 x;

   x = GetKey();
   x.WritePLC(_PLC("BTV_Key"));
}


void main()
{
   ConnectPLC();
...
   Key ( 0, 1, MyClick ); // called if any key is pressed
   Key ( 0, 0, MyClick ); // called if any key is released
                          // to write a zero to the PLC
}
```

RefreshScript();

This command binds the passed function to the current screen as a refresh script. This function is always called before the screen is updated. Depending on a variable value, this script can be used to show animated graphics or to display different text messages. This script must not be too complex, since it is executed at a very high repetition rate (cycle time of approximately 100-200ms).

Caution: Only use static output functions and calculations within this cyclic script. Using edit, display, key or bind commands is not permitted, and would result in a system lock-up. A password command would not make sense either in this context.

```
void RefreshScript(PTR ScriptFunction);
```

Parameter	Description
ScriptFunction	Name of the script function. This function may not have any parameters and can not return a value. Typical definition: void MyScript() { ... }

Fig. 8-8: RefreshScript() parameters

Password();

This command blocks the execution of a script until the user enters the password that is required for the defined level. The execution of the actual script is terminated if the password is incorrect. This command is normally used to protect a screen or a whole submenu. To protect a screen, this command must be placed after Screen().

For repeated entries into password-protected screens, the user is prompted for the password only once. Any further occurrences of the Password() command, at the same or a lower level, do not result in another password prompt. The user will only be prompted for the password when a screen is accessed that requires a higher password level than the one the user is currently in.

The active password is cleared:

- using the ResetPassword() command, that is normally mapped to a key in a menu screen;
- if a key has not been pressed for 30 minutes. This time is adjustable. It is the same that is selected in the parameters as the ‚Screensaver' time.

```
void Password(UINT8 Level);
```

Parameter	Description
Level	Password level: 0-inactive 1-highest level, password one required 2-middle level, password one or two required 3-lowest level, password one, two or three required

Fig. 8-9: Password() parameters

Note: If the password in the system setting is empty, no prompt occurs!

ResetPassword();

This command resets the current password level. It shall be used to lock the unit before a supervisor leaves the machine. The command can be mapped on a key or, for example, be invoked in the main menu.

```
void ResetPassword();
```

The ResetPassword() command does not possess any parameters.

Quit();

This command terminates a running script execution immediately, irrespective of the currently active call level. In contrast to "return" you don not exit the current function, but the entire invocation that resulted from a Key, Menu item, Event....

```
void Quit();
```

The Quit() command does not possess any parameters.

ScreenBack();

This command calls the previous screen. This system function may be assigned to a function key using the Key() command. To create a user-friendly application, the same key should be used for this function in every screen. We recommend a global assignment to the last function key (BTV04 = F6, others F8) and, if possible, a corresponding note on every screen (e.g.: F6-back). However, the system already provides a default solution of this functionality with the ESC key.

The functionality is provided by a previous screen list with ten entries. This means that the return path is available for the last ten screens. Every time ScreenBack() is called, the last entry is erased. To build up a tree structure and guarantee the function of this command, the return path must always be programmed with ScreenBack(). If lateral jumps between the images are permitted in the menu structure, this function can prove insufficient for reaching the main menu. Therefore, you should always assign the application main menu to the ‚Main Menu' key.

```
void ScreenBack();
```

The ScreenBack() command does not possess any parameters.

EditSetBehavior();

This command toggles the editor behavior between ScreenManager and CLM emulations:

```
void EditSetBehavior(UINT8 Mode, UINT8 RemainInField,
UINT8 Reserved2);
```

Parameter	Description
Mode	SCREENMANAGER default: right/left keys to scroll in lists, up/down to leave a list. The cursor remains on the same line CLM/ELC behavior right/left keys to scroll in lists, up/down to leave a list. The cursor moves from the end of one line to the next.
RemainInField	SCREENMANAGER default: The cursor jumps to the next field when it reaches the right-hand of an input field. The cursor remains in the field when 1 is specified. This mode only has an effect on the edit fields that are initialized afterwards. Thus, this behavior can selectively be determined for specific fields only.
Reserved2	reserved

Fig. 8-10: EditSetBehavior() parameters

EditFieldOptions();

This command sets the options for the edit field last created.

```
void EditFieldOptions(UINT8 WriteDirect, UINT8
StartCell, UINT8 FilterAnd);
```

Parameter	Description
WriteDirect	ScreenManager default, the output value is written to the ScreenManager variable after OK has been pressed. The value is written to the ScreenManager variable immediately after the edit field has been left.
StartCell	This field is the first one after opening the screen.
FilterAnd	Special option for CLM parameters; the filter string is smartly ANDed with the existing contents of the field. Filter and previous contents can only be modified if they are compatible with each other. The cursor skips any position that does not satisfy this requirement. Exception for character input: +,- can only be overwritten by +, -

Fig. 8-11: EditFieldOptions() parameters

MenuItem();

This command shows a menu item on the screen. This menu item can be selected using the arrow keys. Selecting the menu item is the same process as selecting an edit field, but the text cannot be modified.

The specified script will be called when the OK key is pressed while this field is active.

```
void MenuItem(INT16 X, INT16 Y, INT16 Width, INT16
Height, STRING s, PTR Script);
```

Parameter	Description
X	Left margin of the text window
Y	Upper margin of the text window
Width	Width of the text window in pixels; strings that do not fit in are truncated 0-default: Length of the transferred string
Height	Height of the text window in pixels 0-default: Height of the transferred string
s	String to be displayed, this can be a constant: "Hello World"', a multi-lingual string or a string variable Like with the Display() command, string variables are updated upon every change.
Script	Function to call if OK is pressed on this field.

Fig. 8-12: MenuItem() parameters

Typical menu screen

```
void MyMenu()
{
   Screen();
   TextAttr(1,1,0,0,1);
   Text(0,0,128,8,"menu");
   TextAttr(1,0,0,0,0);
   MenuItem(10,10,100,0,"1st SubSubmenu", SubScreen1);
   MenuItem(10,20,100,0,"2st SubSubmenu", SubScreen2);
   MenuItem(10,30,100,0,"3st SubSubmenu", SubScreen3);
   MenuItem(10,40,100,0,"4st SubSubmenu", SubScreen4);
}
```

ScrollEvent();

This command calls an event when the cursor reaches the upper or lower border of the screen. This event permits a list to be scrolled on the screen.

```
void ScrollEvent (PTR UpScript, PTR DownScript);
```

Parameter	Description
UpScript	Function is called when the upper margin is reached
DownScript	Function is called when the lower margin is reached

Fig. 8-13: ScrollEvent() parameters

GetCursor();

This function returns the actual cursor position inside an edit field or the currently selected item inside an edit list. This command may be used to program an online user information, such as a help line for a selected command. If programmed in a refresh script, the help line will be updated without pressing OK.

```
UINT16 GetCursor();
```

GetFieldNo();

This function returns the currently selected field number. The fields are numbered in the in order of their appearance in the script. The first one gets the index 0. This command may be used to program an online user information, e. g. a help line for the currently selected field in a refresh script or to react inside a event script in a different way, depending on the actual field position.

```
UINT16 GetFieldNo();
```

8.3 Trapping Error Messages

OnErrorCall();

This function installs the transferred script function as an error handling routine. The transferred function is invoked whenever a communication error occurs. Like the Key or Event commands, this definition is global when it occurs before the first call of "Screen()". It is local when it is invoked within a screen.

```
void OnErrorCall(PTR ErrorScript);
```

Parameter	Description
ErrorScript	Separate error handling routine. This function may not have any parameters and not return any data.

Fig. 8-14: OnErrorCall() parameter

Note: The error handling routine will also be invoked if a serial fault occurs while script processing is waiting for the controller via a WaitCLC() (...CLM,..PLC etc.) command. This is the sole case where a running (waiting...) script can be interrupted by another one. The resulting side effects (modified global variables, for example) within the error script must be taken into account.

Warning: Only static outputs are permitted within an error handling script. Calling Edit, Display or Bind/Read commands is not permitted. The same applies to the Wait...() command that may lead to a new invocation of the error handling script.

GetError();

This is the first call in each error handling script. It provides the application with the main information about the occurred error. Unless it is a serial time-out, the connected controller usually has supplied an error number and in certain cases a text that is returned here. In the event of a serial time-out (controller does not reply) or for binary protocols, this text is produced by the ScreenManager Runtime program. The text types are specified in the ScreenManager Runtime documents and at the end of this document.

```
UINT16 GetError(STRING Text);
```

Parameter	Description
Text	Error text from a controller or from ScreenManager Runtime.

Fig. 8-15: GetError() parameter

The return value is the (controller-related) error number – provided that the controller provides a number.

ClearError();

This command can be used for resetting the error state of the message frame. If a serial fault is not reset, the User Error Script will immediately be followed by the ScreenManager's own error handling routine from Runtime. It is thus possible to trap only certain errors and to have all the other errors still be handled by the system.

```
void ClearError(BOOL Retry);
```

Parameter	Description
Retry	This parameter can be used to specify whether the packet that caused the error shall again be sent to the controller or be discarded.

Fig. 8-16: ClearError() parameter

ClearError() does not possess a return value.

GetErrorTelegrams();

This command is used by the application for reading the serial message frames that caused the error, and for processing or displaying them.

```
UINT16 GetErrorTelegrams(STRING Text1, STRING Text2);
```

Parameter	Description
Text1	Raw form of the data packet that was transferred to the controller. With binary protocols (such as SIS), this string only contains binary data.
Text2	Same as Text1, but this is the reply packet received from the controller.

Fig. 8-17: GetErrorTelegrams() parameters

GetErrorTelegrams() does not possess a return value.

ExceptionWindow();

ExceptionWindow() stores the current screen contents for future restoring with CloseWindow(). To display the error message, outputs can be made on the entire screen. These outputs can be removed after an acknowledgement.

```
void ExceptionWindow();
```

ExceptionWindow() does not possess a return value.

Note: The system only manages a buffer for the restoration of screen contents. ExceptionWindow() may therefore be invoked only once. Using CloseWindow, the buffer must be released before it is invoked again. Stacking several windows is not possible.

CloseWindow();

This command restores the screen contents that were saved with ExceptionWindow().

```
void CloseWindow();
```

CloseWindow() does not possess a return value.

WaitKey();

This is the sole ScreenManager command that permits keys to be programmed sequentially. The program stops here and waits for a user input. It is used for providing the user with the option of, for example, pressing either ESC to discard the serial message frame that caused the error or a different key to retain it.

```
UINT8 WaitKey();
```

The scan code of the pressed key is the return value.

Warning: This command is exclusively intended for acknowledging fatal errors. With the exception of I/O mapper and real-time communication, this command blocks the entire operating system and waits for a key being hit. The same applies to events and timer events! It should therefore never be invoked in a normal script.

8.4 Auxiliary Commands

TextAttr();

This command sets the output mode for all subsequent display commands (Text, Edit, Display). All options remain active until the next TextAttr() or Screen() command sets new options or a Screen() command restores the default values. Default is ’0’ for all parameters.

```
TextAttr(INT8 Charset, UINT8 Revers, UINT8 Bold, UINT8 Italics, UINT8 Alignment);
```

Parameter	Description
Charset	New font, currently available 0-system font 6x8 pixels 1-proportionally spaced font 2..6x8 pixels
Revers	Display in reverse video (black background, white letters)
Bold	Bold print
Italics	Italic print
Alignment	Text alignment: 0-standard alignment (text left-justified, numbers right-justified) 1-text centered 2-right-justified 3-left-justified (including numbers)

Fig. 8-18: TextAttr() parameters

EditOK();

This command corresponds to the predefined function of the OK button. All edited variables of the active screen are accepted permanently and are transmitted to a controller if they have been connected with a bind command.

This function enables the user to redefine the OK key with its own OK script, thus programming a range verification or any other action that must be performed after the OK button has been pressed. To be able to complete the process of setting and writing the variables, the programmer must be able to make the origin OK call once all conditions have been satisfied.

```
void EditOK();
```

EditOK() does not possess any parameters.

EditESC();

This command corresponds to the predefined function of the ESC button. Calling EditESC() restores the previous values in all edit fields of a screen. The variables displayed in italics are displayed normally again after this command has been executed. Like EditOK(), it is usually employed in a script that was assigned to the ESC key. EditESC() is invoked if the required input conditions are not satisfied.

```
void EditESC();
```

EditESC() does not possess any parameters.

StrCopy();

This command copies one string into another one. The source string is truncated if the destination string is too short.

```
void StrCopy (STRING Dest, STRING Source);
```

Parameter	Description
Dest	Destination string, must be a variable.
Source	Source string, may be a variable or a constant.

Fig. 8-19: StrCopy() parameters

StrAdd();

This command adds a second string to the destination parameter. The result is truncated if the destination string is too short.

```
void StrAdd (STRING Dest, STRING SubStr);
```

Parameter	Description
Dest	The destination string, to which the SubStr is added, must be a variable.
Source	SubStr string, may be a variable or a constant.

Fig. 8-20: StrAdd() parameters

To guarantee the compatibility to the preceding version, also function

```
UINT8 strcat (STRING Dest, STRING SubStr);
```

is provided.

StrCopyPart();

This command copies a part of the source string to a destination string. The result is truncated if the destination string proves too short.

```
void StrCopyPart (STRING Dest, STRING Source, UINT16
Start, UINT16 Length);
```

Parameter	Description
Dest	The destination string, to which the SubStr is added, must be a variable.
Source	SubStr string, may be a variable or a constant.
Start	Start position in the source string. The first position is 0.
Length	Length of the substring that shall be copied.

Fig. 8-21: StrCopyPart() parameters

StrGetASCII();

Converts the ASCII value of one character in a string into a UINT8 value.

```
UINT8 StrGetASCII (STRING s, UINT8 Position);
```

Parameter	Description
S	Input string
Position	Position of the character

Fig. 8-22: StrGetASCII() parameters

To guarantee the compatibility to the preceding version, also function

```
UINT8 strGetASCII (STRING s, UINT8 Position, UINT8
Value);
```

is provided.

StrSetASCII();

Sets the ASCII value of a character in the transferred string from the transferred UINT8 value.

```
UINT8 StrSetASCII (STRING s, UINT8 Position, UINT8 Value);
```

Parameter	Description
S	Input string
Position	Position of the character
Value	New value for the specified character

Fig. 8-23: StrSetASCII() parameters

To guarantee the compatibility to the preceding version, also function

```
UINT8 strSetASCII (STRING s, UINT8 Position, UINT8 Value);
```

is provided.

SetIO();

Accesses a single I/O register bit. This command may be used to switch an LED or port and to communicate with the I/O mapper.

```
void SetIO ( INT16 Number, UINT8 Bit, BOOL value );
```

Parameter	Description
Number	Register word
Bit	Bit in this word (0...15)
Value	New value

Fig. 8-24: SetIO() parameters

SetIOegister();

Accesses a complete 16-bit I/O register word. This command can be used, for example, for switching several LEDs or ports at the same time and for communicating with the I/O mapper.

```
void SetIORegister ( INT16 Number, UINT16 value );
```

Parameter	Description
Number	Register word
Value	New value

Fig. 8-25: SetIORegister() parameters

GetIO();

Read access to an individual I/O register bit.

```
BOOL GetIO ( INT16 Number, UINT8 Bit );
```

Parameter	Description
Number	Register word
Bit	Bit in this word (0...15)

Fig. 8-26: GetIO() parameters

Returned value: Status of the selected bit.

GetIORegister();

Read access to a 16-bit I/O register word.

```
UINT16 GetIORegister ( INT16 Number );
```

Parameter	Description
Number	Register word

Fig. 8-27: GetIORegister() parameter

Returned value: Status of the selected register word.

BindIORegister();

Cyclic read access to a 16-bit I/O register word in a variable. Thus, the ScreenManager variable has the value of the specified I/O register assigned in each system cycle. There is no return influence of the I/O register (like with the Bind commands to a controller).

```
Identifier.BindIORegister ( INT16 Number );
```

Parameter	Description
Number	Register word

Fig. 8-28: BindIORegister() parameter

GetLanguage()

Determines the selected language

```
UINT8 GetLanguage();
```

The returned value is the language number set in the parameters.

SetLanguage()

Selecting the currently used language.

```
void SetLanguage(UINT8 NewLanguage);
```

Parameter	Description
Number	Register word

Fig. 8-29: SetLanguage() parameter

The transferred value is temporarily set until the control panel is switched off. Use the SaveParameter function, that is described below, if you wish to save the value permanently.

SaveParameter()

The temporarily modified parameters are written in the flash.

```
void SaveParameter();
```

SaveParameter writes the modified parameters into the flash without restarting the control panel. This process may take up to two seconds. The system does not respond to programmed events during that time. Provided the hardware permits it, real-time communication and I/O mapper remain active during that time.

Typical Voice Selection in an Application:

```
UINT8 ml;
void LanguageOK()
{
      EditOK();
      SetLanguage(ml);
      SaveParameter();
}

void SelectLanguage()
{
      Screen();
      ml = GetLanguage();
      ml.Edit(70,10,0,0,0,0);
      TextAttr(1,0,0,0,0);
      Text(0,10,0,0,"Language:");
      Key(KEY_OK,KEY_PRESSED,LanguageOK);
}
```

rand();

Random number generator

```
INT16 rand();
```

The returned value is a random number between 0 and 32768. Like in ANSI C, this is a pseudo random number sequence that supplies the numbers always in the same sequence after the software has been started.

run_error();

Aborts script processing in the event of a fatal error. This function can be used for testing own applications and for trapping fatal execution errors.

```
void run_error();
```

If the application is aborted and the message "unknown system call" issued, pressing any key reboots the system.

WaitCursor()

```
void WaitCursor();
```

Activates the waiting cursor 

This command, which shall signal to the user that the system is used to full capacity, can be invoked at any point in a compute-bound script. It remains active until the next screen update takes place, and need not be switched off explicitly. This waiting cursor is activated automatically when the Screen() system function or some other system function that causes waiting times is invoked.

TimerEvent();

This function can be used for triggering a script function by a timer event. At a resolution of 1 ms, the accuracy is in the range of one per cent. The calls are made at a jitter that is of the magnitude of the system cycle (typically 100...200 ms). Cyclic calls are fully synchronized. The omitted timer calls are ‚made up' in immediate succession if the system is delayed by waiting commands or longer scripts, or if the specified period is shorter than the system cycle.

Examples:

1. A cyclic event with a period of 800ms leads to 100 calls in 80 seconds. The distances between the individual calls may vary between 600 and 1000ms.
2. A cyclic event with a period of 10ms (significantly shorter than the system cycle – but still valid) would be called ten times in succession every 100ms at a system cycle of 100ms.

Attention: Only static output functions and calculations should be used within a timer script. Using edit, display, key or bind commands is not permitted, and would result in a system lock-up. A password command would not make sense either in this context. If a screen is activated with a timer script it must be ensured that this does not happen periodically. In this case, the user would no longer be able to operate the system.

Note: The system is slowed down (without getting hooked up) if the script repetition time is too short or the total execution time of all timer systems is too long. However, some periodical calls may be skipped (they will be clipped if the time account of the timer is more than one hour overdue). The maximum computing time of timer scripts are limited to 50ms per system cycle.

A single timer event is deleted automatically as soon as it has been completed. Cyclic timers are deleted analogously to the event: Global cyclic timers are implemented before the first Screen() invocation and remain active until power is switched off. Local timers are cleared after the screen is exited. The number of simultaneously active timers is limited to 18. Further invocations of TimerEvent() are ignored.

Note: If the screen changes in a timer script, it can happen that the remaining TimerEvents that belong to the previously active screen are called again.

```
TimerEvent (UINT32 TimeValue, UINT8 Mode, PTR
ScriptFunction);
```

Parameter	Description
TimeValue	Time value in ms (1...2,14 x 10^9 = half the value range of UINT32) Like a refresh script, special variant events with TimeValue = 0 are invoked once per system cycle.
Mode	0 - single timer event after the specified time 1 – periodic timer event

ScriptFunction	Name of the script function. This function need not have any parameters and need not return a value. Typical definition:
	void MyScript()
	{
	}

Fig. 8-30: TimerEvent() parameters

ScreenLock()

This command locks the keyboard of the device. The user must type in at least one password to continue operator input.

Attention: The machine control keys remain active if they are connected to the controller via the I/O mapper. In this case they are working as fast direct-mapped keys without any control over the HMI application.

```
void ScreenLock();
```

ScreenLock() does not possess any parameters.

8.5 System Screens

ParameterScreen();

This command activates the system parameter screen. Programming is fixed in the runtime software and may only be linked to the menu system by using the command (Shift-Help is used only for this example):

```
Key (KEY_SHIFT_HELP,KEY_PRESSED,ParameterScreen).
```

```
void ParameterScreen();
```

ParameterScreen() does not possess any parameters. In order to be able during operation to access interface diagnosis and the counters that have been accumulated there, making the parameters always accessible via the menu system is strongly recommended. Although a restart (and holding the F2 key) that would otherwise be required would also lead to the parameters, all communication and error counters were reset in this case.

RelayScreen();

This command activates the relay mode screen from the parameter interface menu. This permits relay mode to be integrated in own applications without the necessity of running through the entire menu tree of the control panel parameters.

```
void RelayScreen();
```

RelayScreen() does not possess any parameters.

SwitchRelayMode();

This command activates relay mode without displaying the system screen for it. Relay mode remains active until it is switched off again with this command. This permits own relay mode screens to be programmed or this mode to be triggered automatically as a function of external events (e. g. keyswitch...).

The normal serial communication between control field and connected controller rests as long as this mode is active.

```
void SwitchRelayMode(UINT8 On);
```

Parameter	Description
ON=0	Deactivates the Relay Mode.
ON=1	Activates the Relay Mode as long as this function is called again with parameter ON=0.
ON=2	Activates the Relay Mode automatically, if a prompt exists. The internal communication is interrupted for 1 s.

Fig. 8-31: SwitchRelayMode() parameter

Note: When programming an own relay screen you must ensure that the operator cannot exit the screen (using ESC for example) without resetting the mode.

FileSelect();

This command opens the File Manager of the control panel as a file selection menu. The list of the displayed files is restricted to the types that are specified with an extension. Furthermore, the text specified for extension is used as the heading of the file selection.

```
void FileSelect(STRING Name, STRING Extension);
```

Parameter	Description
Name	String for the selection result. Is modified within the FileSelect() function. An empty string is returned when the selection is aborted with ESC.
Extension	Extension of the files to be listed. At the same time, this string is the heading of the selection dialog. Files without extension cannot be selected.

Fig. 8-32: FileSelect() parameters

Example:

```
STRING FileName[80];
void ScreenFileSelect()
{
      FileSelect(FileName,"CLM Program File");
}

void MyScreen()
{
Screen();
...
MenueItem(10,10,0,0,"File Select", ScreenFileSelect);
...
}
```

8.6 Variable Methods

Edit();

The Edit() command initializes an edit field and binds the variable to it. The transferred variable must be global. An edit command with local variables may cause unexpected results or runtime errors. The Screen() command must be executed before Edit() is invoked.

As long as an editing field has not yet been selected, the value may as well be updated by an external controller or an event script. If the cursor is or was in an edit field, the value is copied to an internal buffer. Editing the value does not affect the variable until OK is pressed. OK stores the value in the variable and initiates a serial transfer, if necessary. Pressing ESC does not change the value but redisplays the original value. Both functions may be redefined by a user script to modify the editor's response.

An Edit() field remains active on the screen until the next Screen() call is made. If an Edit() call is made before the very first Screen() it remains active until the system is turned off (seems only expedient in exceptional cases).

Caution: Using the edit command is only allowed in conjunction with global variables and may be used only within screen-creation scripts (i.e. functions that contain the Screen() command). This command may never be used in cyclically invoked scripts (such as Timer, Event, RefreshScript).

Note: The system creates a frame around the area specified by X, Y, Width and Height. This frame must be taken into account when designing the image. Example: The system does not output a position of 0,0 since the frame would begin at –1. A field that would go beyond a margin will not be created.

```
Identifier.Edit(INT16 X, INT16 Y, INT16 Width, INT16
Height, INT16 Character, INT16 Conv);
```

Parameter	Description
X	Left-hand text margin
Y	Upper text margin
Width	Width of the text window in pixels. Character strings that do not fit in are truncated. 0-default: Width of the current font times the number of characters.
Height	Height of the text window in pixels 0-default: Height of the current font
Characters	Width in characters. The width is adjusted if the declared length of a transferred string is smaller. 0-automatic adjustment to the length of a string or automatic default values for numeric variables.
Conv	For FLOAT variables: Number of decimal digits. If a negative value is specified here, the values will be represented in exponential notation. Example: Conv = -4 leads to "+7.6547e+32" The field is in overtype mode. The first digit can only be overtyped by + or –, the following digits by figures. The point and the "e" are skipped. For INT variables: 0-default (decimal) 1-not defined 2-hexa 3-binary For STRING variables – filter: 0-default, 0–9, A–Z, a–z, +.-@ # and <space> 10-A–Z, a–z 11-A–Z

Fig. 8-33: Edit() parameters

Display();

Same functionality as edit, but variable updating without the capability to edit.

Caution: Do not use within timer or refresh scripts. This may cause unexpected results. The Text() command must be used to display a value from a script function.

Caution: Only allowed in conjunction with global variables and may be used only within screen-creation scripts.

```
Identifier.Display (INT16 X, INT16 Y, INT16 Width,
INT16 Height, INT16 Character, INT16 Conv);
```

Parameter	Description
X	Left-hand text margin
Y	Upper text margin
Width	Width of the text window in pixels. Character strings that do not fit in are truncated. 0-default: Width of the current font times the number of characters.
Height	Height of the text window in pixels 0-default: Height of the current font
Characters	Width in characters. The width is adjusted if the declared length of a transferred string is smaller. 0-automatic adjustment to the length of a string or automatic default values for numeric variables.
Conv	For FLOAT variables: Number of decimal digits. If a negative value is specified here, the values will be represented in exponential notation. Example: Conv = -4 leads to "+7.6547e+32" For INT variables: 0-default (decimal) 1-not defined 2-hexa 3-binary

Fig. 8-34: Display() parameters

EditFilter();

Same functionality as Edit(). This function may only be used within screen-creation functions. The transferred variable must be global.

The EditFilter() command can be used for specifying the valid characters for the individual positions. This is useful in conjunction with Indramat ELC or CLM strings.

Caution: Only permitted in conjunction with global variables and may be used only within screen-creation scripts.

```
Identifier.EditFilter(INT16 X, INT16 Y, INT16 Width,
INT16 Height, STRING Filter);
```

Parameter	Description
X	Left-hand text margin, displayed in the editing field
Y	Upper text margin, displayed in the editing field
Width	Width of the text window in pixels. Character strings that do not fit in are truncated. 0-default: Width of the current font times the number of characters.
Height	Height of the window in pixels 0-default: Height of the current font
Filter	The filter string - each position represents a character of the editing field. The meaning of the filter character is shown in the next table.

Fig. 8-35: EditFilter() parameters

Filter character	Permits input of:
0 (zero)	Figures 0–9. The line is cleared after the first input if 0 is on the first position. Same functionality as Edit() in conjunction with numeric variables.
1	Figures 0–9. The cursor steps into the edit field in overwrite mode - the old value is not cleared
7	Figure 0, '+' and '-' permitted. The cursor steps into the edit field in overwrite mode - the old value is not cleared
8	Figures 0–9 and '.' permitted. The line is cleared after the first input if 0 is on the first position.
9	Figures 0–9 '+' and '-' permitted. The line is cleared after the first input if 0 is on the first position.
+	'+' and '-' permitted. The cursor steps into the edit field in overwrite mode - the old value is not cleared
B	0,1 permitted
C	0,1,2 permitted
f	0–9, 'A'–'F' permitted; all characters in capital letters
f	0–9, 'a'–'f' permitted; all characters in lower case letters
Z	'A' – 'Z' permitted, all letters in capital letters
Z	'A' – 'Z', 'a' – 'z' permitted
X	'A' – 'Z', 0–9, '+', '-', '@', '#', '.' and ' ' (space) permitted. The space can be created using the <DEL> key, the special characters using the <2/VW> key.
X	Like X, lower case letters ('a' – 'z') are permitted, too.
K	ELC/CLM specific: '+' and '-' permit a numerical value to be entered. The numerical format depends on the rest of the format string. 'V' permits the input of ELC/CLM variable identifiers that begin with 'V' plus three digits (e.g.'V001').
V	ELC/CLM specific: 0–9 permit any numerical value to be entered. The numerical format depends on the rest of the format string. 'V' permits the input of ELC/CLM variable identifiers that begin with 'V' plus three digits (e.g.'V001').
any other character	This is not a position that can be edited. The cursor skips these fields. This includes the '.' The programmer must ensure that a decimal point in this position already exists.

Fig. 8-36: Filter characters

EditList();

EditList() has the same functionality as Edit(). However, it is only possible to insert numerical values or text from the appended list. When this edit field is active the valid values can be selected from the transferred character list using the cursor keys.

The values from the character list are assigned if Mode = 0 is selected. If the variable contained an invalid value (not contained in the list), this value will be displayed, but not corrected. Hitting the cursor keys inside the field immediately switches over to the first valid list value.

EditList() shows the corresponding list entry instead of the variable if Mode = 1 is selected. The command can be used with any number type. The list begins with the entry zero.

Caution: Only permitted in conjunction with global variables and may be used only within screen-creation scripts.

```
Identifier.EditList(INT16 X, INT16 Y, INT16 Width,
INT16 Height, INT16 Character, INT16 Mode, STRING
List);
```

Parameter	Description
X	Left-hand text margin, displayed in the editing field
Y	Upper text margin, displayed in the editing field
Width	Width of the text window in pixels. Character strings that do not fit in are truncated. 0-default: Width of the current font times the number of characters.
Height	Height of the window in pixels 0-default: Height of the current font
Characters	Width of an entry in characters. This value must be defined.
Mode	0-display variable. Same functionality as Display(). 1-display text entry that corresponds to the numerical value of 'Identifier'. The result can be undefined if the identifier is of the FLOAT or STRING type or if it exceeds the number of list entries.
List	List of the entries as a character string. Each entry must be of the same length. If necessary, it must be filled up with blanks. Typical list of binary states (Width=3): "300..1200.1920038400" . is used for visualizing the blanks.

Fig. 8-37: EditList() parameters

Example:

```
Baud.EditList(50,10,40,10,5,0,"300..9600.1920038400");
```

This example places an edit field at the position 50,10. This field shows the actual value of 'Baud'. When it is edited the user can select one of the four baud rates out of the list with the cursor keys. The input of other values is not possible.

DisplayList();

DisplayList() has the same functionality as EditList(). Values can only be displayed here, not edited. Although TableMode = 0 would work, there would be no difference to Display(). DisplayList() shows the corresponding text entry instead of the variable value if TableMode = 1 is selected. The command can be used with any ordinal type. The first list element is assigned to zero.

Caution: Only permitted in conjunction with global variables and may be used only within screen-creation scripts.

```
Identifier.DisplayList(INT16 X, INT16 Y, INT16 Width,
INT16 Height, INT16 Character, INT16 Mode, STRING
List);
```

Parameter	Description
X	Left-hand text margin, displayed in the editing field
Y	Upper text margin, displayed in the editing field
Width	Width of the text window in pixels. Character strings that do not fit in are truncated. 0-default: Width of the current font times the number of characters.
Height	Height of the text window in pixels 0-default: Height of the current font
Characters	Width of an entry in characters. This value must be defined.
Mode	0-display variable. Same functionality as Display(). 1-display text entry that corresponds to the numerical value of 'Identifier'. The result can be undefined if the identifier is of the FLOAT or STRING type or if it exceeds the number of list entries.
List	List of the entries as a character string. Each entry must be of the same length. If necessary, it must be filled up with blanks. Typical list of binary states (Width=3): `"OffOn."` `.` is used for representing the blank that is recommended here. Even the last entry must be filled up to the defined length.

Fig. 8-38: DisplayList() parameters

Example: `a.DisplayList(100,5,20,10,3,1,"OffOn.");`

Displays the text `"Off"` if "a" is equal to zero. Shows the text `"On"` if "a" is equal to 1.

KonvStr();

Converts any numerical type into a string.

```
Identifier.KonvStr(STRING Dest, STRING Filter);
```

Parameter	Description
Dest	Destination string to store the result.
Filter	Represents the format string. The meanings of the individual format characters are explained in the following table.

Fig. 8-39: KonvStr() Parameter

Parameter	Function
0 (zero), 8, 9	Conversion into number, right-justified, without leading zeros "000000.00" > [23.55]
1	Conversion into decimal numbers with leading zeros. "111111.11" > [000023.55]
.	Position of the decimal point
K	Conversion into decimal numbers with leading zeros and leading +/- "+1111111.111" > [+1234456.123].
8	Figures 0–9 and '.' permitted. The line is cleared after the first input if 0 is on the first position.
9	Figures 0–9 '+' and '-' permitted. The line is cleared after the first input if 0 is on the first position.
+	'+' and '-' permitted. The cursor steps into the edit field in overwrite mode - the old value is not cleared
B	Conversion into binary string "BBBBBBBBB" > [011100101]
F	Hexadecimal string with capital letters "FFFFFF" > [A8D4FF]
f	Hexadecimal string with lower case letters "ffffff" > [a8d4ff]
any other character	Do not use other characters, this may lead to unexpected results.

Fig. 8-40: Format characters

Example:

```
STRING MyString[30];
FLOAT  NewPostion;
...
{
   NewPosition.KonvStr ( MyStr, "+111111.11" );
// N0007 must be a POI !
   MyStr.WriteCLM(1,"N0007",14,23);
...
```

Event();

This function may be used to monitor a variable and to call a script if the variable changes in the specified way. This command can be used for monitoring the limits of values and for displaying a warning or a screen that depends on the state of a variable or an I/O bit.

Caution: The variable must be a global UINT16.

```
Identifier.Event (UINT16 Value, UINT8 Condition, PTR
ScriptFunction);
```

Parameter	Description
Value	Value to compare
Condition	0 - compare for = 1 - compare for bit = 1: (variable & value) != 0 2 - greater than (variable > value) 3 - less than (variable < value) 4 - greater than or equal to (variable >= value) 5 - less than or equal to (variable <= value) 6 - compare for not equal to 7 - Event upon each change (a new value is set after each event) 8 - comparisons for bits cleared: (variable & value) == 0

ScriptFunction	Name of the script function. This function need not have any parameters and need not return a value. Typical definition:
	void MyScript()
	{
	}

Fig. 8-41: Event() parameters

Example of Switching a Screen Using the "PicNum" PLC Variable:

```
UINT16 ScreenNum = 0;

....definition of Screen_1, Screen_33 and Screen_8

void main()
{
...
   ScreenNum.BindPLC(_PLC("PicNum"));

   ScreenNum.Event(1,0,Screen_1);
   ScreenNum.Event(33,0,Screen_33);
   ScreenNum.Event(8,0,Screen_8);

}
```

The selected image number is activated whenever the PLC variable changes.

8.7 Storage List Functions

A list is a dynamic storage class. It can contain any data type in a two dimensional array. Each list element can be read or written to as required. See the explanations and examples in the corresponding chapter of the Section ‚Creating User Programs'.

Prior to using it, the system must employ GetFreeList() to request or create the list. After it has been used, it must be released with CloseList(). All lists are global. If releasing is omitted and further lists are created, the system will abort execution and issue a runtime error at the 50th list.

A single command can be used for storing a complete list as a file in the flash. This file can then be reopened as a flash list. However, this flash list can only be read.

GetFreeList();

Gets the access number (= handle) for a free list from the system. This handle must be used for all subsequent access operations as a parameter.

Caution: There are system calls such as ConnectPLC and ConnectELC that may also use lists to store data. To ensure that their data remains valid you must use this function to get a new empty list from the system.

```
UINT16 GetFreeList();
```

GetFreeList() does not possess any parameters.

Return value: Handle for a free list.

Example of using the "GetFreeList()" command:

```
UINT16 MyList;

void main()
{
...
   MyList = GetFreeList();
...
   x.WriteList(MyList,0,0);
...
}
```

FreeMemory();

This function returns the number of free bytes in the system memory that are available for the user program. This dynamic memory may be used by the following commands: WriteList(), InitProgListELC(), ConnectPLC().

```
UINT32 FreeMemory();
```

FreeMemory() does no possess any parameters.

Return value: Number of free bytes in the RAM system.

Caution: Calling functions that occupy more system memory than available lead to a run-time error. In this case, you must try to release (using the EraseList() command) some lists after they have been used in order to reduce the simultaneously required memory space.

ReadList();

Reads a single element from the list.

```
Identifier.ReadList(UINT16 ListNo, UINT16 Line, UINT16
Element);
```

Parameter	Description
ListNo	The list number (handle) obtained from GetFreeList() or FlashOpenList() identifies the list.
Line	Line with elements
Element	Column with elements

Fig. 8-42: ReadList() parameters

Although the identifier may be any ScreenManager variable type, correct results can only be obtained if it is of the same or a bigger size than the type used for writing. Number types are converted automatically.

WriteList();

Overwrites a single element of the list. The list is expanded accordingly if a line or a column is outside the existing range.

```
Identifier.WriteList(UINT16 ListNo, UINT16 Line,
UINT16 Element);
```

Parameter	Description
ListNo	The list number (handle) obtained from GetFreeList() or FlashOpenList() identifies the list.
Line	Line with elements
Element	Column with elements

Fig. 8-43: WriteList() parameters

InsertList();

Inserts a single element into a list. The rest of the same line is shifted by one digit. The list is expanded accordingly if a line or a column is outside the existing range.

```
Identifier.InsertList(UINT16 ListNo, UINT16 Line,
UINT16 Element);
```

Parameter	Description
ListNo	The list number (handle) obtained from GetFreeList() or FlashOpenList() identifies the list.
Line	Line with elements
Element	Column with elements

Fig. 8-44: InsertList() parameters

InsertLineList()

Inserts a blank line at the specified position.

```
InsertLineList(UINT16 ListNo, UINT16 Line);
```

Parameter	Description
ListNo	The list number (handle) obtained from GetFreeList() or FlashOpenList() identifies the list.
Line	Insertion position

Fig. 8-45: InsertLineList() parameters

EraseLineList()

Deletes a line at the specified position.

```
EraseLineList(UINT16 ListNo, UINT16 Line);
```

Parameter	Description
ListNo	The list number (handle) obtained from GetFreeList() or FlashOpenList() identifies the list.
Line	Line to be cleared

Fig. 8-46: EraseLineList() parameters

EraseList(); CloseList()

Deletes the entire list and releases the memory.

```
EraseList(UINT16 ListNo);
CloseList(UINT16 ListNo);
```

Both commands have the same function.

Parameter	Description
ListNo	The list number (handle) obtained from GetFreeList() or FlashOpenList() identifies the list.

Fig. 8-47: EraseList(), CloseList() parameter

FlashStoreList();

Stores the entire list in a flash file.

```
FlashStoreList(UINT16 ListNo, STRING Name, STRING
Ext);
```

Parameter	Description
ListNo	The list number (handle) obtained from GetFreeList() or FlashOpenList() identifies the list.
Name	File name
Ext	File name extension

Fig. 8-48: FlashStoreList() parameters

FlashOpenList();

This function opens a file stored in the flash as a list that can be read. The return value is a list handle that behaves in the same way as one that is created using GetFreeList(). Write functions, however, cannot be called with it.

```
UINT16 FlashOpenList(STRING Name, STRING Ext);
```

Parameter	Description
Name	File name
Ext	File name extension

Fig. 8-49: FlashOpenList() parameters

9 Linking Variables to an Indramat PLC

9.1 PLC Variable Declaration

A PLC variable is called by its variable name that is used in the PLC program. This name is handled as a character string within the ScreenManager program. The PLC variable name must contain the complete path to the default variable type. If the variable is a part of a function block, the name of the FB instance must be added, separated by a point - as defined in the PLC language.

String Declaration of PLC Variable Names

Every string constant that represents a PLC variable name must use the following declaration syntax:

`_PLC("VarName")` => declare externally and globally in WinPCL

`_PLC("Instanz.VarName")` => complete addressing

Inside the ScreenManager program this syntax behaves like `"VarName"`. In addition, however, a file for the PLC programming software is generated. This file contains all variable names used in the ScreenManager program, and is necessary for creating the MiniMap file.

Creating a MiniMap File

In the PLC programming software the menu item 'Options'-'Status BTV' must be used to select the BTV file generated by the ScreenManager compiler that belongs to the ScreenManager application.

If the PLC Diagnostics and ProView messages are used in the ScreenManager application, the file with the same name as the current PLC project must also be selected.

The PLC programming software now creates a MiniMap file and stores it in the controller. This enables access to the PLC variables. The MiniMap file contains the names and the addresses of all variables used by the ScreenManager application.

Support of WinPCL

The INDRAMAT WinPCL is recognized automatically. ConnectPLC() always waits for a message frame from the PLC and responds according to the connected controller. However, the application must take the deviating declaration of the PLC variables. In the default setting, WinPCL waits for the complete variable name, that begins with the connected PLC instance.

9.2 PLC System Commands

The following ScreenManager system calls are used for PLC access:

ConnectPLC();

ConnectPLC() enables PLC communication. This call must be made before any Screen() call; it blocks the BTV until a PLC is connected. It is possible to make static outputs to the LCD before this call, e.g. to show a company logo.

```
void ConnectPLC();
```

ConnectPLC() does not possess any parameters.

Once the PLC has been connected, any interruption of the connection (e. g. caused by a PLC program download) causes the BTV to be restarted. To be able to simulate the application in off line mode, merely the ConnectPLC() command must be made a comment (e.g. //ConnectPLC). All the other PLC functions may be remain in the program. The are ignored and do not cause an error message to be issued.

Note: To reach the parameter screen when a PLC is not connected – or a connection cannot be established because the interface has been set incorrectly: Booting of the ScreenManager application can be suppressed with the F2 key. This leads directly into the parameter screen.

WaitPLC();

To increase the performance, all variable access procedures are performed in the background. The WaitPLC() command can be used for forcing the BTV unit to wait until all write and read access procedures of the PLC are terminated. This command must be used if equations or conditions depend on a PLC write or read command that was issued immediately before.

```
WaitPLC();
```

WaitPLC() does not possess any parameters.

Example:

```
INT16 a = 0;
{
   a.ReadPLC(_PLC("CountA"));
   Result = a * 22;
...
```

In this example, Result remains zero, because at the moment when the equation is made "a" is not read by the PLC.

A functioning example would be:

```
INT16 a = 0;
{
   a.ReadPLC(_PLC("CountA"));
   WaitPLC();
   Result = a * 22;
...
```

In this example, ‚Result' is only computed after the current variable value of a has been fetched from the PLC. Execution is slower, since the ScreenManager program blocks until all pending serial read and write commands are completed.

9.3 PLC Data Commands

BindPLC();

The BindPLC() method generates a link between two variables in the PLC and the ScreenManager. The ScreenManager variable is cyclically updated with the PLC value.

Global bindings must be made before the first Screen() call. They remain active until the next restart is performed. Every bound variable uses transmission time. Use global bindings only if necessary.

Every local binding that is made after the first Screen() call is released as soon as a new screen is activated.

Changing a variable within an equation (e. .g. a = 99;) does not initiate a command that writes the new value to the PLC. This must be done using the WritePLC(); command.

```
Identifier.BindPLC(STRING PLCName);
```

Parameter	Description
PLCName	PLC variable name

Fig. 9-1: BindPLC() parameter

Note: All numeric types are converted automatically. If the converted value exceeds the valid range of the destination type, the result rolls over. Example: Conversion of 258 to a UINT8 results in a 2.

Depending on the serial protocol definition, reading strings is limited to 45 characters. Longer PLC strings are truncated in the ScreenManager program.

Caution: Only allowed in conjunction with global variables.

The system generates a runtime error if the variable name is not found in the MiniMap file.

ReadPLC();

The value is read from the PLC only once. As with the BindPLC() command, the value is read from the PLC in background operation. The WaitPLC() command must be used if the subsequent lines of program code depend on this read value.

```
Identifier.ReadPLC(STRING PLCName);
```

Parameter	Description
PLCName	PLC variable name

Fig. 9-2: ReadPLC() parameter

Example:

```
STRING MachineName[30];
...
{
   MachineName.ReadPLC(_PLC("MyName"));
   Text(0,0,128,10,MachineName);
...
```

In this example, the name remains blank, because at the moment when text output is made "MachineName" is not read by the PLC. A functioning example would be:

```
STRING MachineName[30];
...
{
   MachineName.ReadPLC(_PLC("MyName"));
   WaitPLC();
   Text(0,0,128,10,MachineName);
...
```

The most effective implementation would be:

```
STRING MachineName[30];
...
{
   MachineName.ReadPLC(_PLC("MyName"));
   MachineName.Display(0,0,0,0,0,0);
...
```

The lines of program code are executed as fast as possible. The PLC text of "MyName" is shown as soon as it is read from the PLC.

Note: All numeric types are converted automatically. If the converted value exceeds the valid range of the destination type, the result rolls over. Example: Conversion of 258 to a UINT8 results in a 2.

Depending on the serial protocol string, reading strings is limited to a maximum of 45 characters. Longer PLC strings are truncated in the ScreenManager program.

Caution: Only allowed in conjunction with global variables.

The system generates a runtime error if the variable name is not found in the MiniMap file.

WritePLC();

A single write command to the PLC is initiated.

```
Identifier.WritePLC(STRING PLCName);
```

Parameter	Description
PLCName	PLC variable name

Fig. 9-3: WritePLC() parameter

Example:

```
INT16 PLC_Var;
...
{
   PLC_Var = 0;
   PLC_Var.WritePLC(_PLC("MyPLCVar"));
// initiates a write of 0 to PLC
   PLC_Var = 1;
   PLC_Var.WritePLC(_PLC("MyPLCVar"));
// initiates a write of 1 to PLC
   PLC_Var = 2;
   PLC_Var.WritePLC(_PLC("MyPLCVar"));
// initiates a write of 2 to PLC
```

Write commands are initiated after every write call to the PLC variable. Every write command requires one communication cycle. If a variable is changed multiple times within a short period, every change is transmitted to the PLC in the order it was entered, so that the PLC is able to track every change.

Note: All numeric types are converted automatically. If the converted value exceeds the valid range of the destination type, the result rolls over. Example: Conversion of 258 to a UINT8 results in a 2.

Depending on the serial protocol string, writing strings is limited to a maximum of 32 characters.

Caution: Only allowed in conjunction with global variables.

The system generates a runtime error if the variable name is not found in the MiniMap file.

BindProViPLC();

This functions connects a ScreenManager variable with a ProVi message function block in the PLC. The message text will be assigned in conjunction with a STRING variable. The message number will be assigned in conjunction with a numerical variable.

```
Identifier.BindProViPLC(STRING FBName, UINT16
Position);
```

Parameter	Description
FBName	Name of the ProVi function block in the PLC
Position	The number of the message in the order of priority if more than one message is active.

Fig. 9-4: BindProViPLC() parameters

Caution: Only allowed in conjunction with global variables.

Caution: This function is only supported when WinPCL V19 is used!

Typical ProVi Messages:

```
void Bild_ProVi()
{
   Screen();
   TextAttr(1,1,0,0,1);
   Text(1,1,238,10,"Startup");
   TEXT_1.BindProViPLC("FbStart2B",0);
   TEXT_2.BindProViPLC("FbStart2B",1);
   TEXT_3.BindProViPLC("FbStart2B",2);
   TEXT_4.BindProViPLC("FbStart2B",3);
   TEXT_5.BindProViPLC("FbStart2B",4);
   TEXT_6.BindProViPLC("FbStart2B",5);
   TEXT_7.BindProViPLC("FbStart2B",6);
   TEXT_8.BindProViPLC("FbStart2B",7);
   TextAttr(1,0,0,0,3);
   TEXT_1.Display(1,15,238,10,0,0);
   TEXT_2.Display(1,27,238,10,0,0);
   TEXT_3.Display(1,39,238,10,0,0);
   TEXT_4.Display(1,51,238,10,0,0);
   TEXT_5.Display(1,63,238,10,0,0);
   TEXT_6.Display(1,75,238,10,0,0);
   TEXT_7.Display(1,87,238,10,0,0);
   TEXT_8.Display(1,99,238,10,0,0);
}
```

TEXT_1...TEXT_8 must be defined as global variables.

ReadProViPLC();

Similar functionality as BindProVi() but the message is read and assigned only once (not cyclically).

```
Identifier.ReadProViPLC(STRING FBName, UINT16
Position);
```

Parameter	Description
FBName	Name of the ProVi function block in the PLC
Position	The number of the message in the order of priority if more than one message is active.

Fig. 9-5: ReadProViPLC() parameters

Caution: Only allowed in conjunction with global variables.

Caution: This function is only supported when WinPCL V19 is used!

BindProViMessage;

The ProVi message of a certain type and module is linked with a variable. This variable is cyclically updated. Concerning string variables the message text is returned. If the message is linked with a numerical variable the message number is returned. Errors, messages, warnings and setup diagnoses are sorted in temporal order. Thereby, the oldest message is always the first. Starting requirements are sorted by message number. The highest number is always on the first place. For this message type and module number no more than 100 messages can be available at the same time. If the PLC output exceeds this number, this messages can not be indicated.

```
void BindProViMessage(UINT8 MessageType, UINT8 ModulNo, UINT16 Position)
```

Parameter	Description
MessageType	1 – Error 2 – Message 10 – Warning 11 – Starting requirement 12 – Setup diagnosis The following values can also be used: 1 – PROVI_TYPE_ERROR, 2 – PROVI_TYPE_MESSAGE, 10 – PROVI_TYPE_WARNING, 11 – PROVI_TYPE_STARTUP, 12 – PROVI_TYPE_SETUP
ModulNo	Module whose message shall be indicated. Only for MessageType 1,2. For all other types the value must be 0.
Position	The message output occurs in sorted order. This parameter determines which message shall be indicated.

Fig. 9-6: BindProViMessage() parameter

ReadProViMessage;

Operates like BindProViMessage; the ProVi message is read in only one time and is not cyclically updated.

```
void ReadProViMessage (INT8 MessageType, INT8 ModulNo,
INT16 Position)
```

Parameter	Description
MessageType	1 – Error 2 – Message 10 – Warning 11 – Starting requirement 12 – Setup diagnosis The following values can also be used: 1 – PROVI_TYPE_ERROR, 2 – PROVI_TYPE_MESSAGE, 10 – PROVI_TYPE_WARNING, 11 – PROVI_TYPE_STARTUP, 12 – PROVI_TYPE_SETUP
ModulNo	Module whose message shall be indicated. Only for MessageType 1,2. For all other types the value must be 0.
Position	The message output occurs in sorted order. This parameter determines which message shall be indicated.

Fig. 9-7: ReadProViMessage

BindProViTime;

Operates like BindProViMessage; only appearance date and time of the message are returned.

```
void BindProViTime (INT8 MessageType, INT8 ModulNo,
INT16 Index, INT16 Format)
```

Parameter	Description
MessageType	1 – Error 2 – Message 10 – Warning 11 – Starting requirement 12 – Setup diagnosis The following values can also be used: 1 – PROVI_TYPE_ERROR, 2 – PROVI_TYPE_MESSAGE, 10 – PROVI_TYPE_WARNING, 11 – PROVI_TYPE_STARTUP, 12 – PROVI_TYPE_SETUP
ModulNo	Module whose message shall be indicated. Only for MessageType 1,2. For all other types the value must be 0.
Position	The message output occurs in sorted order. This parameter determines which message shall be indicated.

Format	Affects the illustration of date and time and influences only string variables. The parameter is indicated by the OR operation of the following flags: PROVI_DATE: The date is indicated. If PROVI_DATE and PROVI_TIME are used together, the order can be adjusted with PROVI_FIRSTDATE or PROVI_FIRSTTIME. PROVI_TIME: The time is indicated. If PROVI_DATE and PROVI_TIME are used together, the order can be adjusted with PROVI_FIRSTDATE or PROVI_FIRSTTIME. PROVI_FIRSTDATE: At first, the date is indicated. Is only efficient, if PROVI_DATE and PROVI_TIME are used together (Default). Must not be used together with PROVI_FIRSTTIME. PROVI_FIRSTTIME: At first, the time is indicated. Is only efficient, if PROVI_DATE and PROVI_TIME are used together. Must not be used together with PROVI_FIRSTDATE. PROVI_TIME_24: The hours from 0-23 are indicated. Must not be used together with PROVI_TIME_12 (Default). PROVI_TIME_12: The hours from 1-12 with AM and PM are indicated. Must not be used together with PROVI_TIME_24. PROVI_TIME_12: The hours from 1-12 with AM and PM are indicated. Must not be used together with PROVI_TIME_24. PROVI_DATE_DM: The date is indicated in format TT.MM.JJ. Thereby, the number of places of the year is indicated by the flags PROVI_DATE_YEAR2 and PROVI_DATE_YEAR4 (Default). Must not be used together with PROVI_DATE_MD. PROVI_DATE_MD: The date is indicated in format MM.TT.JJ. Thereby, the number of places of the year is indicated by the flags PROVI_DATE_YEAR2 and PROVI_DATE_YEAR4. Must not be used together with PROVI_DATE_DM. PROVI_DATE_YEAR2: The year is indicated two-digits. Must not be used together with PROVI_DATE_YEAR4. PROVI_DATE_YEAR4: The year is indicated 4-digit (Default). Must not be used together with PROVI_DATE_YEAR2.

Fig. 9-8: BindProViTime() parameter

ReadProViTime;

Operates like BindProViMessage; the ProVi message is only read in one time and is not cyclically updated. Appearance date and time of the message are returned.

```
void ReadProViTime (INT8 MessageType, INT8 ModulNo,
INT16 Index, INT16 Format)
```

Parameter	Description
MessageType	1 – Error 2 – Message 10 – Warning 11 – Starting requirement 12 – Setup diagnosis The following values can also be used: 1 – PROVI_TYPE_ERROR, 2 – PROVI_TYPE_MESSAGE, 10 – PROVI_TYPE_WARNING, 11 – PROVI_TYPE_STARTUP, 12 – PROVI_TYPE_SETUP
ModulNo	Module whose message shall be indicated. Only for MessageType 1,2. For all other types the value must be 0.
Position	The message output occurs in sorted order. This parameter determines which message shall be indicated.
Format	Affects the illustration of date and time and influences only string variables. The parameter is indicated by the OR operation of the following flags: PROVI_DATE: The date is indicated. If PROVI_DATE and PROVI_TIME are used together, the order can be adjusted by PROVI_FIRSTDATE or PROVI_FIRSTTIME. PROVI_TIME: The time is indicated If PROVI_DATE and PROVI_TIME are used together, the order can be adjusted by PROVI_FIRSTDATE or PROVI_FIRSTTIME. PROVI_FIRSTDATE: At first, the date is indicated. Is only efficient, if PROVI_DATE and PROVI_TIME are used together (Default). Must not be used together with PROVI_FIRSTTIME. PROVI_FIRSTTIME: At first, the time is indicated. Is only efficient, if PROVI_DATE and PROVI_TIME are used together. Must not be used together with PROVI_FIRSTDATE. PROVI_TIME_24: The hours from 0-23 are indicated. Must not be used together with PROVI_TIME_12 (Default).

Format	PROVI_FIRSTTIME: At first, the time is indicated. Is only efficient, if PROVI_DATE and PROVI_TIME are used together. Must not be used together with PROVI_FIRSTDATE. PROVI_TIME_24: The hours from 0-23 are indicated. Must not be used together with PROVI_TIME_12 (Default). PROVI_TIME_12: The hours from 1-12 and with AM and PM are indicated. Must not be used together with PROVI_TIME_24. PROVI_TIME_12: The hours from 1-12 with AM and PM are indicated. Must not be used together with PROVI_TIME_24. PROVI_DATE_DM: The date is indicated in format TT.MM.JJ. Thereby, the number of places of the year are indicated by the flags PROVI_DATE_YEAR2 and PROVI_DATE_YEAR4 (Default). Must not be used together with PROVI_DATE_MD. PROVI_DATE_MD: The date is indicated in format MM.TT.JJ. Thereby, the number of places of the year is indicated by the flags PROVI_DATE_YEAR2 and PROVI_DATE_YEAR4. Must not be used together with PROVI_DATE_DM. PROVI_DATE_YEAR2: The year is indicated two-digit. Must not be used together with PROVI_DATE_YEAR4. PROVI_DATE_YEAR4: The year is indicated as 4-digit number (Default). Must not be used together with PROVI_DATE_YEAR2.

Fig. 9-9: ReadProViTime() parameter

10 Linking Variables to an Indramat NC

10.1 NC Variable Declaration

The Following Definitions Apply to Parameter P0 for Linking to NC Functionalities

These constants have (analogously to the key definitions) already been defined in the ScreenManager system file, and may directly be entered in text form.

```
// CNC requirements in the ScreenManager
#define AAD (1)      // Active angle unit
#define AAS1 (2)     // Actual axis velocity (by axis condition)
#define AAS2 (3)     // Actual axis velocity (by log. axis no.)
#define AAC (4)      // Actual acceleration
#define ACS (5)      // Actual cutting speed of the reference spindle
#define ADN (6)      // Active D correction no.
#define AEM (7)      // Active event monitoring
#define AEN (8)      // Active tool edge number
#define AFO (9)      // Actual feed override
#define AFR (10)     // Actual feed rate
#define AGF (11)     // Active G function
#define AMF (12)     // Active M function
#define AMM (13)     // Active mechanism message
#define APM (14)     // Active NC comment
#define APO1 (15)    // Act. pos. in ma/prg. coordinate (by axis mean.)
#define APO2 (16)    // Act. pos. in ma/prg. coordinate (by log. axis no.)
#define APP (17)     // Number of the active part program
#define SPP (18)     // Number of the preselected part program
#define ARO (19)     // Actual rapid traverse override
#define ASC (20)     // Active spindle for facing (SPC)
#define ASF (21)     // Act. spindle f. G33, G63, G64, G65, G95, G96(SPF)
#define ASG (22)     // Actual spindle gear step
#define ASO (24)     // Actual spindle override
#define ASS (25)     // Actual spindle speed
#define AST (26)     // Active tool spindle (SPT)
#define ATN (27)     // Active T no.
#define AZB (28)     // Active zero offset table (O)
#define CPO1 (29)    // Comm. pos. in ma/prg. coordinates (by axis mean.)
#define CPO2 (30)    // Comm. pos. in ma/prg. coordinates (by log. axis no.)
#define DCD (31)     // D correction single element
#define DCR (32)     // D correction data record (L1, L2, L3, R)
#define DTG1 (33)    // Distance to go (request for axis meaning)
#define DTG2 (34)    // Distance to go (request for log. axis no.)
#define MTC (35)     // Read SW version of individual cards
#define EPO1 (36)    // End pos. in ma/prg. coordinate (by axis mean.)
```

```
#define EPO2 (37)    // End pos. in ma/prg. coordinate (by log. axis no.)
#define MFO (39)     // Maximum feed override
#define MFR (40)     // Maximum feed rate
#define MRO (41)     // Maximum rapid traverse override
#define MSO (42)     // Maximum spindle override
#define MSS (43)     // Maximum spindle speed
#define MTD (44)     // Single machine data element
#define NEV (45)     // Single machine data element
#define NPA3 (46)    // Read single parameters
#define NTN (47)     // Next preselected T number
#define NVS (48)     // NC variable
#define OPD1 (49)    // Optimum position distance (by axis mean.)
#define OPD2 (50)    // Optimum position distance (by log. axis no.)
#define ASN (51)     // Active NC block number
#define PFR (52)     // Programmed feed rate (F value)
#define ABI (53)     // Active NC block
#define SLA1 (54)    // Lag error (request for axis meaning)
#define SLA2 (55)    // Lag error (request for logic axis no.)
#define PSS (56)     // Programmed spindle speed
#define TLD (57)     // Tool data element (by location)
#define TQE1 (59)    // Actual torque (by axis meaning)
#define TQE2 (60)    // Actual torque (by log. axis no.)
#define ZOD (61)     // Single zero offset value
#define DIS3 (62)    // NC program package name
#define DIS6 (63)    // Part program name
```

Note:

- The BindCNC1...7 – instruction always requires the same number of characters to be specified that is specified by the subsequent figure (1..7).
- BindCNC1..7 can be used for any command.
- The number of characters specified by the figure will be interpreted.
- Parameters that are not required must be specified with a "0" !

10.2 NC System Commands

MTC (35) (in preparation)

Reads the firmware version of the individual cards.

```
Identifier.BindCNC1( INT32 MTC );
```

Parameter	Description
MTC	35

Fig. 10-1: MTC parameter

NPA3 (46) (in preparation)

Reads individual parameters (station name, number of memory locations, ...)

```
Identifier.BindCNC4( INT32 NPA3, INT32 PAS, INT32 PAN,
INT32 PEL );
```

Parameter	Description
NPA3	46
PAS	Parameter record number
PAN	Parameter number
PEL	Parameter element

Fig. 10-2: NPA3 parameters

10.3 NC Data Commands (General)

DIS3 (62) (in preparation)

NC program package name

```
Identifier.BindCNC3( INT32 DIS3, INT32 ST, INT32 CS );
```

Parameter	Description
DIS3	62
ST = 0 ST = 1	Memory A Memory B
CS = 0 CS = 1	Package number, package name Package name

Fig. 10-3: DIS3 parameters

DIS6 (63) (in preparation)

NC part program name

```
Identifier.BindCNC5( INT32 DIS6, INT32 ST, INT32 PRO,
INT32 PPN, INT32 CS );
```

Parameter	Description
DIS6	63
ST = 0 ST = 1	Memory A Memory B
PRO	Process number [0...6]
PPN	Part program number [1...99]
CS = 0 CS = 1	Part program number, part program name Part program name

Fig. 10-4: DIS6 parameters

APP (17)

Number of the active part program

```
Identifier.BindCNC3( INT32 APP, INT32 PRO, INT32 CS );
```

Parameter	Description
APP	17
PRO	Process number [0...6]
CS = 0 CS = 1	NC memory, part program number Part program number

Fig. 10-5: APP parameters

SPP (18)

Number of the preselected part program

```
Identifier.BindCNC3( INT32 SPP, INT32 PRO, INT32 CS );
```

Parameter	Description
SPP	18
PRO	Process number [0...6]
CS = 0 CS = 1	NC memory, part program number Part program number

Fig. 10-6: SPP parameters

MTD (44)

Single machine element

```
Identifier.BindCNC5( INT32 MTD, INT32 PN, INT32 I1,
INT32 I2, INT32 ELN);
```

Parameter	Description
MTD	44
PN	Machine data page [1...299]
I1	Control variable 1 [-1000...1000]
I2	Control variable 2 [-1000...1000]
ELN	Data element – number [1...1000]

Fig. 10-7: MTD parameters

ASN (51)

Active NC block number

```
Identifier.BindCNC2( INT32 ASN, INT32 PRO );
```

Parameter	Description
ASN	51
PRO	Process number [0...6]

Fig. 10-8: ASN parameters

ABI (53)

The function reads the active NC block or a user-defined NC block. This permits an NC block display with active NC block and the number of preceding and subsequent NC blocks to be built up. Thus, the display with preceding and subsequent block requires at least three instructions.

```
Identifier.BindCNC3( INT32 ABI, INT32 PRO, INT32 OZB
);
```

Parameter	Description
ABI	53
PRO	Process number [0...6]
OZB(opt)	Preceding or subsequent NC block [-4...4]

Fig. 10-9: ABI parameters

Note: Only the actual NC block is output if the optional parameters are not specified.

Up to ScreenManager version 1V02, the negative value of OZB can only be inserted via a variable that has been computed beforehand.

Example: OZB = 0;

OZB = OZB – 1;

AAD (1)

Representation of the active angle unit (RAD or DEG)

```
Identifier.BindCNC2( INT32 AAD, INT32 PRO );
```

Parameter	Description
AAD	1
PRO	Process number [0...6]

Fig. 10-10: AAD parameters

AAC (4)

Actual acceleration

```
Identifier.BindCNC3( INT32 AAC, INT32 PRO, INT32 CS );
```

Parameter	Description
AAC	4
PRO	Process number [0...6]
CS = 0 CS = 1	"ACC", actual acceleration, unit Actual acceleration, unit

Fig. 10-11: AAC parameters

AGF (11)

Active G function(s)

```
Identifier.BindCNC3( INT32 AGF, INT32 PRO, INT32 CS );
```

Parameter	Description
AGF	11
PRO	Process number [0...6]
CS = 0 CS = 1...21	Total string with up to 21 G codes Single G code

Fig. 10-12: AGF parameters

AMF (12)

Active M function(s)

```
Identifier.BindCNC3( INT32 AMF, INT32 PRO, INT32 CS );
```

Parameter	Description
AMF	12
PRO	Process number [0...6]
CS = 0 CS = 1...16	Total string of up to 16 M functions Single M function

Fig. 10-13: AMF parameters

10.4 NC Data Command (Axis Commands)

CPO1 (29)

Axis command position in machine and program coordinates (request for axis meaning)

```
Identifier.BindCNC5( INT32 CPO1, INT32 PRO, INT32 AXI,
INT32 CSY, INT32 CS );
```

Parameter	Description
CPO1	29
PRO	Process number [0...6]
AXI	Axis meaning [1...12] 1 X axis primary axis 2 Y axis primary axis 3 Z axis primary axis 4 U axis secondary axis 5 V axis secondary axis 6 W axis secondary axis 7 A axis rotary axis 8 B axis rotary axis 9 C axis rotary axis 10 S1 axis spindle 11 S2 axis spindle 12 S3 axis spindle
CSY = 0 CSY = 1	Axis command position in machine coordinates Axis command position in program coordinates
CS = 0 CS = 1	Axis designation, axis command position, unit Axis command position, unit

Fig. 10-14: CPO1 parameters

CPO2 (30)

Axis command position in machine and program coordinates (request for logic axis number)

```
Identifier.BindCNC4( INT32 CPO2, INT32 AXN, INT32 CSY,
INT32 CS );
```

Parameter	Description
CPO2	30
AXN	Axis number [1...32]
CSY = 0 CSY = 1	Axis command position in machine coordinates Axis command position in program coordinates
CS = 0 CS = 1	Axis designation, axis command position, unit Axis command position, unit

Fig. 10-15: CPO2 parameters

APO1 (15)

Actual axis position (actual position) in machine and program coordinates (request for axis meaning)

```
Identifier.BindCNC5( INT32 APO1, INT32 PRO, INT32 AXI,
INT32 CSY, INT32 CS );
```

Parameter	Description
APO1	15
PRO	Process number [0...6]
AXI	Axis meaning [1...12] 1 X axis primary axis 2 Y axis primary axis 3 Z axis primary axis 4 U axis secondary axis 5 V axis secondary axis 6 W axis secondary axis 7 A axis rotary axis 8 B axis rotary axis 9 C axis rotary axis 10 S1 axis spindle 11 S2 axis spindle 12 S3 axis spindle
CSY = 0 CSY = 1	Axis position in machine coordinates Axis position in program coordinates
CS = 0 CS = 1	Axis designation, axis position, unit Axis position, unit

Fig. 10-16: APO1 parameters

APO2 (16)

Actual axis position (actual position) in machine and program coordinates (request for logic axis number)

```
Identifier.BindCNC4( INT32 APO2, INT32 AXN, INT32 CSY,
INT32 CS );
```

Parameter	Description
APO2	16
AXN	Axis number [1...32]
CSY = 0 CSY = 1	Axis position in machine coordinates Axis position in program coordinates
CS = 0 CS = 1	Axis designation, axis position, unit Axis position, unit

Fig. 10-17: APO2 parameters

SLA1 (54)

Lag error (request for axis meaning)

```
Identifier.BindCNC4( INT32 SLA1, INT32 PRO, INT32 AXI,
INT32 CS );
```

Parameter	Description
SLA1	54
PRO	Process number [0...6]
AXI	Axis meaning [1...12] 1 X axis primary axis 2 Y axis primary axis 3 Z axis primary axis 4 U axis secondary axis 5 V axis secondary axis 6 W axis secondary axis 7 A axis rotary axis 8 B axis rotary axis 9 C axis rotary axis 10 S1 axis spindle 11 S2 axis spindle 12 S3 axis spindle
CS = 0 CS = 1	Axis designation, lag error, unit Lag error, unit

Fig. 10-18: SLA1 parameters

SLA2 (55)

Lag error (request for logic axis number)

```
Identifier.BindCNC3( INT32 SLA2, INT32 AXN, INT32 CS
);
```

Parameter	Description
SLA"	55
AXN	Axis number [1...32]
CS = 0 CS = 1	Axis designation, lag error, unit Lag error, unit

Fig. 10-19: SLA2 parameters

EPO1 (36)

End position in machine and program coordinates (request for axis meaning)

```
Identifier.BindCNC5( INT32 EPO1, INT32 PRO, INT32 AXI,
INT32 CSY, INT32 CS );
```

Parameter	Description
EPO1	36
PRO	Process number [0...6]
AXI	Axis meaning [1...12] 1 X axis primary axis 2 Y axis primary axis 3 Z axis primary axis 4 U axis secondary axis 5 V axis secondary axis 6 W axis secondary axis 7 A axis rotary axis 8 B axis rotary axis 9 C axis rotary axis 10 S1 axis spindle 11 S2 axis spindle 12 S3 axis spindle
CSY = 0 CSY = 1	End position in machine coordinates End position in program coordinates
CS = 0 CS = 1	Axis designation, end position, unit End position, unit

Fig. 10-20: EPO1 parameters

EPO2 (37)

Axis end position in machine and program coordinates (request for logic axis number)

```
Identifier.BindCNC4( INT32 EPO2, INT32 AXN, INT32 CSY,
INT32 CS );
```

Parameter	Description
EPO2	37
AXN	Axis number [1...32]
CSY = 0 CSY = 1	End position in machine coordinates End position in program coordinates
CS = 0 CS = 1	Axis designation, end position, unit End position, unit

Fig. 10-21: EPO2 parameters

DTG1 (33)

Distance to go of the axis in machine and program coordinates (request for axis meaning)

```
Identifier.BindCNC5( INT32 DTG1, INT32 PRO, INT32 AXI,
INT32 CSY, INT32 CS );
```

Parameter	Description
DTG1	33
PRO	Process number [0...6]
AXI	Axis meaning [1...12] 1 X axis primary axis 2 Y axis primary axis 3 Z axis primary axis 4 U axis secondary axis 5 V axis secondary axis 6 W axis secondary axis 7 A axis rotary axis 8 B axis rotary axis 9 C axis rotary axis 10 S1 axis spindle 11 S2 axis spindle 12 S3 axis spindle
CSY = 0 CSY = 1	Distance to go in machine coordinates Distance to go in program coordinates
CS = 0 CS = 1	Axis designation, distance to go, unit Distance to go, unit

Fig. 10-22: DTG1 parameters

DTG2 (34)

Distance to go of the axis in machine and program coordinates (request for logic axis number)

```
Identifier.BindCNC4( INT32 DTG2, INT32 AXN, INT32 CSY,
INT32 CS );
```

Parameter	Description
DTG2	34
AXN	Axis number [1...32]
CSY = 0 CSY = 1	Distance to go in machine coordinates Distance to go in program coordinates
CS = 0 CS = 1	Axis designation, distance to go, unit Distance to go, unit

Fig. 10-23: DTG2 parameters

AAS1 (2)

Actual axis velocity (request by axis meaning)

```
Identifier.BindCNC4( INT32 AAS1, INT32 PRO, INT32 AXI,
INT32 CS );
```

Parameter	Description
AAS1	2
PRO	Process number [0...6]
AXI	Axis meaning [1...12] 1 X axis primary axis 2 Y axis primary axis 3 Z axis primary axis 4 U axis secondary axis 5 V axis secondary axis 6 W axis secondary axis 7 A axis rotary axis 8 B axis rotary axis 9 C axis rotary axis 10 S1 axis spindle 11 S2 axis spindle 12 S3 axis spindle
CS = 0 CS = 1	Axis designation, axis velocity, unit Axis velocity, unit

Fig. 10-24: AAS1 parameters

AAS2 (3)

Actual axis velocity (request by axis number)

```
Identifier.BindCNC3( INT32 AAS2, INT32 AXN, INT32 CS
);
```

Parameter	Description
AAS2	3
AXN	Axis number [1...32]
CS = 0 CS = 1	Axis designation, axis velocity, unit Axis velocity, unit

Fig. 10-25: AAS2 parameters

OPD1 (49)

Optimum distance to the center switch (request by axis meaning)

```
Identifier.BindCNC4( INT32 OPD1, INT32 PRO, INT32 AXI,
INT32 CS );
```

Parameter	Description
OPD1	49
PRO	Process number [0...6]
AXI	Axis meaning [1...12] 1 X axis primary axis 2 Y axis primary axis 3 Z axis primary axis 4 U axis secondary axis 5 V axis secondary axis 6 W axis secondary axis 7 A axis rotary axis 8 B axis rotary axis 9 C axis rotary axis 10 S1 axis spindle 11 S2 axis spindle 12 S3 axis spindle
CS = 0 CS = 1	Axis designation, optimum position distance, unit Optimum position distance, unit

Fig. 10-26: OPD1 parameters

OPD2 (50)

Optimum position distance to the zero switch (request by logic axis number)

```
Identifier.BindCNC3( INT32 OPD2, INT32 AXN, INT32 CS
);
```

Parameter	Description
OPD2	50
AXN	Axis number [1...32]
CS = 0 CS = 1	Axis designation, optimum position distance, unit Optimum position distance, unit

Fig. 10-27: OPD2 parameters

TQE1 (59)

Actual torque of the axis (request by axis meaning)

```
Identifier.BindCNC4( INT32 TQE1, INT32 PRO, INT32 AXI,
INT32 CS );
```

Parameter	Description
TQE1	59
PRO	Process number [0...6]
AXI	Axis meaning [1...12] 1 X axis primary axis 2 Y axis primary axis 3 Z axis primary axis 4 U axis secondary axis 5 V axis secondary axis 6 W axis secondary axis 7 A axis rotary axis 8 B axis rotary axis 9 C axis rotary axis 10 S1 axis spindle 11 S2 axis spindle 12 S3 axis spindle
CS = 0 CS = 1	Axis designation, actual torque, unit Actual torque, unit

Fig. 10-28: TQE1 parameters

TQE2 (60)

Actual torque of the axis (request by logic axis number)

```
Identifier.BindCNC3( INT32 TQE2, INT32 AXN, INT32 CS
);
```

Parameter	Description
TQE2	60
AXN	Axis number [1...32]
CS = 0 CS = 1	Axis designation, actual torque, unit Actual torque, unit

Fig. 10-29: TQE2 parameters

AFO (9)

Actual feed override

```
Identifier.BindCNC3( INT32 AFO, INT32 PRO, INT32 CS );
```

Parameter	Description
AFO	9
PRO	Process number [0...6]
CS = 0 CS = 1	"OVR", actual feed override, unit Actual feed override, unit

Fig. 10-30: AFO parameters

MFO (39)

Maximum feed override

```
Identifier.BindCNC3( INT32 MFO, INT32 PRO, INT32 CS );
```

Parameter	Description
MFO	39
PRO	Process number [0...6]
CS = 0 CS = 1	"Omax", maximum feed override, unit Maximum feed override, unit

Fig. 10-31: MFO parameters

ARO (19)

Actual rapid traverse override

```
Identifier.BindCNC3( INT32 ARO, INT32 PRO, INT32 CS );
```

Parameter	Description
ARO	19
PRO	Process number [0...6]
CS = 0 CS = 1	"ROV", actual rapid traverse override, unit Actual rapid traverse override, unit

Fig. 10-32: ARO parameters

MRO (41)

Maximum rapid traverse override

```
Identifier.BindCNC3( INT32 MRO, INT32 PRO, INT32 CS );
```

Parameter	Description
MRO	41
PRO	Process number [0...6]
CS = 0 CS = 1	"Rmax", maximum rapid traverse override, unit Maximum rapid traverse override, unit

Fig. 10-33: MRO parameters

AFR (10)

Actual feed rate

```
Identifier.BindCNC3( INT32 AFR, INT32 PRO, INT32 CS );
```

Parameter	Description
AFR	10
PRO	Process number [0...6]
CS = 0 CS = 1	"Fact", actual feed rate, unit Actual feed rate, unit

Fig. 10-34: AFR parameters

MFR (40)

Maximum feed rate

```
Identifier.BindCNC_MFR( INT32 MFR, INT32 PRO, INT32 CS
);
```

Parameter	Description
MFR	40
PRO	Process number [0...6]
CS = 0 CS = 1	"Fmax", maximum feed rate, unit Maximum feed rate, unit

Fig. 10-35: MFR parameters

PFR (52)

Programmed feed rate

```
Identifier.BindCNC3( INT32 PFR, INT32 PRO, INT32 CS );
```

Parameter	Description
PFR	52
PRO	Process number [0...6]
CS = 0 CS = 1	"F", programmed feed rate, unit Programmed feed rate, unit

Fig. 10-36: PFR parameters

10.5 NC Data Commands (Spindle Commands)

ASC (20)

Active spindle for facing (SPC)

```
Identifier.BindCNC2( INT32 ASC, INT32 PRO );
```

Parameter	Description
ASC	20
PRO	Process number [0...6]

Fig. 10-37: ASC parameters

ASF (21)

Active spindle for G33, G63, G64, G65, G95 and G96 (reference spindle) (SPF)

```
Identifier.BindCNC2( INT32 ASF, INT32 PRO );
```

Parameter	Description
ASF	21
PRO	Process number [0...6]

Fig. 10-38: ASF parameters

AST (26)

Active tool spindle

```
Identifier.BindCNC2( INT32 AST, INT32 PRO );
```

Parameter	Description
AST	26
PRO	Process number [0...6]

Fig. 10-39: AST parameters

ASG (22)

Actual spindle gear step

```
Identifier.BindCNC4( INT32 ASG, INT32 PRO, INT32 SPI,
INT32 CS );
```

Parameter	Description
ASG	22
PRO	Process number [0...6]
SPI	Spindle [1...3]
CS = 0 CS = 1	"g", actual gear pre-stage Actual gear pre-stage

Fig. 10-40: ASG parameters

ASO (24)

Actual spindle override

```
Identifier.BindCNC4( INT32 ASO, INT32 PRO, INT32 SPI,
INT32 CS );
```

Parameter	Description
ASO	24
PRO	Process number [0...6]
SPI	Spindle [1...3]
CS = 0 CS = 1	Spindle name, active spindle override, unit Actual spindle override, unit

Fig. 10-41: ASO parameters

MSO (42)

Maximum spindle override

```
Identifier.BindCNC4( INT32 MSO, INT32 PRO, INT32 SPI,
INT32 CS );
```

Parameter	Description
MSO	42
PRO	Process number [0...6]
SPI	Spindle [1...3]
CS = 0 CS = 1	Spindle name, "max", maximum spindle override, unit Maximum spindle override, unit

Fig. 10-42: MSO parameters

ASS (25)

Actual spindle speed

```
Identifier.BindCNC4( INT32 ASS, INT32 PRO, INT32 SPI,
INT32 CS );
```

Parameter	Description
ASS	25
PRO	Process number [0...6]
SPI	Spindle [1...3]
CS = 0 CS = 1	Spindle name, "act", actual spindle override, unit Actual spindle override, unit

Fig. 10-43: ASS parameters

MSS (43)

Maximum spindle speed

```
Identifier.BindCNC4( INT32 MSS, INT32 PRO, INT32 SPI,
INT32 CS );
```

Parameter	Description
MSS	43
PRO	Process number [0...6]
SPI	Spindle [1...3]
CS = 0 CS = 1	Spindle name, "max", maximum spindle speed, unit Maximum spindle speed, unit

Fig. 10-44: MSS parameters

PSS (56)

Programmed spindle speed

```
Identifier.BindCNC4( INT32 PSS, INT32 PRO, INT32 SPI,
INT32 CS );
```

Parameter	Description
PSS	56
PRO	Process number [0...6]
SPI	Spindle [1...3]
CS = 0 CS = 1	Spindle name, programmed spindle speed, unit Programmed spindle speed, unit

Fig. 10-45: PSS parameters

ACS (5)

Actual cutting speed of the reference spindle

```
Identifier.BindCNC3( INT32 ACS, INT32 PRO, INT32 CS );
```

Parameter	Description
ACS	5
PRO	Process number [0...6]
CS = 0 CS = 1	Reference spindle no., cutting speed, unit Cutting speed, unit

Fig. 10-46: ACS parameters

10.6 NC Data Commands (D Corrections)

ADN (6)

Active D correction number

```
Identifier.BindCNC2( INT32 ADN, INT32 PRO );
```

Parameter	Description
ADN	6
PRO	Process number [0...6]

Fig. 10-47: ADN parameters

DCD (31)

D correction single element

```
Identifier.BindCNC5( INT32 DCD, INT32 PRO, INT32 DCN,
INT32 DEL, INT32 CS );
```

Parameter	Description
DCD	31
PRO	Process number [0...6]
DCN	D correction data record number [1...99]
DEL	D correction element no. (1= L1, 2 = L2, 3 = L3, 4 = R)
CS = 0 CS = 1	Data element name, data element value, unit Data element value, unit

Fig. 10-48: DCD parameters

DCR (32) (in preparation)

D correction data record (L1, L2, L3, R)

```
Identifier.BindCNC4( INT32 DCR, INT32 PRO, INT32 DCN,
INT32 CS );
```

Parameter	Description
DCR	32
PRO	Process number [0...6]
DCN	D correction data record number [1...99]
CS = 0 CS = 1	Data element name, data element value, unit Data element value, unit

Fig. 10-49: DCR parameters

10.7 NC Data Commands (NC Variables and Events)

NVS (48)

NC variable

```
Identifier.BindCNC3( INT32 NVS, INT32 PRO, INT32 VN );
```

Parameter	Description
NVS	48
PRO	Process number [0...6]
VN	NC variable number [0...255]

Fig. 10-50: NVS parameters

AEM (7)

Active event monitoring

```
Identifier.BindCNC2( INT32 AEM, INT32 PRO );
```

Parameter	Description
AEM	7
PRO	Process number [0...6]

Fig. 10-51: AEM parameters

NEV (45)

NC events

```
Identifier.BindCNC2( INT32 NEV, INT32 PRO, INT32 EVN,
INT32 CS);
```

Parameter	Description
NEV	45
PRO	Process number [0...6]
EVN	NC event number [0...31]
CS = 0	All events (EVN parameter without function)
CS = 1	Single event

Fig. 10-52: NEV parameters

10.8 NC Data Commands (Zero Points)

AZB (28)

Active zero offset table (O)

```
Identifier.BindCNC2( INT32 AZB, INT32 PRO );
```

Parameter	Description
AZB	28
PRO	Process number [0...6]

Fig. 10-53: AZB parameters

ZOD (61)

Single zero offset value

```
Identifier.BindCNC7( INT32 ZOD, INT32 PRO, INT32 ST,
INT32 PRO, INT32 BN, INT32 SN, INT32 AN, INT32 CS );
```

Parameter	Description
ZOD	61
ST	NC memory A NC memory B
PRO	Process number [0...6]
BN	Number of the zero offset table [0...9]
SN	Offset type [0...9] 0 = Total of all active offset values 1 = G50/G51 2 = G52 3 = General offset in the zero point table 4 = G54 5 = G55 6 = G56 7 = G57 8 = G58 9 = G59
ON	Axis number [0..9] (corresponds to axis meaning, but feed axes only) 0...8 = axis meanings 9 = torsion angle "Phi"
CS = 0 CS = 1	Axis designation, zero offset value (torsion angle), unit Zero offset value (torsion angle), unit

Fig. 10-54: ZOD parameters

10.9 NC Data Commands (Tools)

AEN (8)

Active tool edge number

```
Identifier.BindCNC3( INT32 AEN, INT32 PRO, INT32 CS );
```

Parameter	Description
AEN	8
PRO	Process number [0...6]
CS = 0 CS = 1	"E", Active tool edge number Active tool edge number

Fig. 10-55: AEN parameters

ATN (27)

Active tool number

```
Identifier.BindCNC3( INT32 ATN, INT32 PRO, INT32 CS );
```

Parameter	Description
PRO	Process number [0...6]
CS = 0 CS = 1	"T", tool number Tool number

Fig. 10-56: ATN parameters

NTN (47)

Next preselected tool number

```
Identifier.BindCNC3( INT32 NTN, INT32 PRO, INT32 CS );
```

Parameter	Description
NTN	47
PRO	Process number [0...6]
CS = 0 CS = 1	"Tsel", next preselected tool number Next preselected tool number

Fig. 10-57: NTN parameters

TLD (57)

Tool data element (request for tool location or tool number)

```
Identifier.BindCNC7( INT32 TLD, INT32 PRO, INT32 S_T,
INT 32 TST, INT32 PON, INT32 TDS, INT32 TDE );
```

Parameter	Description
TLD	57
PRO	Process number [0...6]
S_T	0 = tool location 1 = tool number
TST S_T = 0 S_T = 1	 Storage [0...6] 0 = magazine/revolver 1 = spindle 2 = gripper 3 = position Location number [1...999]
PON S_T = 0 S_T = 1	 TST = 0 [1...9999] TST = 1 [1...4] TST = 2 [1...4] TST = 3 [1...4] Location number [1...999]
TDS = 0 TDS = 1..9	Base tool data Tool tip data
TDE = 1...36	Tool data element For TDS = 0 [3...26] For TDS = 1...9 [1...40]

Fig. 10-58: TLD1 parameters

Call Parameter for Base Tool Data TDS = 0; TDE = 3...36

NAME	Range	DATA TYPE in the SPS	UNIT	DATA ELEMENT	OPTION	SL	TL
Base tool data	(per tool)						
Tool identification							
Index address	hexadecimal long word with 32 bits (read only)		-	01			
ID (tool name)	up to 28 characters (any characters)	STRING28	-	02		X	X
Magazine	0 - 2 (0: magazine/turret, 1: spindle, 2: gripper)		-	03			X
Location	0 – 999		-	04			X
Tool number	1 – 9999999		-	05		X	X
Index number	1 – 999	DINT		06			
Correction type	1 – 5	INT	-	07		X	X
Number of tool edges	1 – 9	USINT	-	08		X	X
Tool status	0/1 (32 status bits)	USINT	-	09			X
Location data							
Old location	1 – 999	INT	-	11			X
Stor. of next setup tool	0 - 2 (0: magazine/turret, 1: spindle, 2: gripper)	INT	-	12			
Loc. of next setup tool	1 – 999	INT	-	13			
Stor. of prev. setup tool	0 - 2 (0: magazine/turret, 1: spindle, 2: gripper)	INT	-	14			
Loc. of prev. setup tool	1 -999	INT	-	15			X
Units			-				
Time unit	0/1 (0: min, 1: cycl.)	USINT		16			X
Length unit	0/1 (0: mm, 1: inch)	USINT	-	17		X	X
Technology data			-				
Tool code	0 - 9	USINT		18		X	X
Representation type	0 - 999	INT	-	19		X	X
User data							
User data 1	$+/- 1.2 * 10^{-38}$ - $+/- 3.4 * 10^{+38}$ and 0 (9 signific. digits)	REAL	any	20	X		X
.	.	.	.	.	.		.
.	.	.	.	.	.		.
User data 9	$+/- 1.2 * 10^{-38}$ - $+/- 3.4 * 10^{+38}$ and 0 (9 signific. digits)	REAL	any	28	X		X
Comment	up to 5 x 76 characters (any characters)		-	99	X	X	

Legend **EL**: = data specific to setup list **WL**: = data specific to tools list

Call Parameters for Tools – Tool Tip Data TDS = 1; TDE = 1...40

NAME	Range	DATA TYPE in the SPS	UNIT	DATA ELEMENT	OPTION	SL	TL
Tool edge data	(per tool edge)						
Tool edge identification							
Tool edge orientation	0 - 8	USINT	-	01		X	X
Tool edge status	0/1 (16 status bits)	WORD	-	02			X
Tool life data							
Remaining tool life	..0.0 - 100.00	REAL	%	03	X		X
Warning limit	..0.0 - 100.00	REAL	%	04	X		X
..Max. utilization time	..0 - 9999999 (0: service life measurement deactivated)	REAL	min or cycle	05	X		X
Geometry data							
Length L1	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	07			X
Length L2	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	08			X
Length L3	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	09			X
Radius R	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	10			X
Wear L1	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	11	X		X
Wear L2	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	12	X		X
Wear L3	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	13	X		X
Wear R	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	14	X		X
Offset L1	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	15	X		X
Offset L2	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	16	X		X
Offset L3	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	17	X		X
Offset R	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	18	X		X
Geometry limit values							
L1_min	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	19	X	X	
L1_max	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	20	X	X	
L2_min	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	21	X	X	
L2_max	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	22	X	X	
L3_min	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	23	X	X	
L3_max	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	24	X	X	
R_min	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	25	X	X	
R_max	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch	26	X	X	
Wear factors							
Wear factor L1	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch/ min or cycle	27	X		X
Wear factor L2	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch / min or cycle	28	X		X
Wear factor L3	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	m or inch / min or cycle	29	X		X
Wear factor R	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	mm or inch / min or cycle	30	X		X
User data							
User data 1	$+/- 1.2 * 10^{-38}$ - $+/- 3.4 * 10^{+38}$ and 0 (9 significant digits)	REAL	any	31	X		X
.	.	.	.	.	.		.
.	.	.	.	.	.		.
User data 5	$+/- 1.2 * 10^{-38}$ - $+/- 3.4 * 10^{+38}$ and 0 (9 significant digits)	REAL	any	35	X		X
User data 6	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	any	36	X		X
.	.	.	.	.	.		.
.	.	.	.	.	.		.
User data 10	-99999.9999 - +99999.9999 or -9999.99999 - +9999.99999	DINT	any	40	X		X

Legend **EL**: = data specific to setup list **WL**: = data specific to tools list

10.10 NC Data Commands (NC Messages)

APM (14)

Active NC note

```
Identifier.BindCNC2( INT32 APM, INT32 PRO );
```

Parameter	Description
APM	14
PRO	Process number [0...6]

Fig. 10-59: APM parameters

AMM (13)

Active mechanism message

```
Identifier.BindCNC2( INT32 AMM, INT32 MN );
```

Parameter	Description
AMM	13
PRO	Process number [0...6]

Fig. 10-60: AMM parameters

11 Linking Variables to an Indramat CLC

11.1 CLC Data Declaration

Parameter	Description	Type
Network Address	CLC device address	UINT8
Command: Class and Subclass	Command class structure	String
Set	Set of specified data	UINT16
Number	Data number	UINT16

Fig. 11-1: BindCLC, ReadCLC, and WriteCLC parameters

These parameters are described in the CLC communication protocol document. Variables may be used for every parameter.

11.2 CLC System Commands

ConnectCLC();

ConnectCLC() enables CLC communication. This call must be made before the first CLC data access. This command looks for the first port (interface), configured as ASCII. This port will then be used for all subsequent CLC commands. If no ASCII port is available, the following error message is issued: "No ASCII port for CLC, off-line mode only. Press any key to continue".

If a CLC port is found, the function looks for a remaining port configured as ASCII or ASCII&SIS-Slave. This port is set up as the CLC relay port.

```
void ConnectCLC();
```

ConnectCLC() does not possess any parameters.

ConnectCLC() must not be invoked in an off line simulation of screens. All the other CLC functions may be remain in the program. The are ignored and do not cause an error message to be issued.

NumberOfCLC()

NumberOfCLC() scans the bus for the number of connected stations. The last number to be scanned is defined in the BTV parameters. The return value is the number of stations found. This command stops script execution until the scan is finished.

```
UINT8 NumberOfCLC();
```

NumberOfCLC() does not possess any parameters.

AddressOfCLC()

AddressOfCLC() returns the station address to the index parameter..

0 = first station address

1 = second station address

n = last station address

n= return value of NumberOfCLC() minus 1. These two commands could be used to generate general-purpose programs for automatic detection.

```
UINT8 AddressOfCLC(UINT8 Index);
```

Parameter	Description	Type
Index	CLC controller, in order of appearance	UINT8

Fig. 11-2: AddressOfCLC() parameter

WaitCLC();

To increase the performance, all variable access procedures are performed in the background. The WaitCLC() command can be used for forcing the BTV unit to wait until all write and read access procedures of the CLC are terminated. This command must be used if equations or conditions depend on a CLC write or read command that was issued immediately before.

```
WaitCLC();
```

WaitCLC() does not possess any parameters.

Example:

```
INT16 a = 0;
{
   a.ReadCLC( 6, "IP", 1, No);
   Result = a * 22;
...
```

In this example, Result remains zero, because at the moment when the equation is made "a" is not read by the CLC. A functioning example would be:

```
INT16 a = 0;
{
   a.ReadCLC( 6, "IP", 1, No );
   WaitCLC();
   Result = a * 22;
...
```

In this example, ‚Result' is only computed after the current variable value of a has been fetched from the PLC. Execution is slower, since the ScreenManager program blocks until all pending serial read and write commands are completed.

11.3 CLC Data Commands

BindCLC();

The BindCLC() command generates a link between ScreenManager and CLC object variables. The ScreenManager variable is cyclically updated with the CLC value.

Global bindings must be made before the first Screen() call. They can only be released by resetting the small operator input unit. Every bound variable uses transmission time. Use global bindings only if necessary.

Every local binding that is made after the first Screen() call is released as soon as a new screen is activated.

If a bound variable is also linked with an edit cell using the Edit() command, it is automatically written to the CLC after it has been modified by the user and the OK button has been pressed.

Changing a variable within an equation (e. .g. a = 99;) does not initiate a command that writes the new value to the CLC. This must be done using the WriteCLC(); command.

```
Identifier.BindCLC( UINT8 Address, STRING Command,
UINT16 Set, UINT16 Number);
```

Parameter	Description	Type
Address	CLC device address	UINT8
Command: Class and Subclass	Command class structure	String
Set	Set of specified data	UINT16
Number	Data number	UINT16

Fig. 11-3: BindCLC() parameters

Note: CLC floating point parameters may only be linked to FLOAT types.

CLC integer parameters may only be linked to INT types. The CLC command for decimal representation of the parameters must be used. Different integer types are converted automatically. If the converted value exceeds the valid range of the destination type, the result rolls over. Example: Conversion of 258 to a UINT8 results in a 2.

Caution: Only allowed in conjunction with global variables.

ReadCLC();

The value is read from the CLC only once. As with the BindCLC() command, the value is read from the CLC in background operation. The WaitCLC() command must be used if the subsequent lines of program code depend on this read value.

```
Identifier.ReadCLC( UINT8 Address, STRING Command,
UINT16 Set, UINT16 Number);
```

Parameter	Description	Type
Address	CLC device address	UINT8
Command: Class and Subclass	Command class structure	String
Set	Set of specified data	UINT16
Number	Data number	UINT16

Fig. 11-4: ReadCLC()-parameters

Example:

```
STRING Firmware[30];


{
   Firmware.ReadCLC( 6, "CP", 1, 100 );
   Text ( 0,0,128,10, Firmware);
...
```

Incorrect: In this example, the name remains blank, because at the moment when text output is made the variable has not yet been read by the PLC. A functioning but time-consuming example would be:

```
STRING Firmware[30];
...
{
   Firmware.ReadCLC( 6, "CP", 1, 100 );
   WaitCLC();
   Text ( 0,0,128,10, Firmware);
...
```

A more effective implementation would be:

```
STRING Firmware[30];
...
{
   Firmware.ReadCLC( 6, "CP", 1, 100 );
   Firmware.Display(0,0,0,0);
...
```

The lines of program code are executed as fast as possible. The CLC value is shown as soon as it is read from the CLC.

Note: CLC floating point parameters may only be linked to FLOAT types.

CLC integer parameters may only be linked to INT types. The CLC command for decimal representation of the parameters must be used. Different integer types are converted automatically. If the converted value exceeds the valid range of the destination type, the result rolls over. Example: Conversion of 258 to a UINT8 results in a 2.

Caution: Only allowed in conjunction with global variables.

WriteCLC();

A single write command to the CLC is initiated.

```
Identifier.WriteCLC( UINT8 Address, STRING Command,
UINT16 Set, UINT16 Number);
```

Parameter	Description	Type
Network Address	CLC device address	UINT8
Command: Class and Subclass	Command class structure	String
Set	Set of specified data	UINT16
Number	Data number	UINT16

Fig. 11-5: WriteCLC() parameters

Example:

```
{
   INT32 My_Var = 0;
   My_Var.WriteCLC( 6, "IP", 1, No );
// initiates a write of 0 to CLC
   My_Var = 1;
   My_Var.WriteCLC( 6, "IP", 1, No );
// initiates a write of 1 to CLC
   My_Var = 2;
   My_Var.WriteCLC( 6, "IP", 1, No );
// initiates a write of 2 to CLC
```

Write commands are initiated after every write call to the CLC variable. Every write command requires one communication cycle. If a variable changes several times within a short period, every change is transmitted to the CLC in the order it was entered.

Note: ScreenManager FLOAT variables may only be written to floating point or text parameters.

11.4 CLC I/O Register Commands

BindIOCLC();

The BindIOCLC() method generates a link between a variable from the ScreenManager and any CLC I/O register bit. The ScreenManager variable is cyclically updated with the CLC register value.

Global bindings must be made before the first Screen() call. They can only be released by resetting the small operator input unit. Every bound variable uses transmission time. Use global bindings only if necessary.

Every local binding that is made after the first Screen() call is released as soon as a new screen is activated.

If a bound variable is also linked with an edit cell using the Edit() command, it is automatically written to the CLC after it has been modified by the user.

Changing a variable within an equation (e. g. a = 99;) does not initiate a command that writes the new value to the CLC. This must be done using the WriteOLC(); command.

```
Identifier.BindIOCLC( UINT8 Address, UINT16 Register,
UINT8 Bit);
```

Parameter	Description	Type
Address	CLC device address	UINT8
Register	CLC I/O Register	UINT16
Bit	Bit in register, following the CLC definition (first bit has the number 1)	UINT8

Fig. 11-6: BindIOCLC() parameters

Any ScreenManager type is permitted. Every value but ’0’ sets the CLC register.

Caution: Only allowed in conjunction with global variables.

ReadIOCLC();

The value is read from the CLC only once. As with the BindIOCLC() command, the value is read from the CLC in background operation. The WaitCLC() command must be used if the subsequent lines of program code depend on this read value.

```
Identifier.ReadIOCLC( UINT8 Address, UINT16 Register,
UINT8 Bit);
```

Parameter	Description	Type
Address	CLC device address	UINT8
Register	CLC I/O Register	UINT16
Bit	Bit in register, following the CLC definition (first bit has the number 1)	UINT8

Fig. 11-7: ReadIOCLC() parameters

WriteIOCLC();

A single write command to the CLC is initiated.

```
Identifier.WriteIOCLC( UINT8 Address, UINT16 Register,
UINT8 Bit);
```

Parameter	Description	Type
Address	CLC device address	UINT8
Register	CLC I/O Register	UINT16
Bit	Bit in register, following the CLC definition (first bit has the number 1)	UINT8

Fig. 11-8: WriteIOCLC() parameters

Any ScreenManager variable type is possible. 0 clears the register, every value but '0' sets the CLC register to 1.

SetIOCLC();

A serial write to the CLC is initiated.

```
void SetIOCLC( UINT8 Address, UINT16 Register, UINT16
Bit, BOOL value);
```

Parameter	Description	Type
Network Address	CLC device address	UINT8
Register	CLC I/O Register	UINT16
Bit	Bit in register, following the CLC definition (first bit has the number 1)	UINT16
Value	New register value	BOOL

Fig. 11-9: SetIOCLC() parameters

ForceIOCLC();

The specified register is forced to a specified value.

```
void ForceIOCLC( UINT8 Address, UINT16 Number, UINT16
Mask, UINT16 Value);
```

Parameter	Description	Type
Address	CLC device address	UINT8
Number	CLC I/O Register	UINT16
Mask	Forced masking: all active bytes are forced to the value of the next parameter.	UINT16
Value	Force register value	UINT16

Fig. 11-10: ForceIOCLC() parameters

ReleaseIOCLC();

All forced registers are released.

```
void ReleaseIOCLC( UINT8 Address );
```

Parameter	Description	Type
Address	CLC device address	UINT8

Fig. 11-11: ReleaseIOCLC() parameter

11.5 CLC Serial Jog Command

CLC Settings for Serial Jogging

The CLC offers a serial jog command with communication monitoring. With this command, a CLC IO register may be set by a serial command and be cleared immediately afterwards if the command is not repeated within an adjustable timeout value. This value is specified in the CLC-Card-Parameter C-0-0016 communication timeout period. Its range is 50...2000 ms.

The minimum repeat time value in conjunction with the ScreenManager project is approximately 100ms. To allow at least one message frame to be repeated, a value of 200ms is recommended. It is possible to display other CLC parameters during jogging. In this case, the update rate will drop by 40% because every 2nd message frame is a jog command.

Strongly recommended:`C-0-0016 = 200 ms`

Caution: Inside of safety gates serial jogging may only be used in conjunction with the BTC06's Liveman and E-Stop switch, which must both be hard wired.

JogCLC();

The JogCLC() function generates a real-time link between a BTV/BTC ScreenManager I/O register bit and a CLC I/O register bit.

Global bindings must be made before the first Screen() call. They can only be released by resetting the small operator input unit. Every local binding that is made after the first Screen() call is released as soon as a new screen is activated. If a local link is active and the system invokes a new screen (e. g. in a event script), the jog function will be terminated before the screen changes. To reactivate a new jog command, the corresponding bit must be cleared and set again (the key must be released and pressed again).

Only one bit can be influenced at any one time. If a 2nd connected bit becomes active it will be ignored and the first operation remains active.

```
JogCLC( UINT8 Address, UINT16 CLC_IO_Register, UINT8
CLC_IO_Bit, UINT16 BTV_IO_Register, UINT8 BTV_IO_Bit);
```

Parameter	Description	Type
Network Address	CLC device address	UINT8
CLC_IO_Register	CLC I/O Register	UINT16
CLC_IO_Bit	Bit in register, following the CLC definition (first bit has the number 1)	UINT8
BTV_IO_Register	BTV/BTC I/O Register	UINT16
BTV_IO_Bit	Bit in register, following the ScreenManager definition (first bit has the number 1)1	UINT8

Fig. 11-12: JogCLC() parameters

Example

This example enables serial jogging of the first axis. In this example the timeout value will be rewritten every time the screen is activated. This is necessary because Jog Control of VisualMotion resets this value to 2000 ms every time it is started. If 2000ms are save enough for the application or if it is impossible, that Jog Control can be activated by VisualMotion, this command may be superfluous.

```
FLOAT fPosX; // position X-axis (global variable)
...

void JogScreen()
{
   INT16 sSaveTimeOut = 200; // local variable for CLC write

   Screen();
   TextAttr(1,1,0,0,1);
   Text(0,0,128,8,"Serial Jogging");
   TextAttr(1,0,0,0,0);
// option: set timeout register to default value
   sSaveTimeOut.WriteCLC(6,"CP",0,16); // Address: 6, CP-0-0016
// wait until this register is written
   WaitCLC();
// cyclic read axis position
   fPosX.BindCLC ( 6, "AP", 1, 102); // Address: 6, AP-1-0102
   Text (0, 20, 50, 8, "X-Axis:");
// cyclic printout axis position on screen
   fPosX.Display (65, 20, 0, 0, 9, 3);
// Bus address 6, 11-3 (CLC Jog Register) 10-15 (BTV Jog- Key)
   JogCLC(6,11,3,10,15);
// Bus address 6, 11-2 (CLC Jog Register) 10-14 (BTV Jog+ Key)
   JogCLC(6,11,2,10,14);
}
```

11.6 CLC List Commands

WriteListCLC()

The *WriteListCLC()* command writes a ScreenManager List into the specified CLC list. This function initiates the writing of the list and blocks script execution until the process is completed. Relay mode is disabled during this time.

```
void WriteListCLC( UINT8 Address, STRING Command,
UINT16 Set, UINT16 Number, UINT16 ListHandle);
```

Parameter	Description	Type
Network Address	CLC device address	UINT8
Command: Class and Subclass	Command class structure	String
Set	Set of specified data	UINT16
Number	Data number	UINT16
ListHandle	ScreenManager ListHandle to use identifies the list if more than one is used.	UINT16

Fig. 11-13: WriteListCLC()-parameters

ReadListCLC()

The *ReadListCLC()* command reads a CLC list into the specified ScreenManager list. This function initiates the reading of the list and blocks script execution until the process is completed. Relay mode is disabled during this time. The list remains in the memory until it is deleted or overwritten.

```
UINT16 ReadListCLC( UINT8 Address, STRING Command,
UINT16 Set, UINT16 Number, UINT8 bEnableColumns);
```

Parameter	Description	Type
Network Address	CLC device address	UINT8
Command: Class and Subclass	Command class structure	String
Set	Set of specified data	UINT16
Number	Data number	UINT16
bEnableColumns		UINT8

Fig. 11-14: ReadListCLC() parameters

12 Linking Variables to an Indramat CLM/DLC/ELC

12.1 CLM Data Declaration

Parameter	Description	Type
Network Address	CLM device address	UINT8
Command	CLM request string	STRING
Start	Start position of selected parameter in CLM reply line	UINT16
End	End position in CLM reply line	UINT16

Fig. 12-1: BindCLM, ReadCLM and WriteCLM parameters

A network address of zero results in a blank – this corresponds to the syntax of the RS232 mode of the CLM (only RS232 MODE! NOT RS232 link. This may still be established via RS485. In industrial applications it is strongly recommended to do this.) Numbers between 1 and 32 are converted into the CLM station address. See also CLM communication protocol document. Variables may be used for every parameter.

Question mark, station number, checksum and CR+LF are added by the system:

Serial CLM command: ?s**K 0402** $csCR LF

Required value of the 'Command' parameter: **"K 0402"**

Serial CLM command: ?s**N0007** $csCR LF

Required value of the 'Command' parameter: **"N0007"**

Note: To communicate with ScreenManager, the checksum option and the serial acknowledgement ('Y') on the CLM side must be activated.

12.2 CLM System Commands

The following ScreenManager system commands are needed for CLM access:

ConnectCLM();

ConnectCLM() enables CLM communication. This command must be executed before the first CLM data access. This command looks for the first port configured as ASCII. This port is used for all successive CLM commands as the communication master port. If no ASCII port is available, the following error message is issued: "No ASCII port for CLM, off-line mode only. Press any key to continue".

If a CLM port is found, the function looks for a remaining port configured as ASCII or ASCII&SIS-Slave. This port is set up as the CLM relay port. This port may be used to connect a PC or another BTV unit in a daisy-chain structure.

```
void ConnectCLM();
```

ConnectCLM() does not possess any parameters.

ConnectCLM() must not be invoked in an off line simulation of screens. All other CLM functions are then ignored by the system and do not generate an error message.

Connecting an ELC via the Binary SIS Protocol

If the first port is set up as an 'SIS Master' the ScreenManager connects to an ELC using the SIS protocol instead of ASCII. If real-time connection is established, the label and jog keys with LEDs and the I/O ports are connected with the ELC. All CLM communication is conducted through this SIS channel. For the ScreenManager program, there is no difference between ASCII and SIS.

ConnectCLMR();

Although ConnectCLMR() has the same function as ConnectCLM(), the peculiarities of a CLM-R with Rollfeed software is taken into account. All the other CLM functions are valid without any changes. Some control message frames, however, are handled in a different way.

NumberOfCLM()

NumberOfCLM() scans the bus for the number of connected stations. The last number to be scanned is defined in the BTV parameters. The return value is the number of stations found. This command stops script execution until the scan is finished.

```
UINT8 NumberOfCLM();
```

NumberOfCLM() does not possess any parameters.

AddressOfCLM()

AddressOfCLM() returns the station address to the index parameter.

0 = first station address

1 = second station address

n = last station address

n= return value of NumberOfCLM() minus 1. These two commands could be used to generate general-purpose programs for automatic detection.

```
UINT8 AddressOfCLM(UINT8 Index);
```

Parameter	Description	Type
Index	CLM device in order of appearance	UINT8

Fig. 12-2: AddressOfCLM() parameter

WaitCLM();

To increase the performance, all variable access procedures are performed in the background. The WaitCLM() command can be used for forcing the BTV unit to wait until all write and read access procedures of the CLM are terminated. This command must be used if equations or conditions depend on a CLM write or read command that was issued immediately before.

```
WaitCLM();
```

WaitCLM() does not possess any parameters.

Example:

```
INT32 a = 0;
{
   a.ReadCLM( 6, "X  00", 6, 15);
   Result = a * 22;
...
```

In this example, Result remains zero, because at the moment when the comparison was made, "a" was not read from the CLM. A working example would be:

```
INT32 a = 0;
{
   a.ReadCLM( 6, "X  00", 6, 15);
   WaitCLM();
   Result = a * 22;
...
```

This example would use a value of "a" that is sure to be read from the CLM. Execution is slower, since the ScreenManager program waits until all pending serial read and write commands are completed.

12.3 CLM Data Commands

BindCLM();

The BindCLM() command generates a link between ScreenManager and CLM data object. The ScreenManager variable is cyclically updated with the CLM value.

Global bindings must be made before the first Screen() call. They can only be released by resetting the small operator input unit. Every bound variable uses transmission time. Use global bindings only if necessary.

Every local binding that is made after the first Screen() call is released as soon as a new screen is activated.

If a bound variable is also linked with an edit cell using the Edit() command, it is automatically written to the CLM after it has been modified by the user.

Changing a variable within an equation (e. .g. a = 99;) does not initiate a command that writes the new value to the CLM. This must be done using the WriteCLM(); command.

```
Identifier.BindCLM( UINT8 Address, STRING Command,
UINT16 Start, UINT16 End);
```

Parameter	Description	Type
Network Address	CLM device address	UINT8
Command	CLM request string	STRING
Start	Start position of selected parameter in CLM reply line	UINT16
End	End position in CLM reply line	UINT16

Fig. 12-3: BindCLM()-parameters

Note: This command may also be used with any numeric variable type if it is used only for reading a value. The received text is automatically converted into a number. If the command is also used for editing variables (using the Edit() / EditFilter() command), only the type STRING may be used. See the application examples for some further tips and exceptions.

Caution: Only allowed in conjunction with global variables.

ReadCLM();

The value is read from the CLM only once. As with the BindIOCLM() command, the value is read from the CLM in background operation. The WaitCLM() command must be used if the subsequent lines of program code depend on this read value.

```
Identifier.ReadCLM( UINT8 Address, STRING Command,
UINT16 Start, UINT16 End);
```

Parameter	Description	Type
Network Address	CLM device address	UINT8
Command	CLM request string	STRING
Start	Start position of selected parameter in CLM reply line	UINT16
End	End position in CLM reply line	UINT16

Fig. 12-4: ReadCLM()-parameters

Example:

```
STRING Firmware[30];
...
{
   Firmware.ReadCLM( 6, "X  19", 6, 34);
   Text ( 0,0,128,10, Firmware);
...
```

Wrong, this writes an empty string, because the variable has not yet been read. A functioning but time-consuming example would be:

```
STRING Firmware[30];
...
{
   Firmware.ReadCLM( 6, "X  19", 6, 34);
   WaitCLM();
   Text ( 0,0,128,10, Firmware);
...
```

This command blocks until the parameter is read. A more efficient implementation would be:

```
...
{
   Firmware.ReadCLM( 6, "X  19", 6, 34);
   Firmware.Display(0,0,0,0);
...
```

The lines of program code are executed as quickly as possible and the Display command shows the CLM value as soon as it is read from the CLM.

Caution: Only allowed in conjunction with global variables.

WriteCLM();

A serial write to the CLM is initiated.

```
Identifier.WriteCLM( UINT8 Address, STRING Command,
UINT16 Start, UINT16 End);
```

Parameter	Description	Type
Network Address	CLM device address	UINT8
Command	CLM request string	STRING
Start	Start position of selected parameter in CLM reply line	UINT16
End	End position in CLM reply line	UINT16

Fig. 12-5: WriteCLM() parameters

Note: May be used only in conjunction with STRING variables, which must have the correct CLM format. Use the KonvStr() command to generate such strings from the numeric types of ScreenManager.

SendCLM();

Sends a serial command to the CLM.

```
SendCLM( UINT8 Address, STRING Command);
```

Parameter	Description	Type
Network Address	CLM device address	UINT8
Command	CLM request string	STRING

Fig. 12-6: SendCLM() parameters

The command contains only the command string itself.

```
"START", "STOP", "CLEAR"...
```

Example - Programming the BTV04 'clear'-key:

```
void ClearCLM()
{
   SendCLM("CLEAR");
}
...
void main()
{
   ConnectCLM();
...
   Key(47,1,ClearCLM); // clear key, on press
...
```

12.4 CLM (DLC) Serial Jog Command

At this time the serial jog function is only implemented in the Indramat DLC.

CLM (DLC) Setup

The CLM controller offers a serial jog command with communication monitoring. With this command, a DLC system input register may be set by a serial command and be cleared immediately afterwards if the command is not repeated within an adjustable timeout value. The value of this time is specified in parameter B004, item five. Valid entries are 1 (=25ms) through 9 (=225ms).

The minimum repeat time value in conjunction with the ScreenManager project is approximately 100ms. To allow at least one message frame to be repeated, a value of 200ms is recommended. It is possible to display other DLC parameters during jogging. In this case, the update rate will drop by 40% because every 2nd message frame is a jog command.

Caution: Inside of safety gates serial jogging may only be used in conjunction with the BTC06 Liveman and E-Stop switch, which are both hard wired.

JogCLM();

The JogCLM() function generates a real-time link between a BTV/BTC ScreenManager I/O register bit and a DLC system input bit.

Global bindings must be made before the first Screen() call. They can only be released by resetting the small operator input unit. Every local binding that is made after the first Screen() call is released as soon as a new screen is activated. If a local link is active and the system invokes a new screen (e. g. in an event script), the jog function will be terminated before the screen changes. To reactivate a new jog command, the corresponding bit must be cleared and set again (the key must be released and pressed again).

Only one bit can be influenced at any one time. If a 2nd connected bit becomes active it will be ignored and the first operation remains active.

```
JogCLM( UINT8 Address, UINT16 CLM_Input, UINT8
CLM_InputBit, UINT16 BTV_IO_Register, UINT8
BTV_IO_Bit);
```

Parameter	Description	Type
Network Address	CLM device address	UINT8
CLM_Input	CLM system input	UINT16
CLM_InputBit	Bit in system input (first bit has the number 0)	UINT8
BTV_IO_Register	BTV/BTC I/O Register	UINT16
BTV_IO_Bit	Bit in register (first bit has the number 0)	UINT8

Fig. 12-7: JogCLM() parameters

Example:

This example enables serial jogging of the first axis.

```
FLOAT fPosX; // position X-axis (global variable)
...

void JogScreen()
{
   Screen();
   TextAttr(1,1,0,0,1);
   Text(0,0,128,8,"Serial Jogging");
   TextAttr(1,0,0,0,0);
// cyclic read axis position
   fPosX.BindCLM( 1, "X  00", 5, 15 );// parameter X00 position
   Text (0, 20, 50, 8, "X-Axis:");
// cyclic printout axis position on screen
   fPosX.Display (65, 20, 0, 0, 9, 3);
// Bus address 1, 0-5 (DLC system input Jog+) 10-14 (BTV Jog+ Key)
   JogCLM(1,0,5,10,14);
// Bus address 1, 0-6 (DLC system input Jog-) 10-15 (BTV Jog- Key)
   JogCLM(1,0,6,10,15);
}
```

12.5 ELC/FLP Command Extensions

ConnectELC();

ConnectELC() enables ELC communication using the binary SIS protocol. This command must be executed before the first ELM data access. The command looks for the first port that is configured as SIS-Master. This port is used for all successive ELC commands as the communication master port. If no SIS port is available, the following error message is generated: "no SIS master port for ELC, off-line mode only, press any key to continue"

If port one is available for ELC, the system looks whether the remaining port is configured as ASCII or ASCII&SIS-Slave. This port is set up as an ELC relay port. This port may be used to connect a PC or another BTV unit (both in ASCII mode) in a daisy-chain structure.

This command opens the port for ELC data access. The specified address will only be used to start real-time data exchange to the specified station. Without disturbing the real-time connection, all other data transfer will be established to the address that is specified in the individual **Bind**, **Read** or **Write** commands.

This command may be used multiple times to select another station for real-time data exchange. Only one real-time connection is active at any one time.

Specifying address = 0 terminates the real-time data connection.

Once this command has been executed, all Ecodrive3 commands may also be used. In special cases, Ecodrive3 (depending on the Ecodrive firmware and the time-out and repetition time settings) may be used together with ELC on the same RS485 bus.

```
void ConnectELC ( UINT8 Address );
```

Parameter	Description	Type
Network Address	ELC device address	UINT8

Fig. 12-8: ConnectELC() parameters

The ConnectELC() command may not be used to simulate screens off-line. All other ELC functions are then ignored by the system and do not generate an error message.

NumberOfELC ()

NumberOfELC() searches the bus for the number of connected stations. The last number to be prompted is defined in the BTV-parameters. The return value is the number of found stations. This command stops the script processing as long as the search is finished.

```
UINT8 NumberOfELC();
```

NumberOfELC() has no parameters.

AddressOfELC ()

AddressOfELC() returns the station address to the index parameter.

0 = first station address

1 = second station address

n = last station address

n= return value of NumberOfELC() minus 1. With this commands it is possible to generate universal programs for automatic recognition.

```
UINT8 AddressOfELC(UINT8 Index);
```

Parameter	Description	Type
Index	ELC device in the order of their appearance	UINT8

Fig. 12-9: AddressOfELC() Parameter

InitProgListELC()

This command loads the NC command and format lists from the ELC. This call must be made before ...NCELC() commands are used.

```
UINT16 InitProgListELC( UINT8 Address, STRING
CommandList, STRING ParameterList );
```

Parameter	Description	Type
Network Address	ELC device address	UINT8
CommandList	In the form of a list, this string contains all commands (as a consecutive character string – matching the EditList() command), that are ready for being used in conjunction with the EditList() commands.	STRING
ParameterList	In the form of a list, this string contains all ELC parameter blocks that are ready for being used in conjunction with EditList() commands.	STRING

Fig. 12-10: InitProgListELC() parameters

Return value: Handle for the ELC format list.

WaitELC();

For faster performance, all variable access should be conducted in the background. WaitELC() is the command to force the BTV unit to wait until all ELC read and write access operations are finished. This command must be used if comparisons or assignments depend on a ELC read or write command that was just issued (forced synchronous reading). This command has the same functionality as WaitCLM().

```
WaitELC();
```

WaitELC() does not possess any parameters.

BindNCELC();

The BindNCLC() method generates a link between a variable from the ScreenManager and an ELC program line parameter. The variable is cyclically updated with the ELC value.

This command extracts the parameter out of the NC line using a specific format for every NC command. To generate this format list, the InitProgListELC() command must first be called to get the current list from an ELC. This command makes it possible to read and write numerical ScreenManager variable types. This is not possible with the BindCLM() command.

If multiple interconnections are established to the same line but to different parameters, the system optimizes these to a single call to the ELC.

If an interconnected variable is also linked with an edit cell using the Edit() command, it is automatically written to the ELC after it has been modified by the user.

Changing a variable within an equation (e. .g. a = 99;) does not initiate a command that writes the new value to the ELC. This must be done using the WriteINCELC(); command.

```
Identifier.BindNCELC( UINT8 Address, UINT16 Line,
UINT16 Parameter );
```

Parameter	Description	Type
Network Address	ELC device address	UINT8
Line	ELC program line	UINT16
Parameter	Parameter number, 0 = Name of the NC command in textual form (in conjunction with a STRING), or the index number within the command list (in conjunction with numerical types).	UINT16

Fig. 12-11: BindNCELC()-parameters

Caution: Only allowed in conjunction with global variables.

ReadNCELC();

Same function as BindNCELC, but reads the value only once.

```
Identifier.ReadNCELC( UINT8 Address, UINT16 Line,
UINT16 Parameter );
```

Parameter	Description	Type
Network Address	ELC device address	UINT8
Line	ELC program line	UINT16
Parameter	Parameter number, 0 = Name of the NC command in textual form (in conjunction with a STRING), or the index number within the command list (in conjunction with numerical types).	UINT16

Fig. 12-12: ReadNCELC() parameters

Caution: Only allowed in conjunction with global variables.

WriteNCELC();

Writes any variable type to an ELC program line. Multiple write commands are collected and transmitted together to the ELC after the script execution.

```
Identifier.WriteNCELC( UINT8 Address, UINT16 Line,
UINT16 Parameter );
```

Parameter	Description	Type
Network Address	ELC device address	UINT8
Line	ELC program line	UINT16
Parameter	Parameter number, 0 = Name of the NC command in textual form (in conjunction with a STRING), or the index number within the command list (in conjunction with numerical types).	UINT16

Fig. 12-13: WriteNCELC() parameters

BindParameterELC();

The method using BindParameterELC() generates a link between a ScreenManager variable and an ELC parameter. The variable is cyclically updated with the ELC value.

This command extracts one item out of the parameter line using the specific format of each parameter. To generate this format list, the InitProgListELC() command must first be called to get the current list from an ELC. This command makes it possible to read and write numerical ScreenManager variables.

If a bound variable is also linked with an edit cell using the Edit() command, it is automatically written to the ELC after it has been modified by the user.

A change of the variable within a comparison (e .g. a = 99;) does not initiate a write process to the ELC. This requires the WriteParameterELC(); command to be used.

```
Identifier.BindParameterELC( UINT8 Address, STRING
Block, UINT16 Number, UINT16 Element );
```

Parameter	Description	Type
Network Address	ELC device address	UINT8
Block	ELC parameter block, e.g. B0, A1...	STRING
Number	Parameter number in this block	UINT16
Element	Element in the parameter line	UINT16

Fig. 12-14: BindParameterELC() parameters

Caution: Only allowed in conjunction with global variables.

ReadParameterELC();

Same function as BindParameterELC, but reads the value only once.

```
Identifier.ReadParameterELC( UINT8 Address, STRING
Block, UINT16 Number, UINT16 Element );
```

Parameter	Description	Type
Network Address	ELC device address	UINT8
Block	ELC parameter block, e.g. B0, A1...	STRING
Number	Parameter number in this block	UINT16
Element	Element in the parameter line	UINT16

Fig. 12-15: ReadParameterELC() parameters

WriteParameterELC();

Writes any variable type to an ELC parameter item.

```
Identifier.WriteParameterELC( UINT8 Address, STRING Block, UINT16 Number, UINT16 Element );
```

Parameter	Description	Type
Network Address	ELC device address	UINT8
Block	ELC parameter block, e.g. B0, A1...	STRING
Number	Parameter number in this block	UINT16
Element	Element in the parameter line	UINT16

Fig. 12-16: WriteParameterELC() parameters

ParameterCountELC();

Returns the number of ELC parameters in the specified block. The ScreenManager gets this information out of the currently loaded format list during runtime.

```
UINT16 ParameterCountELC( STRING Block );
```

Parameter	Description	Type
Block	ELC parameter block, e.g. B0, A1...	STRING

Fig. 12-17: ParameterCountELC() parameters

EditELC();

EditELC() is a special Edit command for elements of ELC programs and parameter lines. Field size, position and format are extracted from the ELC format list. This command should be used to implement NC programs and parameter screens for the ELC. This screens are a part of an INDRAMAT standard application.

Caution: Only allowed in conjunction with global variables and may be used only within screen creation scripts.

```
Identifier.EditELC(UINT16 ProgList, UINT16 Index, UINT16 Element );
```

Parameter	Description
ProgList	Handle returned from InitProgListELC. It identifies the ELC format list loaded from the controller.
Index	NC program: Index of the current NC command – same as the return value from ReadNCELC (*,*,0); ELC parameter: Return value from GetParamFormatIdxELC()
Element	Program line or program number

Fig. 12-18: EditELC() parameters

DisplayELC();

DisplayELC() has the same functionality as EditELC() but without the option of editing a value.

Caution: Only allowed in conjunction with global variables and may be used only within screen creation scripts.

```
Identifier.DisplayELC(UINT16 ProgList, UINT16 Index,
UINT16 Element );
```

Parameter	Description
ProgList	List handle for ELC format list. This must be the return value of the InitProgListELC() command.
Index	NC program: Index of the current NC command – same as the return value from ReadNCELC (*,*,0); ELC parameter: Return value from GetParamFormatIdxELC()
Element	Program line or program number

Fig. 12-19: DisplayELC() parameters

InitProgListFLP(); version 5 and later

This command loads the format lists of the NC commands, PLC commands and parameters from the FLP. This call must be made before the ...NCELC() command or the ...PLCFLP() command is used.

```
UINT16 InitProgListFLP( UINT8 Address, STRING
NcCommandList, STRING ParameterList, STRING
PLCCommandList );
```

Parameter	Description	Type
Network Address	FLP device address	UINT8
NcCommandList	This string contains all NC command format lists that are required in conjunction with ...NCELC() commands.	STRING
ParameterList	This string contains all parameter format lists that are required in conjunction with ...ParameterELC() commands.	STRING
PlcCommandList	This string contains all PLC command format lists that are required in conjunction with ...PLCFLP() commands.	STRING

Fig. 12-20: InitProgListFLP()

Return value: Handle for the FLP format list.

BindPLCFLP(); version 5 and later

The BindPLCFLP() method generates a link between a variable from the ScreenManager and an PLC program line element. The variable is cyclically updated with the FLP value.

This command extracts the parameter out of the PLC line using a specific format for each PLC command. To generate this format list, the InitProgListFLP() command must first be called to get the current list from an FLP. This command makes it possible to read and write numerical ScreenManager variable types. This is not possible with the BindCLM() or BindNCELC() command.

If multiple interconnections are established to the same line but to different elements, the system optimizes these to a single call to the FLP.

If a bound variable is also linked with an edit cell using the Edit() command, it is automatically written to the FLP after it has been modified by the user.

Changing a variable within an equation (e. .g. a = 99;) does not initiate a command that writes the new value to the FLP. This must be done using the WriteIPLCFLP(); command.

Identifier.BindPLCFLP(UINT8 Address, UINT16 Line, UINT16 Parameter);

Parameter	Description	Type
Network Address	FLP device address	UINT8
Line	PLC program line	UINT16
Parameter	Element number, 0 = Name of the NC command in textual form (in conjunction with a STRING), or the index number within the command list (in conjunction with numerical types).	UINT16

Fig. 12-21: BindPLCFLP() elements

Caution: Only allowed in conjunction with global variables.

ReadPLCFLP(); version 5 and later

Same function as BindPLCFLP(), but reads the value only once.

```
Identifier.ReadPLCFLP( UINT8 Address, UINT16 Line,
UINT16 Parameter );
```

Parameter	Description	Type
Network Address	FLP device address	UINT8
Line	PLC program line	UINT16
Parameter	Element number, 0 = Name of the NC command in textual form (in conjunction with a STRING), or the index number within the command list (in conjunction with numerical types).	UINT16

Fig. 12-22: ReadPLCFLP() - element

Caution: Only allowed in conjunction with global variables.

WritePLCFLP(); version 5 and later

Writes any variable type to a PLC program line. Multiple write commands are collected and transmitted together to the FLP after the script execution.

```
Identifier.WriteNCELC( UINT8 Address, UINT16 Line,
UINT16 Parameter );
```

Parameter	Description	Type
Network Address	FLP device address	UINT8
Line	PLC program line	UINT16
Parameter	Element number, 0 = Name of the NC command in textual form (in conjunction with a STRING), or the index number within the command list (in conjunction with numerical types).	UINT16

Fig. 12-23: WritePLCFLP() element

13 Linking Variables to an Indramat ECODRIVE03

13.1 DKC Data Description

SERCOS Parameter Number

The ECODRIVE parameters are addressed via their SERCOS parameter numbers.

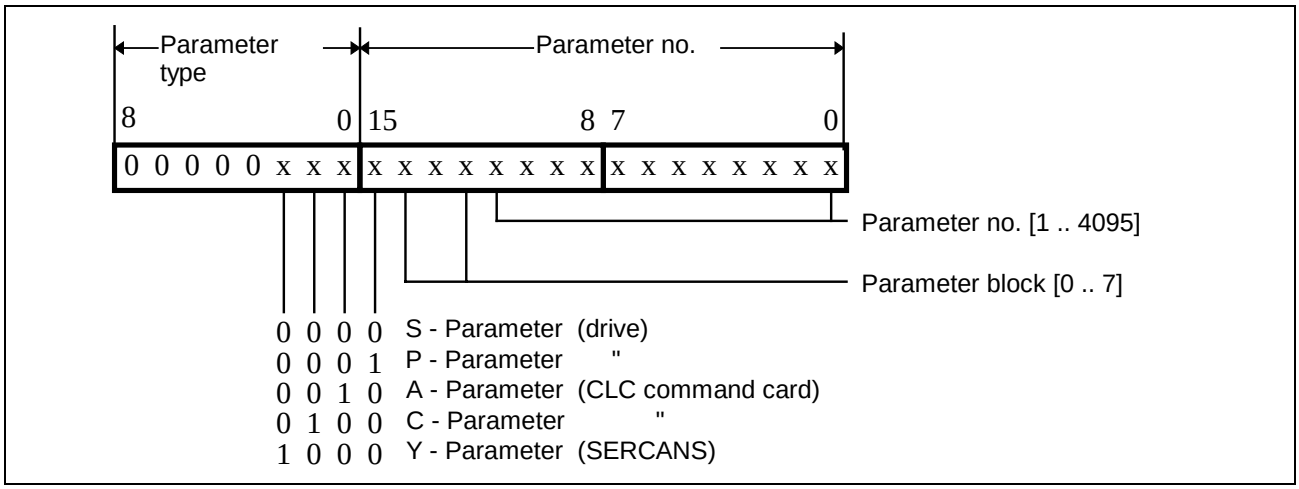


Fig. 13-1: Structure of the SERCOS parameter number

Examples:

The S-0-0010 parameter is 10

The P-0-0109 parameter becomes 2^{15} + 109 = 32877

The ScreenManager library supports the programmer with conversion routines in both directions. For better performance in program sequences (e.g. loops saving multiple parameters) the numerical types should be preferred.

Data Element

Each parameter consists of the following elements:

Element number	Meaning	Return value
0	Data status	UINT16 - see next table
1	SERCOS ident number	UINT or STRING
2	Name	String
3	Attribute	UINT32
4	UNIT	String
5	min. input value	same as data item
6	max. input value	same as data item
7	**User data item**	**Depends on the requested parameter. Type conversion is performed automatically; STRING conversion is supported for any parameter.**
8	Parameter function	0-data 1-command
9	Type	0 - binary value 1 - positive number 2 - number 3 - positive hexadecimal number 4 - STRING 5 - SERCOS parameter number
10	Decimals	Number of fractional part digits (0..15)
11	Number of stored elements For strings: Number of characters	Number of elements in list parameters. A normal parameter returns the value 1. This value may also be written.
12	Max. number of elements For strings: max. length	Maximum number of elements in list parameters. A normal parameter returns the value 1.

Fig. 13-2: DKC data elements

Bit no.	Function	Return value
0	Command status	0-inactive 1-active
1	Command execution	0-interrupted 1-enabled
2	Command ready	0-executed 1-waiting, active
3	Command error	0-no error 1-execution not possible
8	Data status	0-parameters valid 1-parameters invalid

Fig. 13-3: DKC data status

In this special case and in conjunction with BOOL variables, the individual bits may be addressed using the 'ListElement'-parameter.

13.2 DKC Data Commands

ConnectDKC();

ConnectDKC() opens DKC communication. This call must be made before the first DKC data access. This command looks for the first interface that is configured as an SIS master. This interface is used for all subsequent DKC commands as the communication interface. If no SIS master interface is available, the following error message is generated: " no SIS master port for DKC, off-line mode only, press any key to continue".

```
void ConnectDKC();
```

ConnectDKC() does not possess any parameters.

Program execution can be simulated if the ConnectDKC() command is not called. All DKC calls are ignored; an error message is not output.

Note: The ConnectDKC() command has the same functionality as ConnectELC(0).

NumberOfDKC()

NumberOfDKC() scans the bus for the number of connected stations. The last number to be scanned is defined in the BTV parameters. The return value is the number of stations found. This command stops script execution until the scan is finished.

```
UINT8 NumberOfDKC();
```

NumberOfDKC() does not possess any parameters.

AddressOfDKC()

AddressOfDKC() returns the station address to the index parameter.

0 = first station address

1 = second station address

n = last station address

n= return value of NumberOfDKC() minus 1. These two commands could be used to generate general-purpose programs for automatic detection. It becomes useless if all station numbers begin with a 1 and if there are no gaps between them.

```
UINT8 AddressOfDKC(UINT8 Index);
```

Parameter	Description	Type
Index	DKC controller, in order of appearance	UINT8

Fig. 13-4: AddressOfDKC() parameters

WaitDKC();

For faster performance, all variable access procedures are conducted in the background. The WaitDKC() command forces the BTV unit to wait until all ELC read and write access procedures are finished. This command must always be inserted if jumps or calculations depend on variables that were just read.

```
WaitDKC();
```

WaitDKC() does not possess any parameters.

Example:

```
INT32 a = 0;
{
   a.ReadDKC( 6, 84, 7, 0); // read S-0-0084
   Result = a * 22;
...
```

In this example, Result remains zero, because at the moment when the assignment was made, "a" has not yet been read. A functioning example would be:

```
INT32 a = 0;
{
   a.ReadDKC( 6, 84, 7, 0); // read S-0-0084
   WaitDKC();
   Result = a * 22;
...
```

This example would use a value of "a" that is sure to be read from the DKC. The execution is slower, because the ScreenManager program is halted until all buffered read and write commands are completed.

Note: The WaitDKC() command has the same functionality as WaitELC(). Each command waits for the data from the controller.

BindDKC();

The BindDKC() command generates a link between ScreenManager and DKC parameters. The ScreenManager variable is cyclically updated with the DKC value.

Global interconnections must be made before the first Screen() call. They remain active as long as the system is switched on. Since every interconnected variable uses transmission time, global interconnections of variables should only be used if this is really necessary.

Any interconnection made after the first Screen() call is a local one. It will be released when Screen(); is called again.

If a variable that is interconnected with the DKC variable is also connected to an edit cell on the screen using the Edit() command, it will automatically be written to the DKC after it has been changed by the user and the 'OK' key pressed.

Changing an interconnected variable within an equation (e. g. a = 99;) does not initiate the new value to be written to the DKC. This must explicitly be programmed using the WriteDKC(); command.

```
Identifier.BindDKC (UINT8 Address, UINT32 Ident, UINT8
Element, UINT16 ListElement);
```

Parameter	Description	Type
Address	DKC device address	UINT8
Ident	SERCOS parameter as number	UINT32
Element	Parameter element	UINT8
ListElement	Element number of list parameters (the first element is at position 0) or bit number if this command is used in conjunction with a BOOL variable and element 0 (data status).	UINT16

Fig. 13-5: BindDKC() parameters

This command may be used with any variable type. When a string is interconnected, it will automatically obtain the correct format (text, decimal number, hexadecimal or bit string, depending on the SERCOS data attribute) during assignment. The system automatically reads the attribute upon each BindDKC().

Caution: Only permitted in conjunction with global variables.

ReadDKC();

The ReadDKC() command reads a DKC parameter once.

```
Identifier.ReadDKC (UINT8 Address, UINT32 Ident, UINT8
Element, UINT16 ListElement);
```

Parameter	Description	Type
Address	DKC device address	UINT8
Ident	SERCOS parameter as number	UINT32
Element	Parameter element	UINT8
ListElement	Element number of list parameters (the first element is at position 0) or bit number if this command is used in conjunction with a BOOL variable and element 0 (data status).	UINT16

Fig. 13-6: ReadDKC() parameters

This command may be used with any variable type. When a string is interconnected, it will automatically obtain the correct format (text, decimal number, hexadecimal or bit string, depending on the SERCOS data attribute) during assignment. The system automatically reads the attribute upon each ReadDKC().

Caution: Only permitted in conjunction with global variables.

WriteDKC();

The WriteDKC() command writes a DKC parameter.

```
Identifier.WriteDKC (UINT8 Address, UINT32 Ident,
UINT8 Element, UINT16 ListElement);
```

Parameter	Description	Type
Address	DKC device address	UINT8
Ident	SERCOS parameter as number	UINT32
Element	Parameter element	UINT8
ListElement	Element number of list parameters (the first element is at position 0)	UINT16

Fig. 13-7: WriteDKC() parameters

This command may be used with any variable type. When a string is interconnected, it will automatically obtain the correct format (text, decimal number, hexadecimal or bit string, depending on the SERCOS data attribute) during assignment. The system automatically reads the attribute upon each WriteDKC().

13.3 DKC Data Conversion Functions

Str2Ident()

Converts a SERCOS parameter of the STRING type into a numerical value of the UINT32 type.

```
UINT32 Str2Ident(STRING sIdent);
```

Example:

```
{
UINT32 ulIdent1;
UINT32 ulIdent2;
   ulIdent1 = Str2Ident("S-0-0109");
   ulIdent2 = Str2Ident("P-0-0109");
```

The results of this example are:

ulIdent1: 109

ulIden2: 32877 (2^{15} + 109)

Ident2Str()

Converts a SERCOS parameter of the UINT32 type into a STRING.

```
void Str2Ident(STRING sIdent, UINT32 ulIdent);
```

Example:

```
{
STRING sIdent1;
STRING sIdent2;
UINT32 ulIdent2 = 32877;
   Ident2Str(sIdent1, 109);
   Ident2Str(sIdent2, ulIdent);
```

The result is:

sIdent1: "S-0-0109";

sIdent2: "P-0-0109";

14 I/O Mapper

14.1 I/O Overview

The I/O Mapper is used for establishing the logical I/O (input and output) interconnections of the small operator input units. This enables fast internal connections between keys and 24-V terminals, for example, to be established. Implementing simple binary logic connections would be possible too. The I/O Mapper program is a part of a ScreenManager application. It is linked firmly to this application and is transferred to the control panel together with it. There, the I/O Mapper program runs in a fixed deterministic time schedule, independently of the ScreenManager application proper. It is started after the boot process, together with visualization.

An internal memory block is used for managing the I/O systems by the small operator input units. The memory is set up as a linear arrangement of 'registers'. Each register is 16 bit wide. Each bit can assume one of two values: either 1 or 0. This means, it is either ON or OFF. The memory assignment of the small operator input units permits a maximum of 100 I/O registers to be used.

General Arrangement of the I/O Registers

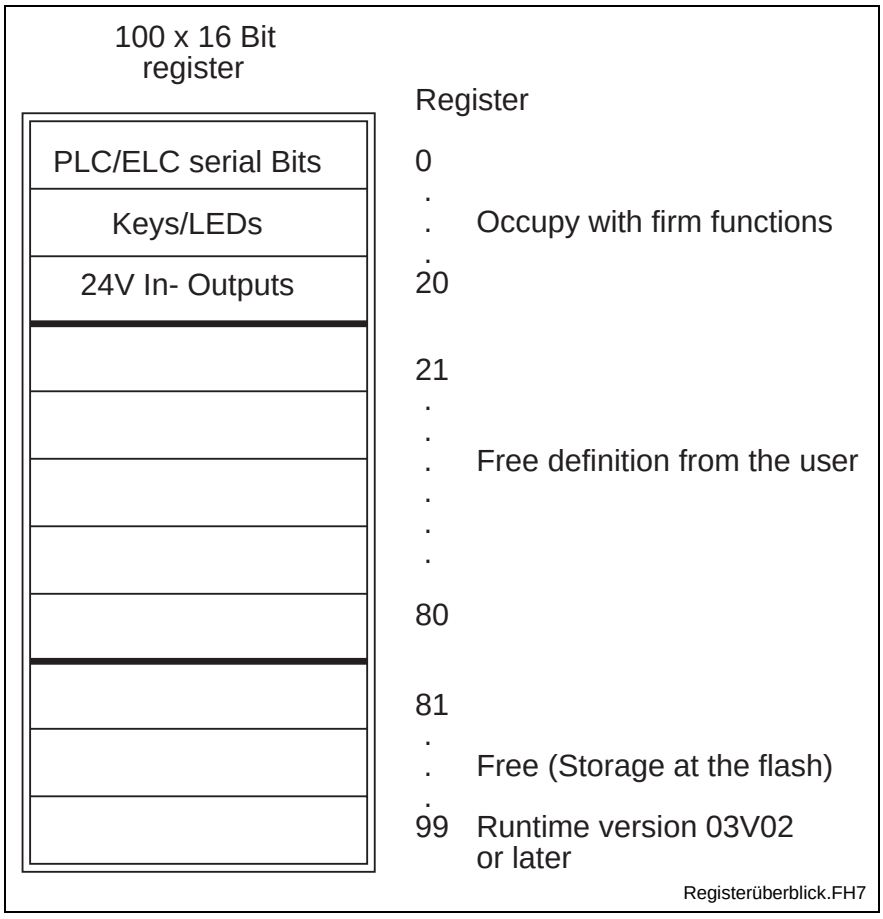


Fig. 14-1: General overview of the register arrangement

Note: Retentive I/O registers (flash) are only available in version 4 or later BTV units.

14.2 Interconnection Logic of the I/O Mapper

Overview

The I/O interconnection logic permits the I/O registers to be manipulated by using Boolean equations. The simple PLC functionality is based on the operators AND, OR and NOT. The PC desktop translates the equations into the CPU machine language of the operator input unit. The translated equations are executed at an invariable clock cycle (min. 2ms, depends on the number of interconnections), and independent of the other user tasks. This permits a predictable response to a large number of time-critical I/O events.

System developers can create application-specific I/O links, store them, and transfer them to all BTV or BTC units.

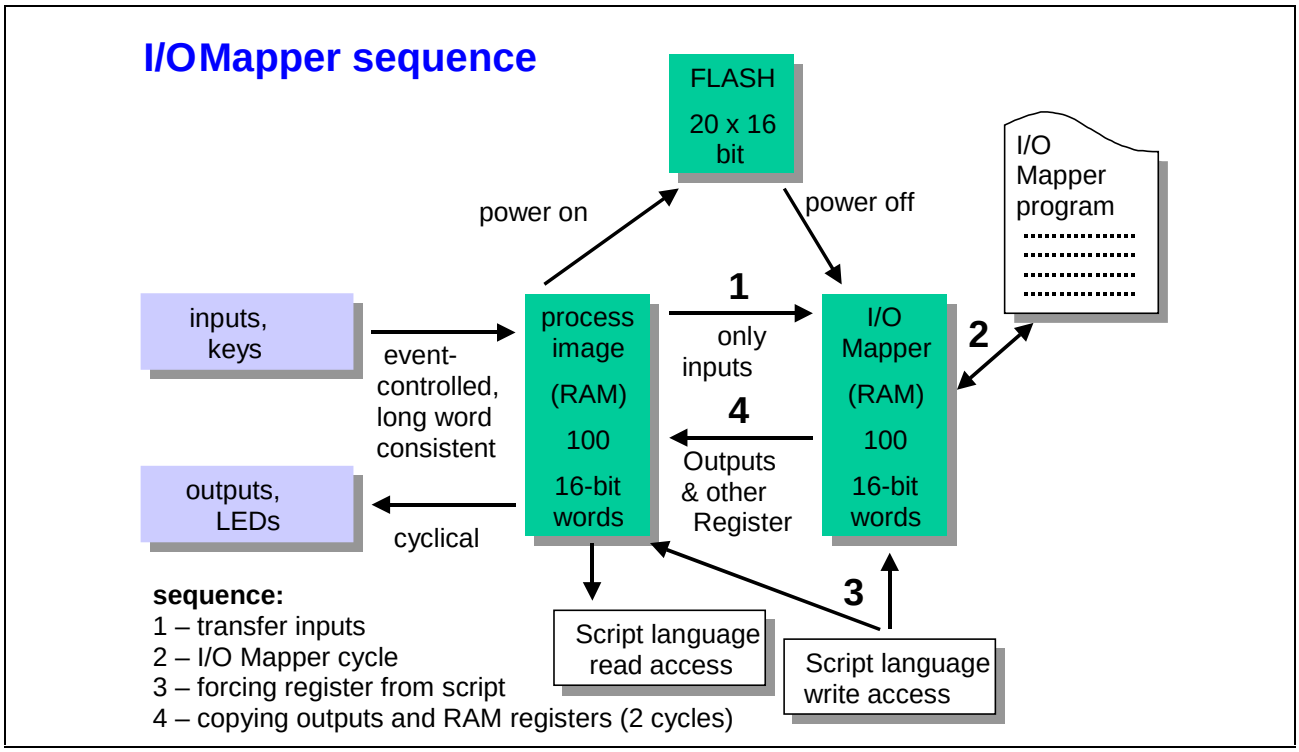


Fig. 14-2: Block diagram of the I/O Mapper

All access procedures to the I/O registers have consistently (cycle synchronous) been implemented. A process image is used for links to the I/O level of the control panel and read access procedures to the registers from the application. The write access of the application is synchronized to happen during the cycle pauses.

Note: If SIS real-time communication is activated, the I/O Mapper program stops immediately and is replaced with a fixed link between I/O and real-time data that runs at a 1-ms frame. The inputs and outputs of the control panel, machine control keys, associated LEDs, handwheel, and override will then exclusively be available to the connected controller. With INDRAMAT ISP, this process runs automatically as soon as it is connected with the control panel (even if an application has not been loaded). With an ELC single-axis controller, it is triggered by a ConnectELC() command with transferred station number. In either case, any previously active I/O Mapper program will **not** be restarted, even if communication with the connected controller is interrupted. In this case, all BTV outputs are set to the safe state 0 after the fixed time-out time of 600ms has expired. It must therefore be ensured that 0 is a safe state for each connected actuator.

Note: Retentive I/O registers (flash) are only available in version 03V02 or later BTV units. The retentive registers are initiated with 0 when a newly loaded application is started (deviating CRC check sum).

Interconnection Logic in Ladder Diagram Format

The I/O Mapper is used for creating the equations of ladder diagram I/O interconnection logic on the screen.

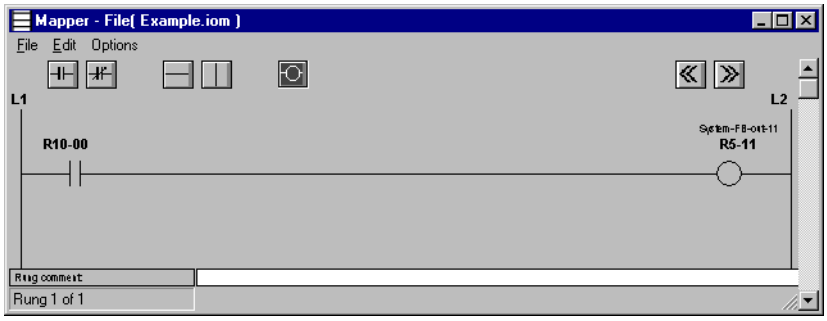


Fig. 14-3: Typical rung in ladder diagram format

If bit 0 of register 10 obtains an input from a physical I/O switch (here: Label1 key on the BTV04 unit), its state (open or closed) changes the state of bit 11 in register 5 (LED of Label1 key). This can be compared with using an electric switch for switching on/off a solenoid or lamp.

Format of an Interconnection String

The Boolean interconnection strings of the I/O interconnection logic are of the following format:

```
R4-1=R1-3&R1-4;comment
R4-1=R1-3&R1-4|((R8-15&!R7-0)|R13-0);comment
```

Syntax for the register: "R4-1" – Register word no. 4, bit no. 1

"=" – assignment

"&", "|" – operators

Note: Blanks are not permitted inside the expression.

Register: Integer value for an I/O register

Bit: Integer value in the range 0 through 15

Operator*:* Logic operator of the I/O interconnection logic (! & |)

The left-hand side of the equation is the result of the right-hand side. The string syntax is analyzed from left to right. The operators are executed during the syntax analysis; they are all of the same priority level.

A semicolon at the end of an I/O interconnection string permits comments to be entered.

Operators of the I/O Interconnection Logic

The following operators are permitted in I/O interconnection logic:

! **NOT**, the complementary operator. All bits are inverted.

& **AND**, bit-by-bit AND. The result is 1 if both bits are 1. Otherwise, the result is 0.

| **OR**, bit-by-bit inclusive OR. The result is 1 if either bit is 1.

( ) Parentheses permit intermediate results to be calculated, or a priority to be forced. A string can contain up to 16 nested operations.

Examples

5-0 = 12-5 & 12-7	5-14=11-1&11-2|!11-5	5-14=11-1&(11-2|!11-5)
Sets bit 0 of register 5 to TRUE if bit 5 of register 12 and bit 7 of register 12 are both TRUE.	Two ANDed bits and an "!OR" operation	Two "ORed" bits with an "AND"

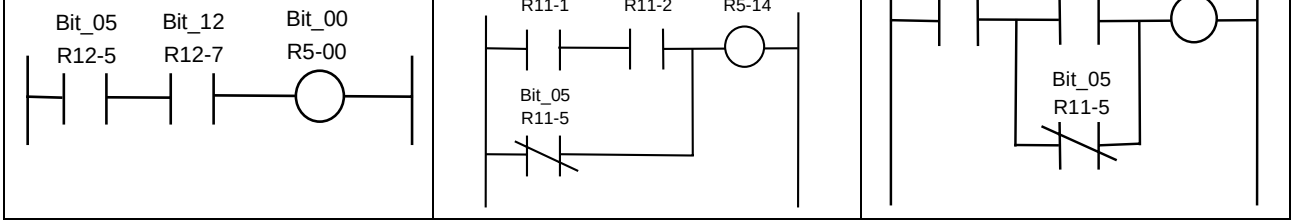


Fig. 14-4: Typical interconnection logic with the possible operators

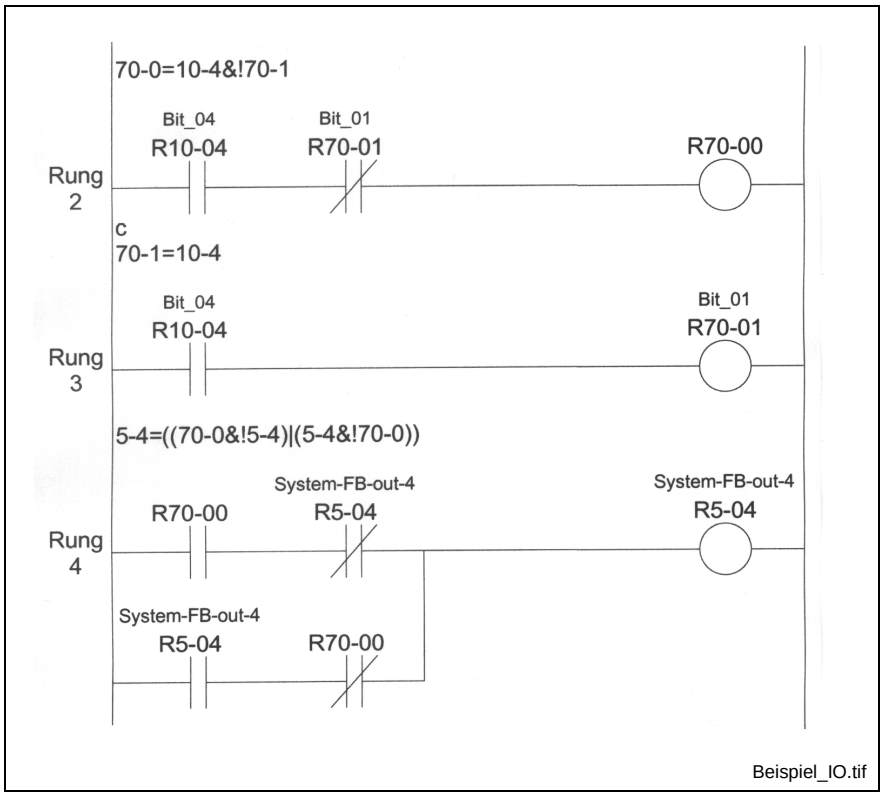


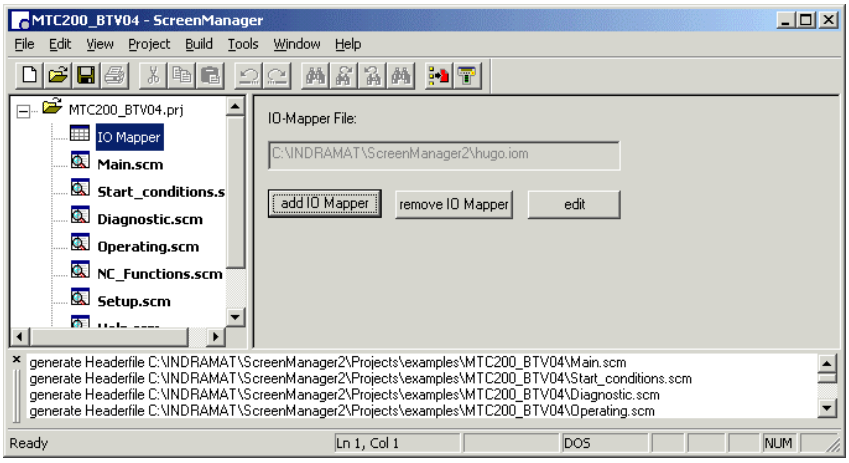
Fig. 14-5: I/O Mapper, typical edge interpretation

14.3 Handling the I/O Mapper Desktop

Invoking the I/O Mapper

From the ScreenManager, the I/O Mapper can be invoked as follows:

Select the I/O Mapper branch from the project tree. The following functions are now available at the right-hand side of the screen:



iom_8bit.bmp

Fig. 14-6: Invoking the I/O Mapper

add I/O Mapper

An existing I/O Mapper file (extension *.iom) can be selected from the subsequently displayed window. This file need not be in the project directory. This proves expedient if several ScreenManager applications shall use a common I/O mapper program. Entering a file name is an alternative possibility of creating a new file that should possibly be created in the default project directory.

Note: An I/O Mapper program that is integrated in a project must contain at least one valid rung. A newly created I/O Mapper file is empty. An immediate compilation of the project is therefore aborted, and an error message of the I/O compiler is issued.

remove I/O Mapper

Removes the inserted I/O Mapper program from the project. The I/O Mapper source file (*.iom) remains.

edit

Opens the graphical I/O editor. This runs as a separate program that is linked (if it has been invoked via the desktop) with the ScreenManager desktop. Pressing the Compile or Download button causes the I/O editor to save the current modifications before the compilation process. Explicitly saving the current modifications of the rungs prior to compilation is therefore not necessary.

Handling the I/O Editor

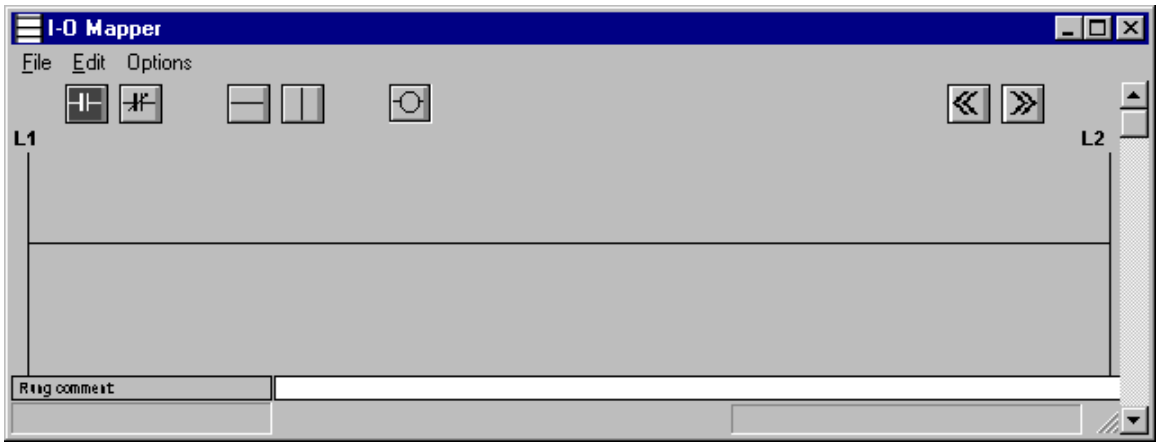


Fig. 14-7: I/O Mapper desktop

"File" Menu

New Opens a new I/O interconnection file in the "I/O Mapper". Is not required if the call is made via the ScreenManager desktop.

Open Used for opening an existing I/O interconnection file (*.iom). Is not required if the call is made via the ScreenManager desktop.

Save Used for saving an I/O interconnection logic in a previously named file.

Save as Used for saving an I/O interconnection logic in a new file whose name must be entered.

Print Output Option for printing the I/O interconnection file, or part thereof, including rungs, comments, equations, and cross references. The printout may also contain a project name and the name of the programmer.

"Edit" Menu

Add Rung A rung is appended at the end of the I/O interconnection logic.

Delete Rung The displayed rung is deleted from the I/O interconnection logic.

Insert Rung A new rung is inserted before the displayed rung into the I/O interconnection logic.

Copy Rung Copies the displayed rung into the clipboard.

Paste Rung The contents of the displayed rung is overwritten by the contents of the clipboard.

Check Checks the rung for missing contacts, coils, shorts and interruptions.

Comment Adds a comment to a rung. The comment may have a maximum of 80 characters.

Find... Used for locating a rung that contains a specific bit of a specific register.

Find Next Used for locating the next network with the same bit within the I/O interconnection logic. This option can be invoked with F3.

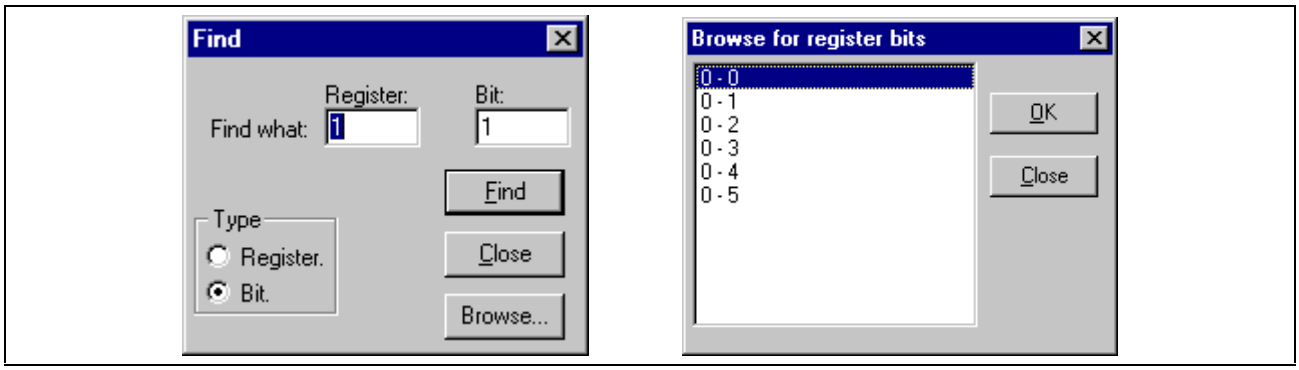


Fig. 14-8: "Find rung" option

"Options" Menu

Text editor: All rungs are shown as Boolean arguments in the form of ASCII text strings.

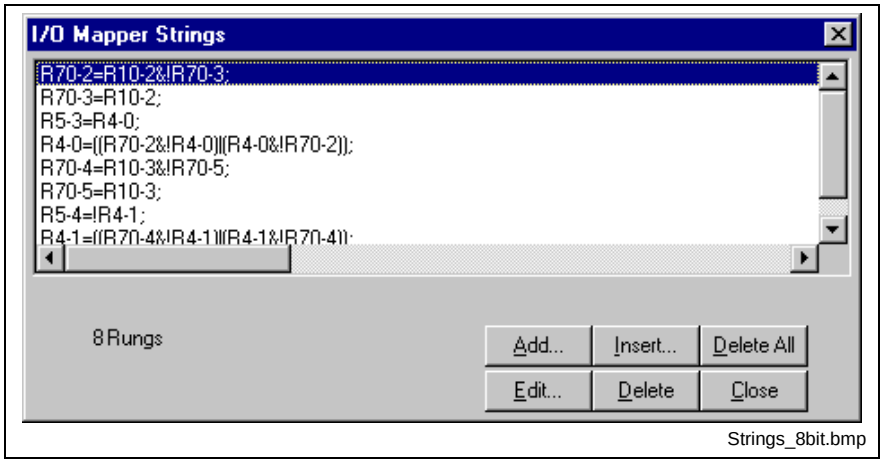


Fig. 14-9: Text editor for displaying and editing ASCII strings

Clicking on a string in the displayed list, or first selecting a string and then clicking on "Edit" invokes the "Edit Mapper String" dialog in which the I/O interconnection strings can be edited.

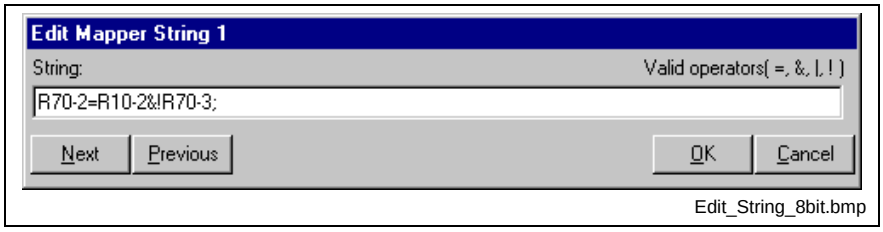


Fig. 14-10: Window for editing an existing string

A semicolon at the end of an I/O interconnection string permits a comment to be appended.

Handling Notes

Certain functions can directly be invoked via functions keys. The following processes are possible:

- F3: Locating rungs
- F5: Previous rung
- F6: Next rung

Functions of the Right-hand Mouse Button

The following functions can be invoked via the right-hand mouse button. This requires the cursor to sit on a cell.

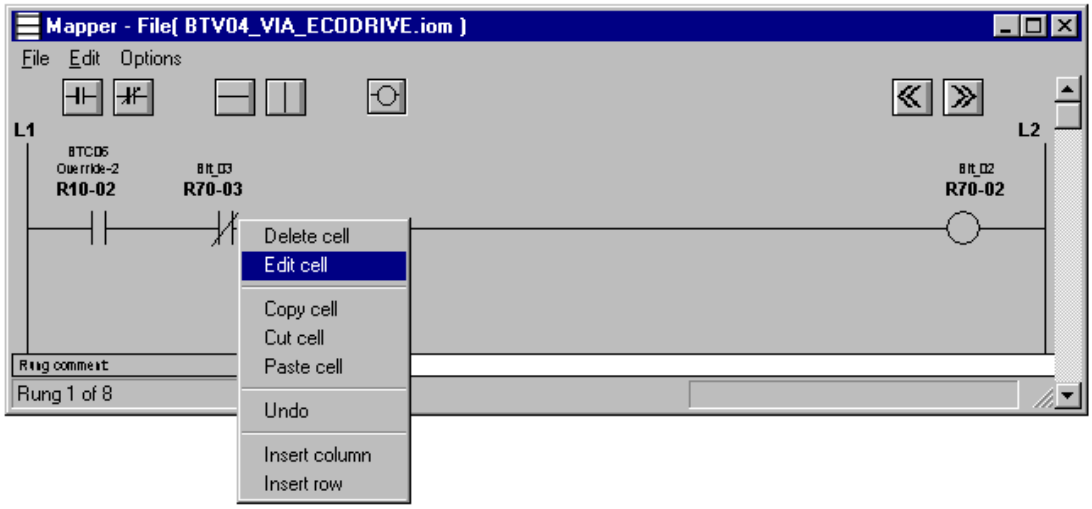


Fig. 14-11: Functions of the right-hand mouse button

Delete cell Deletes the contents of the rectangle the cursor points to.

Edit cell Opens a dialog for editing the register, bit or contact type.

Copy cell Copies the contact or coil the cursor points to into the memory.

Cut cell Cuts the contact or coil the cursor points to and copies it into the memory.

Paste cell Inserts the contact or cell from the memory into the current rung.

Undo Cancels the last editing process.

Insert column Moves the contents of the column the cursor sits in to the right. Existing lines are extended. An error message is issued if the extreme right column is in use.

Insert row Moves the contents of the line the cursor sits in to the next line beneath. Existing lines are extended. An error message is issued if the bottom line is not blank.

15 I/O Register and Keyboard

15.1 Invariably Assigned I/O Registers

In the I/O Mapper, the registers 1-20 are invariably connected with the hardware components (such as keys, LEDs, or inputs and outputs) of the miniature control panel. The allocation of the individual data words is explained in the tables below.

BTV/BTC Register Words

Function	**Register**		
	Word	Write	Read
reserved	0		
reserved	1	X	X
system status register	2		X
24-V inputs (BTV04/05/06 only)	3		X
24-V outputs (BTV04/BTV05/06 only) LED 3 (BTC06.2 only)	4	X	X
LED 1	5	X	X
LED 2 (BTV04 and BTC06.2 only)	6	X	X
override value (BTC06 only)	7		X
handwheel value (BTC06 only)	8		X
current keyboard scan code	9		X
machine key function bits	10		X
key bits 1	11		X
key bits 2	12		X
key bits 3	13		X
PLC/ELC reserved ON	14		X
PLC/ELC reserved OFF	15		X
PLC/ELC input block	16, 17	X	X
PLC/ELC output block	18, 19		X

Fig. 15-1: Overview of the register words

15.2 I/O Mapper System Status Register

The system assigns fixed information to register 2 (Runtime version 03V01 and later):

System Output Register 2	Function
Bit 0	static 0
Bit 1	static 1
Bit 2	OFF-ON edge – active for one I/O Mapper cycle after power-on
Bit 3	10 Hz (50ms ON – 50ms OFF)
Bit 4	5 Hz (100ms ON – 100ms OFF)
Bit 5	2.5 Hz (200ms ON – 200ms OFF)
Bit 6	1.25 Hz (400ms ON – 400ms OFF)
Bit 7	0.625 Hz (800ms ON – 800ms OFF)

Fig. 15-2: System output register

15.3 LED Bit Tables

BTC06

LED in key/function	Register bit
A+	5.0
A-	5.1
B+	5.2
B-	5.3
C+	5.4
C-	5.5
X+	5.6
X-	5.7
Y+	5.8
Y-	5.9
Z+	5.10
Z-	5.11
ESC	6.0
Down	6.1
OK	6.2
Shift	6.3
LEFT	6.4
-	6.5
Right	6.6
Page Down	6.7

LED in key/function	Register bit
0	6.8
Up	6.9
.	6.10
Page Up	6.11
1	6.12
2	6.13
3	6.14
Edit	6.15
4	4.0
5	4.1
6	4.2
Task	4.3
7	4.4
8	4.5
9	4.6
Teach	4.7
R1	4.8
L1	4.9
Help	4.10
Main menu	4.11

Fig. 15-3: LED bits BTC06

BTV04

LED in key/function	Register bit
Jog-	5.0
Clear	5.1
JOG+	5.2
Label 3	5.3
Label 4	5.4
Label 5	5.5
Label 6	5.6
Label 7	5.7
OK	5.8
Main menu	5.9
ESC	5.10
Label 1	5.11
Label 2	5.12
Help	5.14
F6	6.0
F4	6.1
F5	6.2
F1	6.3
F2	6.4
F3	6.5
Buzzer	6.6

Fig. 15-4: LED bits BTV04

BTV05/BTV06

LED in key/function	Register bit
Label 1	5.0
Label 2	5.1
Label 3	5.2
Label 4	5.3
Label 5	5.4
Label 6	5.5
Label 7	5.6
Buzzer	5.7

Fig. 15-5: LED bits BTV05/06

15.4 24-V Terminals, Override, Handwheel

Output Terminals

Terminal X4(BTV05/06) X5(BTV04)	I/O bit
1	4.0
...	...
8	4.7
9	4.8
10	4.9
11	4.10

Fig. 15-6: Output terminals BTV04/05/06

Input Terminals

Terminal X5(BTV05/06) X4(BTV04)	I/O bit
1	3.0
...	...
8	3.7
9	3.8
10	3.9

Fig. 15-7: Input terminals BTV04/05/06

BTC06 Handwheel

Each handwheel increment increases register word 8 by 4; each decrement reduces it by 4. This value is **not** influenced when the handwheel button is pressed for 1 second to reset the digital LED display. Register 8 is initialized to zero when the unit is switched on. It can only be modified by turning the handwheel. Once the maximum value of 65,532 has been reached, the register value continues with 0. The same applies in the opposite sense when the value falls below 0.

Note: Neither the I/O Mapper nor the script language may write to the handwheel register.

BTC06 Override

4-bit override	I/O bit
Bit 0	7.0
Bit 1	7.1
Bit 2	7.2
Bit 3	7.3

Fig. 15-8: BTC06 override

Gray code table

Scale value	Bit 0	Bit 1	Bit 2	Bit 3
0 %				
1 %	X			
2 %	X	X		
4 %		X		
6 %		X	X	
8 %	X	X	X	
10 %	X		X	
20 %			X	
30 %			X	X
40 %	X		X	X
50 %	X	X	X	X
60 %		X	X	X
70 %		X		X
80 %	X	X		X
100 %	X			X
120 %				X

Fig. 15-9: Gray code table

The output signal of the override switch is a 4-bit Gray code switch.

15.5 Keyboard Register Bits and Scan Codes

Keys for HMI Functions

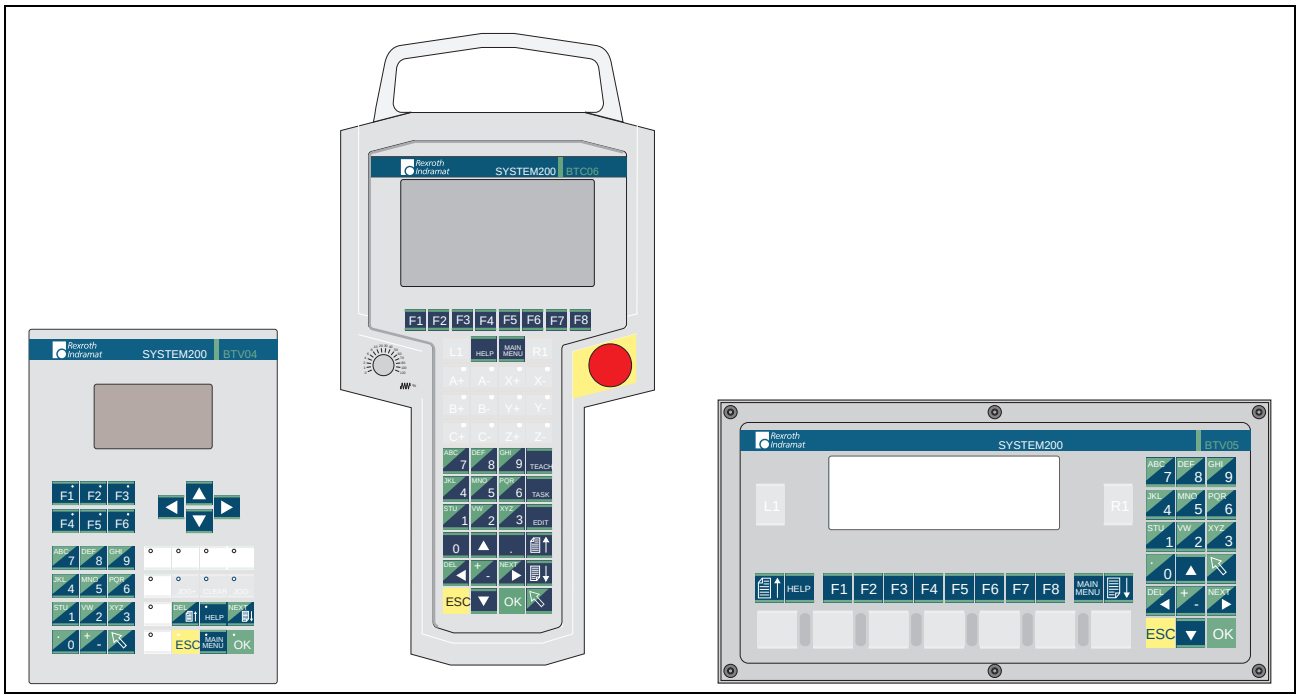


Fig. 15-10: Keys for HMI functions

Key	Scan code (register 9)	Register bit
no key pressed	0	
1	1	11.0
2	2	11.1
3	3	11.2
4	4	11.3
5	5	11.4
6	6	11.5
7	7	11.6
8	8	11.7
9	9	11.8
0	10	11.9
-	11	11.10
+	12	11.11
OK	13	11.12
Decimal point	14	11.13
Top	15	11.14
Bottom	16	11.15
Left	17	12.0
Right	18	12.1

Key	Scan code (register 9)	Register bit
Page up	19	12.2
Page down	20	12.3
DEL	21	12.4
Next	22	12.5
Main menu	23	12.6
Help	24	12.7
Shift OK	25	12.8
Shift ESC	26	12.9
ESC	27	12.10
Teach (BTC06 only)	28	12.11
Task (BTC06 only)	29	12.12
Edit (BTC06 only)	30	12.13
F1	31	13.0
F2	32	13.1
F3	33	13.2
F4	34	13.3
F5	35	13.4
F6	36	13.5
F7 (BTV05, BTC06 only)	37	13.6
F8 (BTV05, BTC06 only)	38	13.7
Shift	39	1215
STU	65	
VW	66	
XYZ	67	
JKL	68	
MNO	69	
PQR	70	
ABC	71	
DEF	72	
GHI	73	
Shift F1	74	13.8
Shift F2	75	13.9
Shift F3	76	13.10
Shift F4	77	13.11
Shift F5	78	13.12
Shift F6	79	13.13
Shift F7	80	13.14
Shift F8	81	13.15
Shift Help	82	12.14

Fig. 15-11: Keys for HMI functions

Keys for Machine Functions

Fig. 15-12: Keys for machine functions

Key			Scan code	Register bit
BTV04	BTV05	BTC06	(register 10)	
Label 1	Label 1	A+	40	10.0
Label 2	Label 2	A-	41	10.1
Label 3	Label 3	B+	42	10.2
Label 4	Label 4	B-	43	10.3
Label 5	Label 5	C+	44	10.4
Label 6	Label 6	C-	45	10.5
Label 7	Label 7	X+	46	10.6
Clear		X-	47	10.7
		Y+	48	10.8
		Y-	49	10.9
		Z+	50	10.10
		Z-	51	10.11
JOG+	L1	L1	52	10.14
Jog-	R1	R1	53	10.15

Fig. 15-13: Keys for machine functions

15.6 Predefined Constants

The compiler automatically links the following "#defines". Thus, they are available in each SCM program.

Typical utilization in a ScreenManager program – writing a heading :

```
...
   TextAttr(1,0,0,0,1);
   Text(0,0,MAX_X,0,"Main Menu");
   TextAttr(0,0,0,0,0);
   Text(MAX_X-(6*8),MAX_Y-8,0,0,"F6-Setup");
   SetIO(REGISTER_LED_1, BIT_LED_LABEL_2);
...Key(KEY_F6, KEY_PRESSED, ParameterScreen);
...
```

Predefined Key Codes

```
#define KEY_PRESSED    (1)
#define KEY_RELEASED   (0)

#define KEY_ASCII_1    (1)
#define KEY_ASCII_2    (2)
#define KEY_ASCII_3    (3)
#define KEY_ASCII_4    (4)
#define KEY_ASCII_5    (5)
#define KEY_ASCII_6    (6)
#define KEY_ASCII_7    (7)
#define KEY_ASCII_8    (8)
#define KEY_ASCII_9    (9)
#define KEY_ASCII_0    (10)
#define KEY_NEG_SIGN   (11)
#define KEY_SIGN       (12)
#define KEY_OK         (13)
#define KEY_DOT        (14)
#define KEY_CUR_UP     (15)
#define KEY_CUR_DOWN   (16)
#define KEY_CUR_LEFT   (17)
#define KEY_CUR_RIGHT  (18)
#define KEY_PAGE_UP    (19)
#define KEY_PAGE_DOWN  (20)

#define KEY_DEL        (21)
#define KEY_NEXT       (22)
#define KEY_MAIN       (23)
#define KEY_HELP       (24)
#define KEY_SHIFT_OK   (25)
#define KEY_SHIFT_ESC  (26)
#define KEY_ESC        (27)
#define KEY_TEACH      (28)
#define KEY_TASK       (29)
#define KEY_EDIT       (30)
#define KEY_F1         (31)
#define KEY_F2         (32)
#define KEY_F3         (33)
#define KEY_F4         (34)
#define KEY_F5         (35)
#define KEY_F6         (36)
#define KEY_F7         (37)
#define KEY_F8         (38)
#define KEY_SHIFT      (39)
```

```
#define KEY_LABEL_1     (40)
#define KEY_LABEL_2     (41)
#define KEY_LABEL_3     (42)
#define KEY_LABEL_4     (43)
#define KEY_LABEL_5     (44)
#define KEY_LABEL_6     (45)
#define KEY_LABEL_7     (46)
#define KEY_LABEL_8     (47)
#define KEY_LABEL_9     (48)
#define KEY_LABEL_10    (49)
#define KEY_LABEL_11    (50)
#define KEY_LABEL_12    (51)
#define KEY_L1          (52)
#define KEY_R1          (53)

#define KEY_STU         (65)
#define KEY_VW          (66)
#define KEY_XYZ         (67)
#define KEY_JKL         (68)
#define KEY_MNO         (69)
#define KEY_PQR         (70)
#define KEY_ABC         (71)
#define KEY_DEF         (72)
#define KEY_GHI         (73)
#define KEY_SHIFT_F1    (74)
#define KEY_SHIFT_F2    (75)
#define KEY_SHIFT_F3    (76)
#define KEY_SHIFT_F4    (77)
#define KEY_SHIFT_F5    (78)
#define KEY_SHIFT_F6    (79)
#define KEY_SHIFT_F7    (80)
#define KEY_SHIFT_F8    (81)
#define KEY_SHIFT_HELP  (82)
#define KEY_SHIFT_MAIN  (83)

#define KEY_A_PLUS      (KEY_LABEL_1)
#define KEY_B_PLUS      (KEY_LABEL_3)
#define KEY_C_PLUS      (KEY_LABEL_5)
#define KEY_X_PLUS      (KEY_LABEL_7)
#define KEY_Y_PLUS      (KEY_LABEL_9)
#define KEY_Z_PLUS      (KEY_LABEL_11)
#define KEY_A_MINUS     (KEY_LABEL_2)
#define KEY_B_MINUS     (KEY_LABEL_4)
#define KEY_C_MINUS     (KEY_LABEL_6)
#define KEY_X_MINUS     (KEY_LABEL_8)
#define KEY_Y_MINUS     (KEY_LABEL_10)
#define KEY_Z_MINUS     (KEY_LABEL_12)

#define KEY_CLEAR       (KEY_LABEL_8)
#define KEY_ENTER       (KEY_OK)
#define KEY_AXIS        (KEY_NEXT)
#define KEY_HOME        (KEY_MAIN)
#define KEY_JOG_F       (KEY_L1)
#define KEY_BLANK       (KEY_LABEL_1)
#define KEY_JOG_R       (KEY_R1)
#define KEY_START       (KEY_LABEL_5)
#define KEY_AUTO        (KEY_LABEL_6)
#define KEY_STOP        (KEY_LABEL_7)
```

Predefined I/O Registers

```
#define REGISTER_LED_1              (5)
#define REGISTER_SCANCODE           (9)
#define REGISTER_KEYBOARD_0        (10)
#define REGISTER_KEYBOARD_1        (11)
#define REGISTER_KEYBOARD_2        (12)
#define REGISTER_KEYBOARD_3        (13)
#define REGISTER_PLC_RESERVED_IN   (14)
#define REGISTER_PLC_RESERVED_OUT  (15)
#define REGISTER_PLC_IN_1          (16)
#define REGISTER_PLC_IN_2          (17)
#define REGISTER_PLC_OUT_1         (18)
#define REGISTER_PLC_OUT_2         (19)

#define BIT_PLC_L                   (0)
#define BIT_PLC_R                   (1)
#define BIT_PLC_LIFEMAN_1           (2)
 #define BIT_PLC_LIFEMAN_2           (3)

#ifdef BTV04
#define REGISTER_INPUT              (3)
#define REGISTER_OUTPUT             (4)
#define REGISTER_LED_2              (6)
```

```
#define BIT_INPUT_1                 (0)
#define BIT_INPUT_2                 (1)
#define BIT_INPUT_3                 (2)
#define BIT_INPUT_4                 (3)
#define BIT_INPUT_5                 (4)
#define BIT_INPUT_6                 (5)
#define BIT_INPUT_7                 (6)
#define BIT_INPUT_8                 (7)
#define BIT_INPUT_9                 (8)
#define BIT_INPUT_10                (9)

#define BIT_OUTPUT_1                (0)
#define BIT_OUTPUT_2                (1)
#define BIT_OUTPUT_3                (2)
#define BIT_OUTPUT_4                (3)
#define BIT_OUTPUT_5                (4)
#define BIT_OUTPUT_6                (5)
#define BIT_OUTPUT_7                (6)
#define BIT_OUTPUT_8                (7)
#define BIT_OUTPUT_9                (8)
#define BIT_OUTPUT_10               (9)

#define BIT_LED_F6                  (0)
#define BIT_LED_F4                  (1)
#define BIT_LED_F5                  (2)
#define BIT_LED_F1                  (3)
#define BIT_LED_F2                  (4)
#define BIT_LED_F3                  (5)

#define BIT_LED_R1                  (0)
#define BIT_LED_L1                  (2)
#define BIT_LED_LABEL_3             (3)
#define BIT_LED_LABEL_4             (4)
#define BIT_LED_LABEL_5             (5)
#define BIT_LED_LABEL_6             (6)
#define BIT_LED_LABEL_7             (7)
#define BIT_LED_OK                  (8)
#define BIT_LED_MAIN                (9)
#define BIT_LED_ESC                (10)
#define BIT_LED_LABEL_1            (11)
#define BIT_LED_LABEL_2            (12)
#define BIT_LED_HELP               (14)
#define BIT_LED_CLEAR               (1)

#define REGISTER_BEEP               (6)
#define BIT_BEEP                    (6)
#endif


#ifdef BTV05
#define REGISTER_INPUT              (3)
#define REGISTER_OUTPUT             (4)

#define BIT_INPUT_1                 (0)
#define BIT_INPUT_2                 (1)
#define BIT_INPUT_3                 (2)
#define BIT_INPUT_4                 (3)
#define BIT_INPUT_5                 (4)
#define BIT_INPUT_6                 (5)
#define BIT_INPUT_7                 (6)
#define BIT_INPUT_8                 (7)
#define BIT_INPUT_9                 (8)
#define BIT_INPUT_10                (9)

#define BIT_OUTPUT_1                (0)
#define BIT_OUTPUT_2                (1)
#define BIT_OUTPUT_3                (2)
#define BIT_OUTPUT_4                (3)
#define BIT_OUTPUT_5                (4)
#define BIT_OUTPUT_6                (5)
#define BIT_OUTPUT_7                (6)
#define BIT_OUTPUT_8                (7)
#define BIT_OUTPUT_9                (8)
#define BIT_OUTPUT_10               (9)

#define BIT_LED_LABEL_1             (0)
#define BIT_LED_LABEL_2             (1)
#define BIT_LED_LABEL_3             (2)
#define BIT_LED_LABEL_4             (3)
#define BIT_LED_LABEL_5             (4)
#define BIT_LED_LABEL_6             (5)
#define BIT_LED_LABEL_7             (6)

#define REGISTER_BEEP               (5)
```

```
#define BIT_BEEP                    (7)
#endif


#ifdef BTV06
#define REGISTER_INPUT              (3)
#define REGISTER_OUTPUT             (4)

#define BIT_INPUT_1                 (0)
#define BIT_INPUT_2                 (1)
#define BIT_INPUT_3                 (2)
#define BIT_INPUT_4                 (3)
#define BIT_INPUT_5                 (4)
#define BIT_INPUT_6                 (5)
#define BIT_INPUT_7                 (6)
#define BIT_INPUT_8                 (7)
#define BIT_INPUT_9                 (8)
#define BIT_INPUT_10                (9)

#define BIT_OUTPUT_1                (0)
#define BIT_OUTPUT_2                (1)
#define BIT_OUTPUT_3                (2)
#define BIT_OUTPUT_4                (3)
#define BIT_OUTPUT_5                (4)
#define BIT_OUTPUT_6                (5)
#define BIT_OUTPUT_7                (6)
#define BIT_OUTPUT_8                (7)
#define BIT_OUTPUT_9                (8)
#define BIT_OUTPUT_10               (9)

#define BIT_LED_LABEL_1             (0)
#define BIT_LED_LABEL_2             (1)
#define BIT_LED_LABEL_3             (2)
#define BIT_LED_LABEL_4             (3)
#define BIT_LED_LABEL_5             (4)
#define BIT_LED_LABEL_6             (5)
#define BIT_LED_LABEL_7             (6)

#define REGISTER_BEEP               (5)
#define BIT_BEEP                    (7)
#endif


#ifdef BTC06
#define REGISTER_LED_2              (6)
#define REGISTER_LED_3              (4)
#define REGISTER_OVERRIDE           (7)
#define REGISTER_HANDWHEEL          (8)

#define BIT_LED_A_PLUS              (0)
#define BIT_LED_A_MINUS             (1)
#define BIT_LED_B_PLUS              (2)
#define BIT_LED_B_MINUS             (3)
#define BIT_LED_C_PLUS              (4)
#define BIT_LED_C_MINUS             (5)
#define BIT_LED_X_PLUS              (6)
#define BIT_LED_X_MINUS             (7)
#define BIT_LED_Y_PLUS              (8)
#define BIT_LED_Y_MINUS             (9)
#define BIT_LED_Z_PLUS             (10)
#define BIT_LED_Z_MINUS            (11)
#define BIT_LED_ESC                 (0)
#define BIT_LED_CUR_DOWN            (1)
#define BIT_LED_OK                  (2)
#define BIT_LED_SHIFT               (3)
#define BIT_LED_CUR_LEFT            (4)
#define BIT_LED_NEG_SING            (5)
#define BIT_LED_CUR_RIGHT           (6)
#define BIT_LED_PAGE_DOWN           (7)
#define BIT_LED_ASCII_0             (8)
#define BIT_LED_CUR_UP              (9)
#define BIT_LED_DOT                (10)
#define BIT_LED_PAGE_UP            (11)
#define BIT_LED_ASCII_1            (12)
#define BIT_LED_ASCII_2            (13)
#define BIT_LED_ASCII_3            (14)
#define BIT_LED_EDIT_LED           (15)

#define BIT_LED_ASCII_4             (0)
#define BIT_LED_ASCII_5             (1)
#define BIT_LED_ASCII_6             (2)
#define BIT_LED_TASK                (3)
#define BIT_LED_ASCII_7             (4)
#define BIT_LED_ASCII_8             (5)
```

```
#define BIT_LED_ASCII_9             (6)
#define BIT_LED_TEACH               (7)
#define BIT_LED_R1                  (8)
#define BIT_LED_L1                  (9)
#define BIT_LED_HELP               (10)
#define BIT_LED_MAIN               (11)
#endif
```

Other Constants

```
#ifdef BTV04
#define MAX_X (128)
#define MAX_Y (64)
#endif

#ifdef BTV05
#define MAX_X (256)
#define MAX_Y (64)
#endif

#ifdef BTV06
#define MAX_X (240)
#define MAX_Y (128)
#endif

#ifdef BTC06
#define MAX_X (240)
#define MAX_Y (128)
#endif

#define INVALID_HANDLE (0xffff)
```

16 Operating System

16.1 Base Functions of Runtime without Application

A BTV unit with installed runtime, but without configuration loaded, first shows the parameter screen. This is caused by an invariably implemented default application. The BTV unit can perform the following actions in this state.

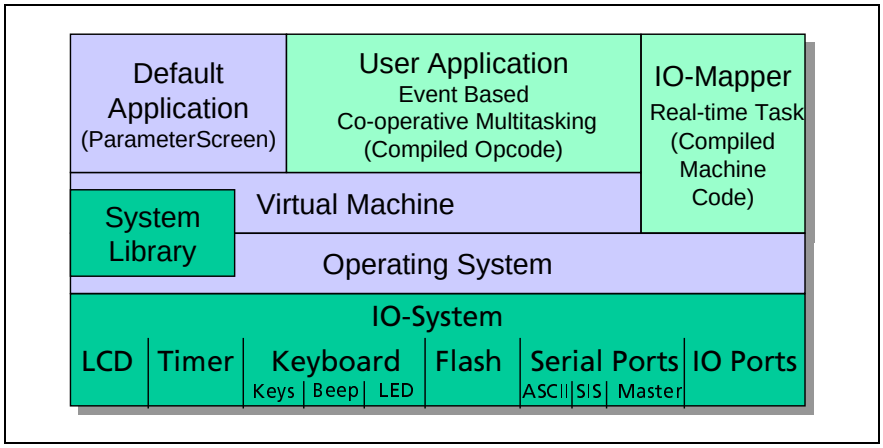


Fig. 16-1: ScreenManager software system - overview.

BTV Parameters

The parameters for the serial interfaces, passwords, screen savers, etc. are selected here. Please refer to the documentation of the individual runtime systems for detailed descriptions.

Manual Relay Mode

A **Manual Relay Mode** menu can be selected to internally interconnect the two serial interfaces of the BTV unit. Thus, there is a variety of applications of the BTV unit as a level or baud rate converter in order to be able to exchange data between different controllers and, for example, a PC. This makes is possible, for example, to reach an Ecodrive on an RS485 bus by connecting the PC to the BTV unit via RS232. With this Relay Mode, the data is transferred without any modification; data transfer is independent of the protocols. This mode can also be invoked and used in some applications. The protocol selected for the interfaces is irrelevant to manual Relay Mode.

Another application could be 'cascading' BTV05 or BTV06 units – a control panel connected via the second interface could optionally be connected to a controller instead of the first one if Relay Mode is activated there. Provided the application has been programmed accordingly, this can be done by the operator. This excludes dual operation of the machine.

Note: On line baud rate selection via the interface (as supported by Dolfi, for example) is not supported.

Flash File system

There is a file system in the control panels that permits data to be stored either from a PC desktop or by the application itself. The associated "Filemanager" is contained in the parameter menu. It permits files to be viewed, individual files or all files (formatting) to be deleted, or the compression of the file system to be triggered. This is a process that definitively removes the gaps that were produced by deleting files. Afterwards, the entire file system is set up in a contiguous block. In this state, Dolfi can be used for reading it from the unit as a separate module. This permits complete file systems to be exchanged between the units (cloning configurations).

I/O Slave on MTC/ISP

The BTV unit immediately establishes communication with the PLC is a BTV interface is configured as a slave and if this interface is connected with an Indramat PLC (MTC-ISP). The real-time data in the SIS protocol is exchanged cyclically. Thus, the selected key LEDs, hardware inputs and outputs, and specific extensions such as override potentiometer or handwheel are available in the PLC without the need of loading an application.

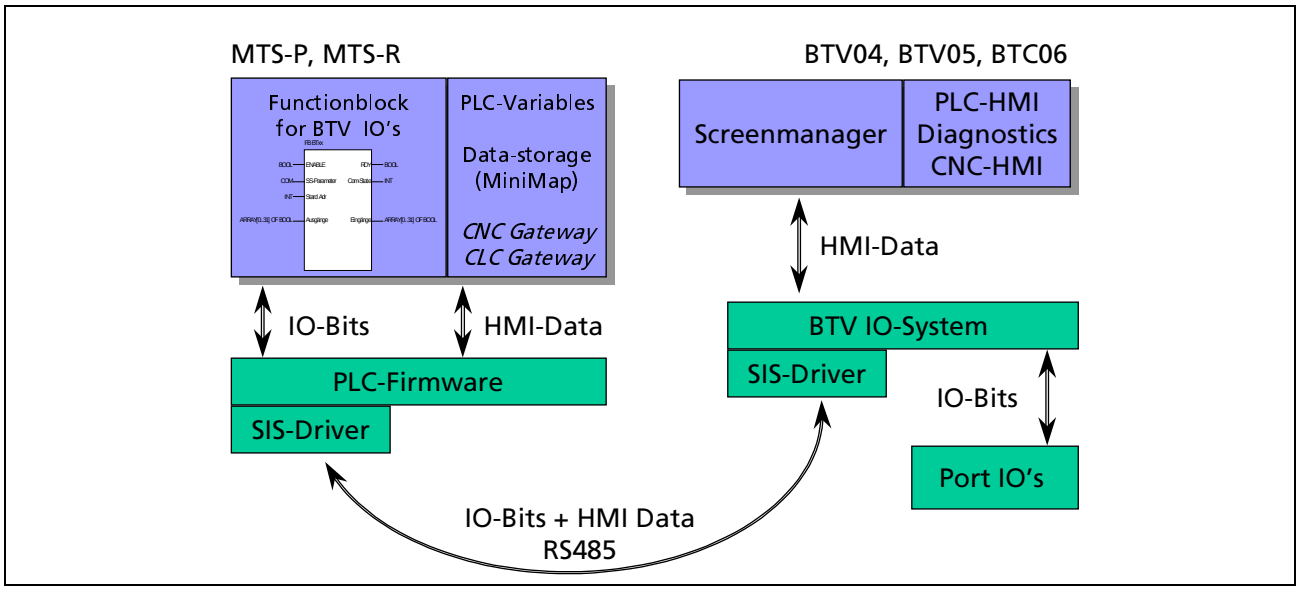


Fig. 16-2: Real-time connection – transfer of the real-time data in the protocol frame of the visualization information

16.2 Event Handling – When Does Which Code Execute ?

The Runtime operating system always executes in an event-driven way. There is no sequentially running code or script. A script that is executed in one piece is assigned to each key actuation or menu call.

Any input or display fields that subsequently exist on the display were created by ScreenManager commands that ensure that they remain permanently on the screen and are updated automatically be the operating system.

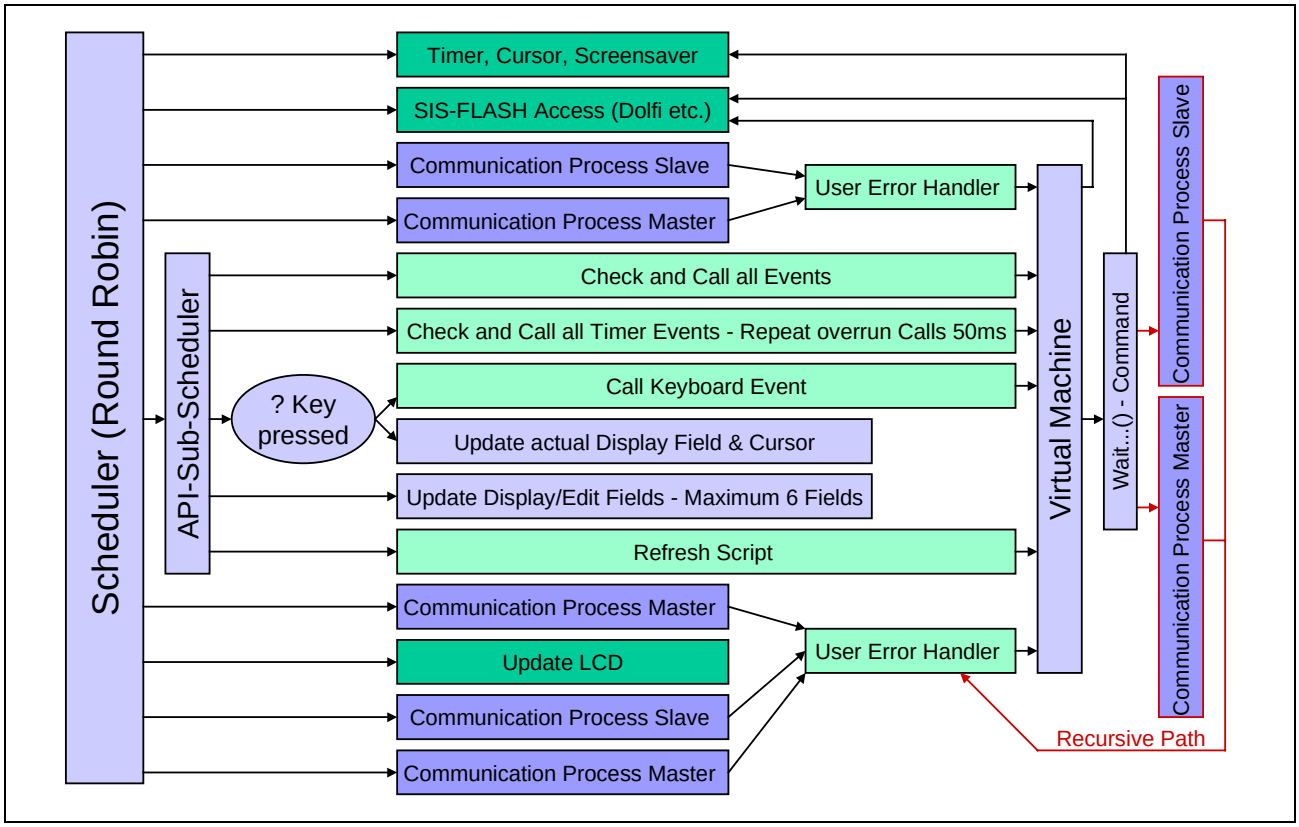


Fig. 16-3: Sequence and interdependencies of the calls in the ScreenManager

The recursive invocation path shown in the figure above can only be used in conjunction with a user error script and the Wait...() commands. This is why the Wait...() calls should never be used in an error handling routine.

Ideally, the response time required by the operating system for a full run is in the range of 100...200 ms. Caution: Long user scripts, blocking or even waiting scripts can extent this time to an undefined value since there is no pre-emptive interruption of a function call of the virtual machine. The benefit of this procedure is that each script call is completely processed on its own (atomic) and the programmer need not think about synchronization mechanisms (semaphores, critical areas, etc.) that are usually required in multi-tasking systems.

Some marginal conditions must be satisfied during implementation if events with predictable response times are required. This makes the system deterministic.

- If possible, do not use any Wait...() commands. Use clever event programming instead: For example: Set a variable that must safely be read to a value that does not occur in the controller, and program an event for modification – the event will then be invoked when a new current value comes from the controller.
- No write access to the file system. It could trigger a compression process that takes a long time.
- Only use short scripts with a predictable execution time (if possible do not use loops - they may take an undefined time).
- Do not perform any access to a connected controller which might cause an error that requires acknowledgement (unless this error must stop execution anyway).
- Do not program any code that could cause runtime errors. For example: A BindPLC for which it has not been ensured that the variable exists in the controller or, a widespread error: Division by zero
- Do not program any timer events that can be invoked faster than this is permitted by the system cycle, and are compute-bound.
- Do not allow any password-free access to the system parameters. The user may trigger file system compression manually.

If you need swift and deterministic reactions, use I/O Mapper or the real-time channels to ELC/ISP whose cycle time is constant and can be guaranteed at any time.

Initialization with "Void Main()"

This function is the entry point of any application. The commands listed in there are executed immediately after power-on. Any other initializations required for running the application are performed here.

Event Sources: Keys, Variable Monitoring, Timer, Refresh Script

Once the Main function is terminated, only the operating system is running. It monitors all event sources like keys, timers, or explicitly programmed variable events (the system is provided with variables and transition conditions; it monitors on its own whether or not these conditions are satisfied, and invokes the associated script function). An activated refresh script may also run during the idle time. Each screen can have its own refresh script assigned.

16.3 Serial Communication – What Happens in the Background ?

Besides the user scripts, the following functions execute in the background.

Real-time Tasks

I/O Mapper and transfer of I/O bits between ELC/MTC/ISP and control panel run as pre-emptive real-time task at a maximum response time in the ms range.

Updating Variables

The controller reads all variables that are associated with controllers asynchronously in the background. Communication with the related controller takes place at the maximum speed possible without idle time. The variables in the ScreenManager program are updated between the execution of scripts. Thus it is ensured that no variable is modified during the execution time of a script.

Automatic Relay Mode CLC/CLM/ELC

If a controller is connected that supports the ASCII protocol, the second interface of the control panel can be used for establishing an additional link between PC and controller without interrupting the communication. The inquiries of the PC are cyclically embedded into the other requests.

Responding to the SIS-FLASH and RAM Services

This includes: Download requests from Dolfi which are processed in the background. RECYCLE.EXE, a free screen hard copy tool functions via the same channel.

16.4 System Functions

All system calls used in the ScreenManager application exist in the Runtime as hard-encoded functions. Whenever a system function is called from the script language, the hard-programmed routine is called directly in the operating system via a branch table.

System Include File

This file (with ScreenManager desktop version 1 of the name kscm_sys.inc) represents the link between script language and branch table in the Runtime. This is why a new system include file is enclosed to new versions with enhanced functions. The call table is always downward compatible. This means that a functioning application can run on all newer runtime versions of the ScreenManager.

If a newer runtime version is available, the additional functions can be used after this Include file has been replaced.

Caution: The System Include file must never be modified or edited manually.

Caution: Version 1 System Include file and the subsequent ones are not compatible ! Never exchange manually or confuse. For version 3, the syntax of these files was modified and enhanced.

```
// kscm system calls version 03A28 00-07-21

INT8 shl8(INT8 a, UINT8 b):0x1600003f;
INT16 shl16(INT16 a, UINT8 b):0x16000040;
INT32 shl32(INT32 a, UINT8 b):0x16000041;

...

FLOAT64 cs16_f64(INT16 a):0x09000029;
FLOAT64 cs32_f64(INT32 a):0x0900002a;
FLOAT64 cf_f64(FLOAT a):0x0900002b;

...

void   Bitmap(INT16 X, INT16 Y, INT16 W, INT16 H, INT16 XOfs, INT16 Yofs, BMP b, INT8 Mode):0x0900002d;
INT16  BitmapWidth(BMP b):0x0900002e;
INT16  BitmapHeight(BMP b):0x0900002f;

UINT8  GetLanguage():0x09000030;
void   SetLanguage(UINT8 NewLanguage):0x09000031;
void   SaveParameter():0x09000032;

void   run_error():0x16000000;
void   ScreenLock     ( ):0x000000ee;

void   WaitCursor():0x1600007d;
void   FileSelect (STRING Name, STRING Ext):0x1600007e;
void   ScreenUpdate():0x1600007f;
INT16  rand():0x16000080;
void   SwitchRelayMode (UINT8 On):0x16000081;
void   RelayScreen():0x16000082;
void   PutPixel (INT16 X, INT16 Y, UINT8 Fill):0x16000083;
void   Circle (INT16 X, INT16 Y, INT16 r, UINT8 Fill):0x16000084;
void   Line (INT16 X, INT16 Y, INT16 X2, INT16 Y2, UINT8 Fill):0x16000085;
UINT8  WaitKey():0x16000086;
void   CloseWindow():0x16000087;
void   ExceptionWindow():0x16000088;
void   GetErrorTelegrams ( STRING t1, STRING t2 ):0x16000089;
void   OnErrorCall ( PTR ErrorScript ):0x1600008a;
void   ClearError ( BOOL bRetry ):0x1600008b;
UINT16 GetError ( STRING t1):0x1600008c;
void   WaitPLC ( ):0x1600008d;
UINT16 GetIORegister  (INT16 Number ):0x1600008e;
void   SetIORegister  (INT16 Number, UINT16 value ):0x1600008f;
void   BindIORegister (INT16 Number)$0x16000090;

void   ConnectCLMR    ( ):0x16000091;

void   JogCLM ( UINT8 Address, UINT16 CLMInput , UINT8 CLMBit , UINT16 SCMRegister , UINT8 SCMBit ):0x16000092;
void   JogCLC ( UINT8 Address, UINT16 CLCRegister , UINT8 CLCBit , UINT16 SCMRegister , UINT8 SCMBit ):0x16000093;

void   ConnectDKC     ( ):0x16000094;

void   WriteDKC (UINT8 Address, UINT32 Ident, UINT8 Element, UINT16 ListElement)$0x16000095;
void   Ident2Str ( STRING s, UINT32 Ident):0x16000096;
UINT32 Str2Ident ( STRING s ):0x16000097;
Void   ReadDKC (UINT8 Address, UINT32 Ident, UINT8 Element, UINT16 ListElement)$0x16000098;
Void   BindDKC (UINT8 Address, UINT32 Ident, UINT8 Element, UINT16 ListElement)$0x16000099;

...

#define KEY_F1          (31)
#define KEY_F2          (32)
#define KEY_F3          (33)
#define KEY_F4          (34)
#define KEY_F5          (35)
#define KEY_F6          (36)
#define KEY_F7          (37)
#define KEY_F8          (38)

...

#define KEY_PRESSED   1
#define KEY_RELEASED  0

#ifdef BTV04
#define MAX_X (128)
#define MAX_Y (64)
#endif

#ifdef BTV05
#define MAX_X (256)
#define MAX_Y (64)
#endif

...
```

Fig. 16-4: Excerpts from the System Include file

17 System Messages

17.1 The ScreenManager Does not Start

When the unit is switched on, the loader first verifies the correct checksum of the loaded software. The INDRAMAT logo is displayed during this time. The logo remains on the screen and "Dolfi Loader active..." appears on the bottom line if the checksum verification fails. In this case you must check the software (is the SWA name compatible with the device type?) and transfer it again with Dolfi. Look out for possible error outputs from Dolfi.

17.2 The Application Does not Start

Once the ScreenManager software has been started, it verifies the correct checksum of a loaded application. The parameter menu starts if the checksum determination fails. In this case, you must check the application (are application and hardware compatible with each other? Has the correct device type been selected on the ScreenManager programming desktop?) and transfer it again. Look out for possible error outputs from Dolfi.

17.3 Start Messages

The following lines are output when the unit is switched on:

BTx0x-SCM-03Vxx May 30 2000 13:12:55 COM1: RS422,38400e Adr1 COM2: RS232, 9600e Adr3 <F2> Parameter Setup file system 255 KB ...blank check ok starting application...	Software version Compile time Port settings Note: F2-Parameter mode Messages from the Flash File system Progress of checking and starting application and I/O mapper program

Fig. 17-1: Start Messages

The following messages are possible if errors occur during the system start:

Flash Fail

Initializing the flash memory failed. The unit cannot be used without Service intervention.

BTx0x-SCM-03Vxx May 30 2000 13:12:55 FLASH fail... – or – Loader must run in flash to start an application	Software version Compile time Flash error

Fig. 17-2: Start messages

The second part of the message tells you that a loader EPROM may still be in use. To be able to execute software with the unit, the loader **must** be copied to the flash. The jumpers on the BTV04/05/06 boards must be in ‚Operate' position. Usually, this is the delivery state of the units. Basically, this part of the information is only for Service.

Load Default, Parameter Rewritten

No valid parameter segment was found in the control panel. The parameter segment was newly written with the factory settings. The unit restarts without interruption. This message need not be acknowledged.

Section x Bad Use y, Parameter Rewritten

The two backup copies of the parameter segment do not agree or one copy is damaged. The first segment or the intact segment is used. The faulty one is rewritten. The unit restarts without interruption. This message need not be acknowledged.

Wrong Parameter Version

"Wrong parameter version, Old parameter erased, new defaults written". The parameters come from a version of the ScreenManager that is no longer compatible. This is only relevant to future extensions when a change is made from a newer version back to an older one. Up to now, there is no version with incompatible parameters. This message need not be acknowledged.

Illegal COMx Parameters

The settings stored in the parameter segment are not possible with the existing hardware. This is only the case if a parameter segment is, for example, transferred from a BTV05/06 to a BTV04 and contains interface settings that are not supported by the BTV04 unit. In principle, however, exchanging parameter segments between the BTV units is permitted (but not to a BTC06 unit). This message need not be acknowledged.

Flash Erase Error

The attempt of initializing the Flash File system failed. Reformatting the flash failed too. This message may be acknowledged, but the file system will not be available. As a remedial action, press F6 to try again an overall clearing of the file system under Flash status in the parameter menu. If this fails too, contact the Service Hotline.

17.4 Error Messages During Operation

More or less fatal errors can occur during operation. The messages described in this chapter are generated within the operating system and can - in contrast to the Script Execution Errors (next chapter) - not be unambiguously assigned to a program location. They occur either during the start or during background processing.

"Press any key..." indicates that program execution can be continued in a more or less unrestricted way after a key has been pressed. "Press any key to reboot..." means that the program cannot be continued and that there will be a restart after the key has been pressed.

CNC Telegram Not Supported

The BindCNC or ReadCNC command was used in conjunction with an invalid first parameter. The selected service is not available. If the loaded application works on other control panels, check whether the latest ScreenManager Runtime is loaded. If this is the case, search the application for a CNC command with invalid parameters. Program execution is continued. The MTC inquiry concerned is discarded.

EEPROM Erase Time-out

This error only occurs when parameters are written in the BTC06 unit. The EEPROM could not be erased. There may be a hardware fault if this error occurs repeatedly.

EEPROM Write Time-out

This error only occurs when parameters are written in the BTC06 unit. The EEPROM could not be written to without errors. There may be a hardware fault if this error occurs repeatedly.

Fatal ReadMiniMapString

This error may not occur under normal circumstances. When this fault occurs, please send the ScreenManager application together with the packed MTC project archive to Indramat.

Fatal LoadProViData

This error may not occur under normal circumstances. When this fault occurs, please send the ScreenManager application together with the packed MTC project archive to Indramat.

Flash Write Error

"Warning: FlashStoreList(), Flash write error". An error occurred when a list was stored in the Flash File system. The list concerned could not be stored permanently. In this case, check the file system in the corresponding parameter menu item, and reformat if necessary.

Handle Was Not Valid

Initiated by CloseList or EraseList in conjunction with an invalid (already released?) handle. Program execution can be continued without any restrictions.

Insert Line Not Supported

The InsertLine function is not yet supported by the present ScreenManager version. Program execution can be continued after a key has been hit.

Map File Create Error

Opening a Flash File for the MTC MiniMap File failed. Check the file system in the parameter menu. Note the error messages that occur during the overall deletion of the flash. Other possible cause – the loaded application is so large that there is no more space for the Flash File system. Observe the specifications of free memory space during the start of the unit. Using the MTC at the control panel requires a minimum of 64 KByte to be reported for the Flash File system.

Map File Write Error

Writing to the Flash File for the MTC MiniMap File failed. Check the file system in the parameter menu. Note the error messages that occur during the overall deletion of the flash.

MTC-Longident Wrong Size

The length of the long ident that is returned from the MTC is invalid. Please check whether the employed versions of ScreenManager and MTC are compatible with each other. The MTC version V20 (WinPCL) is not yet supported by the ScreenManager version 03Vxx and may cause this error message to be issued.

No ASCII Port for CLC

"No ASCII port for CLC, off-line mode only, press any key..." The application requests a ConnectCLC but the "ASCII" protocol has not been selected at any interface of the control panel. Communication with the CLC cannot be established. Program execution can be continued in off line mode after a key has been hit. Use the parameter menu for configuring a port for the ASCII protocol or - the easiest solution - activate the CLC/CLM base settings in the "Intelligent Defaults".

Note: This behavior can deliberately be used for executing a ScreenManager program without a controller and still without error messages (for demonstrations, testing the menu structure, simulations, etc.). At both ports, merely select a protocol that is different to ASCII.

No ASCII Port for CLM

"No ASCII port for CLM, off-line mode only, press any key..." The application requests a ConnectCLM but the "ASCII" protocol has not been selected at any interface of the control panel. Communication to CLM/DLC/ELC-ASCII cannot be established. Program execution can be continued in off line mode after a key has been hit. Use the parameter menu for configuring a port for the ASCII protocol or - the easiest solution - activate the CLC/CLM base settings in the "Intelligent Defaults".

Note: This behavior can deliberately be used for executing a ScreenManager program without a controller and still without error messages (for demonstrations, testing the menu structure, simulations, etc.). At both ports, merely select a protocol that is different to ASCII.

No More List Handles

There are no more handles available for a new list (requested by the application using GetFreeList). The total number of simultaneously open handles is limited to 50 RAM and 50 FLASH handles. The most frequent cause is surely that releasing handles was forgotten in the application or that a new list was allocated cyclically (e.g. in a Refresh or Event Script - programming error).

No More List Memory

The maximum available memory space was exceeded when data was inserted in a list or when a new list was created, Possible causes The amount of data stored in the list was really too large – monitor the value of "FreeMemoryList" in the application in this case. There are 80 Kbyte available for user lists. Another reason can be that releasing unused lists was forgotten in the application so that a larger data quantity has accumulated that is no longer required. Fatal programming error in the application. Program execution is aborted.

No SIS Master Port for ELC

"No SIS master port for ELC, off-line mode only, press any key..." The application requests a ConnectELC but a "SIS master" or "ASCII" protocol has not been selected at any interface of the control panel. (If an SIS_Master Port is not available, the software tries to establish a contact to an ELC via the ASCII protocol). Establishing communication to the ELC is not possible at all. Program execution can be continued in off line mode after a key has been hit. Use the parameter menu for configuring a port for the SIS master protocol or - the easiest solution - activate the ELC base settings in the "Intelligent Defaults".

Note: This behavior can deliberately be used for executing a ScreenManager program without a controller and still without error messages (for demonstrations, testing the menu structure, simulations, etc.). At both ports, merely select a protocol that is different to SIS Master and ASCII.

No SIS Master Port for DKC

"No SIS master port for DKC, off-line mode only, press any key..." The application requests a ConnectDKC but a "SIS master" or "ASCII" protocol has not been selected at any interface of the control panel. Communication to DKC cannot be established at all. Program execution can be continued in off line mode after a key has been hit. Use the parameter menu for configuring a port for the SIS master protocol or - the easiest solution - activate the Ecodrive base settings in the "Intelligent Defaults".

Note: This behavior can deliberately be used for executing a ScreenManager program without a controller and still without error messages (for demonstrations, testing the menu structure, simulations, etc.). At both ports, merely select a protocol that is different to SIS Master.

No SIS Master Port for ELC, No Real-time Connection

"No SIS master port for ELC, no real-time connection, press any key..." The application requests a ConnectELC for a specific station number but a "SIS master" protocol has not been selected at any interface of the control panel. Although communication with the ELC can be established via the ASCII protocol, establishing a real-time connection is not possible. Program execution can be continued without real-time connection after a key has been hit. Use the parameter menu for configuring a port for the SIS master protocol or - the easiest solution - activate the ELC base settings in the "Intelligent Defaults".

Not Enough RAM for List Buffers

This message appears immediately after the system is switched on. There is not enough RAM available for initializing the list function. This message should only appear in test versions. In this case, load the last (previous) version that could be started without any problems.

Only Parameter Not Implemented

The "WriteNCELC" command was used with a single command parameter, but there was no new command specified for the line concerned. Thus, ScreenManager does not know which formatting rules are to be applied for the single parameter. "WriteNCELC" can only be used for writing complete program blocks. Program execution is continued in the background; the error window can be acknowledged with any key. There is no write access to the ELC.

Parameter Write Fail

Saving the parameters in the parameter menu failed. Repeated occurrence of this error indicates a faulty hardware. Working with the unit is still possible. Editing parameters is no longer possible.

Serial Recursive Call

"Serial recursive call, CLM driver error, press any key to reboot." This is an internal procedure error of the CLM/ELC drive that occurs when program blocks are written. This error should never occur during normal operation. Press any key to restart the control panel. In this case, please send the application concerned together with a brief description to INDRAMAT.

Wrong List Handle r

A function that is used for reading from lists was invoked with an invalid list handler. Fatal programming error in the application. Program execution is aborted.

Wrong List Handle w

A function that is used for writing to lists was invoked with an invalid list handler. Fatal programming error in the application. Program execution is aborted.

Wrong Read Offset from CNC

The controller returned an incorrect response to a sequence inquiry to the MTC200 for system fault messages, mechanism messages, or program lines. This error can not usually occur. Please send the application together with an error description and the version descriptions of ScreenManager Runtime **and** MTC firmware to INDRAMAT.

17.5 Script Execution Errors

Array Overflow

A specified array index exceeded the dimensioned array size. This is a programming error in the application program.

Bp Underflow

Fatal address error in the virtual machine. The base pointer leaves the stack area. A possible cause can be that local variables exceeded the maximum stack size (please check the local variable declarations and test by incrementing the stack size of the project options), or a binary code that was generated incorrectly by the compiler. In this case you must try to compile the program again with some modifications.

Divided by 0

There was a division by 0. Error in the application program.

Illegal Segment

Fatal address error in the virtual machine. The specified segment does not exist. Possible cause: A binary code that was incorrectly generated by the compiler. In this case you must try to compile the program again with some modifications. Upgrading to ScreenManager user interface version 3 is strongly recommended.

Jump Out of Segment

Fatal address error in a jump call in the virtual machine. The code segment was exited by a jump address. Possible cause: A binary code that was incorrectly generated by the compiler. In this case you must try to compile the program again with some modifications. Upgrading to ScreenManager user interface version 3 is strongly recommended.

Out of Segment

General address error in the virtual machine. A possible cause can be that local variables exceeded the maximum stack size (please check the local variable declarations and test by incrementing the stack size of the project options), or a binary code that was generated incorrectly by the compiler. The latter chiefly occurs in conjunction with compiler version V01 and an excessive amount of global data. In this case you must try to compile the program again with some modifications. Upgrading to ScreenManager user interface version 3 is strongly recommended.

Stack Overflow

Address error. Invocating a function and/or generating data in the stack. The stack pointer leaves the stack area. A possible cause can be that local variables exceeded the maximum stack size (please check the local variable declarations and test by incrementing the stack size of the project options), or a binary code that was generated incorrectly by the compiler. In this case you must try to compile the program again with some modifications. Upgrading to ScreenManager user interface version 3 is strongly recommended.

Stack Underflow

Address error during the return jump from a function. The stack pointer leaves the stack area. A possible cause can be that local variables exceeded the maximum stack size (please check the local variable declarations and test by incrementing the stack size of the project options), or a binary code that was generated incorrectly by the compiler. In this case you must try to compile the program again with some modifications. Upgrading to ScreenManager user interface version 3 is strongly recommended.

R-stack Overflow

The return stack overflowed. The maximum call depth of the virtual machine is 250 recursive calls. This number was exceeded. Usually, this message is caused by an inadvertent recursion (where a function calls itself). At any case, there is an error in the application program.

Unknown Opcode

The virtual machine encountered an unknown machine command. Possible causes: A binary code that was incorrectly generated by the compiler, or a complete confusion of the machine caused by fatal programming errors. In this case you must try to compile the program again with some modifications. Upgrading to ScreenManager user interface version 3 is strongly recommended.

Unknown System Call

The application called a system function that has not yet been defined in the ScreenManager Runtime version that is installed in the control panel. Please upgrade to the latest version.

17.6 Script Execution Error - Fatal I/O Error

The following errors occur with I/O operations. In most messages, an additional text informs of the exact cause of the error and the ScreenManager command that caused it.

IO elc String Too Long

Fatal error when a program block was built up using the "WriteNCELC" command. This error should not occur when the current software version is used. It points to an incorrect ELC format list. Please send the ScreenManager application together with an error description and all firmware/software versions to INDRAMAT.

Illegal IO Access

An invalid address was specified in an access to the I/O register area of the ScreenManager I/O mapper.

PLC Variable Unknown

The specified ISP variable could not be found in the MiniMap file of the controller. Please check whether all necessary BTV files have been selected in the "BTV status display" menu item and whether a new download to the controller has been performed. MTC/ISP control variables can only be accessed if they have been reported to the ISP programming desktop via this way.

No More Edit Cells

The maximum number (= 50) of simultaneous active links to Display/Edit/Menu item fields was exceeded. In most cases, the cause is a simple programming error: A display, menu item, or edit command is used illegally within a refresh or event script and thus activated repeatedly. This becomes obvious by a "strange" behavior of the cursor (several fields overlap...). This message appears after a few seconds/minutes.

No More Serial Buffers

The maximum number of simultaneous cyclic active links to control variables was exceeded. The maximum values are 50/70 for MTC/ISP and an additional 50 for the remaining controllers. Higher values are not possible (and not expedient either for performance reasons). The data updating time is already in the range of seconds. The most frequent cause is a simple programming error: A bind command is used illegally within a refresh or event script and thus activated repeatedly. This manifests itself by a gradually slowed down communication, and by this message that appears after a few minutes.

Type Not Supported

The Event function was used with a variable different than UINT16.

or

The requested variable type of the controller is not supported or cannot be converted into a ScreenManager type. If possible, declare a different type in the controller.

Variable Must Be Global

An attempt was made to use a local variable together with a Read, Bind Display, Edit or Event command. All these system calls store the address of the transferred variable and process it in the background. This works only with global variables that remain valid after the script function has been executed. In conjunction with the compiler version 1, this message can rarely be issued for global variables. Upgrading to ScreenManager user interface version 3 is strongly recommended in this case. As an immediate aid, compile the program with fewer global variables.

Wrong Map File Entry

An entry that cannot be interpreted was detected in the MiniMap file that was loaded from the MTC/ISP. Download the ISP program again. If this does not help, please send the packed MTC project archive together with the ScreenManager application and a brief error description to INDRAMAT.

Wrong PLC Answer Length

A data package received from the ISP is not of the expected length. Have the version number of ScreenManager and MTC/ISP checked for compatibility. If the error occurs again, please send the packed MTC project archive together with the ScreenManager application and a brief error description to INDRAMAT.

17.7 I/O Error Window

In the event of an incorrect serial access, the ScreenManager activates an error window that usually shows error texts and numbers from the connected controllers. Due to own error handling routines, this error window may be overloaded in the ScreenManager application. Depending on the loaded application, it therefore does not always appear in the described form.

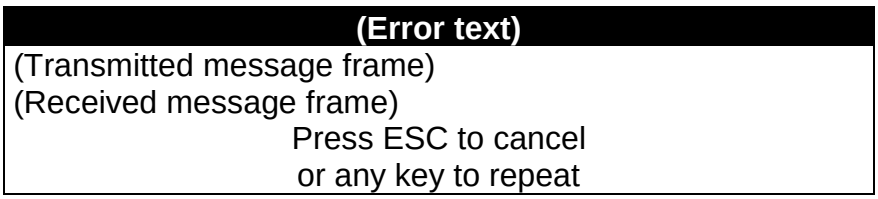


Fig. 17-3: Window of the "default Error Handlers" for serial errors

These errors are always related to **one** serial request. Except for the cases listed below, the error text comes from the connected controller. Where controllers with ASCII protocols are concerned, the transmitted and the received message frames provide additional information that can be used for locating the fault and its cause in the documents. Where controllers with binary protocol are concerned, these two lines contain a hexadecimal dump of the first characters of the message frame that caused the problem.

Press "ESC" to acknowledge. This clears the request from the buffer. Program execution is continued without having read or written this variable.

Pressing any other key for acknowledgment repeats the request. If, for example, the connection to the controller was interrupted, acknowledging this window will cause the communication to be resumed at the point where it was interrupted. No write or read access will be lost.

CLC Is Not Responding

The connected CLC VisualMotion has not responded to a request. Using the time-out selected in the parameters, the request was repeated as many times as is specified in the retry parameter.

CLM Is Not Responding

The connected CLM, DLC or ELC has not responded to a request. Using the time-out selected in the parameters, the request was repeated as many times as is specified in the retry parameter.

CNC Answer Too Short

The MTC has replied to a protocol with too short a message frame. This error should not occur during normal operation when the current software versions are employed.

CNC NAK xx yy

The MTC has replied to a protocol with an NAK. The two numbers xx and yy inform about the error number. Please refer to the MTC publication for further information.

Ecodrive Error 0xNNNN

The Ecodrive3 unit provided an interface error with the hexadecimal code 0xNNNN. The meaning of this code can be found in the drive manuals.

Ecodrive Is Not Responding

The connected Ecodrive3 has not responded to a request. Using the time-out selected in the parameters, the request was repeated as many times as is specified in the retry parameter.

DKC Answer Too Long

The response of the Ecodrive 3 to a message frame was too long. This error should not occur in conjunction with the current software version. Please check the version of the Ecodrive firmware together with the INDRAMAT Application Department.

DKC Answer Too Short

The response of the Ecodrive 3 to a message frame was too short. This error should not occur in conjunction with the current software version. Please check the version of the Ecodrive firmware together with the INDRAMAT Application Department.

DKC Wrong Answer

The response of the Ecodrive 3 to a message frame was incorrect. This error should not occur in conjunction with the current software version. Please check the version of the Ecodrive firmware together with the INDRAMAT Application Department.

18 List of Figures

19 Index

#

,

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

20 Service & Support

20.1 Helpdesk

Unser Kundendienst-Helpdesk im Hauptwerk Lohr am Main steht Ihnen mit Rat und Tat zur Seite. Sie erreichen uns

- telefonisch: **+49 (0) 9352 40 50 60**
  über Service Call Entry Center Mo-Fr 07:00-18:00
- per Fax: **+49 (0) 9352 40 49 41**
- per e-Mail: **service@indramat.de**

Our service helpdesk at our headquarters in Lohr am Main, Germany can assist you in all kinds of inquiries. Contact us

- by phone: **+49 (0) 9352 40 50 60**
  via Service Call Entry Center Mo-Fr 7:00 am - 6:00 pm
- by fax: **+49 (0) 9352 40 49 41**
- by e-mail: **service@indramat.de**

20.2 Service-Hotline

Außerhalb der Helpdesk-Zeiten ist der Service direkt ansprechbar unter

+49 (0) 171 333 88 26
oder **+49 (0) 172 660 04 06**

After helpdesk hours, contact our service department directly at

+49 (0) 171 333 88 26
or **+49 (0) 172 660 04 06**

20.3 Internet

Ergänzende Hinweise zu Service, Reparatur und Training sowie die **aktuellen** Adressen unserer Service- und Vertriebsbüros finden Sie unter **www.indramat.de** – einige Angaben in dieser Dokumentation können inzwischen überholt sein.

Außerhalb Deutschlands nehmen Sie bitte zuerst Kontakt mit Ihrem lokalen Ansprechpartner auf.

- Verkaufsniederlassungen
- Niederlassungen mit Kundendienst

Additional notes about service, repairs and training as well as the **actual** addresses of our sales- and service facilities are available on the Internet at **www.indramat.de** – some information in this documentation may meanwhile be obsolete.

Please contact the sales & service offices in your area first.

- sales agencies
- offices providing service

20.4 Vor der Kontaktaufnahme... - Before contacting us...

Wir können Ihnen schnell und effizient helfen wenn Sie folgende Informationen bereithalten:

1. detaillierte Beschreibung der Störung und der Umstände.
2. Angaben auf dem Typenschild der betreffenden Produkte, insbesondere Typenschlüssel und Seriennummern.
3. Tel.-/Faxnummern und e-Mail-Adresse, unter denen Sie für Rückfragen zu erreichen sind.

For quick and efficient help, please have the following information ready:

1. Detailed description of the failure and circumstances.
2. Information on the type plate of the affected products, especially type codes and serial numbers.
3. Your phone/fax numbers and e-mail address, so we can contact you in case of questions.

20.5 Kundenbetreuungsstellen - Sales & Service Facilities

Deutschland – Germany

vom Ausland: (0) nach Landeskennziffer weglassen!
from abroad: don't dial (0) after country code!

Vertriebsgebiet Mitte Germany Centre	**SERVICE** **CALL ENTRY CENTER** **MO – FR** **von 07:00 - 18:00 Uhr** **from 7 am – 6 pm**	**SERVICE** **HOTLINE** **MO – FR** **von 17:00 - 07:00 Uhr** **from 5 pm - 7 am** + SA / SO	**SERVICE** **ERSATZTEILE / SPARES** verlängerte Ansprechzeit - extended office time - ♦ nur an Werktagen - only on working days - ♦ von 07:00 - 18:00 Uhr - from 7 am - 6 pm -
Rexroth Indramat GmbH Bgm.-Dr.-Nebel-Str. 2 97816 Lohr am Main **Kompetenz-Zentrum Europa** Tel.: +49 (0)9352 40-0 Fax: +49 (0)9352 40-4885	**Tel. +49 (0) 9352 40 50 60** service@indramat.de	**Tel.: +49 (0)172 660 04 06** **oder / or** **Tel.: +49 (0)171 333 88 26**	**Tel. +49 (0) 9352 40 42 22**
Vertriebsgebiet Süd Germany South	Gebiet Südwest Germany South-West	Vertriebsgebiet Ost Germany East	Vertriebsgebiet Ost Germany East
Rexroth Indramat GmbH Landshuter Allee 8-10 80637 München Tel.: +49 (0)89 127 14-0 Fax: +49 (0)89 127 14-490	Bosch Rexroth AG Vertrieb Deutschland – VD-BI Geschäftsbereich Rexroth Indramat Regionalzentrum Südwest Ringstrasse 70 / Postfach 1144 70736 Fellbach / 70701 Fellbach Tel.: +49 (0)711 57 61–100 Fax: +49 (0)711 57 61–125	Bosch Rexroth AG Beckerstraße 31 09120 Chemnitz Tel.: +49 (0)371 35 55-0 Fax: +49 (0)371 35 55-333	Bosch Rexroth AG Regionalzentrum Ost Walter-Köhn-Str. 4d 04356 Leipzig Tel.: +49 (0)341 25 61-0 Fax: +49 (0)341 25 61-111
Vertriebsgebiet West Germany West	Vertriebsgebiet Mitte Germany Centre	Vertriebsgebiet Nord Germany North	Vertriebsgebiet Nord Germany North
Bosch Rexroth AG Vertrieb Deutschland Regionalzentrum West Borsigstrasse 15 40880 Ratingen Tel.: +49 (0)2102 409-0 Fax: +49 (0)2102 409-406	Bosch Rexroth AG Regionalzentrum Mitte Waldecker Straße 13 64546 Mörfelden-Walldorf Tel.: +49 (0) 61 05 702-3 Fax: +49 (0) 61 05 702-444	Bosch Rexroth AG Walsroder Str. 93 30853 Langenhagen Tel.: +49 (0) 511 72 66 57-0 Fax: +49 (0) 511 72 66 57-95	Bosch Rexroth AG Kieler Straße 212 22525 Hamburg Tel.: +49 (0) 40 81 955 966 Fax: +49 (0) 40 85 418 978

Europa (West) - Europe (West)

vom Ausland: (0) nach Landeskennziffer weglassen, Italien: 0 nach Landeskennziffer mitwählen
from abroad: don't dial (0) after country code, Italy: dial 0 after country code

Austria - Österreich Bosch Rexroth GmbH Bereich Indramat Stachegasse 13 1120 Wien Tel.: +43 (0)1 985 25 40 Fax: +43 (0)1 985 25 40-93	**Austria – Österreich** Bosch Rexroth G.m.b.H. Gesch.ber. Rexroth Indramat Industriepark 18 4061 Pasching Tel.: +43 (0)7221 605-0 Fax: +43 (0)7221 605-21	**Belgium - Belgien** Bosch Rexroth AG Electric Drives & Controls Industrielaan 8 1740 Ternat Tel.: +32 (0)2 5830719 Service: +32 (0)2 5830717 Fax: +32 (0)2 5830731 indramat@boschrexroth.be	**Denmark - Dänemark** Bosch Rexroth A/S Zinkvej 6 8900 Randers Tel.: +45 (0)87 11 90 60 Fax: +45 (0)87 11 90 61
Great Britain – Großbritannien Bosch Rexroth Ltd. Rexroth Indramat Division Broadway Lane, South Cerney Cirencester, Glos GL7 5UH Tel.: +44 (0)1285 863000 Fax: +44 (0)1285 863030 sales@indramat.co.uk service@indramat.co.uk	**Finland - Finnland** Rexroth Mecman Oy Rexroth Indramat division Ansatie 6 017 40 Vantaa Tel.: +358 (0)9 84 91-11 Fax: +358 (0)9 84 91-13 60	**France - Frankreich** Bosch Rexroth S.A. Division Rexroth Indramat Avenue de la Trentaine BP. 74 77503 CHELLES CEDEX Tel.: +33 (0)164 72-70 00 Fax: +33 (0)164 72-63 00 **Hotline:** +33 (0)608 33 43 28	**France - Frankreich** Bosch Rexroth S.A. Division Rexroth Indramat 1270, Avenue de Lardenne 31100 Toulouse Tel.: +33 (0)5 61 49 95 19 Fax: +33 (0)5 61 31 00 41
France - Frankreich Bosch Rexroth S.A. Division Rexroth Indramat 91, Bd. Irène Joliot-Curie 69634 Vénissieux – Cedex Tel.: +33 (0)4 78 78 53 65 Fax: +33 (0)4 78 78 53 62	**Italy - Italien** Bosch Rexroth S.p.A. Via G. Di Vittoria, 1 20063 Cernusco S/N.MI Tel.: +39 02 2 365 270 Fax: +39 02 700 408 252378	**Italy - Italien** Bosch Rexroth S.p.A. Via Paolo Veronesi, 250 10148 Torino Tel.: +39 011 224 88 11 Fax: +39 011 220 48 04	**Italy - Italien** Bosch Rexroth S.p.A. Via del Progresso, 16 (Zona Ind.) 35020 Padova Tel.: +39 049 8 70 13 70 Fax: +39 049 8 70 13 77
Italy - Italien Bosch Rexroth S.p.A. Via Mascia, 1 80053 Castellamare di Stabia NA Tel.: +39 081 8 71 57 00 Fax: +39 081 8 71 68 85	**Italy - Italien** Bosch Rexroth S.p.A. Viale Oriani, 38/A 40137 Bologna Tel.: +39 051 34 14 14 Fax: +39 051 34 14 22	**Netherlands - Niederlande/Holland** Bosch Rexroth B.V. Kruisbroeksestraat 1 (P.O. Box 32) 5281 RV Boxtel Tel.: +31 (0)411 65 19 51 Fax: +31 (0)411 65 14 83 indramat@hydraudyne.nl	**Netherlands - Niederlande/Holland** Bosch Rexroth Services B.V. Kruisbroeksestraat 1 (P.O. Box 32) 5281 RV Boxtel Tel.: +31 (0)411 65 19 51 Fax: +31 (0)411 67 78 14
Norway - Norwegen Bosch Rexroth AS Rexroth Indramat Division Berghagan 1 or: Box 3007 1405 Ski-Langhus 1402 Ski Tel.: +47 (0)64 86 41 00 Fax: +47 (0)64 86 90 62 jul.ruud@rexroth.no	**Spain - Spanien** Bosch Rexroth S.A. Divisiòn Rexroth Indramat Centro Industrial Santiga Obradors s/n 08130 Santa Perpetua de Mogoda Barcelona Tel.: +34 9 37 47 94 00 Fax: +34 9 37 47 94 01	**Spain - Spanien** Goimendi S.A. División Rexroth Indramat Parque Empresarial Zuatzu C/ Francisco Grandmontagne no.2 20018 San Sebastian Tel.: +34 9 43 31 84 21 - service: +34 9 43 31 84 56 Fax: +34 9 43 31 84 27 - service: +34 9 43 31 84 60 satindramat-goimendi@adegi.es	**Sweden - Schweden** Rexroth Mecman Svenska AB Rexroth Indramat Division Varuvägen 7 125 81 Stockholm Tel.: +46 (0)8 727 92 00 Fax: +46 (0)8 647 32 77
Sweden - Schweden Rexroth Mecman Svenska AB Indramat Support Ekvändan 7 254 67 Helsingborg Tel.: +46 (0) 42 38 88 -50 Fax: +46 (0) 42 38 88 -74	**Switzerland West - Schweiz West** Bosch Rexroth Suisse SA Département Rexroth Indramat Rue du village 1 1020 Renens Tel.: +41 (0)21 632 84 20 Fax: +41 (0)21 632 84 21	**Switzerland East - Schweiz Ost** Bosch Rexroth Schweiz AG Geschäftsbereich Indramat Hemrietstrasse 2 8863 Buttikon Tel. +41 (0) 55 46 46 205 Fax +41 (0) 55 46 46 222	

Europa (Ost) - Europe (East)

vom Ausland: (0) nach Landeskennziffer weglassen
from abroad: don't dial (0) after country code

Czech Republic - Tschechien Bosch -Rexroth, spol.s.r.o. Hviezdoslavova 5 627 00 Brno Tel.: +420 (0)5 48 126 358 Fax: +420 (0)5 48 126 112	Czech Republic - Tschechien DEL a.s. Strojírenská 38 Zdar nad Sázavou 591 01 Czech republic Tel.: +420 616 64 3144 Fax: +420 616 216 57	Hungary - Ungarn Bosch Rexroth Kft. Angol utca 34 1149 Budapest Tel.: +36 (1) 364 00 02 Fax: +36 (1) 383 19 80	Poland – Polen Bosch Rexroth Sp.zo.o. Biuro Poznan ul. Dabrowskiego 81/85 60-529 Poznan Tel.: +48 061 847 64 62 /-63 Fax: +48 061 847 64 02
Rumania - Rumänien Bosch Rexroth Sp.zo.o. Str. Drobety nr. 4-10, app. 14 70258 Bucuresti, Sector 2 Tel.: +40 (0)1 210 48 25 +40 (0)1 210 29 50 Fax: +40 (0)1 210 29 52	Russia - Russland Bosch Rexroth Wolokolamskoje Chaussee 73 Zimmer 406, 408 RUS – 123424 Moskau Tel.: +7 095/ 232 08 34 +7 095/ 232 08 35 Fax: +7 095/ 232 08 36 info.rex@rexroth.ru	Russia - Russland ELMIS 10, Internationalnaya Str. 246640 Gomel, Belarus Tel.: +375/ 232 53 42 70 Fax: +375/ 232 53 37 69 elmis_ltd@yahoo.com	Turkey - Türkei Bosch Rexroth Otomasyon San & Tic. A..S. Fevzi Cakmak Cad No. 3 34630 Sefaköy Istanbul Tel.: +90 212 541 60 70 Fax: +90 212 599 34 07
Slowenia - Slowenien DOMEL Otoki 21 64 228 Zelezniki Tel.: +386 5 5117 152 Fax: +386 5 5117 225 brane.ozebek@domel.si			

Africa, Asia, Australia – incl. Pacific Rim

vom Ausland: (0) nach Landeskennziffer weglassen!
from abroad: don't dial (0) after country code!

Australia - Australien AIMS - Australian Industrial Machinery Services Pty. Ltd. Unit 3/45 Horne ST Campbellfield , VIC 3061 Melbourne Tel.: +61 (0) 393 590 228 Fax: +61 (0) 393 590 286 Hotline: +61 (0) 419 369 195 terryobrien@aimservices.com.au	**Australia - Australien** Bosch Rexroth Pty. Ltd. No. 7, Endeavour Way Braeside Victoria, 31 95 Melbourne Tel.: +61 (0)3 95 80 39 33 Fax: +61 (0)3 95 80 17 33 mel@rexroth.com.au	**China** Bosch Rexroth Ltd. Wai Gaoqiao Free Trade Zone No.122, Fu Te Dong Yi Road Shanghai 200131 - P.R.China Tel.: +86 21 58 66 30 30 Fax: +86 21 58 66 55 23 roger.shi_sh@boschrexroth.com.cn	**China** Bosch Rexroth (China) Ltd. 15/F China World Trade Center 1, Jianguomenwai Avenue Beijing 100004, P.R.China Tel.: +86 10 65 05 03 80 Fax: +86 10 65 05 03 79
China Bosch Rexroth (China) Ltd. A-5F., 123 Lian Shan Street Sha He Kou District Dalian 116 023, P.R.China Tel.: +86 411 46 78 930 Fax: +86 411 46 78 932	**China** Bosch Rexroth (Changzhou) Co.Ltd. Guangzhou Repres. Office Room 1014-1016, Metro Plaza, Tian He District, 183 Tian He Bei Rd Guangzhou 510075, P.R.China Tel.: +86 20 8755-0030 +86 20 8755-0011 Fax: +86 20 8755-2387	**Hongkong** Bosch Rexroth (China) Ltd. 6th Floor, Yeung Yiu Chung No.6 Ind Bldg. 19 Cheung Shun Street Cheung Sha Wan, Kowloon, Hongkong Tel.: +852 22 62 51 00 Fax: +852 27 41 33 44 alexis.siu@boschrexroth.com.hk	**India - Indien** Bosch Rexroth (India) Ltd. Rexroth Indramat Division Plot. A-58, TTC Industrial Area Thane Turbhe Midc Road Mahape Village Navi Mumbai - 400 701 Tel.: +91 (0)22 7 61 46 22 Fax: +91 (0)22 7 68 15 31
India - Indien Bosch Rexroth (India) Ltd. Rexroth Indramat Division Plot. 96, Phase III Peenya Industrial Area Bangalore - 560058 Tel.: +91 (0)80 8 39 73 74 Fax: +91 (0)80 8 39 43 45	**Indonesia - Indonesien** PT. Rexroth Wijayakusuma Building # 202, Cilandak Commercial Estate Jl. Cilandak KKO, Jakarta 12560 Tel.: +62 21 7891169 (5 lines) Fax: +62 21 7891170 - 71	**Japan** Bosch Rexroth Automation Corp. Service Center Japan Yutakagaoka 1810, Meito-ku, NAGOYA 465-0035, Japan Tel.: +81 (0)52 777 88 41 +81 (0)52 777 88 53 +81 (0)52 777 88 79 Fax: +81 (0)52 777 89 01	**Japan** Bosch Rexroth Automation Corp. Rexroth Indramat Division 1F, I.R. Building Nakamachidai 4-26-44, Tsuzuki-ku YOKOHAMA 224-0041, Japan Tel.: +81 (0)45 942 72 10 Fax: +81 (0)45 942 03 41
Korea Bosch Rexroth-Korea Ltd. 1515-14 Dadae-Dong, Saha-Ku Rexroth Indramat Division Pusan Metropolitan City, 604-050 Republic of South Korea Tel.: +82 (0)51 26 00 741 Fax: +82 (0)51 26 00 747 gyhan@rexrothkorea.co.kr	**Malaysia** Bosch Rexroth Sdn.Bhd. Head Office No. 3, Block B, Jalan SS 13/5 Subang Jaya Industrial Estate 47500 Petaling Jaya - Selangor Tel.: +60 (0) 3 73 44 870 Fax: +60 (0) 3 73 44 864 hockhwa@hotmail.com	**Singapore - Singapur** Robert Bosch (SEA) Pte Ltd. Dept. RBSI-R/SAT 38-C Jalan Pemimpin Singapore 577180 Tel.: +65 35 05 470 Fax: +65 35 05 313 kenton.peh@sg.bosch.com	**South Africa - Südafrika** TECTRA Automation (Pty) Ltd. 28 Banfield Road,Industria North RSA - Maraisburg 1700 Tel.: +27 (0)11 673 20 80 Fax: +27 (0)11 673 72 69 Hotline: +27 (0)82 903 29 23 georgv@tectra.co.za
Taiwan Rexroth Uchida Co., Ltd. No.17, Lane 136, Cheng Bei 1 Rd., Yungkang, Tainan Hsien Taiwan, R.O.C. Tel.: +886 (0)6 25 36 565 Fax: +886 (0)6 25 34 754 indramat@mail.net.tw	**Thailand** NC Advance Technology Co. Ltd. 59/76 Moo 9 Ramintra road 34 Tharang, Bangkhen, Bangkok 10230 Tel.: +66 2 943 70 62 +66 2 943 71 21 Fax: +66 2 509 23 62 sonkawin@hotmail.com		

Nordamerika – North America

USA Hauptniederlassung - Headquarters Bosch Rexroth Corporation Rexroth Indramat Division 5150 Prairie Stone Parkway Hoffman Estates, IL 60192-3707 Tel.: +1 847 6 45 36 00 Fax: +1 847 6 45 62 01 service@indramat.com	USA Central Region - Mitte Bosch Rexroth Corporation Rexroth Indramat Division Central Region Technical Center Auburn Hills, MI 48326 Tel.: +1 248 3 93 33 30 Fax: +1 248 3 93 29 06	USA Southeast Region - Südwest Bosch Rexroth Corporation Rexroth Indramat Division Southeastern Technical Center 3625 Swiftwater Park Drive Suwanee, Georgia 30174 Tel.: +1 770 9 32 32 00 Fax: +1 770 9 32 19 03	USA SERVICE-HOTLINE - 7 days x 24hrs - **+1-800-860-1055**
USA East Region –Ost Bosch Rexroth Corporation Rexroth Indramat Division Charlotte Regional Sales Office 14001 South Lakes Drive Charlotte, North Carolina 28273 Tel.: +1 704 5 83 97 62 +1 704 5 83 14 86	USA Northeast Region – Nordost Bosch Rexroth Corporation Rexroth Indramat Division Northeastern Technical Center 99 Rainbow Road East Granby, Connecticut 06026 Tel.: +1 860 8 44 83 77 Fax: +1 860 8 44 85 95	USA West Region – West Bosch Rexroth Corporation 7901 Stoneridge Drive, Suite 220 Pleasant Hill, California 94588 Tel.: +1 925 227 10 84 Fax: +1 925 227 10 81	
Canada East - Kanada Ost Bosch Rexroth Canada Corporation Burlington Division 3426 Mainway Drive Burlington, Ontario Canada L7M 1A8 Tel.: +1 905 335 55 11 Fax: +1 905 335-41 84 michael.moro@boschrexroth.ca	Canada West - Kanada West Bosch Rexroth Canada Corporation 5345 Goring St. Burnaby, British Columbia Canada V7J 1R1 Tel. +1 604 205-5777 Fax +1 604 205-6944 david.gunby@boschrexroth.ca		

Südamerika – South America

Argentina - Argentinien Bosch Rexroth S.A.I.C. "The Drive & Control Company" Acassusso 48 41/47 1605 Munro Prov. Buenos Aires Tel.: +54 (0)11 4756 01 40 Fax: +54 (0)11 4756 01 36 mannesmann@mannesmannsaic.com.ar	Argentina - Argentinien NAKASE Servicio Tecnico CNC Calle 49, No. 5764/66 1653 Villa Balester Prov. - Buenos Aires Tel.: +54 (0) 11 4768 36 43 Fax: +54 (0) 11 4768 24 13 nakase@usa.net nakase@nakase.com	Brazil - Brasilien Bosch Rexroth Ltda. Av. Tégula, 888 Ponte Alta, Atibaia SP CEP 12942-440 Tel.: +55 (0)11 4414 56 92 +55 (0)11 4414 56 84 Fax sales: +55 (0)11 4414 57 07 Fax serv.: +55 (0)11 4414 56 86 alexandre.wittwer@rexroth.com.br	Brazil - Brasilien Bosch Rexroth Ltda. R. Dr.Humberto Pinheiro Vieira, 100 Distrito Industrial [Caixa Postal 1273] BR - 89220-390 Joinville - SC Tel./Fax: +55 (0)47 473 58 33 Mobil: +55 (0)47 9974 6645 prochnow@zaz.com.br
Mexico Bosch Rexroth S.A. de C.V. Calle Neptuno 72 Unidad Ind. Vallejo MEX - 07700 Mexico, D.F. Tel.: +52 5 754 17 11 +52 5 754 36 84 +52 5 754 12 60 Fax: +52 5 754 50 73 +52 5 752 59 43	Columbia - Kolumbien Reflutec de Colombia Ltda. Calle 37 No. 22-31 Santafé de Bogotá, D.C. Colombia Tel.: +57 1 368 82 67 +57 1 368 02 59 Fax: +57 1 268 97 37 reflutec@inter.net. co		

Printed in Germany

Rexroth Indramat