SIEMENS

# WinCC

# Configuration Manual

## Manual

## Volume 1

C79000-G8276-C139-01

**Edition February 1999**

(We have checked the contents of this manual for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.)

Technical data subject to change

C79000-G8276-C139-01
Printed in the Federal Republic of Germany

Siemens Aktiengesellschaft

# Table of Contents

# Preface

## Purpose of this Manual

This manual introduces you to the configuarion possibilities of WinCC using the following sections:

- a general section about WinCC and its configuration
- an introduction to script processing
- an appendix

The table of contents and the index allow you to quickly find needed information. All the information can also be found in the online documentation (HTML format) which includes additional search functions.

Descriptions of sample projects for an effective and efficient configuration with WinCC can be found in the *WinCC Configuration Manual Volume 2*.

## Prerequisites for using this Manual

Knowledge of WinCC (**Getting Started**) or practical configuration experience using WinCC.

## Additional Support

For technical questions, please contact your local Siemens representative at the responsible branch office.

Their addresses can be found at the "Siemens Worlwide Appendix" of the "S7-300 Programmable Controller System, Hardware and Installtion Manual", in catalogs, and at CompuServe ("go autforum").

You can also direct your questions to our Hotline, which can be reached at:
+49 (911) 895-7000 (Fax 7001).

Additional information can be found on the Internet at the address:
www.aut.siemens.de/coros/html_00/coros.htm.

## Information about SIMATIC Products

Continuously updated information about SIMATIC products can be obtained:

- from the Internet at http://www.aut.siemens.de/
- via fax back service at 08765–93 02 77 95 00

Additionally, the SIMATIC Customer Support provides you with current information and downloads helpful for the application of SIMATIC products. They can be reached:

- from the Internet at http://www.aut.siemens.de/support/html_00/index.shtml
- via the SIMATIC Customer Support Mailbox at +49 (911) 895–7100
- To access the mailbox, use a modem up to V.34 (28.8 kBaud). Set its parameters as follows: 8, N, 1, ANSI, or dial-in via ISDN (x.75, 64 kBit).

The SIMATIC Customer Support can be reached by phone at +49 (911) 895–7000 and by fax at +49 (911) 895–7002. Inquiries can also be made via Internet mail or to the above mailbox.

# 1 Configuration Manual

The configuration manual is part of the WinCC documentation and is primarily concerned with the practical application of WinCC in projects.

## Introduction

This introduction provides you with general information about procedures for realizing HMI projects (Human Machine Interface). Only a few years ago, HMI systems were referred to as operations and monitoring systems.

In the last few years, the requirements for a system that monitors and controls production processes, as well as archives and processes production data, have increased dramatically. To meet those requirements, new HMI systems have been developed.

One of these new systems is WinCC. WinCC certainly is unique, considering its functionality, openness, and up-to-dateness of software technology.

Older generation HMI systems often provided only one way to solve a task. WinCC, on the other hand, almost always gives you multiple solution options. This configuration manual has been written to illustrate the best solution approaches taking into account performance and extent of configuration work.

This description provides you with solution suggestions for the effective utilization of WinCC in plant projects.

We have realized these solution suggestions in WinCC sample projects. These sample projects are contained on the WinCC CD-ROM. You can directly incorporate these solution suggestions into your projects and save valuable time.

## 1.1 Configuration Manual - Information about Structure and Application

### Prerequisite

You should already have practical experience with WinCC before working with this configuration manual. For WinCC newcomers, the **Getting Started** is an ideal introduction. The "Getting Started" covers the most important topics and illustrates them in a sample project. This configuration manual is an addition to the WinCC help system (online and documentation). Special features of objects, properties, and topics are described by the help system, if they were not explained in the configuration manual.

**Note:**

The project environment and dynamization types of WinCC *3.3 Peculiarities of configuration using WinCC* are explained in the chapter.

### Contents and Structure

The configuration manual is divided into six main chapters.

- The first chapter contains the preface, introduction, and general information about this manual.
- The second chapter contains general information about WinCC.
- The third chapter contains general and specific information for a structured and effective implementation of HMI projects.
- The fourth chapter contains a WinCC C language course. For beginners, it contains the most important rules for the application of the WinCC script language. For C experts, the special characteristics of the development environment are described.
- The sixth chapter contains the appendix with subjects, from WinCC *Solutions* and *Tips & Tricks*, that have not been explained in the sample projects.

## Conventions

The configuration manual uses the following conventions:

| **Convention** | **Description** |
|---|---|
| **Important** | Important text passages are shown in bold. |
| *Names* | Dialog box names, buttons, and field names are shown in italics. |
| *Inputs* | Inputs are shown in italics and blue. |
| *Menu →Sub Menu →etc.* | Selections via menus are shown in italics and blue.<br>The arrow indicates the operating sequence. |
| `Program` | C-scripts are always shown in this format: |
| `int` | In C-scripts, keywords are shown in blue. |
| `"Text",'z'` | In C-scripts, the character strings and individual characters are shown in red. |
| `Comments` | In C-scripts, the comments are shown in a smaller and cyan colored font. |
| `By the System` | Program parts that are preset by the system can not be changed. |
| 🖱 | Operation with the left mouse button. |
| 🖱**R** | Operation with the right mouse button. |
| 🖱**D** | Double click with the left mouse button. |
| **Note** | Notes are shaded in gray. |

## How to find Information

The **table of contents** is organized by subject.

Information by keyword can be found from the **index** .

In the **online documentation**, the *Find* tab contains almost all words listed in alphabetical order. A 🖱**D** on the desired word displays all chapters containing the word.

# 2 WinCC - General Information

This section describes the structure, concept, and working method of WinCC in greater detail.

## 2.1 WinCC - The Concept

In general, WinCC offers you three solution approaches for a configuration:

- using WinCC standard tools
- using existing Windows applications with WinCC via DDE, OLE, ODBC, and ActiveX
- using Visual C++ or Visual basic to develop your own applications embedded into WinCC

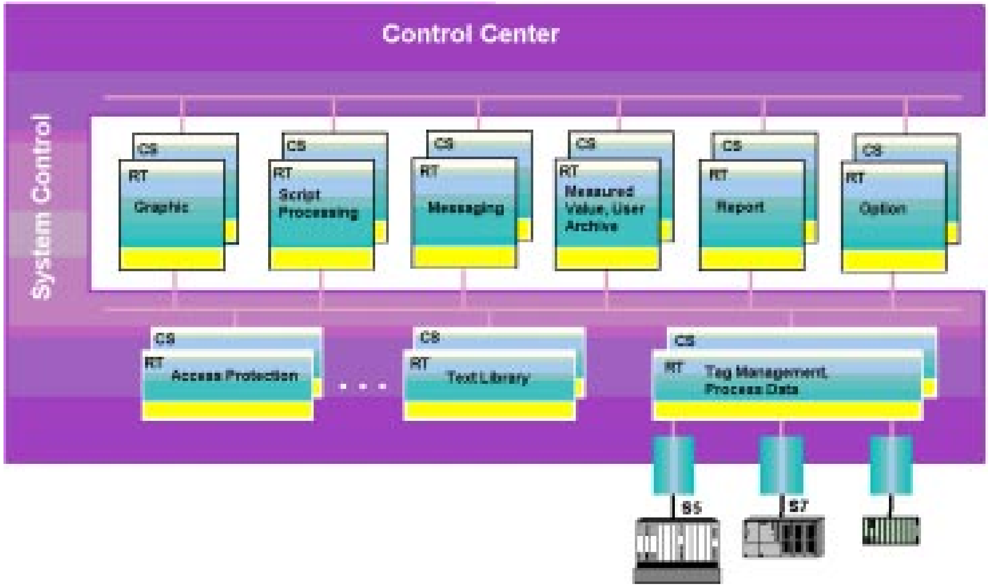
For some, WinCC is the HMI system for fast and cost-effective configurations, for others it is an infinitely expandable system platform. The modularity and flexibility of WinCC gives you completely new possibilities for planning and implementing automation tasks.

### The Operating System: Basics Of WinCC

WinCC is based on Microsoft's 32-bit operating systems (currently Windows 95 and Windows NT 4.0). These are the standard operating systems of the PC platform.

### The modular Structure of WinCC

WinCC offers system modules for the visualization, messaging, acquisition, and archiving of process data, as well as for the coordinated integration of user defined application routines.
You can also integrate your own modules.

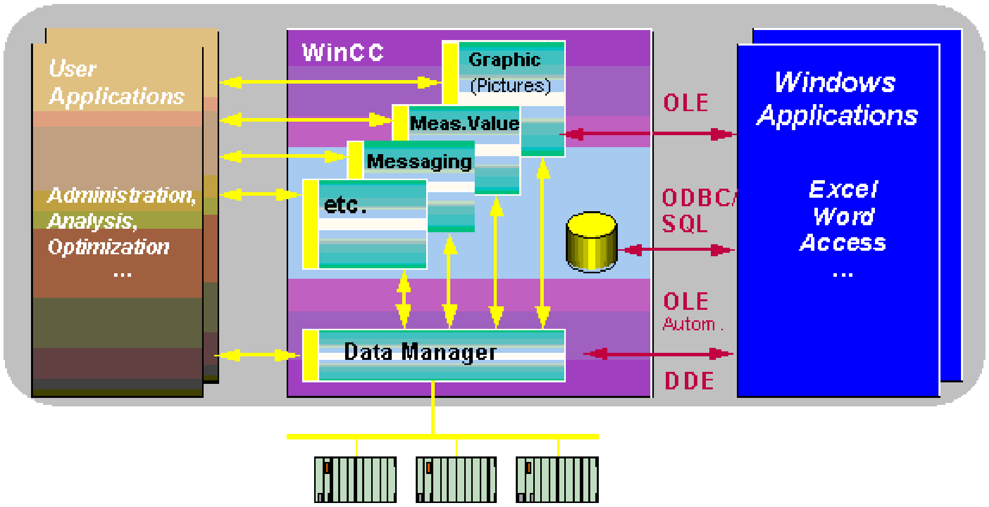


## 2.1.1 The Interfaces of WinCC

### The Openness of WinCC

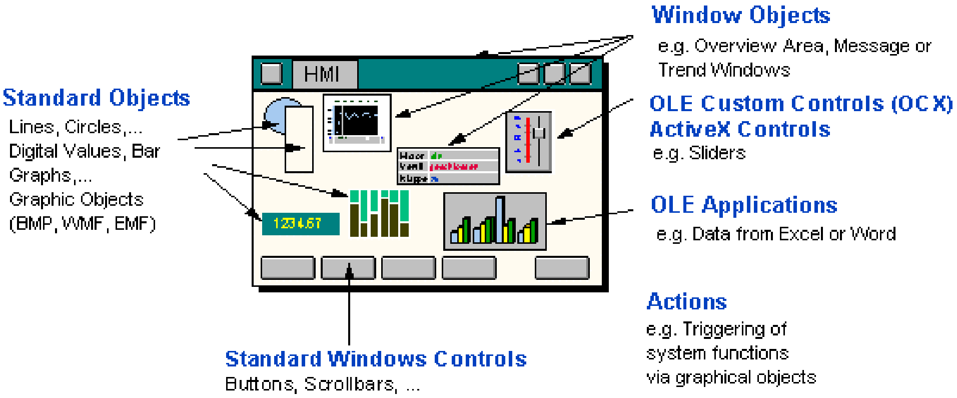
WinCC is completely open to any user add-on. This openness is accomplished through WinCC’s modular structure and its powerful programming interface.

The following picture illustrates the connection possibilities of various applications.



### Integration of External Applications into WinCC

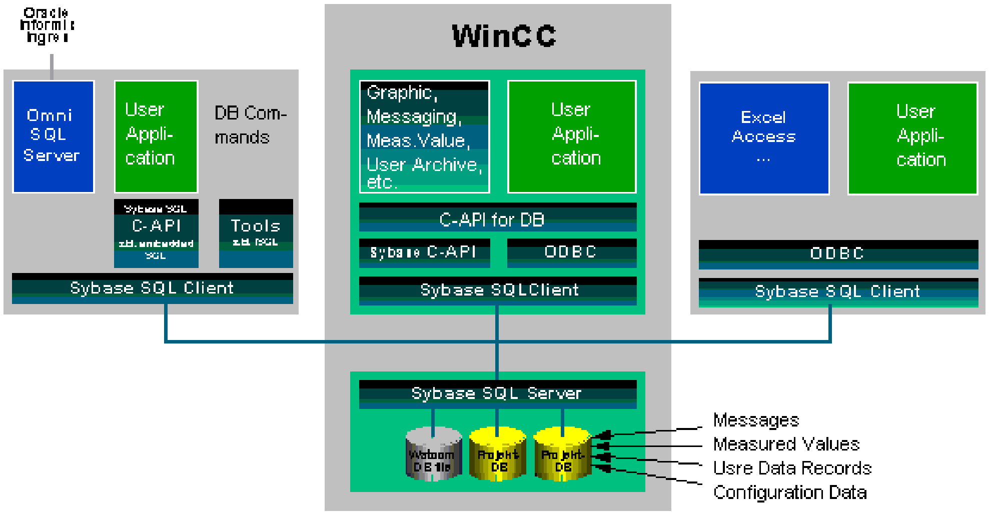
WinCC gives you options to integrate other applications and application modules **uniformly into the operator interface of the process.**
As pictured below, OLE application windows and OLE Custom Controls (32-bit OCX objects) / ActiveX Controls can be integrated into WinCC applications as if they were WinCC objects.



## WinCC Data Maintenance

In the following diagram, WinCC makes up the entire middle section. The diagram shows that the **Sybase SQL Anywhere standard database** is subordinate to WinCC. This database is used to store (transaction-secured) list-oriented configuration data (e.g. tag lists, message texts), as well as current process data (e.g. messages, measured values, user data records). The database has server functionality. WinCC can access the database via ODBC or the open programming interface (C-API) as a client.

Other programs can, of course, do the same. This gives a **Windows spreadsheet** or a **Windows database** direct access to the data of the WinCC database, regardless of whether the application is executed on the same computer or on a networked station. With the help of the database query language SQL and the appropriate Connectivity Tools (e.g. ODBC driver), other clients (for example **UNIX based databases** like Oracle, Informix, Ingres, etc.) can have access to the WinCC database. This also works the other way around. Nothing should stand in the way of **integrating WinCC into a process or company wide concept**.



## 2.2 WinCC - Terms and Abbreviations

This section contains a collection of terms about WinCC in alphabetical order. Many of these terms are probably already known to you:

| | |
|---|---|
| HMI | Human Machine Interface |
| PLC | Programmable Logic Controller |
| CS | Configuration System |
| RT | Runtime |

# 3 Configuration - General Subjects

In this section, you will find a great deal of information, instructions and ideas on how to manage projects using WinCC. Some of this information is not specific to WinCC. Ideally, these stipulations (configuration rules) should have the same quality as a style guide for configuring and designing runtime projects.

## 3.1 Before beginning the project

Before you begin with configuration, you should lay down a number of specifications and conduct some structuring work. This:

- simplifies configuration
- improves the clarity of the project
- simplifies working as a team
- improves stability and performance
- simplifies maintenance of the projects

Clear specification of the structural guidelines is a basic prerequisite for the setting-up or expansion of a corporate standard.

**These specifications can be divided into two categories:**

### Specifications for configuration

Before you configure, you should specify the following:

- specify the name of the WinCC project
- specify the names of the tags.
- specify the names of the WinCC pictures.
- specify rules for creating the scripts and actions.
- specify rules for configuration (corporate standards, library function, working in a team).
- specify the mode and method of documenting the project.

### Specifications for runtime project

Specifications which relate to the runtime project (result of configuration). These specifications depend heavily on the application (e.g. automobile industry, chemicals, machinery manufacturers). The following specifications should be performed:

- specify the user interface (screen layout, font and font size, language in runtime, representation of the objects).
- specify the control concept (picture hierarchy, control philosophy, user rights, valid keys).
- specify the colors to be used for messages, limit values, states, text etc.
- specify the modes of communication (type of connection, type of and cycles for updating).
- specify the typical application specification (number of alarms, archive values, trends, clients etc.).
- specify the logging and archiving methods.

## 3.2 Specifications in detail

In this section of the manual, we will lay down specifications which we will use in our sample projects. These specifications are intended to be used as a type of template when creating your own projects.

**Note:**
In our sample projects, the names of projects, pictures, tags and comments in the scripts are in English.

### Defaults of the configuration tools

In most of the editors in WinCC, certain properties can be set by means of defaults. In this way, WinCC supports your own particular style of configuration and can therefore be optimally configured for specific tasks.

**Note:**
An example of this are the options under *Graphics Designer → Tools → Settings...* . You can find a detailed description of this topic in the online Help of the *Graphics Designer*.

## 3.2.1 Specification: WinCC project name

### General

The name of a WinCC project is also suggested as the default name for the folder in which all the data is stored. You can change the folder name when initially creating the project or at a later stage (in Windows Explorer).

### Parameters / limits

With the exception of certain special characters - for example, \ ? ’ . ; : / - all characters are allowed. Numerical values from 0 to 9 are likewise allowed.

### Specification

In the example projects described in the second part of the Configuration Manual, the following specification applies to the project name:

a...a_nn

where:

a Type designation (a-z, A-Z, no special characters).
_n Serial number to distinguish between a number of projects of one type; numbers 0 to 9, range 00 to 99.

Example: cours_00.mcp or pictu_01.mcp

### Notes on general use

The WinCC project name can be used, for example, to distinguish between different sections of plant.

**Note:**

When updating the documentation, you can print out the project name in the printouts. This makes it easier to associate and find information.

## 3.2.2 Specification: tag names

### General

Tag names are no longer restricted to a maximum of 8 characters. Despite this, you should avoid making them too long. If you adhere to strict rules when allocating tag names, you will find this to be tremendously advantageous during configuration.
When creating WinCC projects, structuring tag management is one of the key tasks necessary to ensure quick and effective configuration and high-performance processing during runtime (in scripts).

Before defining the tag names, you must take a number of special characteristics relating to the structuring of tag management in WinCC into consideration. Creating groups only affects the way in which tags are displayed in the tag management during configuration. Group names in now affect the uniqueness of the tag names. The tag names used in a WinCC project must be unique. Their uniqueness is verified by the system.

WinCC helps you select tags in many different ways, e.g. through sorting according to columns (names, creation date, etc.) or through the use of filters. However, you may find it useful if the tag name contains additional information.

### Specification

The following specifications apply to tag names used in the sample projects dealt with in this manual:

xxxy_z...z_a...a_nn

where:

| x | Type | |
|---|---|---|
| | BIN | Binary tag |
| | U08 | Unsigned 8-bit value*(unsigned)* |
| | S08 | Signed 8-bit value*(signed)* |
| | U16 | Unsigned 16-bit value |
| | S16 | Signed 16-bit value |
| | U32 | Unsigned 32-bit value |
| | S32 | Signed 32-bit value |
| | G32 | Floating-point number 32-bit IEEE 754 |
| | G64 | Floating-point number 64-bit IEEE 754 |
| | T08 | Text tag 8-bit character set |
| | T16 | Text tag 16-bit character set |
| | ROH | Raw data type |
| | TER | Text reference |
| | STU | Structure types |

y Origin
r pure read tag from the PLC (read)
w write and read tag from the PLC (write)
i internal tag in WinCC without link to PLC
x Tag with indirect addressing (a text tag containing a tag name)

_z Group (corresponds to plant sections or buildings)
_Lack ... e.g. name of the plant section

_a Tag name - for example, name of measurement point
_EU0815V10 ... e.g. name of the measurement point

_n Serial number of the instance; numbers 0 to 9, range 00 to 99.

## Parameters / limits

The following restrictions apply when assigning tag names:

- The special character @ should be reserved for WinCC system tags, the use of this character is possible in principle, however.
- The special characters ’ and % cannot be used.
- The special character " and the character string // should not be used since they have a special meaning in C scripts (introduction or termination of a character string and introduction of a comment).
- no spaces.
- no distinction is made between uppercase and lowercase letters in tag names.

## Notes on general use

The tag names assigned in our examples are only suggestions.
When using the tags in scripts and Excel, you may find it useful to keep to a fixed length for the separate parts of the tag name (if necessary, using 0 or x as a filler).
Large quantities of tags can be created and maintained very effectively and simply in, for example, Excel. If the tag names have a fixed structure, it is considerably easier to create the tag lists in Excel. You can then import these tag lists created in Excel into the current WinCC project using the program *\SmartTools\CC_TagImportExport\Var_exim.exe* which is on your WinCC CD.

## 3.2.3 Specification: picture names

### General

If you want to address pictures in scripts or external programs, you will find it very helpful to use a fixed structure when assigning the picture names. You should also put some thought into deciding on the length of the picture names. Names (file names) that are too long are more likely to hinder clarity (making selections in list boxes, calls in scripts, etc.). Experience has shown a maximum length of 40 characters to be advisable.

### Parameters / limits

The following restrictions apply when assigning picture names:

- maximum length of 255 characters.
- any characters apart from specific special characters - for example, / " \ : ? < >.
- no distinction is made between uppercase and lowercase letters in picture names.

### Specification

The following specifications apply to picture names used in the projects dealt with in this manual:

aaaaa_k_x...x_nn

where:

a Picture identification (a-z, A-Z, no special characters) for grouping of the pictures.
course... e.g. name of the pictures in the C course

_k Picture type, identification of the picture type 0 to 99

| | |
|---|---|
| _0 | Start picture |
| _1 | Overview picture |
| _2 | Button picture |
| _3 | Plant picture |
| _4 | Detail picture |
| _5 | Message picture |
| _6 | Trend picture |
| _7 | ... |
| _8 | ... |
| _9 | Diagnostic pictures (for testing or commissioning only) |

_x Name for describing the picture function (a-z, A-Z, no special characters), maximum of 30 characters long.
_chapter ... e.g. name of the chapters in the C course

_n Serial number of the type; numbers 0 to 9, range 0 to 99.

## Notes on general use

The picture names assigned in our examples are only suggestions. You must, however, use the name convention we use for some of the scripts supplied.

## 3.2.4 Specification: scripts and actions

### General

You can create your own scripts and actions in WinCC projects. The names you assign should be of an **explanatory** nature. This makes things a lot easier when using scripts later.

Using a proportional font tends to be a nuisance when configuring in the Global Script (editor). For this reason, choose a font with a constant character width (e.g. Courier) to make things easier to read.

The scripts should be accompanied by appropriate and adequate comments. The amount of time spent writing comments is out of all proportion compared with the amount of time you need to comprehend a badly commented program. Although this fact is well appreciated by all, it is still often ignored.

### Specification

The following specifications apply to scripts used in the projects dealt with in this manual:
We use the proportional font *Courier New* in size 8;
all tag names and comments are in English.

### Notes on general use

You will find a detailed description of how to use scripts, actions and editors in Section *4.1 The development environment for scripts in WinCC.*

## 3.2.5 Specification: the user interface

### General

It is essential that you take the greatest of care when setting up the user interface. All objects created in the *Graphics Designer* are displayed on the screen in the user's office. The pictures created are the only interface between the machine and the user. For this reason, you must take great care creating them since they play an important role in ensuring the success of a project. It goes without saying that operation of the plant is more important than the appearance of the screen but in the long term sloppily created pictures can mar the impression made by and possibly even increase the costs of maintaining plant that has otherwise been well thought out.
**These are the pictures that the users (customers) see every day.**

In a screen display system, information about the current status of the plant is presented to the users solely by means of the pictures displayed. This interface must, therefore, provide **information in as comprehensive and easily understood a manner as possible**.
WinCC allows you to configure the user interface precisely as you want it. How you lay out the user interface of your own particular system depends on the hardware you use, on the demands during processing and on specifications that already exist.

### The users

When you are configuring the user interface, you must make the users, for whom configuration is after all being performed, the focus of your contemplations.

If you succeed in giving the users the information they need and do so in a clear manner, the result will be a **higher level of quality in production and fewer failures**. The amount of maintenance work necessary will also be reduced.

The users need as must information as they can get. Using this data as the foundation, the users can take the decisions that are essential to keeping the process running and with a high level of quality. The main job of the users is not to respond to alarms (the process has then already been thrown off balance), but to use their experience, knowledge of the process and the information provided by the operating system to predict the direction in which the process is developing. The users should be able to counteract irregularities before they arise. WinCC gives you the ability to edit and display this information to the users effectively.

### How much information should you pack into a picture?

When deciding on the amount on information that should be incorporated into a picture, there are two aspects that must be weighed against each other to achieve a balanced relationship:

- if a picture contains too much information, it will be difficult to read and searching for information will take too much time. The probability of errors being made by the users is also increased.
- if a picture contains too little information, the amount of work the users have to do is increased. They lose track of the process and have to change picture frequently in order to find the information required. This leads to delayed responses, control inputs and instability of the process being controlled.

Investigations have shown that experienced users want **as much information as possible in every picture**, so they don't have to change picture as often.
In contrast, beginners become confused and uncertain of what to do when a lot of information is packed into one picture. They either can't find the right information or can't find it in time.
But experience has taught us one thing: **a beginner soon becomes experienced, but an experienced user will never again become inexperienced**.

## Hiding information

The information displayed should be important and easily understood. You can **hide (mask out) certain pieces of information (e.g. the tag numbers, measurement point identifiers)** until they are needed.

## Displaying the information

When displaying analog values, combine them with pointer instruments with digital values.
Graphical representation of values (e.g. pointer instruments, bar graphs ...) makes it far easier and quicker for users to identify and grasp information.
In order to avoid problems from arising in the unlikely but possible event of a user being color-blind, important changes to an object (state) should be indicated by using not only a different color but also a different format.
Important information must always be immediately recognizable as such in a picture. This means good use of contrasting colors is essential.

## Color coding

The human eye picks up colors quicker than e.g. text. Working with color coding can make you far quicker at establishing the current status of the various objects, but it is important that you set up and at all times observe a consistent color coding scheme. Uniform color specifications for displaying states in a project (e.g. red for error/fault) are already standard. Corporate conventions already in force at the customers' must be taken into account.

## Displaying text

To make text easier to read, you should keep to a number of simple rules.

- The size of the text must be matched to the importance of the information contained in the text, but also to the distance the user will probably sit away from the screen.
- Lowercase letters should be preferred. They require less space and are easier to read than uppercase letters, even if the latter are easier to read from a distance.
- Horizontal text is easier to read than vertical or diagonal text.
- Use different fonts for different types of information (e.g. measurement point names, notes, etc.).

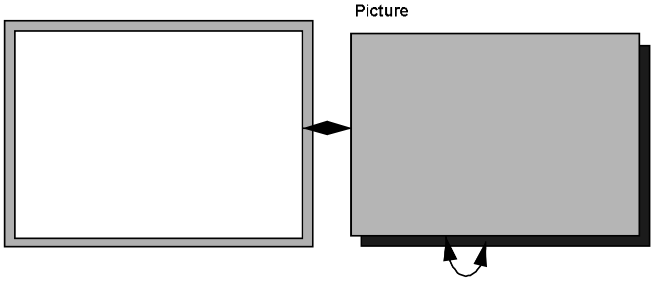
## Stick to your concept

Whatever concept you decide to use, you must always stick to it throughout the entire project. In this way, you support intuitive control of the process pictures. User errors become less likely.
This also applies to the objects used. A motor or pump must always look the same whichever picture it is depicted in.

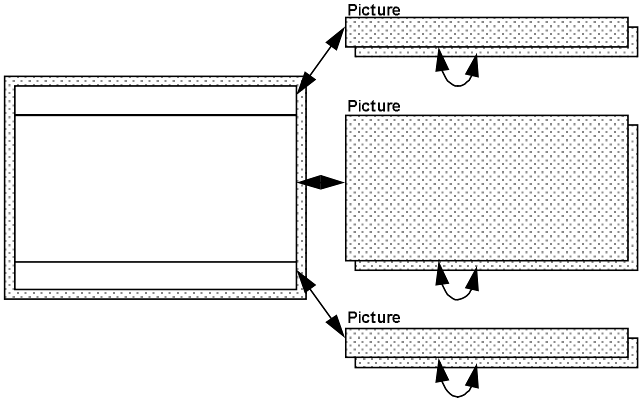
## Screen layout

If standard PC monitors are being used, experience has shown that it makes sense to split the screen into three sections: the overview section, the workspace section and the keys section.
If, however, your application runs on a special industry PC or operator panel with integral function keys, this method of sectioning the screen contents doesn't always make sense.

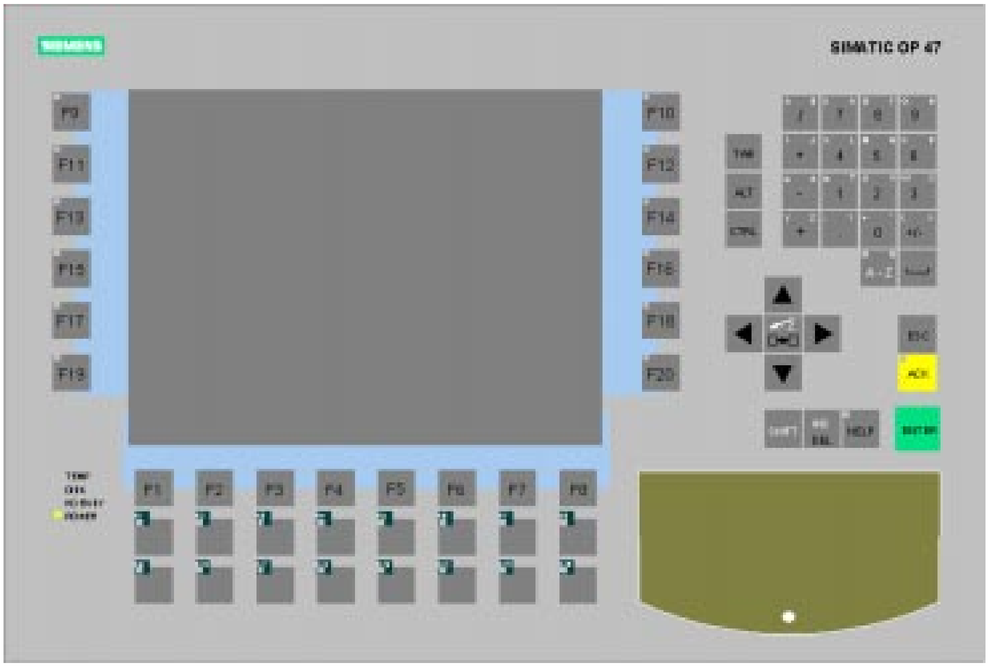
Pictures occupy the entire area of the screen



The screen is split into the overview, keys and plant pictures sections



Example of an operator panel



## Parameters / limits

The size of the individual pictures can be set as you wish within fixed limits (min 1 x 1, max. 4096 x 4096 pixels). In the case of single-user systems with a 17" monitor, we recommend you use a maximum resolution of 1024 x 768 pixels. With multi-user systems (multi-VGA), you may find a higher resolution useful.
In the case of operator panels, the technology used usually restricts the resolution available (TFT from 640 x 480 through 1024 x 768).

## Specification

The following specifications apply to pictures used in the projects dealt with in this manual:

### Resolution

In our sample projects, we use a resolution of 1024 x 768 and 800 x 600 pixels in exceptional cases. The color setting of your PC must be set to a minimum of 65536 colors for our sample projects to be displayed correctly.

### Texts

Different types of texts are written in different fonts: the names of measurement points are written in Courier, pure descriptions and all other texts and text displays in Arial. MS Sans Serif and System fonts are used in Windows-style information boxes.
The font size is adjusted when and as necessary.

## Information in the picture

Whenever it seems to make sense, we hide certain pieces of information in pictures. This information is only displayed when it is required (manual operation or automatic).
We also use a number of different screen layouts in our projects. If a picture contains a large number of controllable objects, we provide information on how to use them in the form of tool tips.

## Screen layout

We will configure the basic options for laying out the screen. In the other projects, however, we apply the subdivision into header, working area and key area.

## Notes on general use

You can reuse the basic layout of the concepts used for your own projects.

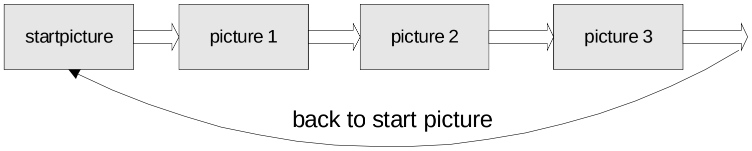
## 3.2.6 the control concept

### General

You control your process application under WinCC by means of the usual inputting means: a keyboard, mouse, touch screen or industry joystick. If your computer is located in an industrial setting with extreme conditions and where it would be impossible to use a mouse, you can use **tab orders** and the **alpha cursor**. The tab orders move you through controllable fields, while the alpha cursor moves you to the input fields.
Every operation can be locked against access by unauthorized persons.

### Opening pictures

The concept for opening pictures depends on a number of factors. Crucial to this concept is the number of pictures and the structure of the process that is to be displayed.
In small applications, the pictures can be arranged as a ring or FIFO buffer.



If you are working with a large number of pictures, a hierarchical arrangement for opening the pictures is imperative. Select a simple and permanent structure so that the operators can quickly learn how to open the pictures.

Of course, it goes without saying that pictures can be opened directly, and this may well make good sense for very small applications (e.g. a cold-storage depot).

### Hierarchy

**A hierarchical structure makes the process easily comprehensible,** simple to handle and provides, if necessary, rapid access to detailed information.
A common and frequently used hierarchical structure consists of three layers.

### Layer 1

Categorized under Layer 1 are the overview pictures.
This layer mainly contains information about the different system sections present in the system and about how these system sections work together.
This layer also indicates whether an event (message) has occurred in lower layers.

### Layer 2

Categorized under Layer 2 are the process pictures.
This layer contains detailed information about a specific process section and shows which plant objects belong to this process section.
This layer also indicates which plant object an alarm refers to.

### Layer 3

Categorized under Layer 3 are the detail pictures.
This layer provides you with information about individual plant objects, e.g. controllers, valves, motors etc. It displays messages, states and process values. If appropriate, it also contains information concerned with interaction with other plant objects.

## Specification

The following specifications apply to projects developed in the course of creating this manual:
We will be using a number of different control concepts in our projects and will point out the differences.

## Notes on general use

Our projects should only be regarded as a stimulus when you create your own control concept. When extending plant, you must take the existing control concepts into account. Many users will find that their company already has corporate conventions and standards that must be adhered to when conducting configuration.

**Note:**
The optional WinCC package known as **Basic Process Control** offers a ready-made control concept. This optional package also contains other useful and powerful functions (e.g. storage).

## 3.2.7 Specification: color definition

### General

The subject of colors is a very popular point of discussion with respect to HMI systems. WinCC allows you to freely select the colors used for lines, borders, backgrounds, shading and fonts. You have the choice of all those colors supported by Windows. Naturally, the colors, and the other graphic elements too, can be changed during runtime in WinCC. Color definition is particularly important in ensuring that configuration is inexpensive and that the processes are represented clearly.

Colors should always be defined for the following areas. The colors can be defined in accordance with DIN *EN 60073* which corresponds to *VDE 0199*, but this must always to agreed on together with the user:

- colors for messages (arrived / departed / acknowledged)
- colors for states (on / off / faulty)
- colors for character objects (leads / fill levels)
- colors for warning and limit values

### Specification

The following specifications apply to colors used in the projects dealt with in this manual:
The color setting of your PC must be set to greater than 256 colors for the sample projects to be displayed correctly.
To make it easier for you to find your bearings, we will use a different background color for the separate topics (tags, C course, picture configuration) dealt with in the sample projects. The background color in the overview and keys sections is darker.
In alarm logging, a specific color code is assigned to every message class and to every message type assigned top a message class.

### Notes on general use

After defining the colors, you should if necessary adjust the default settings of WinCC.

You will find a table for coding the color values in *C Actions* in the Appendix, Section *5.1.6 Color table*.

## 3.2.8 Specification: the update cycles

### General

When specifying the mode of updating, it is crucial that the overall system is looked at. What is updated and how often updating is carried out. Choosing the wrong update cycles can have negative effects on the performance of the HMI system.
When looking at an overall system (PLC - communication - HMI), changes should be detected where they occur, namely in the process (PLC). In many cases, it is the bus system that poses the bottleneck for data transmission.

When specifying the mode of updating measured values, you must pay attention to how quickly the measured value actually changes. It makes absolutely no sense at all to update the actual value for the temperature control of a boiler with a capacity of approx. 5,000 l with a 500 ms cycle.

### 32-bit HMI system

WinCC is pure 32-bit HMI system based on Windows 95 and Windows NT. These operating systems are optimized for event-driven control action. If you take this principle into account when configuring with WinCC, performance problems will be a rarity, even when your are handling very large volumes of data.

### Specification

The following specifications apply to updating in the projects dealt with in this manual: Insofar as the task definition permits, updating is performed driven by events. Since we work predominantly with *internal tags,* we often trigger change of the tags. When using external tags, this can lead to increased system load depending on the process driver connection. If communication allows event-driven transfer, it has to be chosen for time-critical data. Non-critical data can then be fetched by the HMI in suitable cycles (polling procedure).

### Notes on general use

You will find a detailed description of the use of update cycles in Section *3.3.1 Update cycles - how and where you set them.*

## 3.2.9 Specification: the user rights

### General

When operating plant, it is necessary to protect certain operator functions against unauthorized access. A further requirement is that only certain persons have access to the configuration system.
You can specify users and user groups and define various authorization levels in the *User Administrator*. These authorization levels can be linked to the control elements in the pictures.
The user groups and users can be assigned different authorization levels on an individual basis.

### Specification

In sample projects *course_00* and *varia_00*, every user is authorized to control operation of the project.
In sample project *pictu_00*, users can only control operation of the project after logging on. The password is the same as the project name (*pictu_00*).
The buttons used for selecting the individual topics are linked to the authorization level known as *Project control*.

### Notes on general use

You will find a description of how to assign user rights in the Configuration Manual, Part 2, in the example project *pictu_00* of Section 3.3 Shutdown WinCC / access enable.

## 3.2.10 Specification: alarm message reporting

### General

WinCC supports two different message procedures:

- The **bit message procedure** is a universal procedure which permits messages to be reported from any automation system. WinCC monitors the signal edge change of selected binary tags itself and derives message events from it.
- **Sequenced reporting** requires that the automation systems generate the messages themselves and send them in a predefined format to WinCC with a time stamp and possibly with process values. It is this message procedure which makes sequenced ordering of messages from different automation systems possible. Refer to Section *5.2 Documentation of S5 Alarm Logging*.

What is to be reported?
When specifying which events and states are to be reported, many people follow what they see as the safest route and set the software to report all events and changes in state. This leaves it up to the users to decide which messages they look at first.
If too many events are reported in a plant, experience shows us that important messages are only picked up too late.

### Notes on general use

How the messages are displayed and which messages are selected for archiving can be changed and customized to suit your own requirements.

## 3.2.11 Specification: implementation

### General

It makes particularly good sense if you use a fixed structure for storing data when implementing a project. Specification begins with deciding which drive the WinCC project is to be created. The next step concerns setting up the folder structure etc.
Experience has shown us that it makes sense to store all the data of a project under one folder which contains corresponding subfolders. You will find this method advantageous when processing a project, but even more so when backing up data.

**Note:**
PC configurations differ greatly. To avoid any problems this may cause when assigning the drive on which a project is to be processed, we suggest you use **virtual drives**. Assignment of a folder to a virtual drive can be changed at any time.

### Specifying folders

In addition to the folders that are created by WinCC, create further folders for Word, Excel and temporary files if required.

## 3.3 Peculiarities of configuration using WinCC

The following chapters deal with topics which cut across all aspects of configuration using WinCC.
These topics supplement the online help in WinCC.

## 3.3.1 Update cycles - how and where you set them

Specifying the update cycles is one of the most important setting procedures performed in the visualization system. The settings influence the following properties:

- picture buildup
- updating the objects of the picture currently open at the visualization station (*Graphics Designer*)
- editing background scripts (*Global Script*)
- activating the data manager and process communication

Other time variables are set during measured value processing (*Tag Logging*) under the archiving times.

### Data manager

The instantaneous tag values are requested by the data manager, the main administrator for tag management, in accordance with the update cycles set. See Chapter *3.3.2 Dynamization in WinCC*.

The data manager acquires the new process data via the communication channels and supplies these values to the applications. This requesting of data therefore means there is a switchover between different tasks (*Graphics Designer*, *Data manager* etc.). Depending on the configuration, this can lead to very different system loads.

### 3.3.1.1 Updating in the picture

### Picture updating

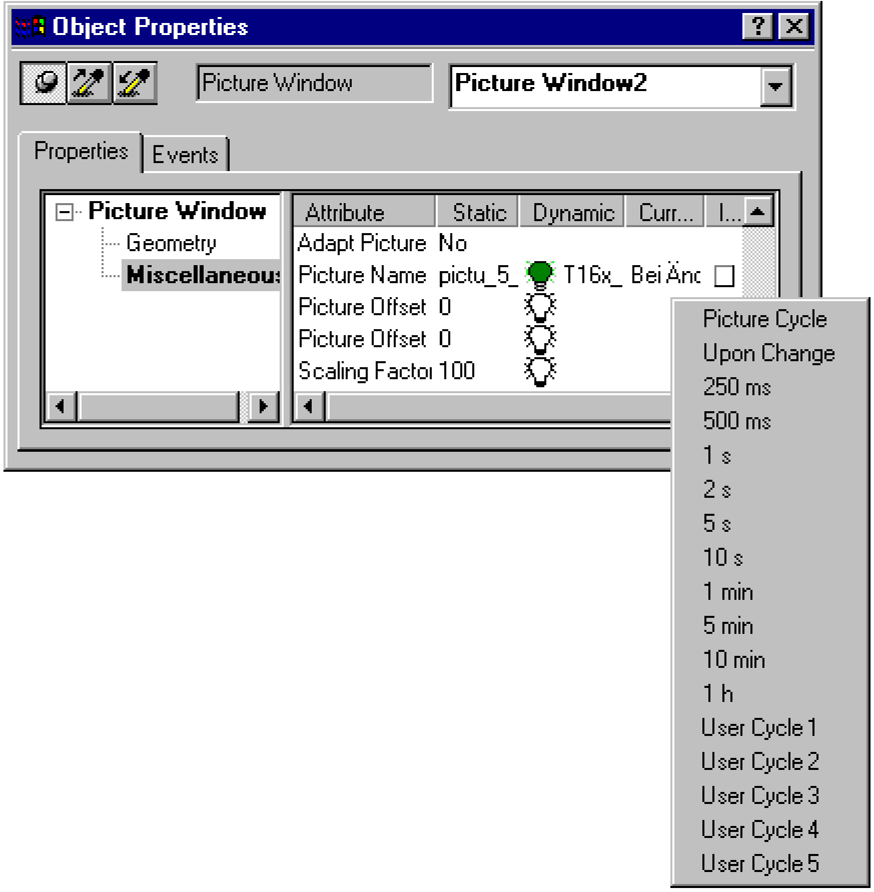
Updating of the individual properties of the objects in the picture refers to the objects that are dynamized after the picture is opened. The task of the updating cycle is to establish the current state of the particular object in the picture. The update cycle of the dynamic objects can be set by the person configuring the system or by the system itself for the following types of dynamization:

| **Dynamization type** | **Default setting** | **Configuration customization** |
|---|---|---|
| Configuration dialog | Tag trigger, 2 sec.<br>or<br>event trigger (e.g. control) | Customizing the time cycles |
| Dynamic Wizard | You can choose from the following depending on the type of dynamization<br>• event trigger<br>• time cycle<br>• tag trigger | Customizing the time cycles, events or tags |
| *Direct link* | event trigger | |
| Tag link | Tag trigger, 2 sec. | Customizing the time cycles |
| *Dynamic dialog* | Tag trigger, 2 sec. | Customizing the time cycles, tag triggers |

| Dynamization type | Default setting | Configuration customization |
|---|---|---|
| *C action* for properties | Time cycle of 2 seconds | Customizing the time cycles, tag triggers<br>Direct reading from the PLC |
| Object property | Setting depending on the dynamics | Editing the Update cycle column |

The update cycles to be selected are specified by WinCC and can be added to by users defining their own time cycles.

Selecting the update cycles, e.g. for property of an object:



Object Properties

Picture Window | Picture Window2

Properties | Events

| Attribute | Static | Dynamic | Curr... | I... |
|---|---|---|---|---|
| Adapt Picture | No | | | |
| Picture Name | pictu_5_ | T16x_ | Bei Änc | □ |
| Picture Offset | 0 | | | |
| Picture Offset | 0 | | | |
| Scaling Factoı | 100 | | | |

## 3.3.1.2 Types of update cycle

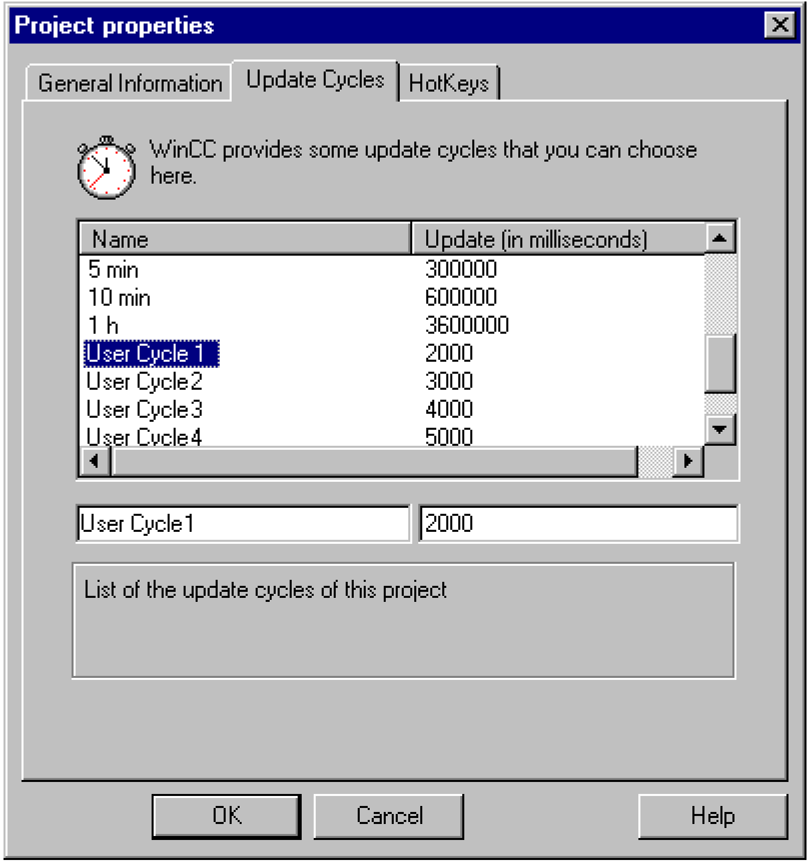
We distinguish between the following types of update cycles:

| Type | Default setting |
|---|---|
| Default cycle | Time cycle of 2 seconds |
| time cycle | 2 seconds |
| tag trigger | 2 seconds |
| Picture cycle | 2 seconds |

| Type | Default setting |
|---|---|
| Window cycle | upon change |
| User-defined time cycles | User cycle 1: 2 sec.<br>User cycle 2: 3 sec.<br>User cycle 3: 4 sec.<br>User cycle 4: 5 sec.<br>User cycle 5: 10 sec. |

## User cycle

Up to 5 **project-related** user cycles can be defined. If the project name has been selected in the left tree structure in the *Control Center*, you can use the button on the toolbar to open the *Project properties* dialog box. The *Project Properties* dialog box presents, on the *Update Cycles* tab, user cycles 1 to 5 for the project-related definition at the end of the list of set standard update cycles. Only these user cycles can be parameterized.



WinCC provides some update cycles that you can choose here.

| Name | Update (in milliseconds) |
|---|---|
| 5 min | 300000 |
| 10 min | 600000 |
| 1 h | 3600000 |
| User Cycle 1 | 2000 |
| User Cycle2 | 3000 |
| User Cycle3 | 4000 |
| User Cycle4 | 5000 |

User Cycle1 2000

List of the update cycles of this project

These user cycles enable you to define time cycles other than the ones already available (e.g. 200 ms).
You can define user cycles for any length of time between 100 ms and 10 hours. You can give user cycles any name you wish.

These project-related units of time can be used for selected objects whose update cycle must be modified at a later time. Once reason for changing the time cycles could be to effect an optimization. The user-defined update cycles also make it possible for you to subsequently modify the set time cycle from a **single** central point. In this case, the individual objects of the pictures no longer have to be adjusted as well. This is why this method of defining user cycles should be preferred if you want your projects to be **maintenance-friendly**.

### 3.3.1.3 What the update cycles mean

Before you begin putting the possible update cycles to use, we must first take a look at what the various update cycles mean.

We distinguish between the following types of update cycles:

| **Type** | **Meaning** |
|---|---|
| Default cycle | time cycle |
| time cycle | The property or action of the individual object is updated after the time set. This means that each of the tags is requested **individually** by the data manager. |
| tag trigger | In accordance with the cycle time set and once the time interval has elapsed, the tags are determined by the system and checked for value changes.<br>If the value of at least one selected tag changes during the time frame set, this acts as a trigger for the properties or actions dependent on this.<br>**All** tag values are requested **together** by the data manager. |
| Picture cycle | Updating of the properties of the current picture object and all objects that are triggered by means of the Picture Cycle update cycle. |
| Window cycle | Updating of the properties of the window object and all objects that are triggered by means of the Window Cycle update cycle. |
| User-defined time cycle | Time units that can be defined specifically for a project. |
| *C action* for direct reading from the PLC | Values can be read directly from the PLC by means of internal functions in the *C actions*. Further editing of the subsequent instructions in the *C action* is only continued after the process values have been read (synchronous reading). |

**Note:**

Requesting of the current tag value by the data manager leads in each case to a change of task and to a data exchange between the individual tasks. In addition, the tag values must be requested by the data manager via the communication channel of the programmable controllers connected. Depending on the type of communication, this is done by means of request telegrams sent to the communication interface (FETCH) and data telegrams sent back from the programmable controller to WinCC.

### 3.3.1.4 Instructions on how to use the update cycles

We recommend the following settings when using the update cycles, depending on the respective type of cycle:

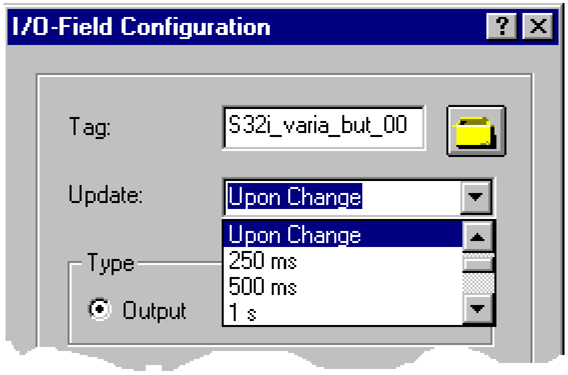
| **Type** | **Default setting** | **Recommendation for configuration** |
|---|---|---|
| Default cycle | Time cycle of 2 seconds | *Dynamic dialog* or *C action*:<br>if tags are interdependent, you should at all events make use of tag triggering. This reduces the number of task changes and communication between the tasks.<br>Tag triggering *upon change* may only be used selectively, since it can lead to greater system load! The tags are then checked constantly for changes. This polling mechanism always leads to a greater load on the system.<br>We recommend a cycle of 1 to 2 seconds for standard objects. |
| time cycle | 2 seconds | Make each time cycle dependent on the object type or the object property. The inertia of process components (tank fillings or temperatures in contrast to switching operations) should likewise be taken into account.<br>We recommend a cycle of 1 to 2 seconds for standard objects. |
| Tag trigger | 2 seconds<br>(for *dynamic dialog*) | If this update option is configurable (depends on the dynamics type), **preference** should be given to using it! If tags are interdependent, always taken into account **all** tags that are responsible for a change to the property or for execution of the action. Only those tags contained in the list act as triggers for updating of the dynamized property or action.<br>**Tag triggering** *upon change* should only be used selectively. As soon as one of the selected tags has changed, the trigger for this property or action is tripped. This polling mechanism leads to a greater load on the system. |
| Picture cycle | 2 seconds | This cycle should only be shortened if the dynamized properties of the picture object themselves change in a shorter time interval and therefore have to be updated. Lengthening the picture cycle reduces the load on the system. |
| Window cycle | upon change | This setting makes sense if you are dealing with a picture window that is opened, for example, for adjusting process variables (process box).<br>If the picture window is displayed constantly for informational purposes (e.g. screen layout), updating of the window and its contents should be set to tag triggering or a time cycle. |
| *C action* for direct reading from the PLC | | The internal functions (e.g. GetTagWordWait ) for synchronous reading of process values (direct from the PLC) should only be used selectively. Application of these functions requires polling by the system (script |

| Type | Default setting | Recommendation for configuration |
|---|---|---|
| | | control) and therefore leads to a greater communication load. |

The following examples show where each of the update cycles is set:

## Configuration dialog

This dialog box appears when you configure a *Smart Object →I/O Field*. It can also be opened by means of 🖱**R** above the object concerned, however.



I/O-Field Configuration

Tag: S32i_varia_but_00

Update: Upon Change

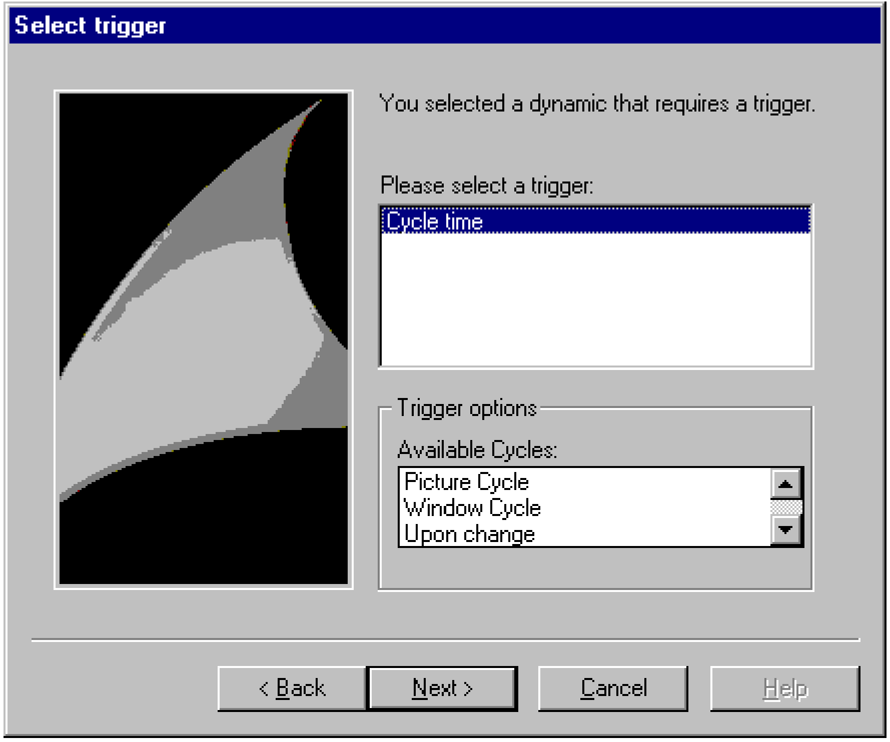
Upon Change
250 ms
500 ms
1 s

Type

● Output

## Dynamic Wizard

This page appears when you select *Dynamize Properties* on the *Standard Dynamics* tab in the *Dynamic Wizard* with 🖱**D**.



Select trigger

You selected a dynamic that requires a trigger.

Please select a trigger:

Cycle time

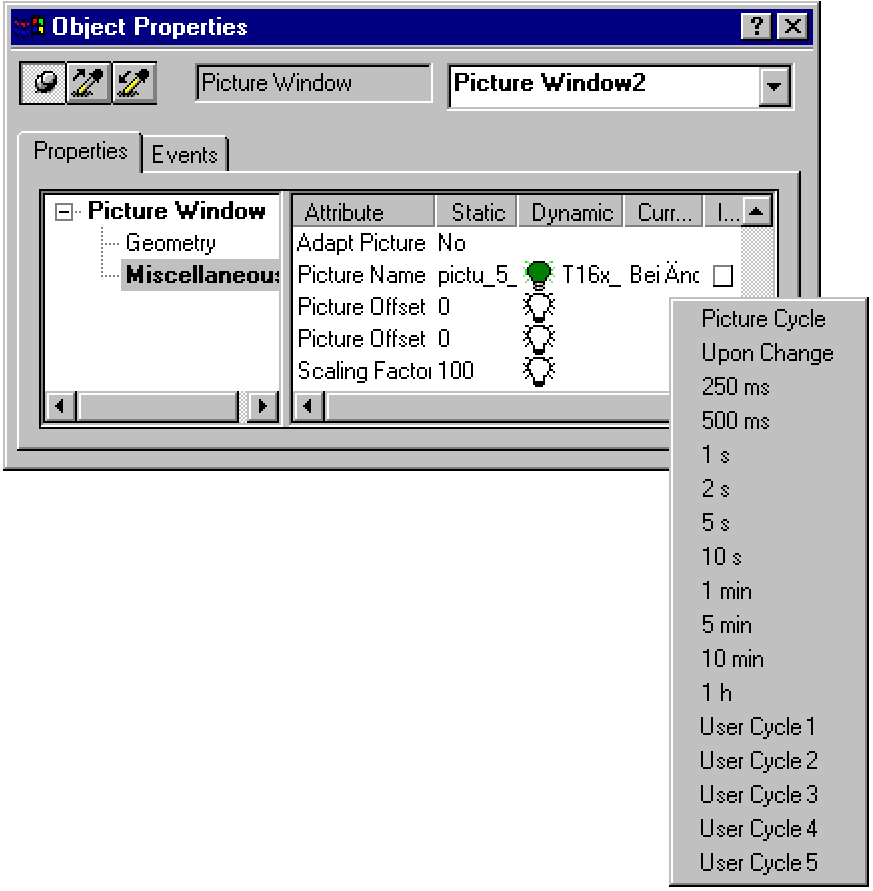
Trigger options

Available Cycles:

Picture Cycle
Window Cycle
Upon change

## Tag link for object property

This menu appears when you select the *Update* column with 🖱**R** in the event of a tag-dynamized object property.



Object Properties

Picture Window | Picture Window2

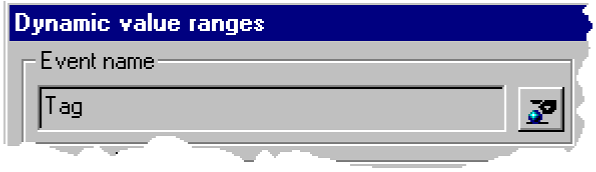
Properties | Events

- Picture Window
  - Geometry
  - Miscellaneous

| Attribute | Static | Dynamic | Curr... | I... |
|---|---|---|---|---|
| Adapt Picture | No | | | |
| Picture Name | pictu_5_ | ● T16x_ | Bei Änc | □ |
| Picture Offset | 0 | 💡 | | |
| Picture Offset | 0 | 💡 | | |
| Scaling Factor | 100 | 💡 | | |

## Dynamic dialog

If you select the trigger button when in a *dynamic dialog box*, you select the dialog box for changing the update cycle.

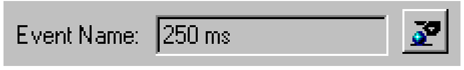


Dynamic value ranges

Event name: Tag

## C action for property

If you select the trigger button in the editor while handling a *C action*, you select the dialog box for changing the update cycle.

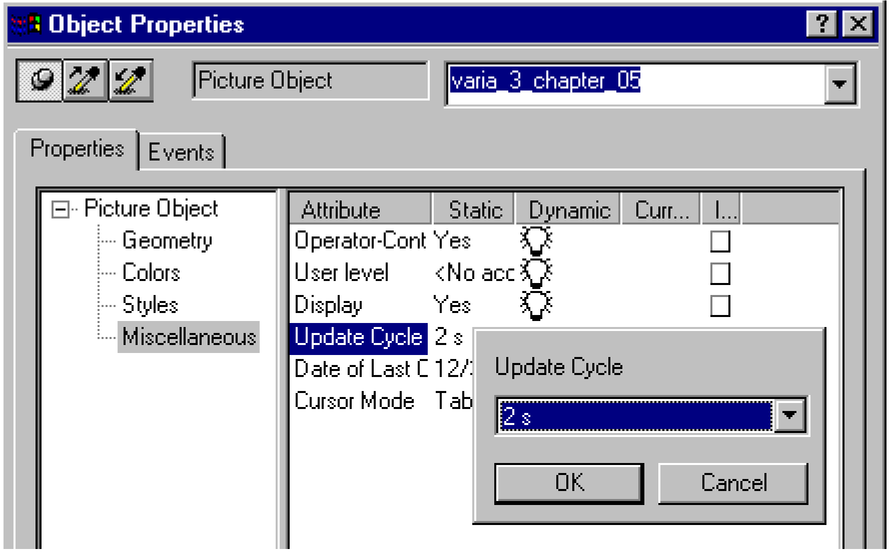


Event Name: 250 ms

The update cycles set by default can be changed as follows:

## Picture cycle

Changing the picture cycle



Object Properties

Picture Object | varia_3_chapter_05

Properties | Events

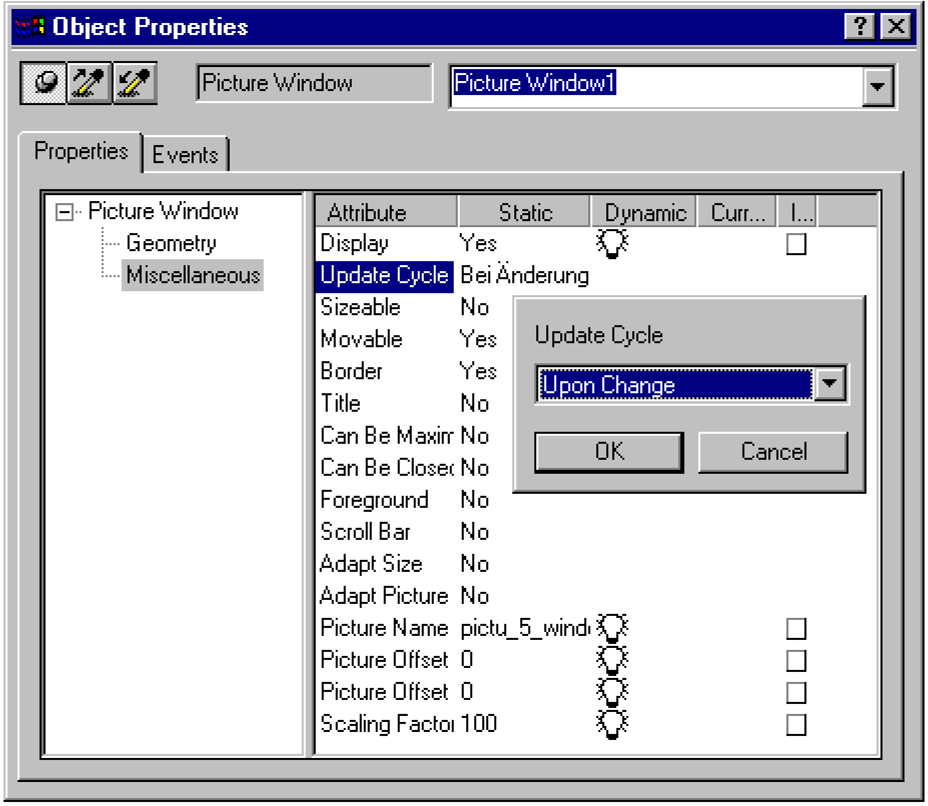
- Picture Object
  - Geometry
  - Colors
  - Styles
  - Miscellaneous

| Attribute | Static | Dynamic | Curr... | I... |
|---|---|---|---|---|
| Operator-Cont | Yes | 💡 | | □ |
| User level | <No acc | 💡 | | □ |
| Display | Yes | 💡 | | □ |
| Update Cycle | 2 s | | | |
| Date of Last C | 12/: | | | |
| Cursor Mode | Tab | | | |

Update Cycle: 2 s

## Window cycle

Changing the window cycle



Object Properties

Picture Window | Picture Window1

Properties | Events

- Picture Window
  - Geometry
  - Miscellaneous

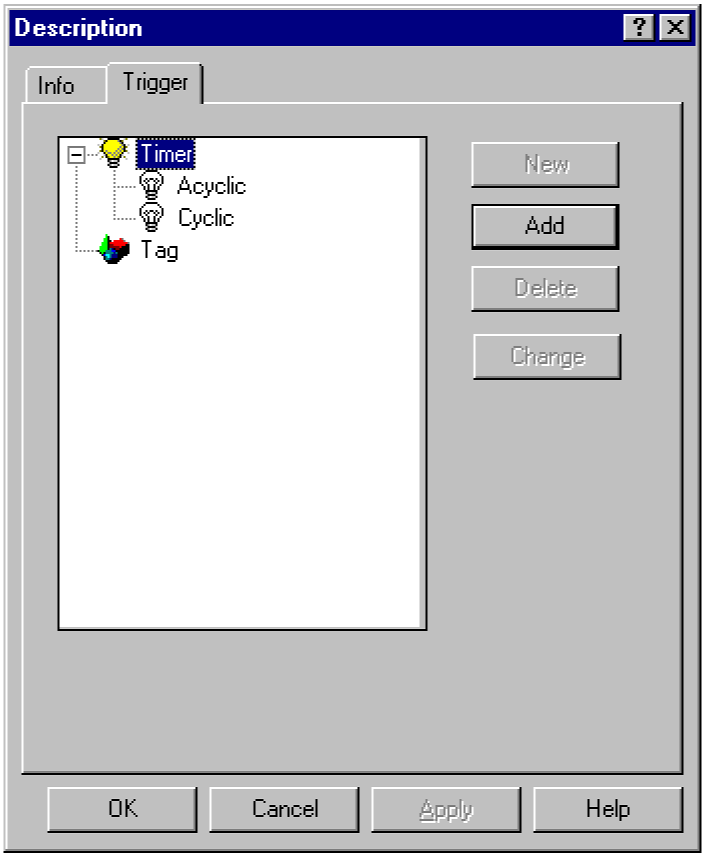
| Attribute | Static | Dynamic | Curr... | I... |
|---|---|---|---|---|
| Display | Yes | 💡 | | □ |
| Update Cycle | Bei Änderung | | | |
| Sizeable | No | | | |
| Movable | Yes | | | |
| Border | Yes | | | |
| Title | No | | | |
| Can Be Maxim | No | | | |
| Can Be Closed | No | | | |
| Foreground | No | | | |
| Scroll Bar | No | | | |
| Adapt Size | No | | | |
| Adapt Picture | No | | | |
| Picture Name | pictu_5_windo | 💡 | | □ |
| Picture Offset | 0 | 💡 | | □ |
| Picture Offset | 0 | 💡 | | □ |
| Scaling Factor | 100 | 💡 | | □ |

Update Cycle: Upon Change

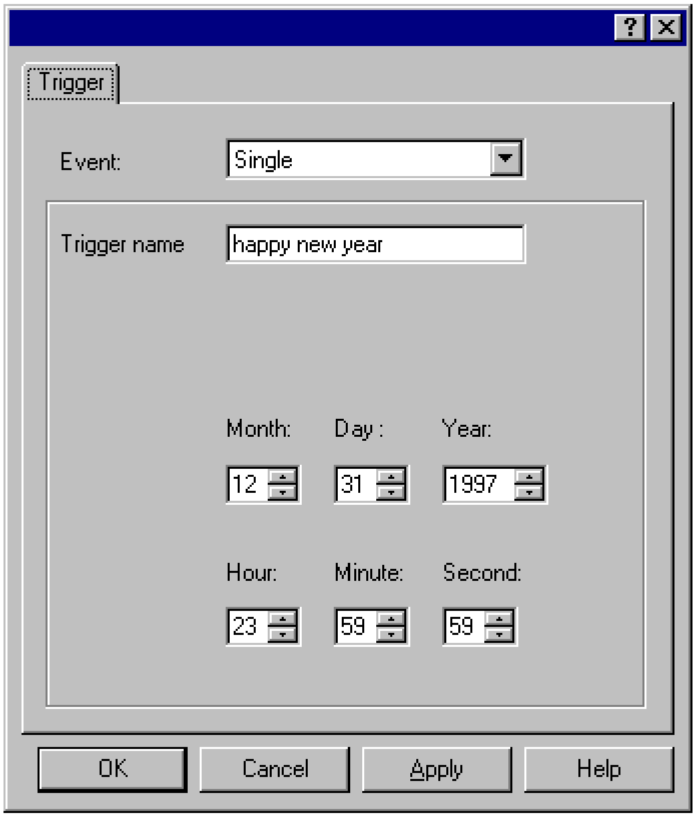
### 3.3.1.5 Executing background scripts (Global Script)

The execution of background scripts (*Global Script*) is dependent on various variables depending on the configuration:

- time trigger (cyclic execution, exception: acyclic = single (or non-recurrent))
  - time cycle
  - time
- Event trigger



or only once



Event: Single

Trigger name happy new year

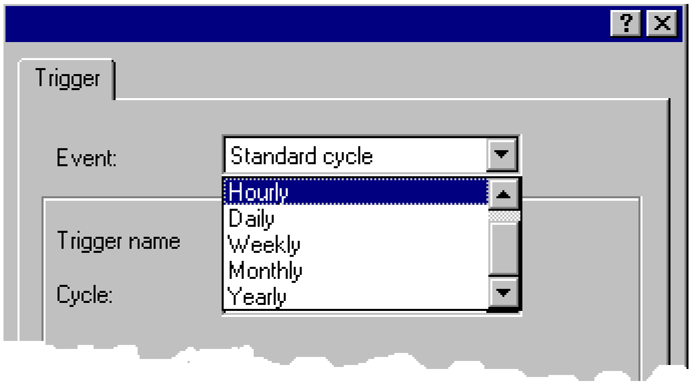
Month: 12 Day : 31 Year: 1997

Hour: 23 Minute: 59 Second: 59

## Time cycle

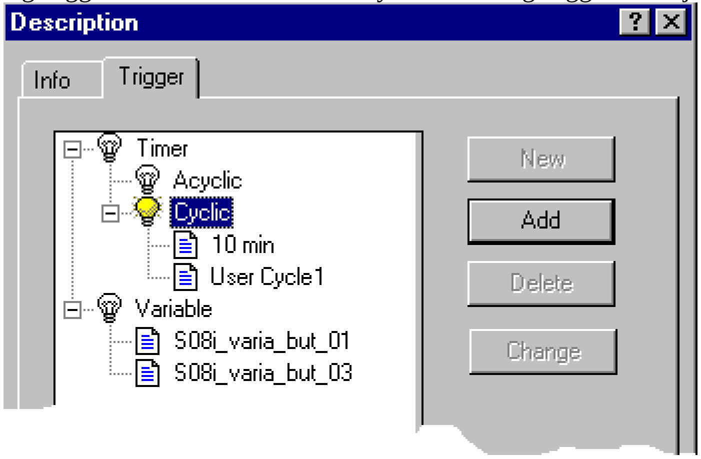
The configured time factor of the global action defines **when** the defined sequence of actions must be processed. Besides the standard cycle already described and the corresponding time settings from 250 ms through 1 h (or user cycles 1 through 5), you can also select from the time triggers:

- Hourly (minute and second)
- Daily (hour, minute, second)
- Weekly (day of week, hour, minute, second)
- Monthly (day, hour, minute, second)
- Yearly (month, day, hour, minute, second)

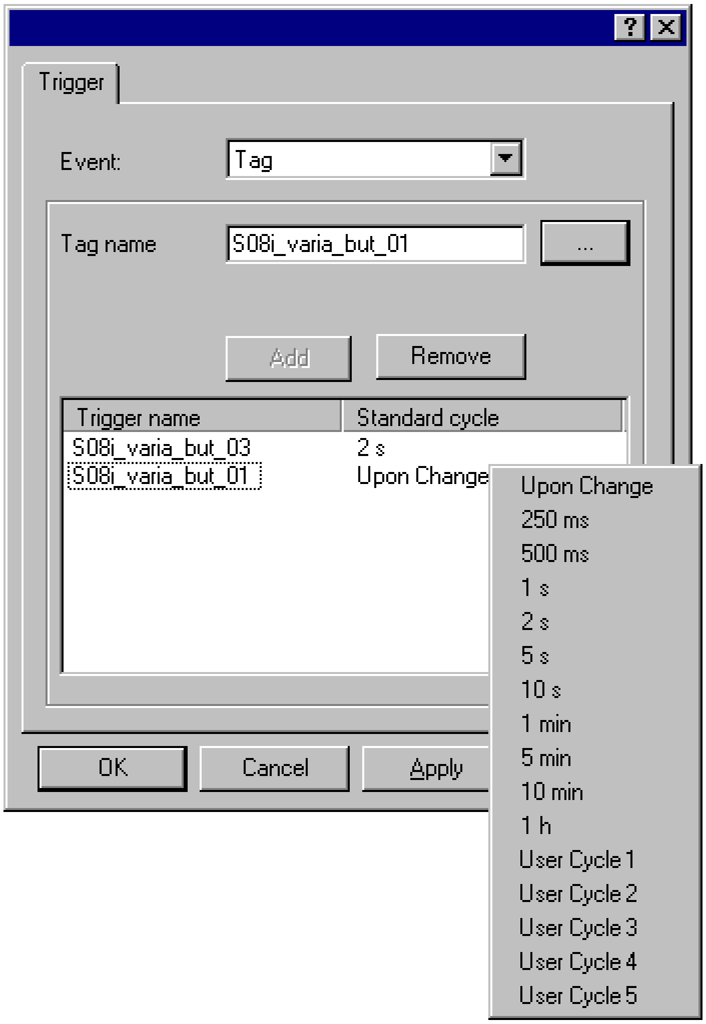


## Tag trigger

If the action is activated dependent on one or more tags, the event trigger must be set as a tag trigger. You do this in same way as for the tag trigger for object properties.



The 2-second cycle is set by default. The person configuring the system can, however, choose from the following time factors instead of the default value:



At the start and at the end of the set time frame, the values of the tags selected are determined. If the value of at least one of the tags has changed, the trigger for the global action is tripped.
Note the high system load when triggering *upon change*. This setting is not always appropriate. The same advice applies here as does for the Update object.

**All** actions you define as global actions are **not** checked and activated **object-linked**, i.e. only dependent on the time cycles or event triggers set. For this reason, only use the global actions selectively and avoid unnecessary action steps, in order not to put too heavy a load on the system. Neither use too many nor too many short time cycles for executing your actions.

## 3.3.2 Dynamization in WinCC

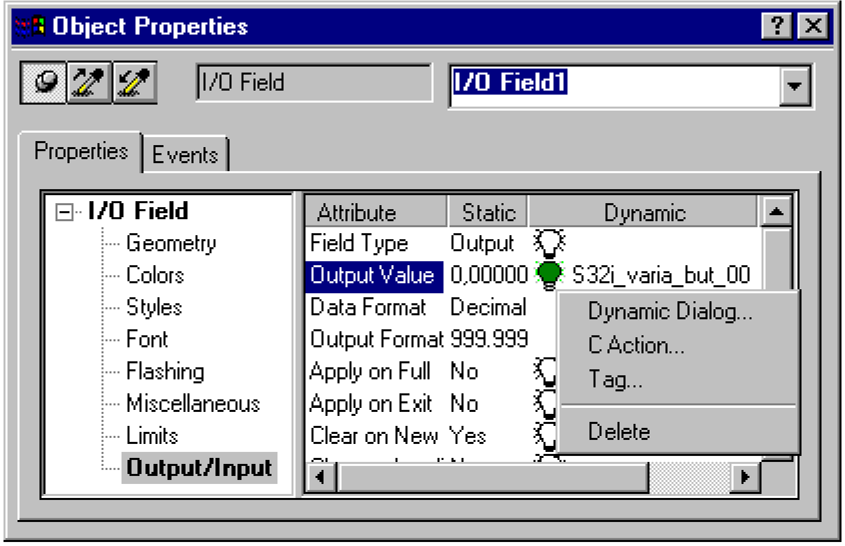
### Definition

The concept of dynamization in WinCC refers the changing of states (e.g. position, color, font, etc.) and the response to events (e.g. mouse action, keyboard action, value change, etc.) during runtime.

Each element in the graphics window is viewed as an independent object. The graphics window itself is likewise an object of the object type known as *Picture Object*.
Each object in the WinCC *Graphics System* has *Properties* and *Events*. With just a few exceptions, these properties and events can be dynamized. The small number of exceptions mainly concerns *properties* and *events* that have no effect during runtime and they do not have a symbol which indicates they can be dynamized.

### 3.3.2.1 Dynamizing the properties

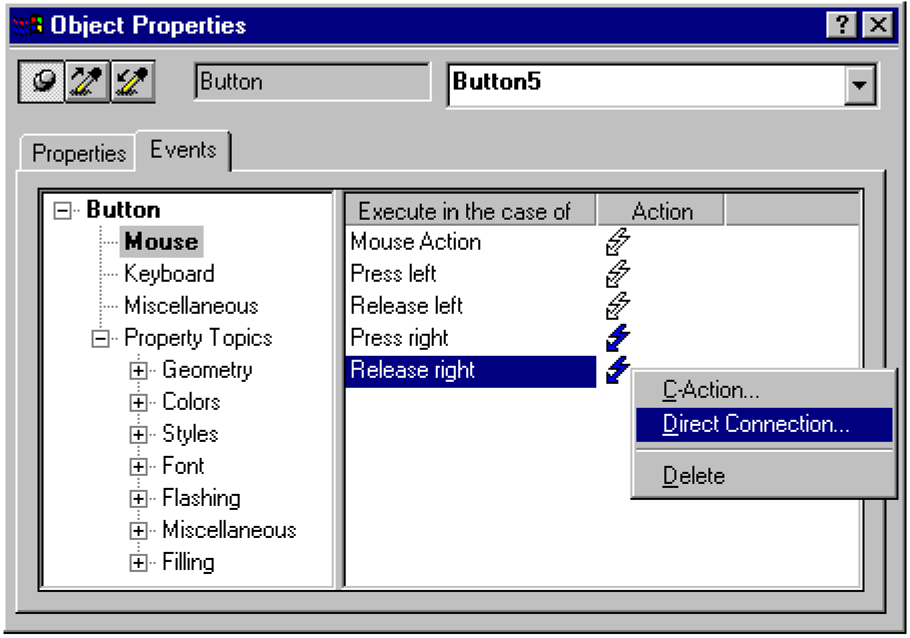
The properties of an object (position, color, font, etc.) can be set static and be changed dynamically during runtime.
All properties with a light bulb in the *Dynamic* column can be dynamized. Once a property has been dynamized, a colored symbol, which depends on the type of dynamization, is displayed instead of the white light bulb. Topics (e.g. Geometry) that have been dynamized are shown in bold type.



| Attribute | Static | Dynamic |
|---|---|---|
| Field Type | Output | |
| Output Value | 0,00000 | S32i_varia_but_00 |
| Data Format | Decimal | |
| Output Format | 999.999 | |
| Apply on Full | No | |
| Apply on Exit | No | |
| Clear on New | Yes | |

### 3.3.2.2 Dynamizing the events

The events of an object (e.g. mouse action, keyboard action, value change, etc.) can be requested during runtime and evaluated dynamically.
All events with a lightning bolt symbol in the *Action* column can be dynamized. Once an event has been dynamized, a colored lightning bolt, which depends on the type of dynamics, is displayed instead of the white lightning bolt. Topics (e.g. Miscellaneous) that have been dynamized are shown in bold type.



### 3.3.2.3 Types of dynamization for objects

The objects of a plant picture can be dynamized in a number of different ways. The separate standard dialog boxes in which dynamization is performed are oriented to different target areas and to some extent lead to different results.

#### Overview

| Type of dynamization | A | B | Advantage | Disadvantage |
|---|---|---|---|---|
| *Dynamic Wizard* | x | x | Standard prompted method when configuring | Only for certain types of dynamization. Always generates a *C action*! |
| *Direct link* |  | x | The **quickest** method of dynamizaing in the picture; **maximum** performance during runtime | Restricted to **one** link and can only be used in a picture. |
| Tag link | x |  | Simple to configure | Limited options for dynamization |
| Dynamic dialog | x |  | Quick and easy to understand and use; for ranges of values or a number of alternatives; high performance during runtime | Cannot be used for all types of dynamization. |
| *C action* | x | x | Almost unlimited possibilities for dynamization thanks to the powerful script language (Ansi-C) | Possibility of errors due to wrong C instructions<br>slower performance compared with other types of dynamization, therefore always check whether or not you can achieve your aim through a different type of dynamization. |

#### Legend:

| | |
|---|---|
| A | Dynamizing the object property |
| B | Dynamizing the object event |

## Opening the dialog boxes for dynamization

| Dialog box | Open command |
|---|---|
| Configuration dialog | Not all objects have such a dialog box.<br>Automatic when creating these objects.<br>*Select object in picture →Press and hold down SHIFT key →* 🖱**D**.<br>*Select object in picture →* 🖱**R** *Open pop-up menu with →Configuration Dialog* |
| Dynamic wizard | *Select object in picture→Select property or event →Select wizard and start with* 🖱**D**. The *dynamic wizard* must have been selected via *View → Toolbars....* |
| Direct link | *Select object in picture →Display object properties →Select Events tab →In the Action column, open pop-up menu with* 🖱**R** *→Select Direct Connection.* |
| Tag link | *Select object in picture →Display object properties →Select Events tab →In the Dynamic column, open pop-up menu with* 🖱**R** *→Select Tag... →In dialog box, select relevant tag and click OK.* |
| Dynamic dialog | *Select object in picture →Display object properties →Select Properties tab →In the Dynamic column, open pop-up menu with* 🖱**R** *→Select Dynamic Dialog. →In dialog box, configure appropriate dynamics and click Apply.* |
| C action | *Select object in picture →Display object properties →Select Properties or Events tab →In the Dynamic or Action column, open pop-up menu with* 🖱**R** *→Select C Action... →In dialog box, configure appropriate C action and click OK* |

## Results and presentation

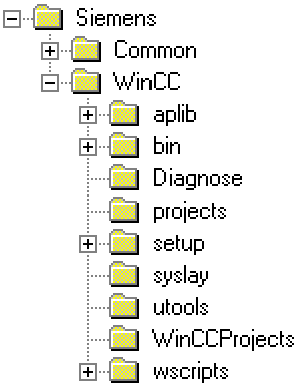
| Dialog box | Result | Presentation (symbol) |
|---|---|---|
| *Dynamic Wizard* | A *C action* is always generated. | *Green lightning bolt* |
| *Direct link* | | *Blue lightning bolt* |
| Tag link | | *Green light bulb* |
| Dynamic dialog | Automatically generated *C action* (InProc); this *C action* can be subsequently expanded but the performance advantage is lost in the process | *Red lightning bolt*<br>Upon change in *C action*; switch to *Green lightning bolt* |
| *C action* | Configured C script | *Green lightning bolt*<br>*Yellow lightning bolt* - the action still has to be created |

## 3.3.3 WinCC system environment

WinCC is installed under the default installation path *C:\Siemens\WinCC\*. You can however change the path during installation.

### 3.3.3.1 Folder structure of the WinCC system

The folder structure of WinCC looks as follows (without options and examples):



```
Siemens
    Common
    WinCC
        aplib
        bin
        Diagnose
        projects
        setup
        syslay
        utools
        WinCCProjects
        wscripts
```

### Files in the default WinCC folder

In the default WinCC path, the following folders and files are important for the persons configuring and commissioning the system:

| Folder | File name, extension | Comment |
|---|---|---|
| Diagnostics | License.log | Current logbook entries regarding the license checks and/or violations. |
| | License.bak | The logbook file with the license information from the last startup. |
| | WinCC_Op_01 .log | Operator messages generated by WinCC during runtime. |
| | WinCC_Sstart_ 01.log | System messages generated by WinCC on startup. An important file when **troubleshooting**. The file contains messages about missing tags and wrongly executed scripts. |
| | WinCC_Sys_0 1.log | System messages generated by WinCC during runtime. An important file when **troubleshooting**. The file contains messages about missing tags and wrongly executed scripts. |
| | S7chn01.log | System message of the channel used (in this case S7) |
| aplib | Library path | The header files, all standard functions and all internal functions are stored in subfolders. |

## Files in the default WinCC folder

In the default WinCC path, the cross-project functions and symbols are stored under the following folders:

| Folder | Subfolder, file name | Comment |
|---|---|---|
| aplib | library.pxl | Symbols of the default library of WinCC. |
|  | Report, Wincc, Windows | Folders for standard functions; they can be adjusted **at any time**. |
|  | Allocate, C_bib, Graphics, Tag | Folders for internal functions; they **cannot** be adjusted. |
| syslay |  | All print layouts that are automatically copied by WinCC to the project path into the folder *prt* when creating a project. |
| wscripts | Dynwiz.cwd | *Dynamic Wizard* of the *Graphics Designer.* You can write your own scripts at any time. These scripts are given the extension **.wnf** |
|  | wscripts.deu | This path contains the script files for *German*. This path depends on the language installed. |
|  | Wscripts.enu | This path contains the script files for *English*. Since English is the default langauge, this path is always created. |
|  | Wscripts.fra | This path contains the script files for *French*. This path depends on the language installed. |

## Files in the default WinCC folder

When you install WinCC, the following application programs are stored in the folders shown below:

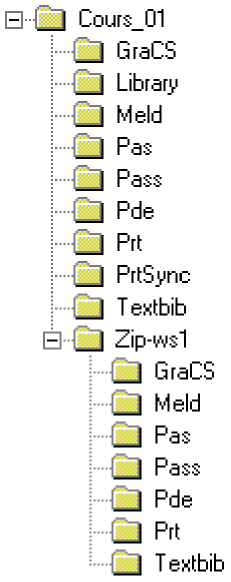
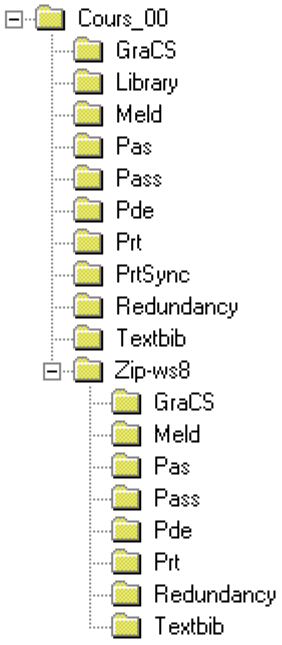
| Folder\file | Comment |
|---|---|
| \sqlany\isql.exe | Interactive program used for looking at the data in the database of a WinCC project. |
| \bin\Wunload.exe | Assistant (wizard) used for emptying the online tables in the database of the WinCC project, e.g. removing the messages and measured value data stored.<br>The wizard automatically sets the runtime tables for unloading; additional tables can however be added or removed from the list by the user at any time.<br>This tool must be used offline with a WinCC project. It cannot be used in runtime mode. Messages and measured values can be swapped during runtime by means of the optional *STORAGE* package. |
| \bin\Wrebuild.exe | Wizard for reconstructing the database; it **cannot** be used in runtime mode! |
| \SmartTools\CC_GraficTools\metaVw.exe | Viewer for graphics files (e.g. print jobs, exported symbols) in EMF format (extended meta file). |
| \SmartTools\CC_GraficTools\wmfdcode.exe | Viewer for graphics files in WMF format (windows meta file). |
| \SmartTools\CC_OCX_REG\ocxreg.exe | For registering or canceling registration of additional OLE Control components (OCX). |
| \SmartTools\CC_OCX_REG\Regsvr32.exe | Is called by ocxreg.exe. |

## 3.3.4 WinCC project environment

**Note:**
Create a special project folder for WinCC projects, e.g. WinCC_Projects. In this way, you keep the WinCC system and the configured data totally separate from each other, simplifying the task of data backup in the process. You also avoid the danger of losing danger (through operator errors) if you need to uninstall WinCC at some time.

### 3.3.4.1 WinCC project folder structure

A project under WinCC consists of a complete folder structure with corresponding content. Once a new project has been created in the *Control Center* (by choosing *File* → *New* from the menu), a folder structure is generated as follows:

| WinCC as standard | WinCC with options |
|---|---|
| <br>Cours_01<br>GraCS<br>Library<br>Meld<br>Pas<br>Pass<br>Pde<br>Prt<br>PrtSync<br>Textbib<br>Zip-ws1<br>GraCS<br>Meld<br>Pas<br>Pass<br>Pde<br>Prt<br>Textbib | <br>Cours_00<br>GraCS<br>Library<br>Meld<br>Pas<br>Pass<br>Pde<br>Prt<br>PrtSync<br>Redundancy<br>Textbib<br>Zip-ws8<br>GraCS<br>Meld<br>Pas<br>Pass<br>Pde<br>Prt<br>Redundancy<br>Textbib |

## Contents of the project folders

| Folder | Extension | Comment |
|---|---|---|
| Project path | .db | The database with the configuration data. |
| | rt.db | The database with the runtime data, measured values and messages |
| | .mcp (master control program) | Main file of the WinCC project. The project is opened with this file. |
| | .pin | Project.pin |
| GraCS | .pdl (picture design language) | The configured pictures. |
| | .sav | Backup files of the picture files with the last configuration state |
| | .bmp (bitmap),<br>.wmf (windows meta file),<br>.emf (extended meta file) | Picture files |
| | .act (action) | Exported *C actions* |
| | .pdd | *Default.pdd* setting parameters for the graphic editor (default settings of the individual objects in the Object Palette) |
| library | .h (header) | *Ap_pbib.h* (project function declarations) |
| | .pxl | *Library.pxl* (project symbol library) |
| | .fct | Project functions |
| | .dll (dynamic link library) | Own function libraries that have been created in a C development environment. |
| Meld | | |
| Pas | .pas (action definition) | Project actions that run as background actions dependent on the trigger set. |
| Pass | | |
| Pde | | |
| Prt | .rpl (report picture language)<br>.rp1 (line layout) | Page layouts for print jobs<br>All the predefined WinCC default layouts begin with @. All system variables (including tags) are identified by this prefix. |
| Computer name e.g. Zip-ws1 | \GraCS\GraCS.ini | Initialization file for the graphic editor |

### Optional: files which can be created during configuration

| Folder | Extension | Comment |
|---|---|---|
| To some extent freely definable | .ini | Initialization file for the simulator with information for the call. |
| | .sim | Internal tags with settings for simulation |
| | .csv | Exported texts from the text library. |
| | .txt | Exported messages from the message system (*Alarm Logging*) |
| | .emf | Print jobs that write your print results to a file. |
| | .log | Log files |
| | .xls<br>.doc<br>.wri | Files that have been created using other applications but are used in the WinCC project. |

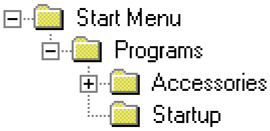
## 3.3.5 Automatic WinCC project startup

### Requirement

The HMI system (WinCC) on the system should start automatically when Windows is started. The HMI station operator does not need to know anything about using the version of Windows loaded (e.g. calling WinCC under Windows 95 or Windows NT).

### Solution

WinCC is started automatically in the startup routine of the PC on startup. This function is set in the *Startup* folder of Windows.



- Start Menu
  - Programs
    - Accessories
    - Startup

### Create a link

| Step | Procedure in NT 4.0: |
|---|---|
| 1 | In Windows Explorer, change to the *WinNT\Profiles\All Users\Startmenu\Programs\Startup* folder. WinNT is the folder in which Windows NT has been installed. |
| 2 | Create a new link in the folder with 🖱**R** → *New* → *Link*. |
| 3 | Establish the link to the program called *mcp.exe* (Master Control Program) in the *\bin* folder of WinCC. |
| 4 | Give the link a name. |

As a result, the WinCC *Control Center* starts automatically. WinCC itself starts automatically with the project that was processed or activated last.
To start a system in runtime mode, the project must therefore be exited while active.

**Note:**
If the key combination *CTRL + SHIFT* is not locked (barred) and it is pressed during startup of WinCC, WinCC starts in configuration mode even if the project was exited in active mode.

The operator now sees the familiar start screen of the system. To prevent operators from being able to inadvertently or intentionally switch to configuration (which is running in the background) or use Windows applications they should or must not use, it is necessary to take appropriate precautionary measures. Operators must also not be able to open the database runtime window, since they can close the WinCC database link from there.

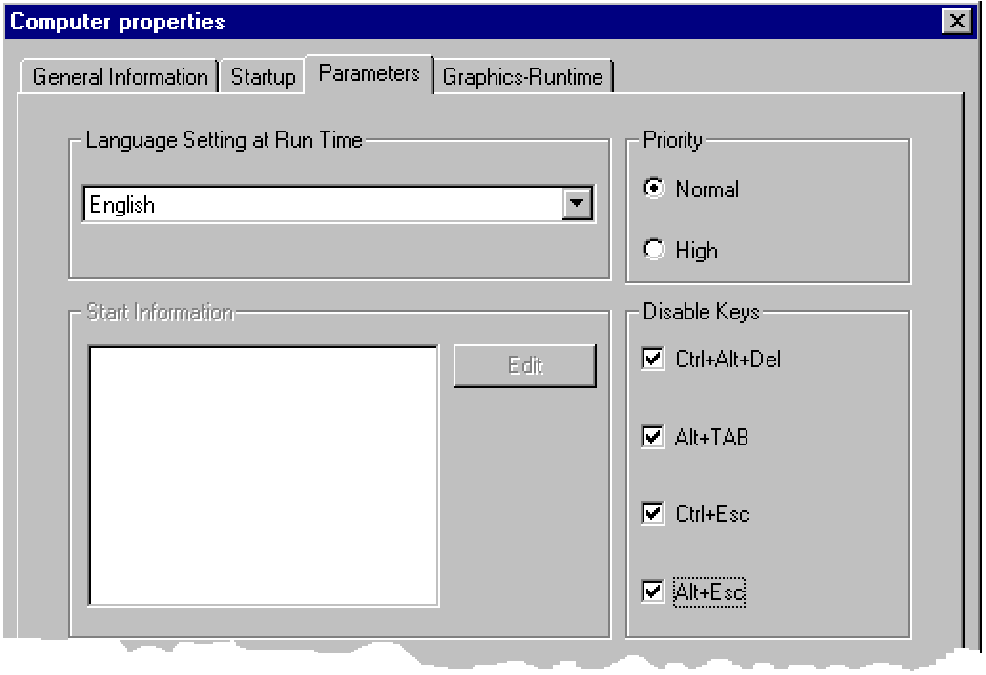
## Barred selection functions:

Operators should have no way of switching from WinCC runtime to:

- the *Control Center* of WinCC (configuration environment).
- the runtime window of the *SQL database* of WinCC (Sybase SQL Anywhere), since they could use this route to close the WinCC database link. WinCC would then no longer be able to continue working.
- the Windows *task bar*, since this can be used to start **all** of the programs installed.
- the current task window, since the application can be closed.

## Settings required for the computer

The following key combinations must be locked so that operators do not have these options open.



Language Setting at Run Time: English

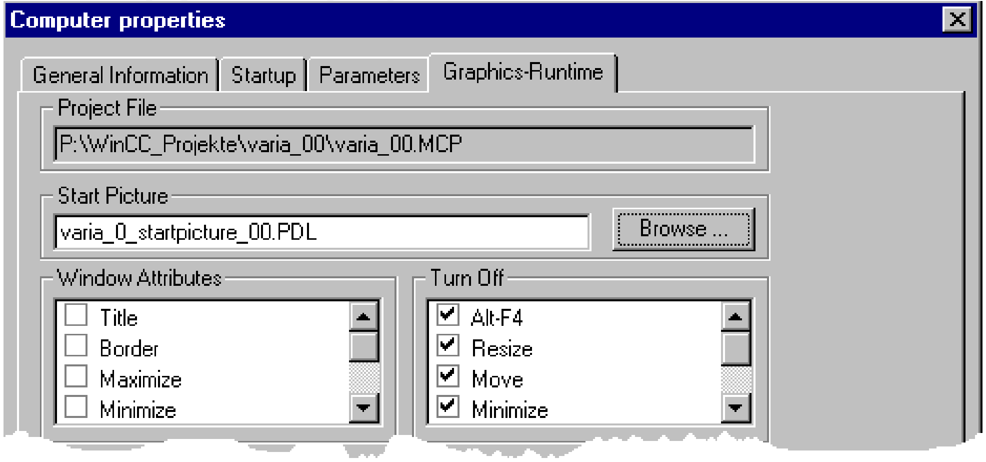
Priority: ● Normal ○ High

Disable Keys: ☑ Ctrl+Alt+Del ☑ Alt+TAB ☑ Ctrl+Esc ☑ Alt+Esc

Locking is performed in the *Control Center* in the *Computer Properties* dialog box. Please refer to the online help of WinCC or the operating system you are using for the exact meaning of the key combinations.

## Settings required for runtime

The plant picture could also be closed by means of the standard Windows keys; i.e. this route could also be used to exit WinCC. To prevent this, the following settings must be made for the properties of the plant picture:



Project File: P:\WinCC_Projekte\varia_00\varia_00.MCP

Start Picture: varia_0_startpicture_00.PDL

Window Attributes

□ Title
□ Border
□ Maximize
□ Minimize

Turn Off

☑ Alt-F4
☑ Resize
☑ Move
☑ Minimize

Locking is performed in the *Control Center* at *Computer Properties*.
Please refer to the online help of WinCC or the operating system you are using for the exact meaning of the key combinations.

- If *Resize* and *Minimize* are not deactivated, access can be gained to the user interface of the operating system.
- *Close* (not shown in the figure above) must also be deactivated. Otherwise users can exit runtime and access the Configuration System.

**Note:**
If all or some of the keys cited above are locked, the person configuring the system and Service personnel must be enabled to access the configuration system by means of defining a particular key configured precisely for this purpose. The same is also the case to enable the system to be shut down properly.

These functions must not be readily operable. You must therefore also **secure access** to this key, e.g. for Service personnel, under the property *Password*.

## 3.3.6 Coordinated exiting of WinCC

### Controlled exit

A WinCC station must never be switched off without shutting down the operating system. The use of an *EMERGENCY SHUTDOWN switch* is not suitable for the HMI system. For this reason, an appropriate button must be configured to enable operators to exit the system correctly without requiring any additional knowledge.

### Power failure

In order to prevent data from being lost in the event of power fluctuations or a complete power failure, a detailed **data backup concept** must be worked out and applied for the HMI system.

At all events, a UPS (uninterruptible power supply) should be included in your plans for the WinCC station. This can be provided by connecting the station to the plant's own UPS or by connecting a separate UPS to the WinCC server. This applies both to single-user and multi-user systems, irrespective of the operating system being used (Windows 95 or Windows NT).
The UPS used must also include special shutdown software for Windows 95 or Windows NT (as appropriate), so that the operating system and all active applications are exited automatically without any loss of data in the event of a power failure and shutdown after a specified time interval; e.g. APC UPS 600 with power shutdown software for Windows 95 and Windows NT.

### Notes on how to install a UPS

The WinCC station must have a serial interface to which a UPS with appropriate test software can be connected. If there isn't a serial interface free on the station (e.g. they are all occupied by a printer or PLC), you must install an additional interface card.
Serial interfaces that are used for connecting two or more peripherals (e.g. via switches) are not supported by the majority of UPS systems and are certainly not recommended, since the system needs to be monitored constantly.
An appropriate monitoring service is installed in the operating system. This monitoring service then has to be assigned shutdown parameters, so that coordinated and orderly exiting of the system is guaranteed. The shutdown process for application software must be activated under all circumstances, so that WinCC is closed without any loss of data in the event of a shutdown. You must choose a *Save time* before shutdown, that is sufficiently long enough for all the applications that are active to be exited properly.

Most software packages for UPS systems offer time-controlled *shutdown*, e.g. for the weekend or for nighttime. This function can be used to obtain **definite exiting of the WinCC system without the operator performing a single control action**.

## 3.3.7 Data backup

### When is data to be backed up?

A WinCC project must be repeatedly saved and backed up at the following times and for the following reasons:

- during the configuration phase.
- before exporting or importing data (e.g. when importing tags, multilingual picture texts, message texts and multilingual message texts).
- before rebuilding or unloading the WinCC database.
- before editing the database using tools such as interactive SQL access.
- the configuration data must be backed up for installation on the destination system at the end user.
- before taking over data for a similarly structured project.

### What media are suitable?

| Medium | Advantage | Disadvantage |
|---|---|---|
| Floppy disks | Can be read by almost any system. | Insufficient capacity (even when data is compressed). |
| ZIP disks | Inexpensive, adequate capacity, direct and quick access possible via Windows; simple to install; portable, for use on the plant. | |
| Streamers (e.g. in the network) | Automatic backup possible (daily), very high capacity. | Mostly available only in office environment, no direct access to the data due to the special format used for saving. |
| Hard disk on another PC (e.g. laplink) | No need to deal with other media, data can be used directly. | Slow, unsuitable for large volumes of data. |
| MOD | High level of data integrity, reusable, backup of messages and measured value possible in runtime mode. | A special drive is required for reading and writing. |
| CD-ROM | High capacity, can be read by almost any system, suitable for long-term archiving. | A special drive is required for writing, medium not reusable. |

## Reducing the project before data backup

In order to be able to back up the project in a state that is as **reduced** as possible before actually backing up data for the end user or for taking over project data, the following data can be deleted or reduced by means of auxiliary programs beforehand.

- All backup files in the *\GraCS*.sav* folder of the project.
- If you haven't created any layouts of your own (Report Designer) for the documentation, the system layouts can be deleted from the *\Prt* folder. When a new project is created, all system layouts are automatically copied from the *\Siemens\WinCC\syslay* folder to the project folder.

## What data must be backed up?

If only the data of a WinCC project are to be backed up, the following individual files and folders with all the files contained in them must be backed up
**from the project folder:**

- the files **.mcp, *.pin and *.db*
- the folders *\GraCS* and *\Library*
- if you have created your own actions, the folder *\Pas*
- if you have created your own print layouts, the folder *\Prt.*

If you have created cross-project components (standard functions, objects in the project library), the following files must also be backed up from
**the default WinCC folder:**

- the files *\Siemens\WinCC\aplib*.fct*
- the file *\Siemens\WinCC\aplib\library.pxl*

This data is **not** generated when WinCC is reinstalled.

## 3.3.8 Copying a backed-up WinCC project to a new destination computer

### Installing the system software

The WinCC software is installed on integral systems either by means of the Configuration dialog which is called automatically or by means of the Installation CD supplied with the WinCC package and the licensed disks that belong to it.

### Supplements

If special supplementary packages (e.g. options packages or add ons) or special communication interfaces or interfaces to other Windows programs (e.g. WORD, EXCEL etc.) are used in your project, these packages must also be installed on the new destination computer. The relevant authorizations for options packages, add ons or communication interfaces (channel DLLs) must also be installed on the destination computer. Note that **all** the necessary authorizations (for all channel DLLs used) must be imported on the new computer to enable you to work with the WinCC project.

### Windows software

If OLE links to other Windows programs, e.g. to WORD, ClipArts or EXCEL, are used in the WinCC pictures, the corresponding software package, depending on the type of the OLE link, must likewise be installed on the destination computer, i.e. entered in the Windows registration.

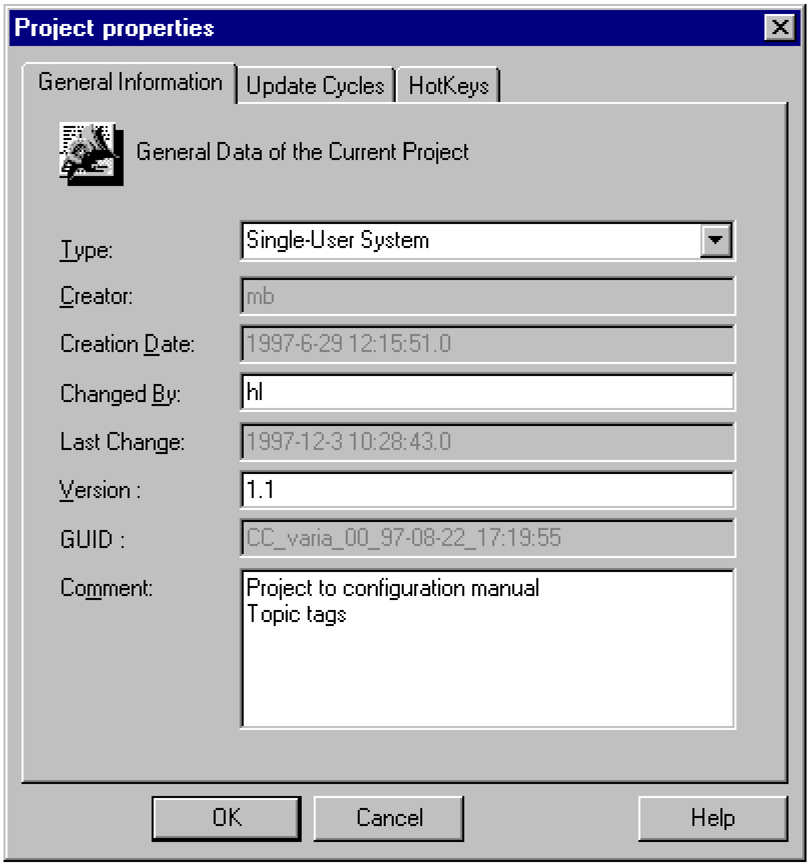
### OXC, ActiveX

If other OCX components (OLE Control, ActiveX) from bought-in software packages are used, they must also be stored in the Windows registration. Register or check the registration entries of these OCX components, e.g. using the tool *SmartTools\CC_OCX_REG\ocxreg.exe* supplied with the WinCC CD. If a registration to OLE objects or to OCX objects is not found on startup of the WinCC project in runtime mode (as well as during configuration using the *Graphics Designer*), this object is displayed in the picture with the remark *Unknown Object*.

### Network

If the project has been configured for a **multi-user system**, the network must be fully installed on the WinCC destination computer before the WinCC data is copied. Make a note of the necessary computer names in the configured computer environment, since you will need them when assigning the parameters of the project copied. The computer name is also required as a parameter for a single-user system. Therefore, you must know the name of the destination computer or find its name by means of the Windows Control Panel. If the computer in question is a single-user system that is not connected to a network, the Windows computer name will be set.

### Copying the data and starting the project

| Step | Procedure when taking over data |
|---|---|
| 1 | Create a project folder (e.g. *WinCC_Projects*) |
| 2 | Copy the complete backed-up path to this project folder as a subfolder (e.g.: *\WinCC_Projects\Varia_00*. The name of the WinCC project folder can be changed if desired. When renaming the files in the project folder (varia_00.mcp, varia_00.db, varia_00.pin, varia_00.log), make sure that all the files are given the same name (with the exception of the extension). |
| 3 | Open the project in the WinCC *Control Center*. |
| 4 | If necessary, change the project-specific settings. If the type has to be changed, additional adjustments must also be made to the *Computer properties*. These adjustments are described in the WinCC Help.<br><br> Project properties<br>General Information<br>Update Cycles<br>HotKeys<br>OK<br>Cancel<br>Help<br>General Data of the Current Project<br>Type: Single-User System<br>Creator: mb<br>Creation Date: 1997-6-29 12:15:51.0<br>Changed By: hl<br>Last Change: 1997-12-3 10:28:43.0<br>Version : 1.1<br>GUID : CC_varia_00_97-08-22_17:19:55<br>Comment: Project to configuration manual<br>Topic tags |
| 5 | Check the *computer name* and modify it, if necessary, in the *Computer Properties* dialog box on the *General Information* tab. If the computer name in the WinCC project does not match up with the computer name of this destination system, an error message is displayed when you activate the project, i.e. when you start it up in runtime mode. |
| 6 | If you have used your own *standard functions* (*.fct) in the project, the backed-up standard functions must be copied to the default WinCC path *\Siemens\WinCC\aplib* and then made known in the function tree.<br>• With the project open, call the *Global Script* editor.<br>• Recreate the declaration structure by choosing *Options →Regenerate Header* |

| Step | Procedure when taking over data |
|---|---|
| | from the menu.<br>The new functions can now be seen in the function tree. |
| 7 | Activate the project in order to check that it starts up correctly. |

### 3.3.9 Reuse - reusing parts of a project in a new or existing project

#### The reason for reusing data

A WinCC project can be generated in different ways. The chief aspects in this regard are the reusability of existing project parts from similar projects and the taking over of data from preconfigured sample projects.

#### Project team

Similar tasks are specified in this regard for a configuration team, since a WinCC project must in the end be merged again to form one project.
A WinCC project consists of individual files (e.g. pictures) **and** the configuration data in the database (Alarm Logging, Tag Management).

#### Data in the database

Data that is stored in the configuration database **cannot** be created in two separate projects and subsequently merged together.
For this reason, a **basic project**, which is used for this type of configuration, must be created when configuring database information (e.g. the structure of Alarm Logging). This basic project must be **backed up** before **every** change to the database (for intermediate steps, too). If something goes wrong when making a change, you can retrieve the status that prevailed before the change was made.

**Note:**
Note that every change made in the database may influence the structure and accessing of the database. Lots of unnecessary changes (possibly with deletions) can lead to the WinCC database no longer being optimally compiled and consequently to a loss of performance.

#### Single-user system

While the next configuration step, e.g. on Alarm Logging, is being carried out on the basic project, you must never under any circumstances make a change in the database at some other point (e.g. archiving – *Tag Logging*) in a WinCC *single-use system*.

#### Multi-user system

In contrast, a project is being configured on a *multi-user system*, changes can be made to the configuration in different areas of the database simultaneously. For example, one person configuring the system can edit Alarm Logging while another system configurer edits the archiving system (measured-value acquisition).

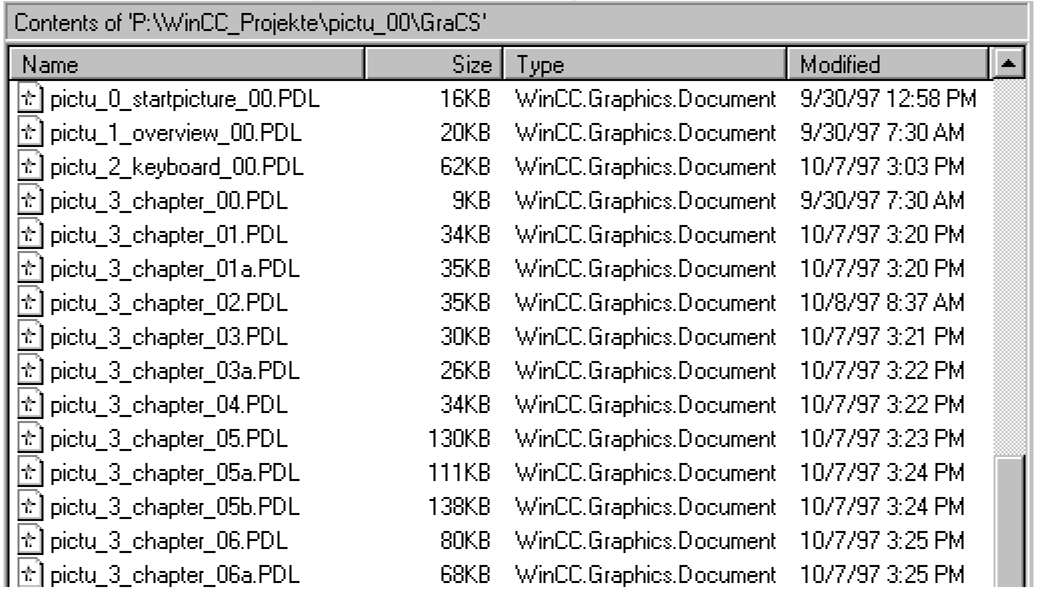
#### Conversion from single-user to multi-user system

Every project can be created on a WinCC *multi-user configuration system* and converted back to a single-user system on the destination computer. If a change of configuration between the configuration station(s) and the customer's system is planned beforehand, no elements specific to WinCC, that are oriented to a multi-user system, may be used. For

example, no internal tags can be used on the local computer if the system being configured is actually for a single-user system at the customer.
When creating a new project (irrespective of whether it is for a single-user or multi-user system), you must know beforehand whether or not the project is to be based on an existing basic project with preconfigured Alarm Logging and/or archiving data. This data stored in the database can only be transferred to other projects through reconfiguring or, if possible, through export - import.

### 3.3.9.1 Taking over pictures

Configured pictures can be taken over at any time. You can copy the picture files (_*.pdl_ ) from the source folder to the target folder of the WinCC project path, _\GraCS_, in two different ways. The first method is to do it directly using the Windows Explorer (useful if you have to copy several picture files).
Below is an extract from the project for picture configuration:



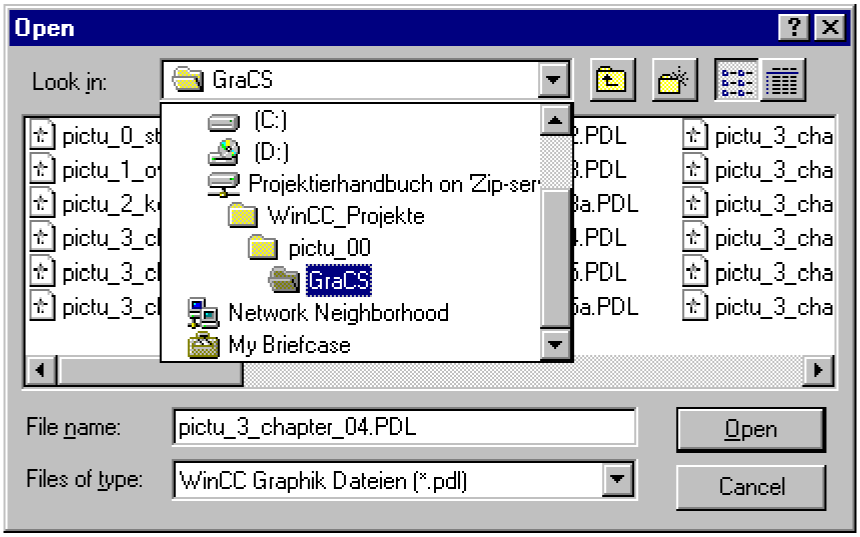
Contents of 'P:\WinCC_Projekte\pictu_00\GraCS'

| Name | Size | Type | Modified |
|---|---|---|---|
| pictu_0_startpicture_00.PDL | 16KB | WinCC.Graphics.Document | 9/30/97 12:58 PM |
| pictu_1_overview_00.PDL | 20KB | WinCC.Graphics.Document | 9/30/97 7:30 AM |
| pictu_2_keyboard_00.PDL | 62KB | WinCC.Graphics.Document | 10/7/97 3:03 PM |
| pictu_3_chapter_00.PDL | 9KB | WinCC.Graphics.Document | 9/30/97 7:30 AM |
| pictu_3_chapter_01.PDL | 34KB | WinCC.Graphics.Document | 10/7/97 3:20 PM |
| pictu_3_chapter_01a.PDL | 35KB | WinCC.Graphics.Document | 10/7/97 3:20 PM |
| pictu_3_chapter_02.PDL | 35KB | WinCC.Graphics.Document | 10/8/97 8:37 AM |
| pictu_3_chapter_03.PDL | 30KB | WinCC.Graphics.Document | 10/7/97 3:21 PM |
| pictu_3_chapter_03a.PDL | 26KB | WinCC.Graphics.Document | 10/7/97 3:22 PM |
| pictu_3_chapter_04.PDL | 34KB | WinCC.Graphics.Document | 10/7/97 3:22 PM |
| pictu_3_chapter_05.PDL | 130KB | WinCC.Graphics.Document | 10/7/97 3:23 PM |
| pictu_3_chapter_05a.PDL | 111KB | WinCC.Graphics.Document | 10/7/97 3:24 PM |
| pictu_3_chapter_05b.PDL | 138KB | WinCC.Graphics.Document | 10/7/97 3:24 PM |
| pictu_3_chapter_06.PDL | 80KB | WinCC.Graphics.Document | 10/7/97 3:25 PM |
| pictu_3_chapter_06a.PDL | 68KB | WinCC.Graphics.Document | 10/7/97 3:25 PM |

The second type of picture application functions by opening a picture file (_bild.pdl_) in _Graphics Designer_ by choosing the option _File → File Open._ The picture is then stored in the current picture folder (_\GraCS_) by choosing _File → Save As_ from the menu. This procedure is suitable if the picture files are being used as a basis and will be customized immediately.

### Restriction

With trend windows, message windows, table windows, application works, albeit to a limited extent.



Look in: GraCS

File name: pictu_3_chapter_04.PDL

Files of type: WinCC Graphik Dateien (*.pdl)

**Note:**
The references relating to objects from the previous project no longer match up. The relevant references must be defined in the new project.

These references can be:

### References in pictures

- Structures from the data type area
- Internal or external tags
- System:tags
- Message window
- Archive window templates (trend window or table window for process-value or user archives)
- Picture objects that are stored as a bit map or meta file (e.g. for status displays or graphic objects)
- Other pictures that are used as graphic or process boxes or as pop-up windows
- Project functions used
- Access authorization

The following references must also be defined:

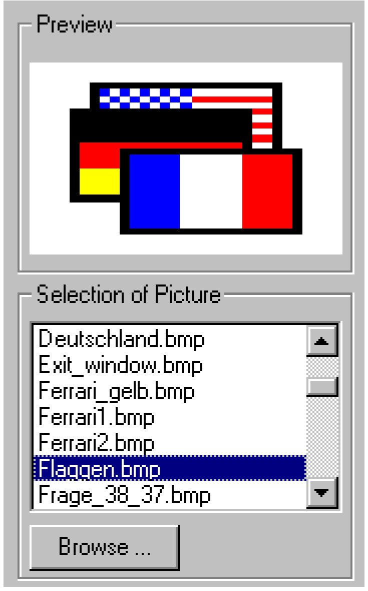
- Definition of the structures under data types, e.g. controller or templates structures for user objects
- Definition of the communication channels and logical links with the definition of the tags (possibly with tag groups).
- Definition of the internal tags and system tag names (beginning with @).

- Definition of new or similar window templates and linking of the application windows (for *Alarm Logging* and *Tag Logging*).
- Transfer of the picture elements (*.bmp and/or *.emf) by copying them from the *\GraCS* folder.
- Transfer of picture window contents by copying the other picture files (*.pdl) from the *\GraCS* folder.
- The project functions used must be copied from the source project to the *\library* folder of the new project. In addition, these functions have to be stored in the function tree using the *Global Script* editor by choosing *Generate Header*. This approach has already been described in greater detail in Section *4.1 The development environment for scripts in WinCC*.
- Access rights used (e.g. Open Picture) must be specified in the *User Administrator* editor. The access rights used (e.g. for control buttons) must be defined for the group specifications.

### 3.3.9.2 Taking over symbols and bit maps

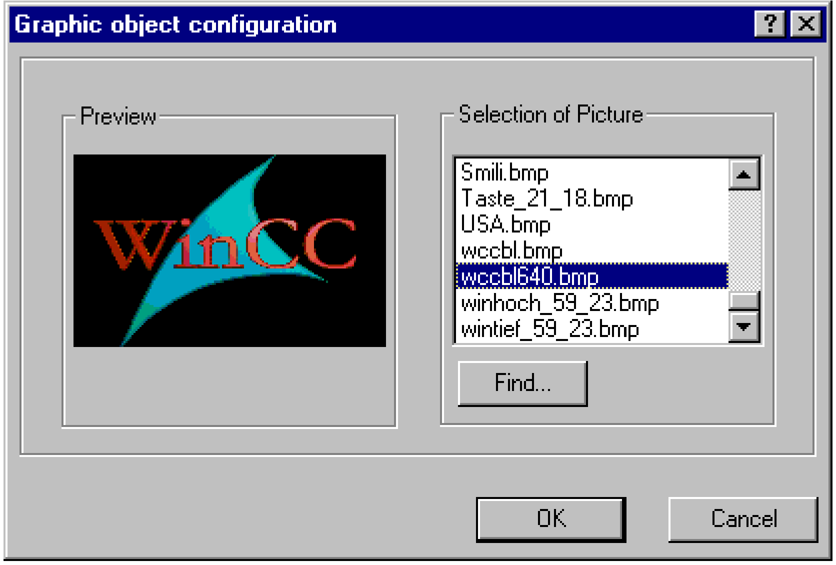
## Copying

Symbols for *status displays* or *graphic objects* in picture files are stored as separate files in the picture folder of the project. This is done by copying the desired symbol files (*.emf or *.bmp) to the target folder, *\GraCS*, of the new project. These pictures are now immediately available in the selection list for *status displays* or *graphic objects* (see the Object Palette in the *Graphics Designer*). Here is a section of the configuration dialog of the *status display*.



Deutschland.bmp
Exit_window.bmp
Ferrari_gelb.bmp
Ferrari1.bmp
Ferrari2.bmp
Flaggen.bmp
Frage_38_37.bmp

Picture browser for the *graphic object*:



Smili.bmp
Taste_21_18.bmp
USA.bmp
wccbl.bmp
wccbl640.bmp
winhoch_59_23.bmp
wintief_59_23.bmp

## Importing

Symbols can be embedded in a picture using the method described above or they can copied directly into the graphic you are currently editing by choosing *Insert* → *Import*. In the latter case, you don't have to copy a file; you import the symbol you want directly by accessing the path of the source project (*\GraCS*) and then importing the symbol file you want (*.bmp, *.emf, *.wmf). After being imported, the symbol is then immediately displayed as an object in the picture (top left).

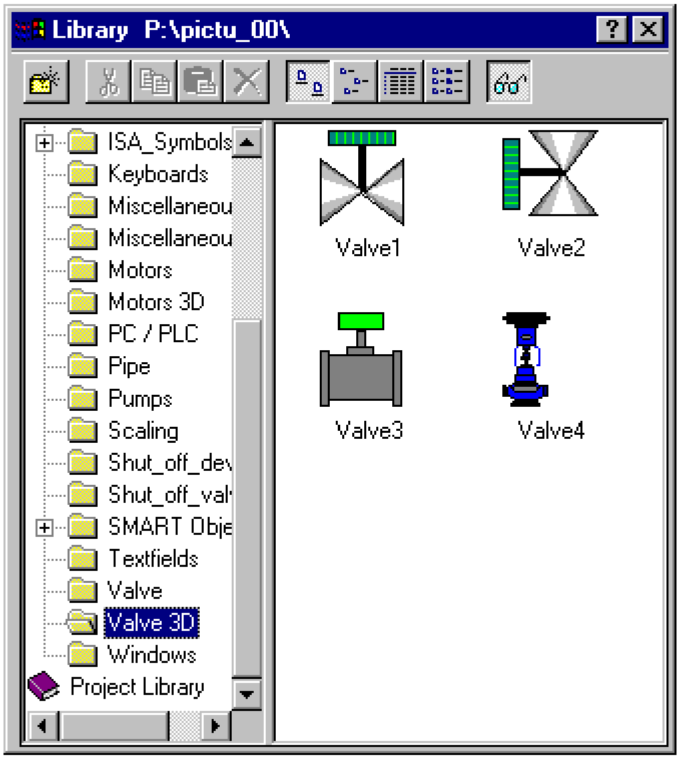
If symbols are stored in a project library, the project library can be used in another project by being taken over in its entirety. How this is done is described in the following chapter.

### 3.3.9.3 Taking over a project library (with preconfigured symbols and user objects)

## Global library

If symbols are stored in a project library, this library can be used in another project by copying the file *library.pxl* to the *\library* path.

The preconfigured modules can now continue to be used at any time in the new project.

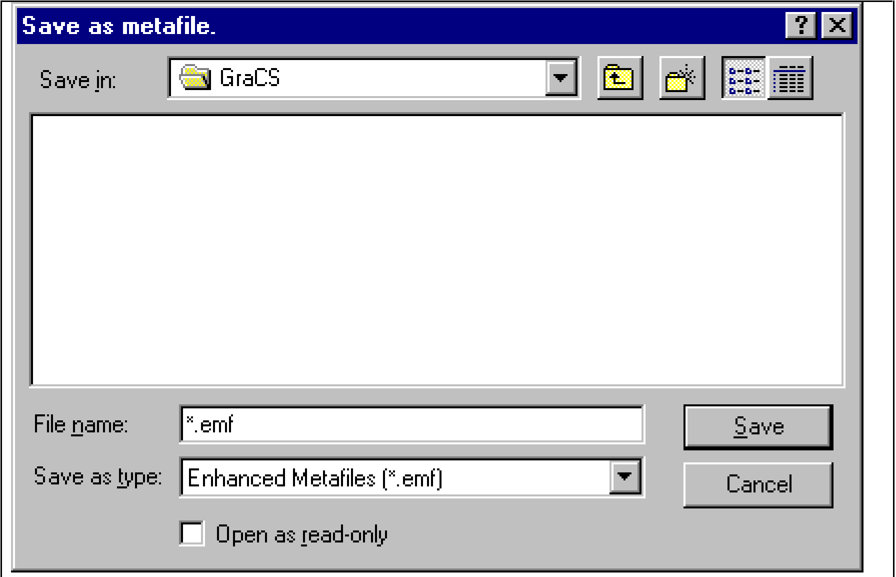


**Note:**
Please take into account the fact that linked symbols may refer you to unavailable references (or tags) that you must define first. Depending on the configuration of these symbols, the actions or links associated may have to be adjusted. For this reason, after using symbols from the library, check which properties/event links are already present and whether these may have to be adjusted.

## Individual symbols

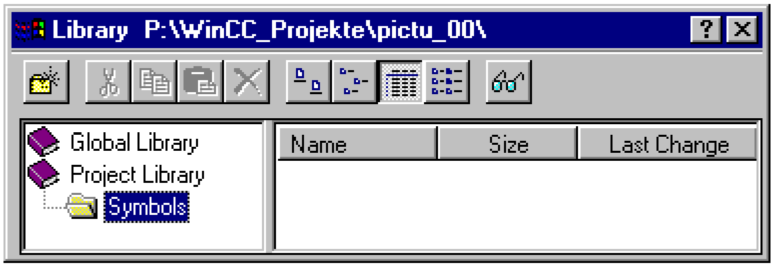
If only a few specific symbols from the project library are used in the new project, these symbols are exported individually (symbol file *.emf).

| Step | Procedure: taking over symbols |
|---|---|
| 1 | Open library. |
| 2 | Select the symbol you want using the 🖱 and press and hold down the mouse key to drag the symbol to the picture (Drag&Drop). |
| 3 | Call the dialog box for saving the symbol by selecting *File →Export...* . |

| Step | Procedure: taking over symbols |
|---|---|
| | <br>Save as metafile.<br>Save in: GraCS<br>File name: *.emf<br>Save as type: Enhanced Metafiles (*.emf)<br>☐ Open as read-only<br> |
| 4 | Save the symbol. |

## New project library

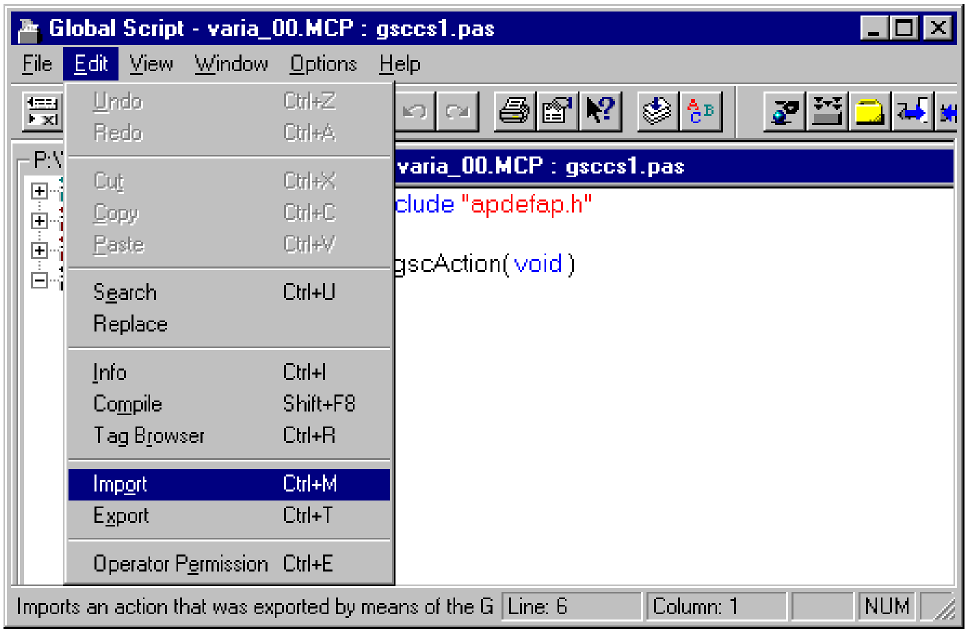
These exported symbols are now available as separate symbol files and can be used individually by being imported. If these symbols are used frequently in the project, they should be integrated again into the new project library. You do this by calling the symbol library, in particular the project library.
Create your own symbol folder, e.g. with the aid of the folder icon in the toolbar of the library window and copy the imported symbols to this folder by means of *Drag&Drop*. In this way, you can take over some of your symbols from projects and add additional specific symbols to create a new project-specific library again.



Library P:\WinCC_Projekte\pictu_00\

## 3.3.9.4 Taking over global actions

Actions that are frequently required in a project or that are to be copied from one object action to another object action are stored as separate files. These files are stored in the *\GraCS* folder with the extension *.act*. They can be taken over at any time by being copied from the source folder to the target folder.
An action file is saved using the *C Actions* editor by means of the *Export Action* button on the toolbar to the destination file you name (having the extension *.act* for action).



```
clude "apdefap.h"

gscAction( void )
```

Imports an action that was exported by means of the G

A stored action file is taken over for an object action in the picture of the new project by means of the *Import Action* button in the toolbar. Read the description carefully in Section *4.1 The development environment for scripts in WinCC*.

**Note:**
Actions used frequently can also be defined as project or standard functions.

### 3.3.9.5 Taking over tags

The tag management of WinCC can be supplemented in a number of different ways:

- By reading in S5 data tags or S7 data tags using the Assistant (*Dynamic Wizard*)
- By taking over *S7 tags* using the *PCS7 mapper*
- By importing and exporting text lists using the auxiliary tool known as *Var_Exim*
- By interactive accessing of the database tables (tag tables)
- By means of specially programmed assistants (*Dynamic Wizard*) or programs that create new data in tag management with WinCC API functions

The last two options cited require very good knowledge of working with SQL databases and of programming via the application interface. They should only be used by persons who have such knowledge.

Before the data is transferred to the destination project, you therefore have to clarify where the basis for the destination project is. If there is already a large number of tags present in the tag management of WinCC, the tag list of WinCC should be imported into the destination project. The *internal tags* must at all events be taken over from the *tag management* of WinCC. This is done using the tool *Var_Exim.exe*.

## Taking over S5-/S7 data tags using Dynamic Wizards

The data area definitions generated using STEP5/STEP7 software can be read into the tag management of WinCC with the aid of *Dynamic Wizards*. The following activities have to be performed:

| Step | Procedure: taking over S5 or S7 data |
|---|---|
| 1 | **Back up the data** of the project. Changes are made in the database. |
| 2 | Export the assignment list using the STEP software. A file with the name *prj_zuli.SEQ* is created. |
| 3 | Remove any special symbols (e.g. for program calls) that are not required for import into WinCC from this exported file. You can do this with a typical text editor, such as Wordpad. The assignment list must not contain any blank lines. |
| 4 | Open the destination project in the WinCC *Control Center*. The project must be in configuration mode (Runtime not activated!). |
| 5 | Open the *Graphics Designer* editor. In any picture in the *Dynamic Wizard* (display by choosing *View → Toolbars...*), select the *Import Functions* tab. Then select the *Import S7 – S5 Assignment List* function. In the following you have to name the *source file (.seq)* with its folder (by means of the button ). You also have to specify the logical connection into which the tag descriptions of the assignment list have to be mounted.<br>The data is now entered into the WinCC tag management. The tag names used in a WinCC project must be unique. The tags are added to the existing WinCC tag management. The tag name is the key to this. |

## Taking over tags using the auxiliary program

The links (channel DLL, logical link and link parameters) must already be defined before being imported into the destination project.

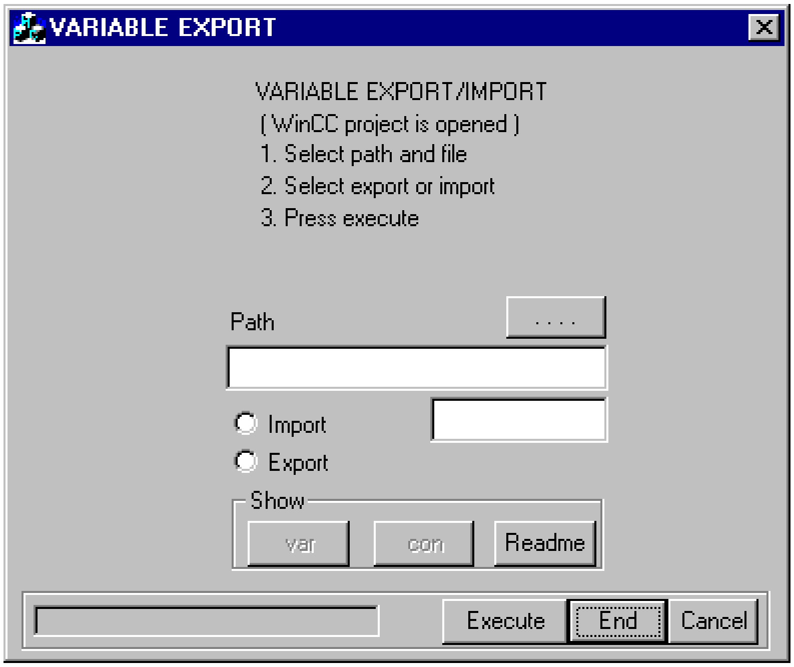
**Note:**

To achieve automatic generation of links and data entries in the WinCC database, you could compile a special program which carries out such definitions automatically via the WinCC API interface. In this way, existing plant data can be added to automatically. This program must be written by a specialist in WinCC API programming or SQL programming.

The tags defined in the tag management can be **exported** at any time as a text file to supplement the list of tags. Afterwards, this data generated must be **imported** back into the tag management of the project. The generated files are stored in CSV (comma separated values) format and can be read and processed further in any editing program.
For this there is a special application on the WinCC CD-ROM in the folder called *\SmartTools\CC_VariablenImportExport*. This Windows program is provided as an auxiliary program:

- for transferring (exporting) the data of the tag management to external storage
- for importing tag data already generated externally
- for bulk data configuration

The following steps now have to be taken to export or import the data:

| Step | Procedure: importing / exporting tags |
|---|---|
| 1 | Open your WinCC project in the WinCC *Control Center*. |
| 2 | Define links (channel DLL – logical link – link parameters) that are not currently available but that will later be required for importing. Only do this, however, in the new project. This may required a second export-import procedure. |
| 3 | Activate the program called *Var_exim* by means of 🖱**D**. The user interface of this program is displayed in the following.<br><br>VARIABLE EXPORT/IMPORT<br>( WinCC project is opened )<br>1. Select path and file<br>2. Select export or import<br>3. Press execute<br> VARIABLE EXPORT<br>Path<br>. . . .<br>○ Import<br>○ Export<br>Show<br>var<br>con<br>Readme<br>Execute<br>End<br>Cancel |

## Import - Export

The following settings are necessary for exporting or importing the data:

| Location, action | Import | Export |
|---|---|---|
| *Path* | Select the project folder which contains the files for tag import by selecting the file named <*project_name*>*.mcp*. The files which contain the data to be imported must be located in the same folder as the project file. | Enter the project folder for tag export by selecting the file named <*project_name*>*.mcp*. |
| Action | Select *Import*.<br>If you want existing tag data to be overwritten, select *Import overwrite*. | Select *Export*. |

| Location, action | Import | Export |
|---|---|---|
| *Execute* | Click *Execute.*<br>The dialog box then displayed shows the parameters set and executes the procedure after you have clicked *OK*.<br>Due to the checks performed, import takes more time. | Click Execute.<br>The dialog box then displayed shows the parameters set and executes the procedure after you have clicked *OK*. |
| Status display | End export/import | End export/import |
| Tag file<br>*Name_vex.csv* | Basis for import:<br>consists of a header and data records. | The tag list generated is stored in this file as text. You can open it directly by means of the *var* button or edit it in a text editor (Notepad) or in EXCEL. |
| Tag file<br>*Name_cex.csv* | Basis for import:<br>consists of a header and data records (structural components). | This file contains the currently configured links which are referred to in the tag file. You can open it directly by means of the *con* button or edit it in a text editor (Notepad) or in EXCEL. |
| Data structure file<br>*Name_dex.csv* | Basis for import:<br>consists of a header and data records. | If tags with data structure types are included, this file containing structural information is also generated. You can edit its contents using a text editor (Notepad) or EXCEL. |
| Diagnostics file<br>*Diag.txt* | A diagnostics file containing information as to which tags were not able to be imported. | |

The program can be exited by clicking the *End* button.

## List of tags

The structure of the tag lists is explained in the table below.

| Field | Type | Description |
|---|---|---|
| Varname | char | Tag name |
| Conn | char | Connection |
| Group | char | Group name |
| Spec | char | Internal tag or address (matching type of link) |
| Flag | DWORD | |
| Common | DWORD | |
| Ctyp | DWORD | Tag type<br>1 BIT<br>2 SBYTE<br>3 BYTE<br>4 SWORD<br>5 WORD<br>6 SDWORD |

| Field | Type | Description |
|---|---|---|
| | | 7 DWORD<br>8 FLOAT<br>9 DOUBLE<br>10 TEXT_8<br>11 TEXT_16<br>12 Raw data type<br>13 Field<br>14 Structure<br>15 BITFELD_8<br>16 BITFELD_16<br>17 BITFELD_32<br>18 Text reference |
| CLen | DWORD | Length of the tags |
| CPro | DWORD | Internal or external tag |
| CFor | DWORD | Format conversion |
| Protocol | | |
| P1 | BOOL | Error at upper limit |
| P2 | BOOL | Error at lower limit |
| P3 | BOOL | Conversion error |
| P4 | BOOL | Write error |
| P5 | BOOL | |
| P6 | BOOL | |
| L1 | BOOL | Substitute value on upper limit error |
| L2 | BOOL | Substitute value on lower limit error |
| L3 | BOOL | Start value |
| L4 | BOOL | Substitute value on link error |
| L5 | BOOL | Upper limit valid |
| L6 | BOOL | Lower limit valid |
| L7 | BOOL | Start value valid |
| L8 | BOOL | Substitute value valid |
| LF1 | double | Upper limit |
| LF2 | double | Lower limit |
| LF3 | double | Start value |
| LF4 | double | Substitute value |
| Scaling | | |
| SCF | DWORD | 1 when scaling is defined |
| SPU | double | Range of values, process from |
| SPO | double | Range of values, process up to |
| SVU | double | Range of values, tag from |
| SVO | double | Range of values, tag up to |

## Connection list

| Field | Type | Description |
|---|---|---|
| Conname | char | Logical tag name |
| Unit | char | Channel Unit |
| Common | char | General |
| Specific | char | Specific link parameter |
| Flag | DWORD | |

## Data structure list

| Field | Type | Description |
|---|---|---|
| Datastructure | short | Data structure name or component name |
| Type ID | short | Identification (is used in tag list under Ctyp) |
| Creator ID | short | |

In order to be able to continue processing the text lists in EXCEL (Version 7.0 or 8.0), you must open the exported files of file type *Textfiles [*.prn; *.txt; *.csv]*.

## Specifications

Customization of the tag data in the text list is performed under the following special specifications:

| Type | In text list | In WinCC |
|---|---|---|
| Connections | General description | If not available, logical links **must** be redefined! |
| | Special description of channel DLL | If not available, channel DLLs **must** be redefined! |
| Tag group | No group information<br>If groups are defined in the project, which do not contain any tags, these groups are also not exported. | The group information is automatically generated when the first tag of a group. |
| Tags | General description | |
| | Specific description<br>Respective channel DLL or internal tag | Channel DLL or internal tag |
| | On export, the missing sections are replaced by *. | Tags whose specific description is missing are not imported! |
| Tags of Data structure type | Assignment corresponding to the data structure definition of the structure list. | Is assigned to the data type. |
| Data structure definition | | Must be defined by the person configuring the system. |
| Limits | Are not exported or imported via the text list. | Must be defined by the person configuring the system. |

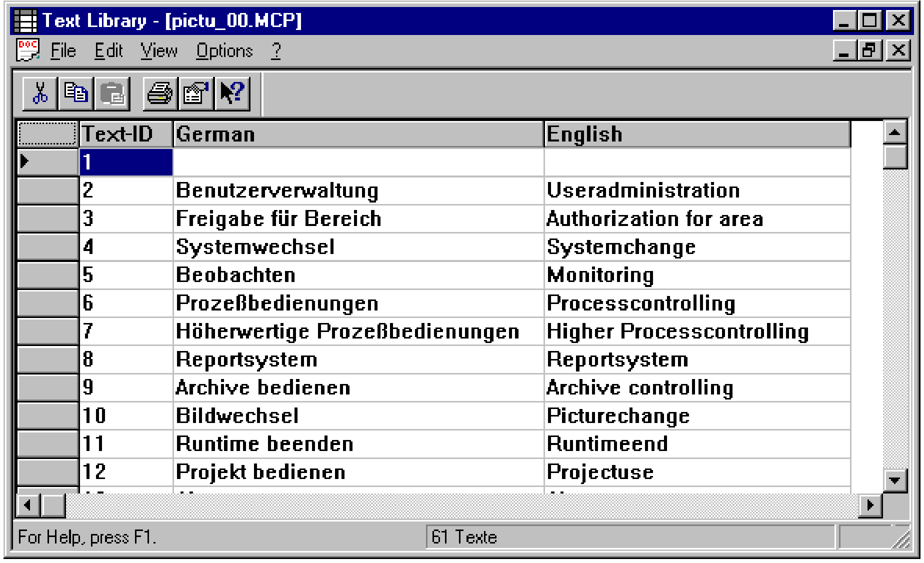
**Before** you import the modified or new tags, **backup the data** of your project. Changes are made in the database. These database changes cannot be undone in WinCC.

## 3.3.9.6 Taking over multilingual texts (from pictures, in messages)

### Multilingual texts of the pictures

The multilingual texts stored in the pictures can be transferred to the new project by copying the picture itself.

The respective language must be added to the text library in the new project. Check the settings in the Text Library. Each language must have its own column!



| Text-ID | German | English |
|---|---|---|
| 1 | | |
| 2 | Benutzerverwaltung | Useradministration |
| 3 | Freigabe für Bereich | Authorization for area |
| 4 | Systemwechsel | Systemchange |
| 5 | Beobachten | Monitoring |
| 6 | Prozeßbedienungen | Processcontrolling |
| 7 | Höherwertige Prozeßbedienungen | Higher Processcontrolling |
| 8 | Reportsystem | Reportsystem |
| 9 | Archive bedienen | Archive controlling |
| 10 | Bildwechsel | Picturechange |
| 11 | Runtime beenden | Runtimeend |
| 12 | Projekt bedienen | Projectuse |

For Help, press F1.
61 Texte

Should only some of the languages be taken over in the new project?
Since the entire text information per picture is stored, reconfiguring in the respective language must be carried per picture. You should consider whether or not it makes more sense to delete the texts already configured. Switching over to runtime is made possible only by means of specially configured keys and could consequently be restricted for the project. If you nevertheless want to delete the texts for a language that has already been included, we recommend you use the export and import function via the auxiliary tool known as *language*.

### Taking over pictures with text references

If text references are used in the pictures taken over, the following data must also be taken over:

- the associated tags (export or redefinition) from the tag management of the WinCC project
- the texts from the text library
- The text reference tags must be provided with the valid text identification numbers (text IDs). Check whether the text IDs still match up with the associated texts.

### Taking over texts from the text library

If only some of the texts from the text library are taken over, you must adjust the text IDs accordingly. Texts from the text library can be taken over by means of the export/import mechanism in the *Text Library* editor.

## 3.3.9.7 Taking over messages

The information basis for messages (alarms) requires

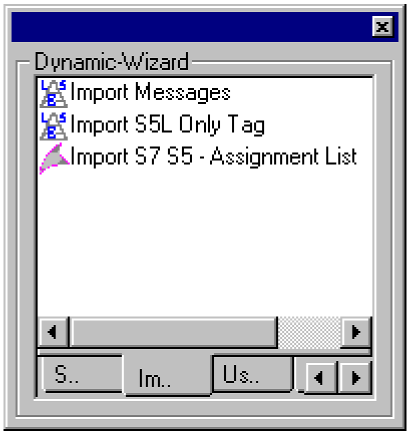
- modification and
- a great deal of configuration work due to the volume (bulk data).

For this reason, the option of taking over message data from previous projects is used very frequently. The following methods of taking over messages can be used depending on the source of the message data:

- taking over already configured message information from predecessor systems (e.g. COROS)
- importing (single) messages from an existing WinCC project
- importing message information from the concept phase

## Taking over messages from COROS

The procedure for the sources quoted above looks as follows:

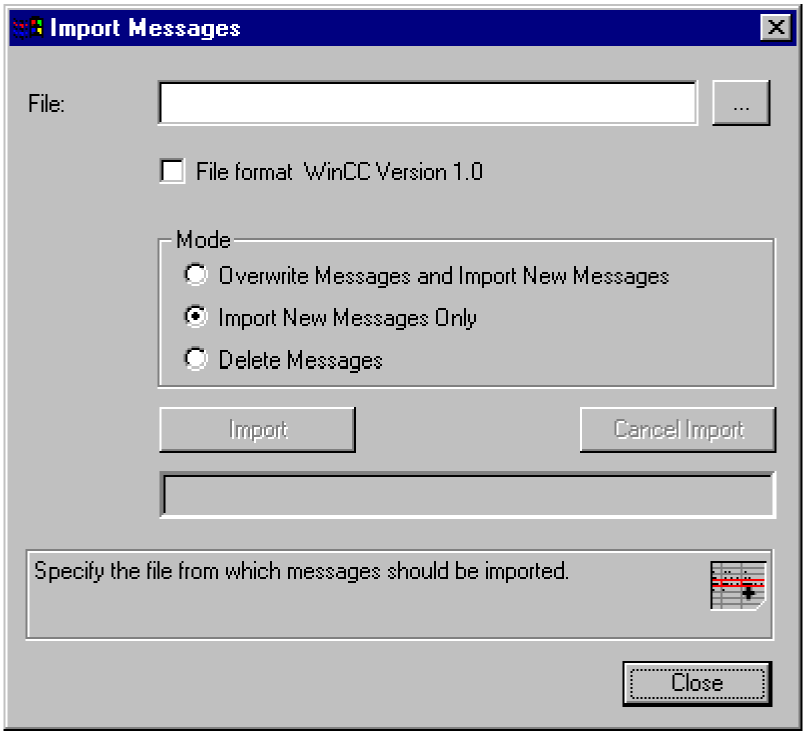
| Step | Procedure: taking over existing COROS message texts |
|---|---|
| 1 | In COROS, export message information (*mldtexte.txt*) |
| 2 | In WinCC, read in this message file using the *Dynamic Wizard Import Functions → Import Messages*. The data is now imported into the current WinCC project.<br><br> Dynamic-Wizard<br>Import Messages<br>Import S5L Only Tag<br>Import S7 S5 - Assignment List<br>S..<br>Im..<br>Us.. |

## Taking over messages from a WinCC project

If messages are taken over from an existing project, you must first clarify whether or not the destination message system (Alarm Logging) has been set up in the database structure in the same way. The differences can be seen, for example, in the user and process value blocks. If possible, organize the destination data blocks according to sequence (as well as according to the length of the text elements) straight away. Otherwise you will have to made adjustments in the individual columns before importing.

| Step | Procedure: taking over existing WinCC message texts |
|---|---|
| 1 | Open the *Alarm Logging* editor in the current project. |
| 2 | Call the message export procedure by choosing *Messages →Export Single Messages* from the menu. |
| 3 | Specify the destination text file in which the message information to be exported is to be stored.<br>Under *Selection...*, select the messages to be exported; use the criteria, e.g. Number, Message class. |
| 4 | Start the export procedure by clicking *Export*. A text file containing items of message information separated by commas is now created. |
| 5 | Close the current project and open the new one. Open the *Alarm Logging* editor again and define the necessary message classes and message types. Define one message per message type, in order to retain the basic framework for the following import. |
| 6 | To export these basic messages, select *Messages → Export Single Messages...*. Repeat steps 3 and 4. |
| 7 | Now open, e.g. in EXCEL, the message file of the source project and the message file of the destination project. The columns are separated by commas.<br>Compare the structure of the message blocks and, if necessary, make adjustments by reorganizing or renaming columns. |

| Step | Procedure: taking over existing WinCC message texts |
|---|---|
| | Enter the index 0 in each of the blocks with the text IDs. This means that the texts are automatically organized in the text library when imported. Under no circumstances may the old ID numbers be retained!<br>The modified file must once again be saved as a text file. |
| 8 | Now call the import procedure by selecting *Messages →Import Single Messages....* |
| 9 | Now specify the source text file with the exported message information. You now have to decide whether existing messages are to be overwritten during importing. The messages are assigned by means of the respective message number, which must be uniquely defined in the project. |
| 10 | The messages are then imported and supplement your existing message system (Alarm Logging) with the message information already configured. Check the assignments imported. |


Import Messages

File:

□ File format WinCC Version 1.0

Mode
- ○ Overwrite Messages and Import New Messages
- ● Import New Messages Only
- ○ Delete Messages

Specify the file from which messages should be imported.

**Note:**
If message data is taken over from a WinCC V 1.10 project, you must pay attention to the column headers in the message text file!

### Importing message information from the concept phase by means of EXCEL tables

The message information is already available in an EXCEL table. These messages can be taken over by incorporating the columns into the message structure of the WinCC project. This must be done by building on a WinCC message file. This file is created in the following manner:

| Step | Procedure: creating a message structure |
|---|---|
| 1 | Open the new project in the WinCC *Control Center*. Open the *Alarm Logging* editor and define the necessary message blocks, message classes and message types. Define one message per message type, in order to retain the basic framework for the following import. |
| 2 | To export these basic messages, select *Messages → Export Single Messages...*. |
| 3 | Specify the destination text file in which the message information to be exported is to be stored. |
| 4 | Start the export procedure by clicking *Export*. A text file containing items of message information separated by commas is now created. |
| 5 | In EXCEL, open the message file and the freshly created message file of the destination project. The columns are separated by commas.<br>In the table, create a copied line for the corresponding message class/message type. Take over the message texts etc. from the source data and enter them in the associated blocks. For example: *Block 1 → Message text*.<br>**Number** all of the message lines, e.g. starting from 1. This can be done very quickly in EXCEL with the aid of the numbering in the message number column. |
| 6 | Enter the index 0 in each of the blocks with the text IDs. This means that the texts are automatically organized in the text library when imported. Under no circumstances may the old ID numbers be retained!<br>The modified file must once again be saved as a text file. |
| 7 | In the Alarm Logging editor, call the import procedure by selecting *Messages → Import Single Messages...*. |
| 8 | Now specify the source text file with the exported message information. You now define the parameters in such a way that existing messages are overwritten. The messages are assigned by means of the respective message number, which must be unique in the project. |
| 9 | The messages are then imported and supplement your existing message system (Alarm Logging) with the message information already configured. Check the assignments imported. |

## 3.3.9.8 Taking over measured values

Since the specifications for measurement points and the definitions of the process value archives and user archives together with their properties are integrated directly into the database structure, it is not possible to take over measured values (without direct accessing of the database, which requires a sound knowledge of the database). This means that either these archives and measurement points must be reconfigured or the data is taken over automatically at the start of configuration by copying an entire basic project.

### 3.3.9.9 Taking over print layouts

Copy the desired print layouts (**.rpl* for page layouts or **.rp1* for a line layout) from the source folder to the *\PRT* folder of the new project.

### 3.3.9.10 Taking over global actions

Copy the desired global actions or background actions (**.pas*) from the source folder to the *\Pas* folder of the new project.

### 3.3.9.11 Taking over project functions

**.fct*) from the source folder to the *\library* folder of the new project. To make these functions known to the project, you have to activate the option *Options→Generate Header* in the *Global Scripts* editor. You will find a detailed description of how to do this in Section *4.1 The development environment for scripts in WinCC*.

### 3.3.9.12 Using standard functions

Contrary to *project functions*, *standard functions* don't have to be copied. These functions are immediately available for the project, since they are known to all WinCC projects on the station.

### 3.3.9.13 Taking over user administration

Since the specifications for user groups, users and access rights are integrated directly into the database structure, it is not possible to take over user administration. This means that a reconfiguration is necessary.
There is one way round this: the data is taken over automatically at the start of configuration by copying an entire basic project.

## 3.3.10 Working without a mouse

Plant pictures are in many cases controlled under WinCC by means of the mouse. The mouse click or mouse action is the event among the respective types of dynamization, which is used most frequently and in the greatest number of different variants (left or right mouse button, press or release). There are, however, systems which are controlled using both a mouse and a keyboard or even just a keyboard. Operator panels, for example, are exclusively controlled by means of a keyboard.

### 3.3.10.1 Working with a keyboard

A keyboard offers the following input options:

- Function keys F1 through F12
- Special function keys (e.g. Operator panel function keys SF10)
- Standard keyboard inputs
- Moving around by means of input fields or control buttons, by means of arrow keys or special keys

Configuration of the control actions without a mouse must looked at separately for the following areas of configuration:

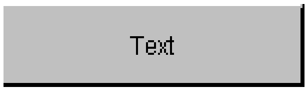
- Control buttons in the plant picture (e.g. for changing pictures)
    - by means of function keys
    - by means of special keys
    - by means of standard keys
- "Any key" control
- Moving around by means of control objects
- Input fields in the plant picture
    - input/output fields
    - special input objects (check box, ...)

- Alarm Logging (message windows)
    - control actions by means of function keys
    - control actions by means of specially configured keys

- Tag Logging (trend or table windows)
    - control actions by means of function keys
    - control actions by means of specially configured keys
- Starting a print job by means of a key
- Logging on or off by means of the keyboard

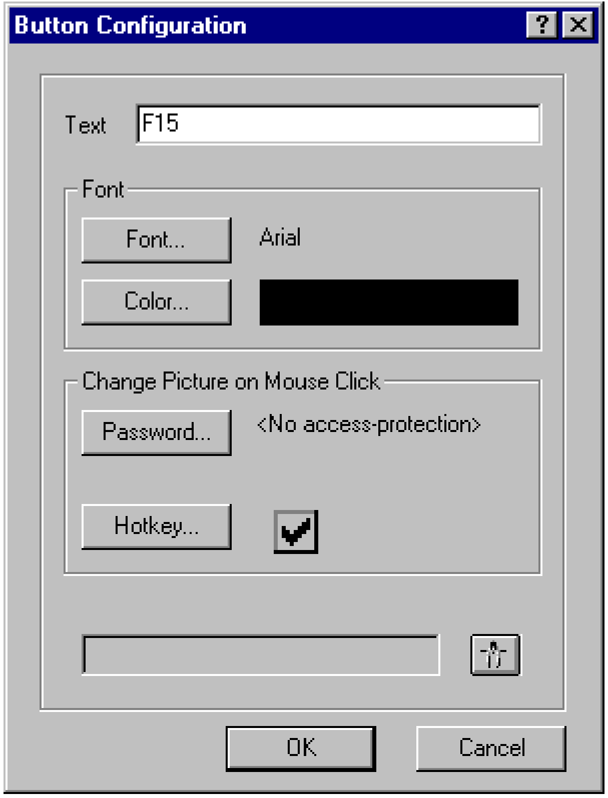
Control buttons are configured in the typical *Windows style*. This is why you will find the standard Windows control button in the Object Palette. You can add other graphical elements to this button at any time.

## Control buttons



## Control actions by means of function keys

Function keys F1 through F12 on the standard keyboard are frequently used as (additional) keyboard action for control buttons for changing pictures in the plant picture hierarchy. These function keys can be assigned at any time to the configured Windows buttons as hot keys. A hot key offers you the quickest way of initiating the function assigned to it.


Button Configuration

Text: F15

Font: Arial

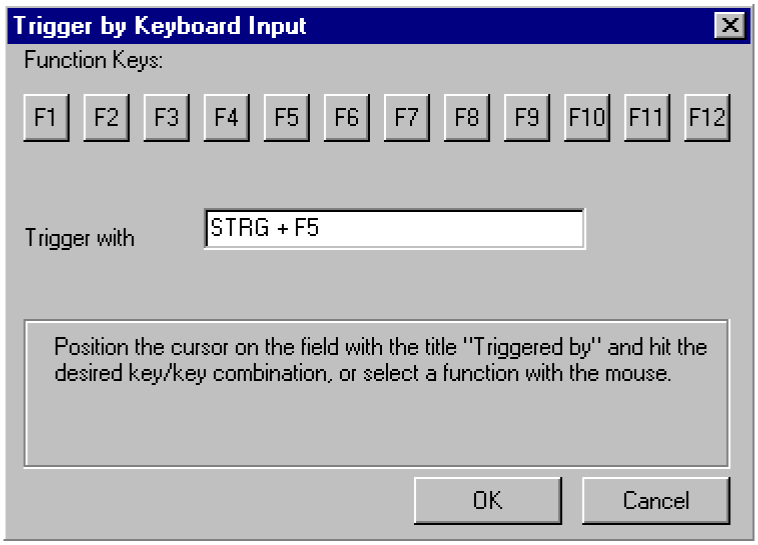
Change Picture on Mouse Click

<No access-protection>

Hotkey: ☑

Hot keys can be assigned, for example, to the functions keys mentioned. These keys are already displayed in the configuration dialog box as selection buttons.

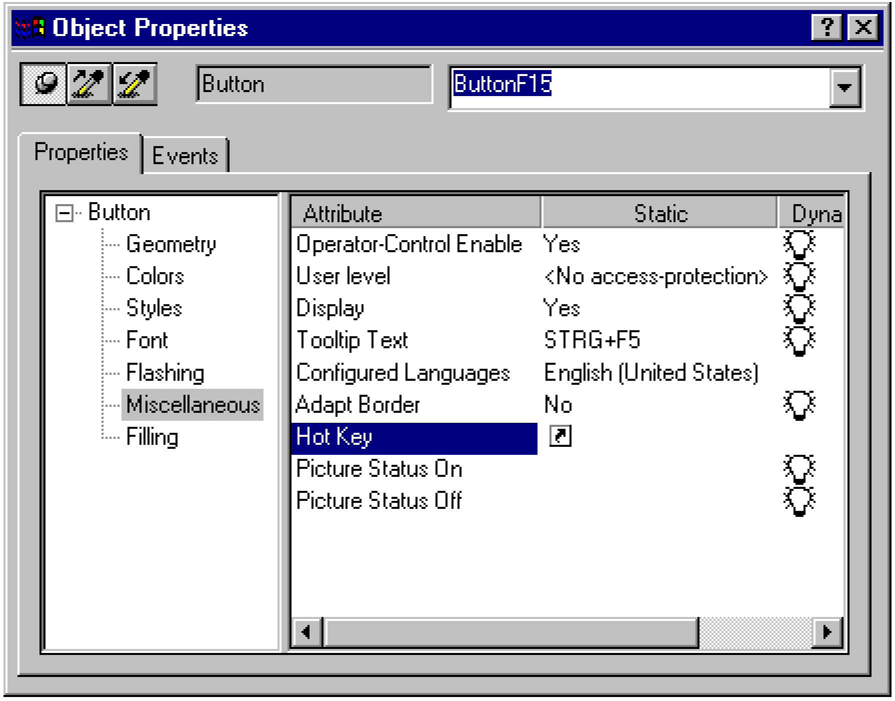
If you need to combine a key with, for example, the SHIFT key or CTRL key, then simply enter the desired key sequence (e.g. SHIFT+F2) directly into the input field by pressing the respective keys. There is no need to enter any special codes.
The key combination you have selected is displayed in the input field.



Trigger with: STRG + F5

Position the cursor on the field with the title "Triggered by" and hit the desired key/key combination, or select a function with the mouse.

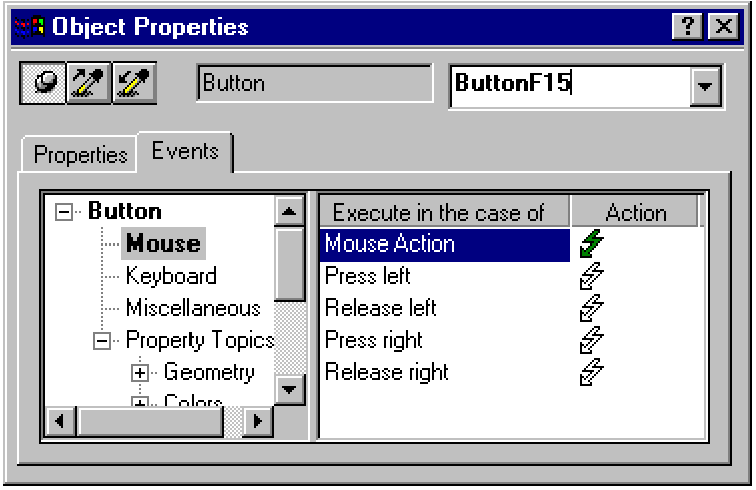
This key selection is stored in the properties of the object and can be modified either by means of the configuration dialog box or directly by means of the property *Miscellaneous → Hot key*.



| Attribute | Static | Dyna |
|---|---|---|
| Operator-Control Enable | Yes | 💡 |
| User level | <No access-protection> | 💡 |
| Display | Yes | 💡 |
| Tooltip Text | STRG+F5 | 💡 |
| Configured Languages | English (United States) | |
| Adapt Border | No | 💡 |
| Hot Key | ↗ | |
| Picture Status On | | 💡 |
| Picture Status Off | | 💡 |

## Action for hot key

The action which is to be triggered by entering the function key (hot key definition) must be stored under the events of the Windows button, the event under which it must be stored being the **Mouse Action**. The event is triggered when the mouse button is released, but only when the mouse pointer is positioned over the object when pressed and released. If there is no action stored under the *mouse action (click)* event, but e.g. only under the (similar) event known as *Press left* (i.e. the left mouse button), the action will **not** be triggered by the function key! When configuring, please also note that the function key can only be used once in the picture.



## Special function keys

If you use special function keys, e.g. the operator panel buttons F13, S1 etc., for controlling the pictures, these keys must be reassigned to key combinations. The F13 key could, for example, be reassigned to the combination SHIFT+F1. In addition to the usage for the key combinations chosen in the visualization mechanism as described above, the combinations are also defined specific to particular devices. You will find special keyboard settings for this purpose, which depend on the respective devices used. For example, a file named *FI25.key* is provided for the settings of the key codes for an industry PC. These device-specific files are used for storing the codes for the function keys. After adjusting the device keyboard definition - per function key with the respective hexadecimal code - and activating the new keyboard codes, these keys in the plant pictures can then be used for working in the pictures.

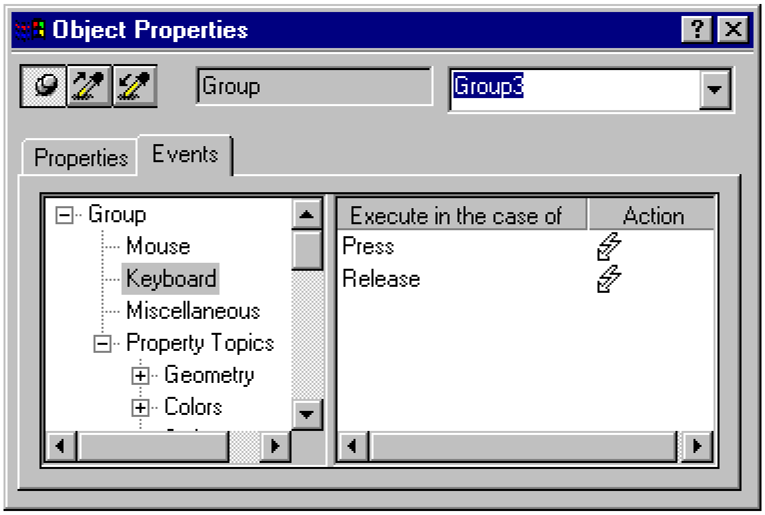
## Standard keys

If the action is not assigned to a function key input but to a key on the standard keypad, e.g. the letter *m*, this key is stored as a hot key under the *Windows button* object.

## "Any key" control

The design of the control buttons for the plant pictures can itself be created. You can find other control buttons, e.g. in the user library under *3D buttons* or *user objects*.
Objects you have designed yourself that are not based on the Windows button cannot be configured as hot keys. All other objects must be configured for button control by means of the *Button* event of the object. The following keyboard events are available per object:

- Press
- Release



This button event must be available to enable configuration of keyboard control actions. When using predefined buttons from the user library, you must therefore first check whether or not this button is suitable for working with only a keyboard and without a mouse. The shift buttons of the user objects, for example, are not always enabled for keyboard control actions. This means you may have to make adjustments.

Preconfigured button (key) actions (e.g. scrolling through the picture hierarchy) can be found in the optional packages (e.g. *Basic Process Control* - *Picture Tree Manager* etc.).

If one of these objects is used as a control element, the button (key) which triggers the action is configured to respond to either the *Keyboard - Press* or *Keyboard - Release* event. Either a *direct link* or a *C action* can be configured as the action.
The triggering button event is either

- an "any key" action or
- a selected key on the standard keypad.

If the event is an "any key" event, a *direct link* can be used. In contrast, if a specific key input has to be checked, a *C action* must be used. The *C action* checks the key code entered, character for character, before continuing the actual sequence of the action:

```
if (nChar==S)
    OpenPicture("Start.pdl");
```

If key combinations are queried, further checks must take place after the first keyboard code is queried. To this end, the special function „GetAsyncKeyState“ must be used in the following way:

```
#include "apdefap.h"
void OnKeyDown(char* lpszPictureName, char* lpszObjectName, char*
                 lpszPropertyName, UINT nChar, UINT nRepCnt, UINT
                 nFlags)
{

#pragma code("user32.dll");
int GetAsyncKeyState(int vKey);
#pragma code();

int nRet;
//Query on F1
if (nChar==112) {
   //If F1, query if SHIFT operates
   nRet = GetAsyncKeyState(VK_SHIFT);
   //Bit set for key pressed
   if (nRet & 0x8000) {
        // Here you can edit further
        // instructions that are initiated for the
        // corresponding key combination
}
   //If F1, query whether CTRL operates
   nRet = GetAsyncKeyState(VK_CONTROL);
   //Bit set for key pressed
   if (nRet & 0x8000) {
        // Here you can edit further
        // instructions that are initiated for the
        // corresponding key combination
        }
   //If F1, query if ALTGR operates
   if (nFlags & KF_ALTDOWN) {
        // Here you can edit further
        // instructions that are initiated for the
        // corresponding key combination
        }
   //If F1, query if ALT operates
   nRet = GetAsyncKeyState(VK_MENU);
   //Bit set for key pressed
   if (nRet & 0x8000) {
        // Here you can edit further
        // instructions that are initiated for the
        // corresponding key combination
        }
   }// End of keyboard query
}
```

### 3.3.10.2 Moving around by means of control objects (input fields and control fields)

The mouse can be used to click directly onto every controllable object. Whether or not objects can be controlled is indicated by the mouse pointer changing. How can these objects be controlled without a mouse?

### Alpha cursor / Tab order

You can move between the controllable objects in runtime mode by means of the movement keys. A distinction is made between:

- alpha cursor objects (I/O objects) and
- tab order objects.

Input/output objects are selected by means of the alpha cursor (the Tab key or SHIFT+Tab key combination).
All control elements (whether controllable by mouse, keyboard or both) can be integrated into control by means of the tab order. The I/O fields can be integrated into control by means of both the alpha cursor and the tab order.

## TAB sequence

Using the so-called *TAB order* (it can be set by choosing *Edit →TAB Order →Alpha Cursor or →Tab orders* from the menu) you can influence the order in which operator-controllable objects are addressed in RUNTIME mode. The object currently selected can be visualized in Runtime. This is the runtime cursor, which can also be turned off (*Computer properties → Graphics-Runtime).* Windows-style buttons are always shown with a dashed quadrangle on the button.

Moving around between the controllable elements depends on the settings for the Graphics Runtime (*Computer properties → Graphics-Runtime).*

| Movement | Standard keys | Key settings |
|---|---|---|
| Up, Down<br>left, right | Cursor (arrow) keys or<br>Tab (next) or SHIFT+Tab (previous) | Other key settings by means of *Computer properties →Graphics-Runtime →Cursor control: keys* |
| Alpha cursor / Tab order | Tab order | Switching between alpha cursor and tab order by means of hot keys (*Computer properties → Graphics-Runtime →Hot keys)* or self-defined keys (using the internal function *SetCursorMode*) |
| Movement in tables (cursor group) | Normal, i.e. line-by-line editing<br>When the cursor has reached the end of the cursor group, it remains in this position. | Changing the characteristics (behavior) by means of *Computer properties → Graphics-Runtime →Cursor control: characteristics* |

## Input/output fields

Configured input/output fields can be written in directly after being selected via the keyboard, i.e. you can enter your new data immediately. A pure output field (*Properties → Input/Output → Field Type→Output)* cannot be changed.
Input is confirmed as a function of the configured property (*Properties → Output/Input →Apply on Exit*) by means of ENTERIn contrast, the ESCAPE key (ESC) exits input without saving the changes (if any) made.

## Other input objects

In addition to the typical analog input fields, Windows applications offer other input options. These particular objects can be found in the Object Palette under *Windows Objects*

- Check Box
- Option Group

The individual selection fields of the check box or option group are set by means of the spacebar, moving across the individual components in the box or group performed by means of the Up/Down keys (e.g. arrow keys). This is the key assignment already set by default.
Another input object is the **text list object**. You can make a selection by means of an opened list that is dependent on the entries configured:

This object can likewise be controlled by means of the standard keyboard. There is no need to store a special configuration for the keyboard control action.

The list is opened by pressing ENTER, moving around in the list performed by means of the Up/Down keys and confirmation of the current selection by pressing ENTER.

Further input objects could be used by means of the OCX elements in WinCC. Control and configuration of these objects depends, however, on the events and properties available which have been defined specific to the object. This must be clarified in each individual case.

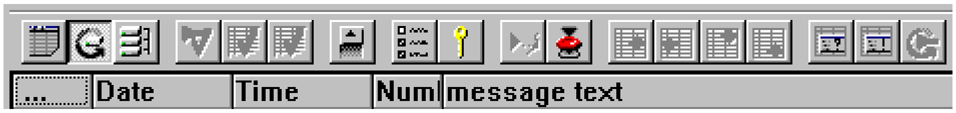
### 3.3.10.3 Alarm Logging function keys for the toolbar buttons

In the message windows, different control buttons are set in the toolbar, which are controlled by means of the mouse (standard).

The most frequent control actions in a message window are

- selecting a message for acknowledgement
- moving up/down through the message list
- scrolling through the message list

**Note:**
When opening the message window or through a further control action, control must lie in the message window (application window) and not in the main window. Depending on the current controllability, the key actions (or function keys) act on the function key bar of the main window or on the stored key actions of the message window.



This can be achieved, for example, by setting the current focus of control in this section of the window. The focus of control is normally set by clicking with the mouse.
Using the keyboard, the focus can be set in message windows by means of the following configurable routes:

- changing the window by means of a hot key
- setting the focus by means of a control button or
- setting the focus directly to a defined element in the message window on opening the picture.

Changing (switching) to the message window by means of a quick control action (hot key) which can be used in the same way for all window changes, i.e. in trend windows too, is defined in the startup parameters of the Graphics Runtime. The key combination (e.g. CTRL+W) is entered under *Computer properties→Graphics-Runtime →Hot keys → Window On Top*.

Once the message window has been opened, the buttons in the toolbar can be activated directly by means of this key sequence.

Direct setting of the focus of control in the message window is, however, achieved by means of the internal function *Set_Focus*. A *C action* on making a key action or on Open Picture (*Picture Object →Events →Miscellaneous→ Open Picture*) can therefore influence activation of control of the message window.

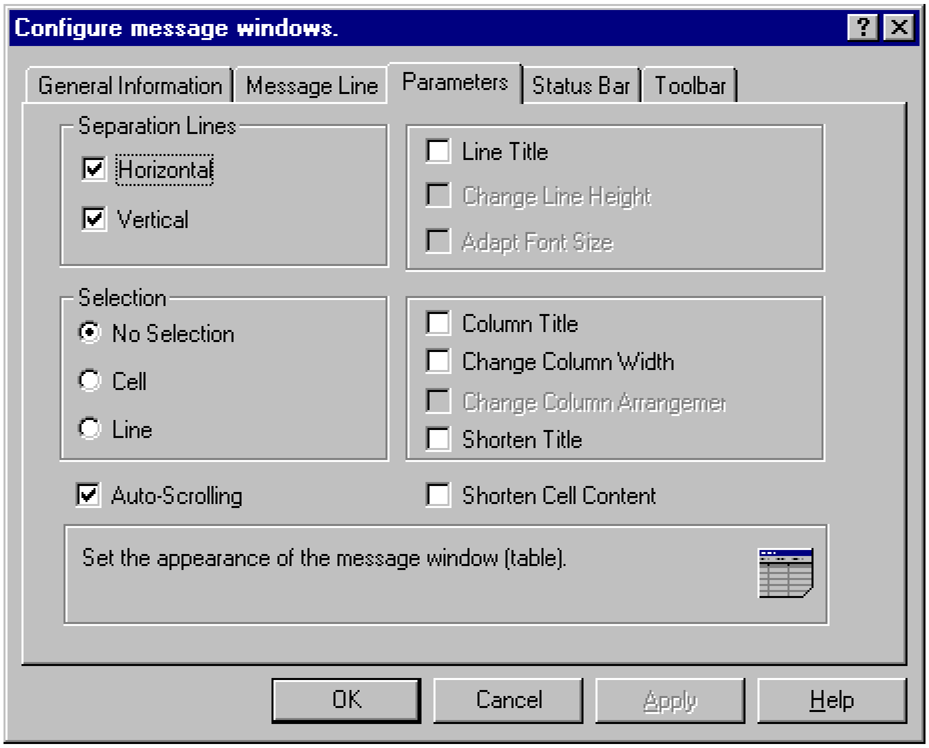
The functions for the focus of the picture can be found in the Global Script editor under *Internal functions →graphics →set →focus.* For example, the following function is called to set the focus of control:

```
Set_Focus(picture name, object name);
```

The name of the main window (picture name) and the application window (object name) must be entered as the parameters.

A message in the message window is selected by choosing the message line. When the message window is opened, the cursor is positioned in the youngest message (the last message in the message picture). Whether or not it is actually possible to select a message in the window or to scroll through the messages depends on the whether the scroll mechanism is active or not.

The picture scroll mechanism can be turned on/off either by means of a button in the toolbar or directly in the configuration of the message window template. When a picture is opened, the scroll mechanism is actively turned on in the *Alarm Logging →Message Window Templates→ Parameters →AutoScrolling editor.*



Separation Lines
☑ Horizontal
☑ Vertical

□ Line Title
□ Change Line Height
□ Adapt Font Size

Selection
● No Selection
○ Cell
○ Line

□ Column Title
□ Change Column Width
□ Change Column Arrangemer
□ Shorten Title

☑ Auto-Scrolling
□ Shorten Cell Content

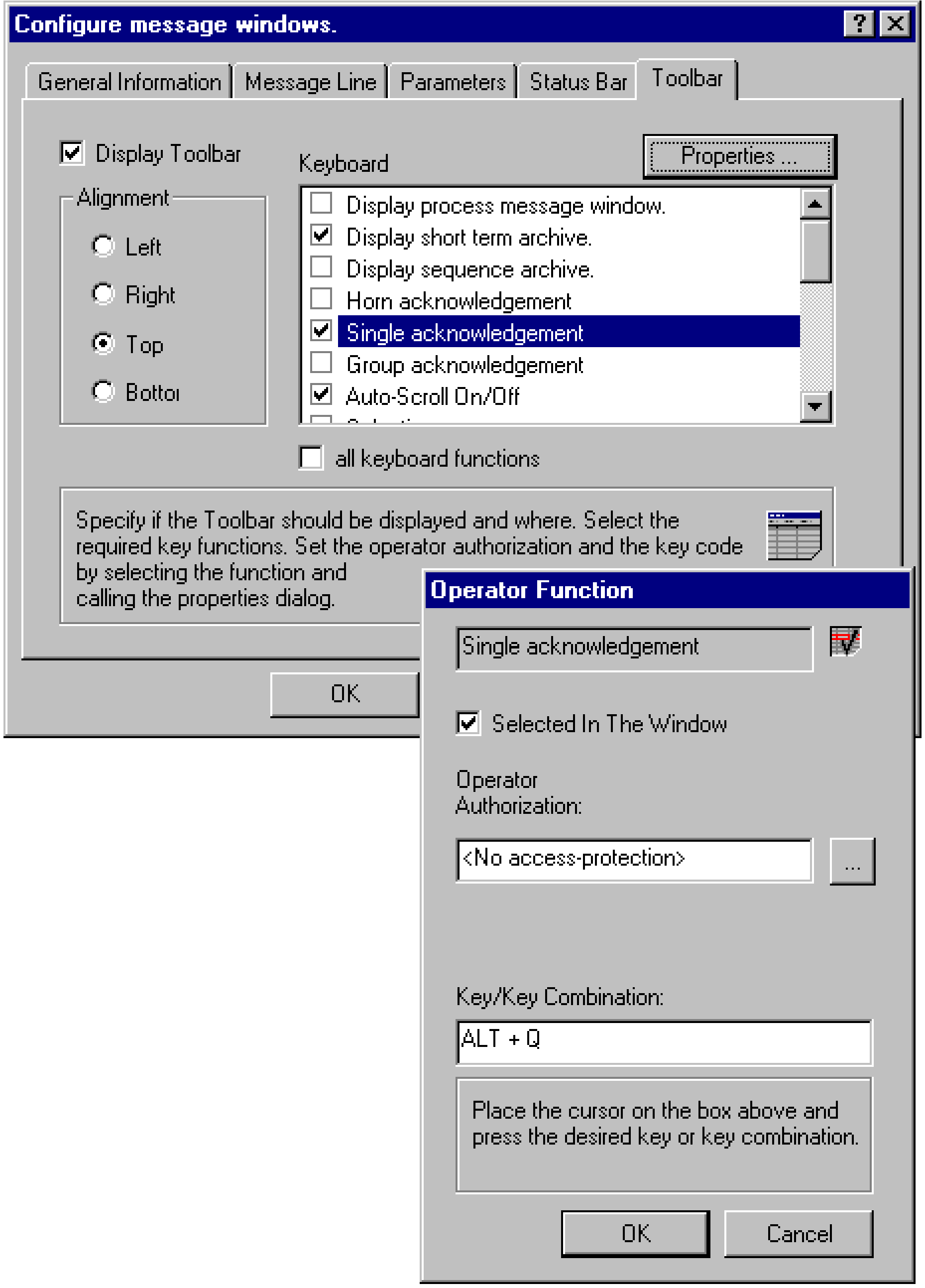
Set the appearance of the message window (table).

If scrolling and the focus of control in the message window are active, the movements can be performed as follows:

| Movement | Standard keys | Key settings |
|---|---|---|
| up/down through the message list | Arrow keys | Individual message lines |
| Start, end of message list | Home, End keys | Start, end of message picture |
| Scrolling | Scrolling keys (Page Up/Page Down) | Several message lines |

In addition to being able to activate the buttons on the toolbar (such as acknowledgement of the selected message) by means of standard mouse action, you can also define control actions by means of function keys.
A separate key action can be stored under the properties for each button displayed on the toolbar (*Alarm Logging →Message Window Templates →Toolbar →Key Crossed and Selected →Properties*).

For example:



Configure message windows.

General Information | Message Line | Parameters | Status Bar | Toolbar

☑ Display Toolbar

Alignment
- ○ Left
- ○ Right
- ● Top
- ○ Bottoi

Keyboard

- □ Display process message window.
- ☑ Display short term archive.
- □ Display sequence archive.
- □ Horn acknowledgement
- ☑ Single acknowledgement
- □ Group acknowledgement
- ☑ Auto-Scroll On/Off

□ all keyboard functions

Specify if the Toolbar should be displayed and where. Select the required key functions. Set the operator authorization and the key code by selecting the function and calling the properties dialog.

Operator Function

Single acknowledgement

☑ Selected In The Window

Operator Authorization: <No access-protection>

Key/Key Combination: ALT + Q

Place the cursor on the box above and press the desired key or key combination.

These configuration steps enable you to assign a key combination to each and every one of the buttons used for the message window. You should therefore also define key actions to enable the message window to be controlled by means of the keyboard.

### 3.3.10.4 Alarm Logging - toolbar buttons designed specifically for a plant

All the toolbar buttons are specified by WinCC and they cannot be changed. If a given button layout is specified for the plant to be configured, you must deactivate the WinCC toolbar (i.e. no toolbar) and design the relevant buttons yourself. All of these new button objects can be designed to satisfy the wishes of the customer, e.g. given icons.

The functionality assigned to a particular button must, however, still be configured as an associated action. In the *C action* of the associated event (e.g. Press button), the corresponding *standard function* must be selected from the function tree.

The functions available for assigning to buttons can be found in the Global Script editor under *Standard functions → alarm.* For each button in the toolbar, the list contains a function which corresponds to it. For example, the Single Acknowledgement button is called by means of the following function:

```
OnBtnSinglAckn(window name);
```

The name of the *message window template* must be entered as the parameter.
These actions can also be used for buttons you have designed yourself by means of mouse control.
A number of examples of such buttons can be found in the optional packages for the alarm system (e.g. *Basic Process Control* - Horn acknowledgement etc.).

### 3.3.10.5 Tag Logging function keys for the toolbar buttons

In the trend or table windows used for displaying measured values (also known as trend displays), various buttons are assigned to the toolbar, which are normally selected using the mouse.

The most frequent control actions in a trend window are

- scrolling through the measured values (time axis)
- selecting a time frame
- selecting trends
- using the read ruler

After opening the trend window, the current trend curve is displayed depending on the configuration.

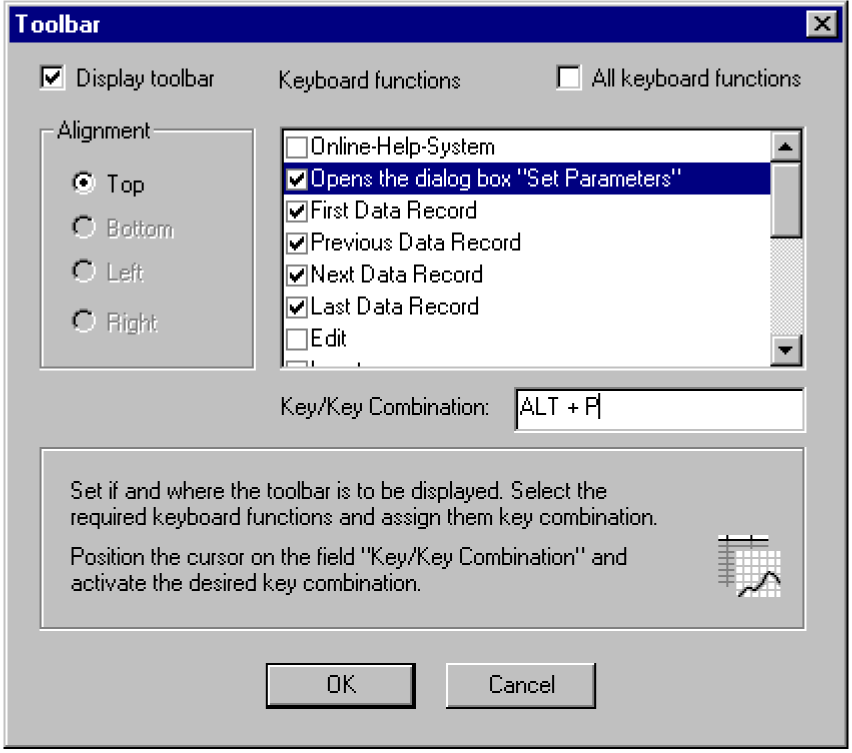
**Note:**
When opening the message window, control must lie in the trend window (application window) and not in the main window. Depending on the current controllability, the key actions (or function keys) act on the *function key bar* of the main window or on the stored key actions of the trend or table window.

This can be achieved, for example, by setting the current focus of control in this section of the window. The focus of control is normally set by clicking with the mouse.
Using the keyboard, the focus can be set in trend or table window by means of the following configurable routes:

- changing the window by means of a hot key
- setting the focus by means of a control button or
- setting the focus directly to a defined element in the message window on opening the picture.

Implementation of these different variants can be found in this chapter under the description of Alarm Logging.

In addition to being able to activate the buttons on the toolbar (such as selection of a time frame) by means of standard mouse action, you can also define control actions by means of function keys. By default, the individual buttons are occupied by the function keys F1 through F10.
A separate key action can be stored under the properties for each button displayed in the toolbar (*Tag Logging →Trend window templates →Template →Toolbar →Configure...*).
For example:



Toolbar

☑ Display toolbar

Keyboard functions

□ All keyboard functions

Alignment

● Top
○ Bottom
○ Left
○ Right

□ Online-Help-System
☑ Opens the dialog box "Set Parameters"
☑ First Data Record
☑ Previous Data Record
☑ Next Data Record
☑ Last Data Record
□ Edit

Key/Key Combination: ALT + P

Set if and where the toolbar is to be displayed. Select the required keyboard functions and assign them key combination.

Position the cursor on the field "Key/Key Combination" and activate the desired key combination.

These configuration steps enable you to assign a key combination to each and every one of the buttons used for the trend or table window. You should therefore also define key actions to enable the trend or table window to be controlled by means of the keyboard.

The following standard keys can be used in the trend window once the respective function key has been activated:

| **Movement** | **Standard keys** | **Key settings** |
|---|---|---|
| Read ruler | Arrow keys | For moving the read line left or right. |
| Zooming | For selecting zoomed window. | For setting and activating substitute input aid for mouse(see system settings).<br>INS and arrow keys enable you to define the zoomed window. |
| Dialog boxes, e.g. archive tag selection | Tab key | For moving between the input fields |
| | Arrow keys | For moving within tag selection or tab selection |
| | + key (- key ) | For displaying or closing the tree of the archive tags |

| Movement | Standard keys | Key settings |
|---|---|---|
| | Spacebar | For canceling selection or a selection. |
| | ENTER key | For confirming and exiting a dialog box |
| | ESC | For canceling a dialog box. |

### Tag Logging - toolbar buttons designed specifically for a plant

All the toolbar buttons are specified by WinCC and their design cannot be changed. If a given button layout is specified for the plant to be configured, you must deactivate the WinCC toolbar (i.e. no toolbar) and design the relevant buttons yourself. All of these new button objects can be designed to satisfy the wishes of the customer, e.g. given icons.
The functionality assigned to a particular button must, however, still be configured as an associated action. In the *C action* of the associated event (e.g. Press button), the corresponding *standard function* must be selected from the function tree.

The functions available for assigning to buttons can be found in the Global Script editor under *Standard functions* → *TAGLOG (*or additionally → *TEMPLATE* for the table functions*).* For each button in the toolbar, the list contains a function which corresponds to it. For example, the read ruler is called by means of the following function:

```
TlgTrendWindowPressLinealButton(picture name);
```

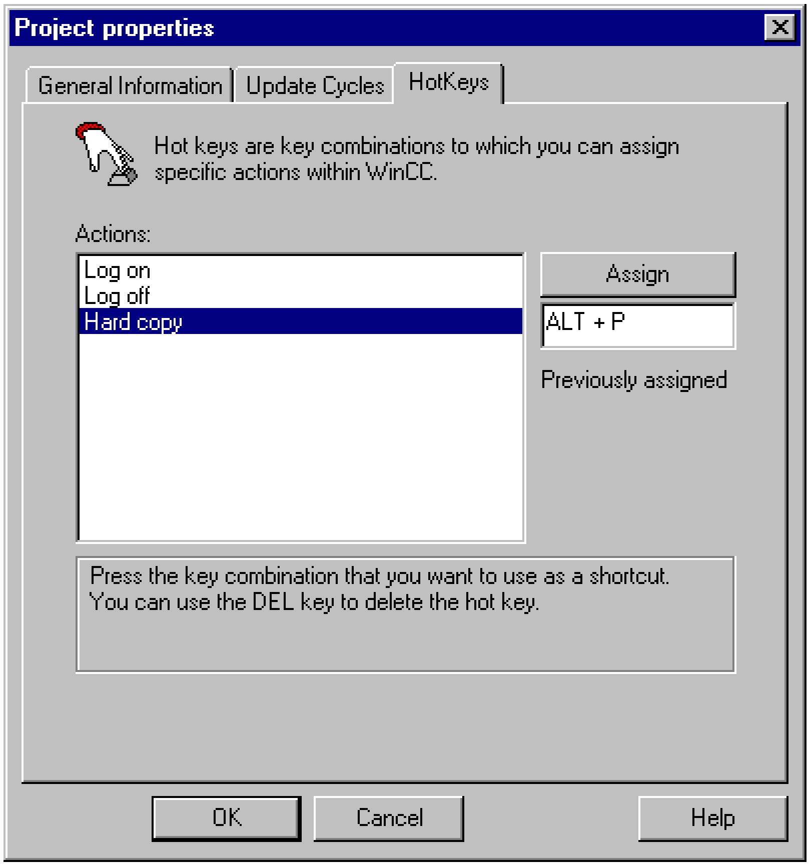
The name of the *trend window template* must be entered as the parameter.

These actions can also be used for buttons you have designed yourself by means of mouse control.

## 3.3.10.6 Starting a print job

A print job can be started in a number of different ways. In the *Control Center*, for example, printing is activated directly by selecting from the list of print jobs. You can, however, also create a Print button in the plant window itself, which you can used to start the print job.
This button already exists in the toolbar for the message lists and can, as described on the previous pages, be activated by a function key or a key you have designed yourself.

A printout of the screen contents - a hardcopy - can be activated in every picture by means of a hot key. This hot key is set globally in the project properties. To do this, select *Project properties* → *HotKeys* → *Hard copy* in the *Control Center* and define the hot key by directly entering the key combination (by pressing the respective key(s) on the keyboard).



Hot keys are key combinations to which you can assign specific actions within WinCC.

Actions:

- Log on
- Log off
- Hard copy

ALT + P

Previously assigned

Press the key combination that you want to use as a shortcut. You can use the DEL key to delete the hot key.

If you configure your own button for printing a defined print job in the plant picture, the action must be triggered by a *C action*. As described at the beginning of the chapter, the button is activated by, for example, a function key (see hot key) or a keyboard action (e.g. the D key). The *C action* must correspondingly be configured to the event in question (e.g. *Mouse Action* or *Keyboard - Press*). WinCC makes this functionality available with the functions in *Standard Functions→Report →ReportJob*.

```
ReportJob("Documentation Control Center","PRINTJOB");
```

The function requires two transfer parameters, the first of which is the name of the print job and the second of which defines the print destination (printer or scren).

### 3.3.10.7 Logging in or out

In addition to the hot keys that can be configured for the login and logout procedures, you can also configure a button which calls the Login dialog box. You can also log out by means of a key action. For this purpose, you have to configure a separate button, which can, for example, be activated both by a mouse action and via the keyboard. You can also set a function key control action by means of the hot key property of the button. The various variants of button and key action are described in detail at the beginning of this chapter. The function which must be used for login and logout is a WinCC application function. This function must be configured as a *C action*. Attach the *C action*, for example, to the event known as *mouse click* or the event *press key*.
The following function is used for logout:

```
#pragma code ("useadmin.dll")
#include "PWRT_api.h"
#pragma code()

PWRTLogout();
```

Login is activated by the WinCC C application function *PWRTLogin()*.
Here is an example of how this function could be used:

```
#pragma code ("useadmin.dll")
#include "PWRT_api.h"
#pragma code()
PWRTLogin(c);
```

The dialog box that pops up can be controlled by means of the standard keys:

| **Movement** | **Standard keys** | **Key settings** |
|---|---|---|
| Individual input fields | Tab key (forward) or SHIFT+Tab (backward)<br>Arrow keys | For moving the read line left or right. |
| Confirm (OK) | ENTER key | For exiting a dialog box and confirming the input. |
| Cancel (Abort) | ESC | For canceling a dialog box or input. |

## 3.3.11 Module technology

Picture module technology is a crucial strategy for enabling quick and simple configuration and the reusability and maintainability of configured picture components.

A configured process box is used, for example, for several process components of the same kind (e.g. valves or controllers).

The original picture window configured can now be reused for the control modules that are to be worked with and visualized in the project. This is done in accordance with the following principles:

- copying a picture window and reconnecting the tag fields
- using a picture window whose tag fields are assigned on being called (indirect link)
- using user objects with prototypes and objects created from them
- creating prototype pictures and integrating them
- creating OCX picture modules and integrating them as WinCC OCX objects

### Comparing the different techniques

These techniques differ very greatly with respect to how they are applied, the complexity of configuration and their possibilities. For this reason, we will start by comparing the alternatives.

| Type | Advantage | Disadvantage |
|---|---|---|
| Copy of picture windows | Simple procedure | All object links must be changed.<br>Changes to picture buildup lead to complex post-processing. |
| Picture window with indirect link | • Once-only buildup of the picture window with simple *C actions*<br>• Reuse without copying the basic picture window | Changes to picture buildup lead to complex post-processing. |
| Smart objects | Once-only buildup of the object with link by means of existing Dynamic Wizards | • Changes to picture buildup lead to post-processing, i.e. picture regeneration.<br>• Cannot be changed centrally. |
| Prototype pictures | • Once-only buildup of the object<br>• Can be changed centrally. | (Good) knowledge of C necessary. |
| OCX | • Simple integration into configuration of WinCC as an object in the picture<br>• A later modification of the OCX object doesn't lead to post-processing in the objects generated, except when changes are made to the object | Must be created by writing a program (C++, VB 5); cannot be created by means of WinCC configuration. |

| Type | Advantage | Disadvantage |
|---|---|---|
| | properties<br>• High level of performance<br>• Other graphical possibilities<br>• Buying-in of new objects (e.g. PCS7 modules) | |

If only a few simple picture modules are used in the project, one of the first variants is adequate enough to satisfy the person configuring the system. These variants can be implemented without any great training being required.
The user object is particularly suitable for simple objects of low to medium complexity and tag link. If you foresee that the object will require a number of changes, you will find it makes sense to get to grips with the concept of prototype pictures.
If the graphic modules are of a complex nature or if comprehensive processing work is required, preference should be given to OCX technology. The OCX objects available will in the future grow stronger and stronger in this field.
In the following chapters, we will show you the various types of picture module configuration and how they are put to use in the plant pictures. This will enable you to form you own picture of the different variants and their applications in your projects.

### 3.3.11.1 Process box as a picture module

Specific information boxes pop up in the plant pictures to display the current states of a device (controller, valve, motor etc.) or to enable you to specify set values. These process boxes typically contain both current states (actual values) and set values, which can be entered by the privileged operator.

### Creating the information box

This information box is created as a picture window whose components are connected to the corresponding (process) tags.

| Step | Type | Configuration |
|---|---|---|
| 1 | Data structures | Define the data structures to be used in the picture module by means of the tag management, e.g. motor with actual value, set value, on-off switch. |
| 2 | Picture module | Using the *Graphics Designer*, configure a picture which displays the device states, e.g. bars and I/O fields, and control buttons. The size of the picture window (picture object property - X variable and Y variable) must correspond to the target size of the picture window. |
| 3 | Defining the tags | Define the (process) tags in the tag management, e.g. Motor_T01 of the (structure) data type Motor Type, which is used for the process box. |
| 4 | Tag link | Now dynamize the individual picture components, e.g. I/O fields, bars etc., by connecting them to the corresponding (process) tags. |
| 5 | Picture window | In the plant picture, create a picture window object and connect it to the picture window contents created under steps 2 through |

| Step | Type | Configuration |
|---|---|---|
| | | 4 by means of the Picture window name property. |
| 6 | Properties - Settings | This picture window object should not be shown when the picture is opened. Therefore, the property *Display* must be set statically to No.<br>The appearance of the picture window, with the Windows buttons and title etc., must also be defined in the properties of the picture window. |
| 7 | Calling the picture window | This picture window must be caused to pop up by, e.g. clicking a button or working on the device itself. Design a button which is connected to popping-up of the picture window object (e.g. by means of a *direct link*). |

This picture window object, the picture window contents and the corresponding call of the picture window (button) can be reused in a similar form for further devices. All you have to do is copy the picture window object, the picture module and the button. The references must be adapted each time. The picture window object and the button can both be copied by being dragged to and dropped in the graphics library (e.g. project library).

## Customizing the picture modules

The following individual steps must therefore be carried out when using the picture module created:

| Step | Type | Configuration |
|---|---|---|
| 1 | Process tags | Define a new process tag, e.g. Motor_T02, for the defined data structure. |
| 2 | Copy of the picture module | Make a copy of the picture window contents (Motort02.PDL) and change all permanently stored references (e.g. instead of Motor_T01.ActValue, now Motor_T02.ActValue). |
| 3 | Copy of the picture window | Make a copy of the picture window object in the destination plant picture (by dragging and dropping from the graphics library). Adapt the reference to the picture window contents under *Properties... →Picture name* (Motor02.PDL). |
| 4 | Copy of the button | Make a copy of the button in the destination plant picture (by dragging and dropping from the graphics library). Adapt the reference to the new picture window object in the *direct link* (*Object →Picture Window2 →Display).* |

In this way, the individual picture windows and their contents can be created for each device and reused by being copied. As can already been seen, the work this necessitates is that of adjusting the permanently stored references for the picture window contents. For this reason, there is a more simple method of achieving reusability through indirect addressing. The amount of adaptation work required should be kept to a minimum.
An alternative solution to this is to configure the picture module without a link to the picture window. This means that the picture module itself is configured as a non-displayed object in the plant picture. The great disadvantage of this, however, when changing the picture module is that a change must be made in all the pictures in which this picture module is used.

### 3.3.11.2 Picture module with indirect addressing

So far, the individual components of the picture module have been permanently connected to the corresponding (process) tags. If the link is not made by means of a permanent configuration but dynamically during runtime, the picture module created can be used with far greater flexibility. This dynamic linking of (process) tags is implemented by means of indirect addressing of the individual components in the picture module. This means that no direct link is made to the (process) tags; the link made is only to the *container* which will carry the current names of the corresponding (process) tags during runtime.

The adaptational and reusability characteristics of a picture module can in this way be simplified considerably.
Configuration is carried out in a similar way to that explained in the steps described above. Here are the actually steps to be carried out:

| Step | Type | Configuration |
|---|---|---|
| 1 | Specification of data | Define the data to be used in the picture module by means of the tag management, e.g. Motor001_ ActValue, Motor001_SetValue, Motor001_Switch, on the one hand and specify the name container for the individual components that are to be used in the picture module, e.g. ActV_Name, SetV_Name etc., on the other. You initialize these tags with a name, e.g. Motor001_SetValue. |
| 2 | Picture module | Using the *Graphics Designer*, configure a picture which displays the device states, e.g. bars and I/O fields, and control buttons. The size of the picture window (picture object property - X variable and Y variable) must correspond to the target size of the picture window. |
| 3 | Tag link | Now dynamize the individual picture components, e.g. I/O fields, bars etc., by connecting them to the corresponding container tags which contain the names of the corresponding tags. You must, however, state in the link that the tag is only the name of the actual (process) tags. You do this by ticking the *indirek.Adr.* column in each case. |
| 4 | Picture window | In the plant picture, create a picture window object and connect it to the picture window contents created under steps 2 through 3 by means of the Picture window name property. |
| 5 | Properties - Settings | This picture window object should not be shown when the picture is opened. Therefore, the property *Display* must be set statically to No.<br>The appearance of the picture window, with the Windows buttons and title etc., must also be defined in the properties of the picture window. |
| 6 | Calling the picture window | This picture window must be caused to pop up by, e.g. clicking a button or working on the device itself. Design a button which is connected to popping-up of the picture window object (e.g. by means of a *direct link*). |
| 7 | Graphics library | The picture window object and the button are copied to the library (by being dragged and dropped), so they can be reused. |

### 3.3.11.3 User objects

User objects and the corresponding dynamic wizards can be used to create picture modules which are easy to reuse. The copy made of the picture module can be connected to the corresponding current (process) tags by means of simple configuration using the wizard.
A user object is a graphic object designed by the person configuring the system (e.g. a combination of several objects), whose large number of properties and events are reduced to the essential properties and events by means of a configuration dialog. This user object is dynamized by being defined as a prototype by means of the corresponding wizard.

The following steps must be taken

| Step | Type | Configuration |
|---|---|---|
| 1 | Data structures | Define the data structures to be used in the picture module by means of the tag management. |
| 2 | Picture module | Using the *Graphics Designer*, configure a user object with the user-defined properties. |

A user object is formed from a group of WinCC objects. Initially, these objects are not configured with dynamics. All the objects that are to be combined to form the user object are selected and the configuration dialog of the user object is called:

In this dialog box, all the properties of the objects are now declared to be the properties of the user object, which are later to be dynamized. The basic properties for an object (e.g. the position and size) have already been stored for the user object.
Each individual property of the combined objects can be selected in the dialog box and added to the new user object (per Drag&Drop) as a *user-defined property or event*.

Each of these properties can be assigned a new (language-independent) attribute name by the user and also the language-dependent property name (e.g. for configuration in English). Properties which should not be visibly displayed in the properties dialog, but which are used for example in scripts, can be hidden by using the @ character. This means that only a few properties and events (to be dynamized) can be displayed. All the others are hidden.

The user object you have designed must now be dynamized. A specific wizard is provided to enable you to do this.

| Step | Type | Configuration |
|---|---|---|
| 3 | Dynamization | Call the dynamic wizard called *Add dynamics to the prototype*.<br>As a template (i.e. prototype), link each individual property of the object to the corresponding structural component of the data structure defined.<br>The *structure member* is selected for interconnection by means of the tag browser.<br>The wizard, however, only saves the name of the structural component on the property linked (e.g. Value). Each individual property must be linked separately.<br>This object is now a dynamic object, which, however, has only been linked as a prototype and is not active during runtime. This means that it cannot be updated during runtime. |
| 4 | Copying to the graphics library | Copy this prototype to the graphics library as an object. |

The prototype user object is copied to the graphics library so that it can, for example, be reused again and again. One example of a dynamic object are the pointer instruments in the WinCC library (user library, user objects, pointer instruments).

The prototype object is inserted into the destination plant picture as a copy of the prototype. This copy now has to be linked to the real (process) tags from the tag management.

| Step | Type | Configuration |
|---|---|---|
| 5 | Tag | Define a (process) tag for the data structure defined in step 1; this tag is to be used for the user object. |
| 6 | Creating an instance | Copy the prototype object from the graphics library to the plant picture by dragging and dropping it.<br>Link this object to the (process) tag by means of the dynamic wizard called Link a prototype to a structure.<br>The wizard automatically links all necessary structural components of the tags to the correct property of the prototype picture module by replacing each prototype tag link on each property with the actual tag link.<br>You now have and object which is updated with the current tag values during runtime. |

### 3.3.11.4 Dynamic instance (linking prototype to a structure)

*Link a prototype to a structure* dynamic wizard, there is also a wizard called *Make a prototype dynamic*. How does it differ from linking to a structure and which steps have to be modified?

In contrast to permanent linking of an object to tags, the picture modules can also be linked dynamically. This means that the instance to the runtime is set first depending on the current contents of a tag. For example, the above picture module is not linked permanently to the tag; the name of the tag is kept dynamic. The current name of the tag must then be determined by means of a text tag. This text tag, which contains the current name of the tag, must be linked to the picture module.

In contrast to permanent instanciation, i.e. linking to a structure, the following steps must be modified when configuring:

| Step | Type | Configuration |
|---|---|---|
| 2 | Picture module | Using the *Graphics Designer*, configure a user object with the user-defined properties as described above. The user object must contain a *Static Text* component whose *Text* property is taken over as a user-defined component.<br>This Text property is assigned an attribute name, *Tagname*. This tag name is used for dynamic linking of the (process) tags. |
| 5 | Tag | Define a (process) tag for the data structure defined in step 1; this tag is to be used for the user object. |
| 6 | Creating an instance | Copy the prototype object from the graphics library to the plant picture by dragging and dropping it.<br>Link this object to the (process) tag by means of the dynamic wizard called Link a prototype to a structure.<br>The wizard automatically links all necessary structural components of the tags to the correct property of the prototype picture module by replacing each prototype tag link on each |

| Step | Type | Configuration |
|---|---|---|
| | | property with the actual tag link.<br>You now have and object which is updated with the current tag values during runtime. |

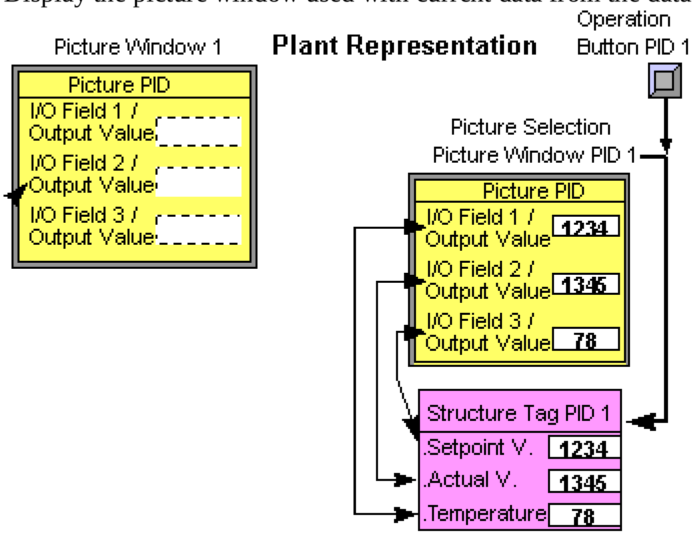
When the dynamic user objects and prototypes are used, the person configuring the system must make sure that the necessary *C actions* have already been stored on the objects. They must not be deleted, since otherwise the entire functionality of the module is lost.

### 3.3.11.5 Prototype pictures

The technology of the prototype pictures goes one step further. When prototypes are used, the concept can be structured so flexibly that a change made to the prototype is automatically followed by an adjustment to the objects created. This flexibility requires the linking of very flexible C scripts.
The technology of the prototype pictures operates using template pictures which can be repeatedly integrated into one or more parent pictures. A template picture is only a template, which is brought to life only once integrated into a real object. An object based on a template (= prototype picture) is brought into being by what it known as an instanciation".A number of instances (i.e. real objects) can be created for one template.

#### Template picture window

Display the picture window used with current data from the data manager.


Picture Window 1
Plant Representation
Operation Button PID 1
Picture PID
I/O Field 1 / Output Value
I/O Field 2 / Output Value
I/O Field 3 / Output Value
Picture Selection Picture Window PID 1
Picture PID
I/O Field 1 / Output Value
1234
I/O Field 2 / Output Value
1345
I/O Field 3 / Output Value
78
Structure Tag PID 1
.Setpoint V.
1234
.Actual V.
1345
.Temperature
78

Subsequent changes are made at a central point (in the template) and affect all applications (instances). This makes this a very efficient program which does away with the laborious task of dragging changes to many different places.

In a parent picture, up to 30 instances (i.e. objects) of a particular template type can be displayed. If different prototypes are used, it is possible to use more than 30 objects.

The prototype pictures are picture modules which are copied to the library after being created, so they can be reused. The picture modules used are used in the plant pictures as instances of the template. These copies show the current data, e.g. of controllers or motors, which is visualized in the module. The corresponding controller or motor components are displayed automatically.

Not only simple picture modules, but complex ones too can be created. A picture module can consist of several components that overlap each other either partially or fully, but that make up a single common unit. For example, all the data that relates to a motor, such as a view of the current status, progress data, maintenance data, etc., can be combined in one object and updated as and when required. If you have a number of motors of the same type, all you have to do is create the picture module once and then make copies of it. Everything else is automatic.

The following steps must be followed to create the picture module:

| Step | Type | Configuration |
|---|---|---|
| 1 | Data structures | Define the data structures to be used in the picture module by means of the tag management. |
| 2 | Picture window buildup | Using the *Graphics Designer*, configure the contents of the picture module, e.g. bars, I/O fields etc. |
| 3 | Linking data structure with picture window buildup | Link the picture components with the individual data structure components with special C scripts (from the sample project). |
| 4 | Picture window | Link the picture module to the picture window object. |
| 5 | Graphics library | Copy the picture window to the graphics library. |

The picture module can now be used in the plant pictures by being retrieved from the graphics library.
Let’s first take a look at how the template picture is created.

| Step | empty | Configuration |
|---|---|---|
| 1 | | Creating a tag structure (data type, structure type) in the data manager; this is where the number of tags that go to make up the structure is defined (member tags), what the tags are called and of which data type each of the tags is (BIT, SHORT, etc.). For example, PID with set value, actual value and temperature as structural components. |
| 2 | | Creating a picture which is to be used as a picture module. This is usually smaller than the size of the monitor screen and can be dimensioned accordingly. Each picture that has been created in its own right can be used to create a template.<br>The picture is created using the graphics editing functions, and graphics tags such as I/O fields, bars etc. are positioned in it but not linked to tags.<br>Internal relationships (*direct links*) between graphic objects, such as change-controlled transfer of the output value of an I/O field to a bar, are configured in this picture. |
| 3 | | The graphic fields are now linked to the structural components of the corresponding tag structure. This link configuration links the tags at type level (template only), but not yet with concrete process objects.<br>For this purpose, you will find a pre-prepared sample project on |

| Step | empty | Configuration |
|---|---|---|
| | | the WinCC CD (\Samples).<br>In the project library (\Template) of this project is a user object called *TemplateInit*, which performs this interconnection. This is located in the graphics library and can be dragged and dropped from here into a picture that is to be standardized.<br>*TemplateInit* already has a complete script logic. This logic uses what is known as a *ConnectionTable*, which during configuration is filled in as a table and contains precisely the underscored entries quoted above. This is method used to define the link between the properties and the structural components.<br>The buildup of these links can be set within a template or also from outside. To this end, the special project graphics library contains user objects which to the naked eye look like simple buttons, but which in fact contain parameterizable information about the template to be called up.<br>All of the scripts for implementing this already prepared generation of the prototype must be copied to the destination project. See the final steps described at the end of the chapter under steps 8 - 10. Without these project functions, these prototypes cannot be implemented. |
| 4 | | This picture is to be used as a process box.<br>To do this, create a picture window object in a temporary picture (i.e. it is only required for this intermediate step) and link the *Picture Name* property of this object to the picture that contains the picture module. |
| 5 | | Copy this picture window object to the graphics library by dragging and dropping it. |

This template can now be used repeatedly in the plant pictures. The link to the process tags is made automatically when the name is entered.

| Step | Type | Configuration |
|---|---|---|
| 6 | Defining the tags | Define a (process) tag for the corresponding data type (e.g. PID_1 of type PID). |
| 7 | Creating an instance | Copy the picture window module from the graphics library.<br>Immediately assign the picture window object name to the name of the (process) tag that is to be used (e.g. PID_1): set *Picture window object → Picture window →Object name → Static* 🖱**D** to PID_1 |

When the picture module is positioned in the picture, it is given the name of a structured (process) tag whose values contain the status data of a process object. During runtime, then, the picture module retrieves the status data automatically.

A number of C scripts are used for these prototype pictures, which are or have been stored as project functions. To be able to use the C scripts that have already been prepared, you have to adopt the following scripts from the sample project. Proceed as follows:

| Step | Type | Configuration |
|---|---|---|
| 8 | Copying the function files | From the path <Projectpath>\Library, copy all the required functions (.fct), e.g. LinkConnectionTable.fct, from the sample project to your project folder <Projectpath>\Library. |
| 9 | Making known in the project | In WinCC, call the Global Script editor (*Global Script →* 🖱**R** → *Open*) to make these new functions known in the following:<br>you can now make the new functions known to the function tree of the project functions by clicking the *Regenerate Header* button. The new functions can now be seen in the list of project functions. |
| 10 | Taking over user objects | The user objects are made available by means of the project library. In a new project in which you have so far not stored any of your own symbols in the library, you can simply copy the library from the sample project to your project:<br>copy <Projectpath>\Library\library.pxl → to your path <Projectpath>\Library<br>Otherwise, use the export mechanism to transfer the user objects from one project to the other.<br>Export the desired symbols in the sample project as .emf files (*File →Export...)* and import these .emf symbols in your own project into a temporary picture by means of *Insert →Import...*. Transfer the symbols to your project library by dragging and dropping them. Use a separate folder also to do this, e.g. Template. |

Picture modules (standardized picture segments or templates) offer a very large savings effect. They are modeled as objects with tag references and are, for example, stored in the graphics library. They are taken from this graphics library, positioned in the plant picture and automatically supplied with data during runtime. There is no longer any need to configure links for individual tags with graphic segments such as I/O fields, bars etc.

When using these prototype pictures, there are different ways in which the picture module components are supplied with the current names. This is done by means of the following variants, which we will briefly name here:

- The instance name determines itself when the picture window is **opened**:
  to this end, a specified project function (EnableTemplateInstance) is stored under the Open Picture event in the picture window itself.
- **input/output tag**, which is automatically read in the picture window by means of a script; this is how the instance name is determined:
  this is done by using the pre-prepared user object, *InstanceCallButton+Template*.
- A **button** transfers the instance name directly to the open picture window:
  this is done by using the pre-prepared user object, *InstanceCallButtons+Template*.

You can find detailed examples of these variants in the sample project on the WinCC CD (\Samples).

### 3.3.11.6 OCX objects

OCX or ActiveX objects are picture modules which are available as loadable components. WinCC offers a number of them, e.g. the WinCC Digital/Analog Clock Control.

These modules can be very easily incorporated into the plant pictures.

| Step | Type | Configuration |
|---|---|---|
| 1 | Inserting an OCX object | In the Object Palette of the *Graphics Designer* under Smart objects, select the type *OLE Control* (OCX). Drag the object into the plant picture by holding down the 🖱 and in the dialog box that then pops up, select the element you want. |
| 2 | Linking properties | The object inserted likewise has properties and events. Which properties and event are available depend on the specific OCX itself.<br>A link to a tag can be created, e.g. using the property Process driver connection. |

## Creation

The OCX picture modules must be created using a separate development environment. You can use, for example, Microsoft Visual C++ 5 or Microsoft Visual Basic 5.
This method is used to modify and improve the picture modules. The OCX modules are very powerful, but they cannot be created with WinCC configuration resources. In this case, you will always have to resort to an external method of creating or modifying.
In contrast with change-friendly and powerful OCX programming, the technology of the prototype pictures can get by with pure WinCC resources. This means that you do not need to have any knowledge of OCX programming.

There are already a large number of such modules available today. Among other things, complete modules are available as PCS7 faceplates in connection with the integration of the two worlds of Operation&Observation (Bedienen&Beobachten) and PLC programming (PCS7) in the plant sector.

## Registration

The OCX modules created or bought in must be registered on the respective WinCC station. You can see which OCX objects are available on the WinCC station by looking in the selection dialog of the *Graphics Designer* (see description above). All the OCX elements registered on the computer are listed in the dialog box. An OCX element is stored on the computer in the form of a file with the extension .OCX or .dll.
If a module has not yet been registered, you can do so in the WinCC OLE Control dialog box. The dialog box contains a Register... button (obviously for registering the module) and an Unregister button for removing the registration of the components currently selected. The relevant file must be on the WinCC station for registration to be carried out.

The compatibility and mode of functioning of the OCX components must be tested by the person configuring the system. Only those OCX modules marked using WinCC have been used and tested in the WinCC environment.

## 3.3.12 Online configuration (runtime) - notes and restrictions

A number of points must be noted with respect to online configuration.
For various reasons, a small number of changes are not able to be made while online or can only be made under certain conditions, or the changes only become effective at a later point in time.

### Control Center

The following changes are not adopted:

- changing the type of a computer in the computer list

During runtime, the following configuration steps are not possible:

- deleting/renaming tags
- changing the data type of a tag

### Alarm Logging

The following changes are not adopted:

- changing the archives/reports
- changing the group messages
- every message after a total of 500 single messages during runtime

During runtime, the following configuration steps are not possible:

- no restrictions

### Tag Logging

The following changes are not adopted:

- no restrictions

During runtime, the following configuration steps are not possible:

- tables of the user archives can be created but not changed
- deleting data in Tag Logging and user archives

Exceptions for configuration during runtime:

- the runtime API of Tag Logging can be used to edit and delete the tables of the user archives

## Global Script

The following changes are not adopted:

- changes made to a wizard script are only adopted after restarting the *Graphics Designer*.
- modified wizard scripts

During runtime, the following configuration steps are not possible:

- no restrictions

## Report Designer

The following changes are not adopted:

- changes to the message sequence report, since once started, it always remains active in runtime and doesn't reload the layout information

During runtime, the following configuration steps are not possible:

- no restrictions

## Redundancy

The following changes are not adopted:

- the computer name of the partner cannot be transferred to a third computer
- the AutoSwitcher cannot be changed, i.e. you must configure at the outset to where the AutoSwitcher is to be switched. It does, however, also switch back if the other one fails.

During runtime, the following configuration steps are not possible:

- no restrictions

## SIMATIC S7 Report Suite or S/7PMC channel

The following changes are not adopted:

- no diagnostics parameters relating to the S7Chn.ini (not published) are adopted online.
- all changes to communication addresses are indeed adopted online, but are only evaluated when a connection is set up.

During runtime, the following configuration steps are not possible:

- no restrictions

### Text Library

The following changes are not adopted:

- no restrictions
  - *File →Send Changes to Active Project*.
  - In Alarm Logging, the changes are copied to the text library by means of *File → Save*.

During runtime, the following configuration steps are not possible:

- no restrictions

### User Administrator

The following changes are not adopted:

- changes to user authorization only become effective after logging off and on again.

During runtime, the following configuration steps are not possible:

- no restrictions

# 4 WinCC - C Course

Experience has shown us that the script language of WinCC poses a problem that newcomers to C initially find extremely difficult to solve. For many applications, it is perfectly adequate if you provide the available functions with the appropriate parameters and use just these functions.

**Note:**
It is not absolutely necessary for you to create your own functions or actions when dynamizing objects, since the software already offers a *dynamic dialog*. This dialog creates executable and expandable *C actions*. *Dynamic dialogs* run in a special memory area of the operating system and are therefore considerably quicker than *C actions***.** The functionality of the *dynamic dialogs* is limited, however. A *dynamic dialog* can be converted to a *C action* at a later stage, but this does lead to a loss of performance.
The quickest method of dynamization is, however, the *direct link*. But this method also suffers from certain restrictions (e.g. only for objects in the picture) compared with *C actions.*

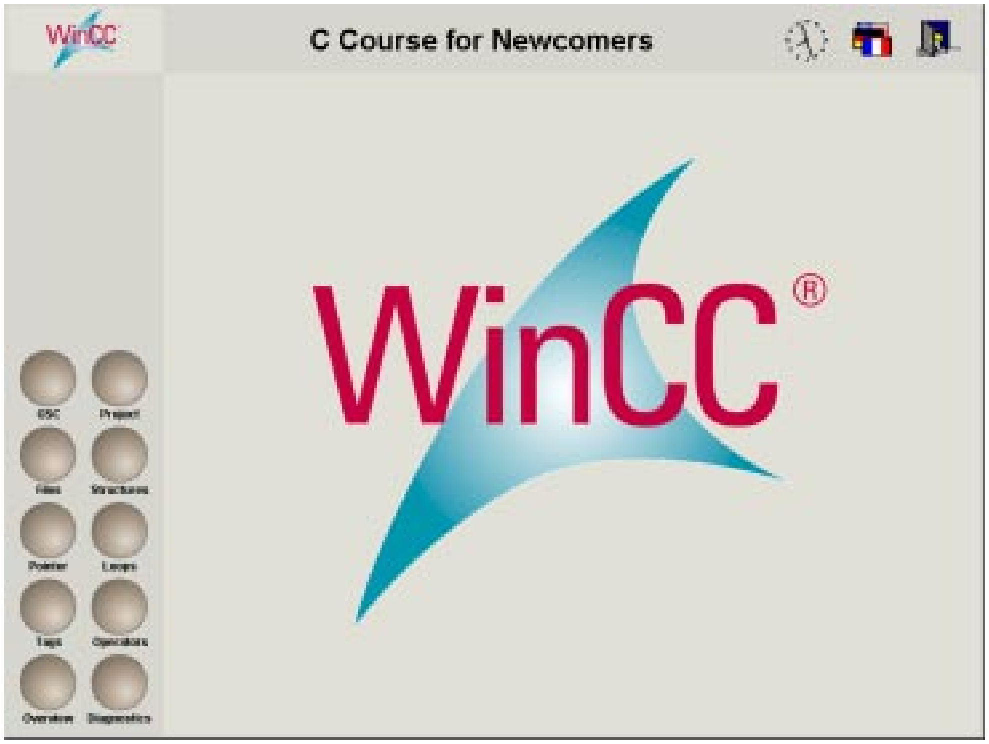
*C actions* in the *Graphics Designer* and*Global Script* editors offer users with very demanding tasks an extremely powerful script language. If, in your initial euphoria when developing, your scripts turn out to be somewhat larger and more complex, you can integrate them into WinCC as executable *dll (Dynamic Link Library)* functions and improve performance considerably in this way.

## Target group

This "C course" is intended to illustrate the most important points of the programming language C to newcomers. It should also provide all those who already program in C with a better insight into the special features of this development environment.

The examples used for this course have been configured in the WinCC project with the name *Cours_00*.

## Start screen of the C Course



We have configured a picture with a number of examples for each topic covered in this course. You open the individual pictures by clicking the corresponding button in the toolbar with the 🖱.

The *Diagnostics* button is used to turn the WinCC diagnostics window on/off. In the *C actions*, all outputs are output to this configured diagnostics window by means of *printf*. If in the individual examples the relevant button is clicking with the 🖱**R**, the source code of the example in question is displayed.

## 4.1 The development environment for scripts in WinCC

*C actions* in WinCC's *Graphics Designer* and *Global Script* editors offer you an extremely powerful script language. The syntax of the script language corresponds to Standard C (per ANSI standard) and is processed interpretatively. This means that the instructions formulated in the script language are only checked for syntax and not immediately executed on being compiled.

*Global Script* is the generic term for *C functions (standard and project functions)* and *global actions* which can, depending on their type, be used for the entire project or even for all projects.

### 4.1.1 Functions and actions in WinCC

*C actions* (functions and actions) are executed during runtime and serve to both visualize and control the process.
Functions and actions can be used to change object properties and they enable you to react to events.

#### The structure of a function

*Functions* consist of two different areas:

| | |
|---|---|
| **Header of the function** | `Type of the return value`<br>`Function name`<br>`(Type of Parameter1 Parametername1,`<br>`....)` |
| **Body of the function** | `{`<br>`Declare tags;`<br><br>`Commands;`<br><br>`return;`<br>`} // End of function` |

#### The function header

The header describes the environment of the function (interface to the outside world).
Here is an example of a *function header*:

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
```

The header of the *function* describes the name (How is the function called (i.e. invoked)?) and transfer parameters (which data is transferred to the *function* and which *return value* is supplied by the function) of the function.

The *function header* is automatically displayed when the *C action* is called and doesn't have to be written by the person configuring the system.
The *function header* shows the person configuring the system which *return value* has to be returned and which data is already available in the function.

Different function headers are displayed depending on the type of dynamization of a property or event.

#### Function header of an event

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char*
                 lpszPropertyName)
{
//Function body
}
```

The *return value* with these actions is always of the type *void* (i.e. nothing). The command *return;* can be left out of the *C actions*, since it is executed automatically at the end of every *C action*.

## Function header of a property

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char*
                   lpszPropertyName)
{
//Function body with return value
}
```

In the case of the *C actions* for object properties, the new setting of the property to be made dynamic is delivered by means of the current *return value* of the C action. The type of the *return value*, e.g. a number (in our example, *long* ) or a character string (e.g. *char* *), is defined in the header of the *C action*.

## The function body

The body describes the content of the function (**What does the function do?**).

```
/* Declaration of the tags */
int number1;

/* Value assignments to the tags */
number1=123;

/* Output of the tag values in the diagnostics window */
printf("The first number is %d\n",number1);
}
```

The individual commands are called in the *function body*. The *function body* begins with a left curly bracket and ends with a right curly bracket.
The *function body* defines additional data which are required during processing of the *function* (local data). This data is automatically deleted at the end of the *function*.
Each individual command in the *function body* is concluded by a semicolon.

## Return value

The *return value* of the *function* (e.g. the new color or new PositionX) is returned at the end of the function in the *function body*.

Formulation of the C sequences is based on existing functions. These functions can be displayed and used by means of the function tree. These project-wide and cross-project functions are formulated in the *Global Script* editor, with a distinction being make between the following types of function:

## Standard functions

WinCC enables you both to create new *standard functions* and to modify existing ones. *Standard functions* are known, i.e. available, to all projects and are divided into categories. You edit them in the *Control Center* using the *Global Script editor.*

*The standard functions* are used for dynamizing *graphic objects* and *archives. Standard functions* can be called in *project functions*, in other *standard functions* and in *actions*.

## Project functions

WinCC enables you both to create new *project functions* and to modify existing ones.The *Project functions* are only known, i.e. available, in the *project* in which they have been created. This is **the only difference** to the *standard functions*.
You edit them in the *Control Center* using the *Global Script editor.*

Actions which are used more than once should be defined as *project functions* or *standard functions*.
*Project functions* can and should be categorized in the same way as the *standard* and *internal functions*.
The advantages of *project functions* are:

- their **reusability.** Once the *C action* has been configured and extensively tested, it can be used again and again at any time without requiring any additional configuration and renewed extensive testing.
- Simple **modification**. If a modification is made to the *C action*, this modification is only effective at one location – in the *project function*. You don't have to search for and adapt the configured dynamics in all pictures. This simplifies not only configuration, but also maintenance and troubleshooting work.
- a **reduction in the data e of pictures** and consequently quicker times for opening pictures. If project- and standard-function calls are used in the picture actions, the overall volume of the data in the pictures is reduced.
- Project functions can be protected against unauthorized modifications by the use of a password (**user authorization**) (protection of configuration data and configuration know-how).

## Internal functions

*Internal functions* cannot be created or modified and are known, i.e. available, to all projects.
*internal functions* to dynamize *graphic objects*, *archives*, *project functions*, *standard functions*, and *actions*.
*Internal functions* are divided into categories.

## Actions

Actions (global and on the object) can be created and modified. These actions are only known, i.e. available, to the project in which they have been created.
You use project, standard and internal functions:

- in *actions* on the object
- in *actions* which are created in the *Dynamic Dialog*
- for making *alarms*, *process-value*, *compressed* and *user archives*.

**Note:**

Actions are processed interpretatively. You must therefore count on the load on the system being higher whenever you use a large number of or wide-ranging actions. For this reason, it is better if you replace wide-ranging actions with their own *DLLs* (Dynamic Link Libraries).

You use actions during runtime for the purpose of process control. The execution of actions is triggered by (believe it or not!) a *trigger*. To this end, you have to activate the project.

You create a global action using the *Global Script* editor. Start this editor from the *Control Center*.

## 4.1.2 The editors for functions and actions

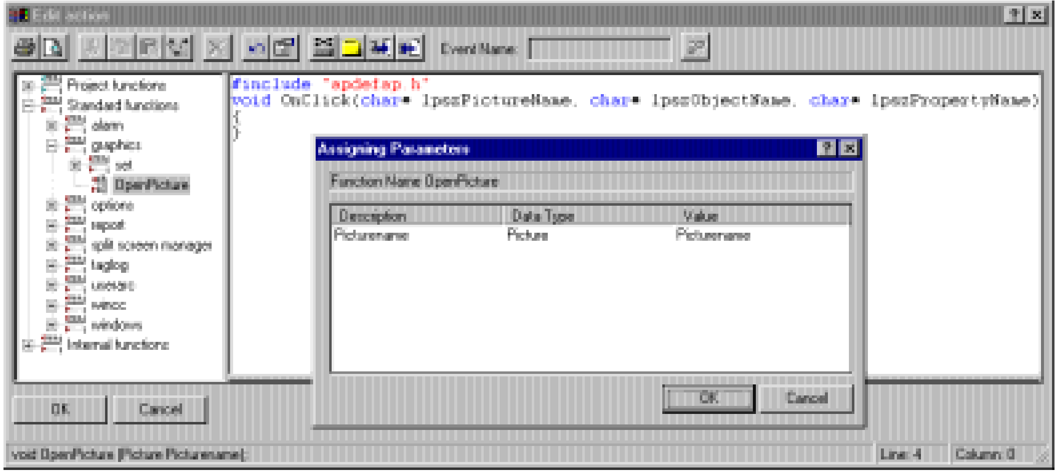
Functions and global actions are edited using the *Global Script* editor.
A distinction is made between project and standard functions. These functions are used in pictures in an object-oriented manner as a *C action* on the object, or in the message system (Alarm Logging).
In contrast, the global actions are not object-oriented and are only edited in certain events (triggering by time or tag).

The differences between these two editors are to be found in the way they are called and in the scope of their functions.

### The editor for actions on the object

For dynamizing the *objects*, the editor is used in the *Graphics Designer*. This editor can only be used for editing the *function body*. The *function header* is generated automatically when the editor is called.
In the picture below, the parameters of the function inserted are assigned in the corresponding dialog box.

Example of editing an action on the object:



```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
}
```

Function Name OpenPicture

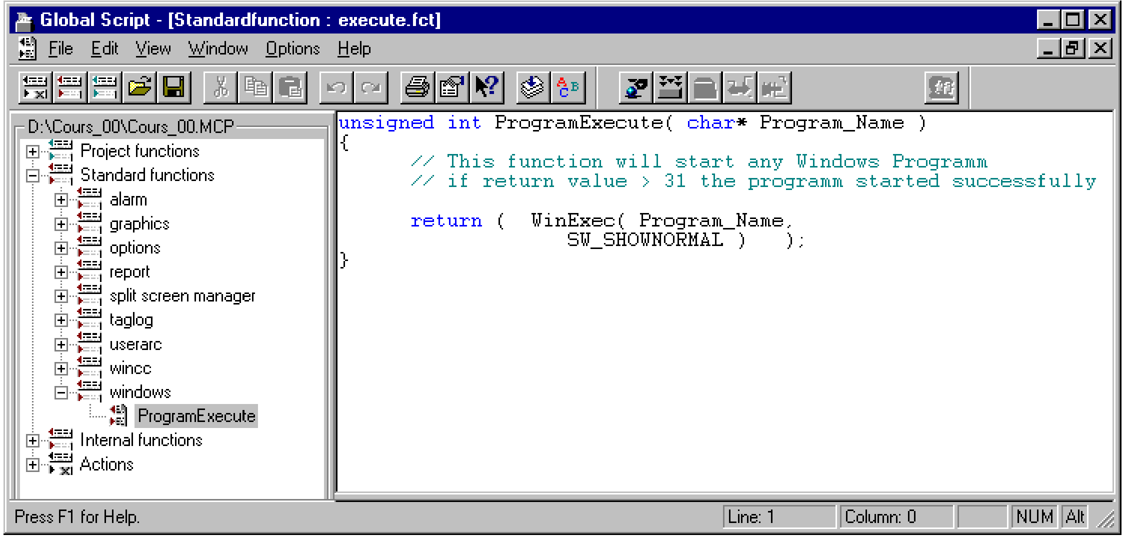
| Description | Data Type | Value |
|---|---|---|
| Picturename | Picture | Picturename |

void OpenPicture (Picture Picturename);

### The Global Script editor

The editor is called from the *Control Center*. It can be used to edit *functions* and *actions*. *Actions* on the *object*, however, cannot be edited with this editor. When *actions* are being edited, the *function header* is protected; that is, it cannot be changed. In the case of *functions*, both the *function header* and the *function body* can be edited.

Example of editing a standard function using the Global Script editor.



```
unsigned int ProgramExecute( char* Program_Name )
{
      // This function will start any Windows Programm
      // if return value > 31 the programm started successfully

      return (  WinExec( Program_Name,
                   SW_SHOWNORMAL )    );
}
```

The control functions listed below apply for both editors.

## Key actions in the editor

| Function | Key action |
|---|---|
| New line | RETURN (ENTER) |
| Delete character to right | DEL |
| Delete character to left | Backspace |
| Move to beginning of line | HOME |
| Move to end of line | END |
| Move to beginning of text | CTRL+HOME |
| Move to end of text | CTRL+END |
| Moving the insertion point | with the cursor keys |
| Cut selected text | CTRL+X |
| Copy selected text | CTRL+C |
| Insert (paste) text from clipboard | CTRL+V |

### Mouse actions in the editor

| Function | Mouse action |
|---|---|
| Select text | with left mouse button |
| Select one word | double-click with left mouse button |
| Reposition insertion point | with left mouse button |

### Other editing functions:

| Function | Explanation |
|---|---|
| Insert mode | Mode in which text is inserted. |
| Selected text | Selected text is replaced by the next character entered using the keyboard. |
| Select an area of text | Position the insertion point at the beginning of the area to be selected, hold down the Shift key and position the insertion point to the end of the area to be selected. |
| Extend a selection | Hold down the Shift key and position the insertion point to the end of the area to be selected. |

## 4.1.3 Creating functions and actions

### Creating a new function

The following steps must be followed to create a standard or project function:

| Step | Procedure: creating a standard or project function |
|---|---|
| 1 | Formulate the function. |
| 2 | Add to the function information. |
| 3 | Compile the function. |
| 4 | Save the function and, if necessary, rename it. |
| 5 | If necessary, generate the header files |

Through following these 5 steps, you have created your standard or project function.

The header file named *apdefap.h* integrates the header file named *ap_glob.h* which comprises the declaration of the *standard functions*.
*apdefap.h* and *ap_glob.h* are located in the *<WinCC Installation Folder>\APLIB* folder.
If you create a new *standard function* and use other *standard functions* there, you must integrate either the *apdefap.h* or the *ap_glob.h* header file.

New *standard functions* are added to the existing *standard functions* in the *<WinCC Installation Folder> \APLIB* on the *server*.
In contrast, the *project functions* defined in the project are stored on the *server* under *<Projectname>\Library* (the project path). The function headers (known as the function prototypes) of the new *project functions* are stored in the declaration file (for C functions), *Ap_pbib.h,* in the *<Projectname>\Library* folder.

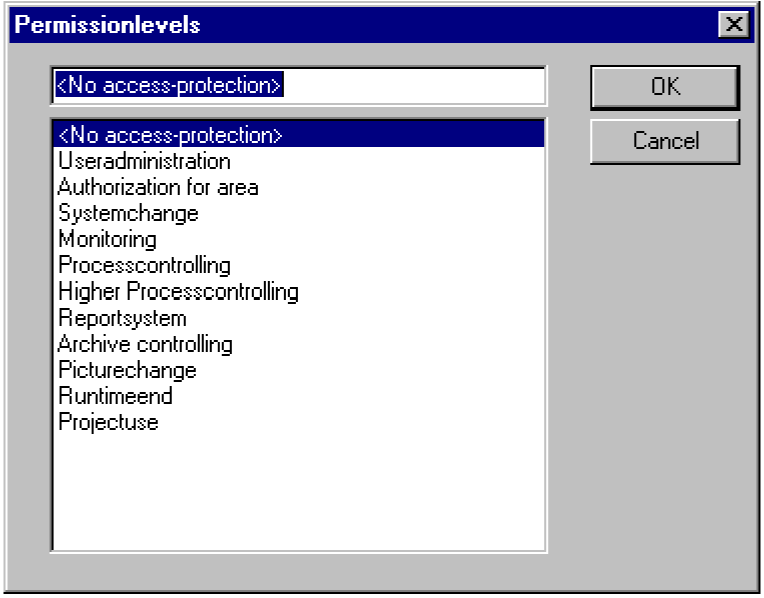
### Creating a new action

The following steps must be followed to create a global action:

| Step | Procedure: creating an action |
|---|---|
| 1 | Formulate the action. |
| 2 | Add to the action information. |
| 3 | Specify the trigger(s) as the triggering event. |
| 4 | Compile the action. |
| 5 | Assign operator authorization (permission). |
| 6 | Save the action and, if necessary, rename it. |

Through following these 6 steps, you have created a global action.

The *Global Script* editor allows you to link the action to operator authorization: *Edit* → *Operator Permission*.



Permissionlevels

<No access-protection>

<No access-protection>
Useradministration
Authorization for area
Systemchange
Monitoring
Processcontrolling
Higher Processcontrolling
Reportsystem
Archive controlling
Picturechange
Runtimeend
Projectuse

When you create a new action, the header file named *apdefap.h* from the project folder *...\<Projectname>\LIBRARY* is integrated automatically.

If there is no *project function* contained in the action, the header file from the *...\APLIB* folder is integrated.
This is how the *standard function* and the *project function* come to exist in the actions.

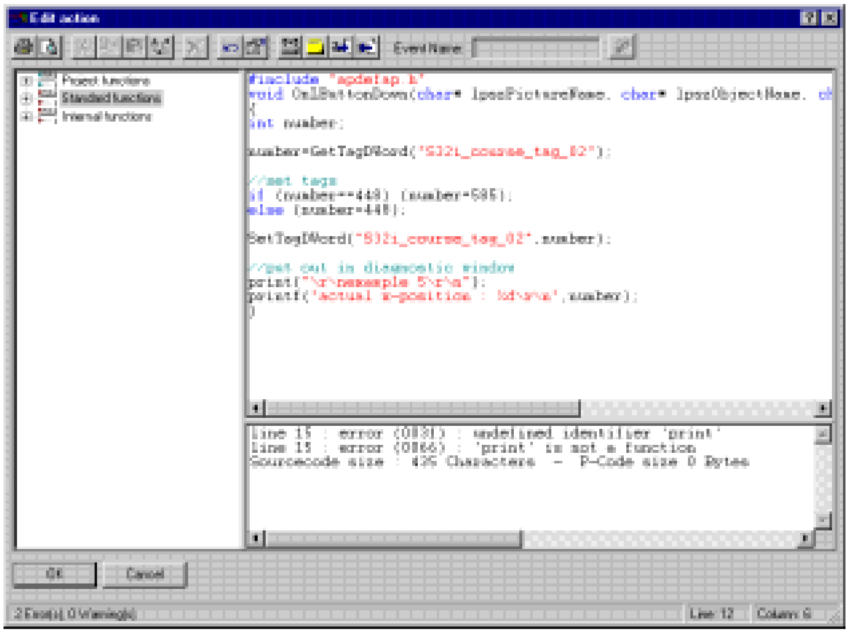
In multi-user systems, a distinction is made between global and local actions.

Global actions are stored in the project folder in such a way that they are not linked with the particular computer (*...\<Projectname>\PAS*). They are active on all computers.
Local actions are stored in the project folder in such a way that they are linked with the computer (*...\<Projectname>\<Computername>\PAS*). This means that local actions are active only the respective computer.

If the *Global Scripts* runtime module is added to the startup list, **all** global actions that belong to the project are activated. This activation takes place on startup in runtime mode.

## 4.1.4 Testing functions and actions

### Error messages



```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, ch
{
int number;

number=GetTagDWord("S321_course_tag_02");

//set tags
if (number==448) (number=585);
else (number=448);

SetTagDWord("S321_course_tag_02",number);

//put out in diagnostic window
print("\r\nexample 5\r\n");
printf("actual x-position : %d\r\n",number);
}
```

```
line 15 : error (0831) : undefined identifier 'print'
line 15 : error (0866) : 'print' is not a function
Sourcecode size : 426 Characters  -  P-Code size 0 Bytes
```

2 Error(s), 0 Warning(s) Line 12 Column 6

You can read up on the syntax messages (*errors* or *warnings*) listed in the *error window* in any book on C or manual on the C compiler (e.g. Microsoft Visual C). These are standard error messages output by every C compiler when the syntax check is run.
*Errors* always lead to the action not being able to be executed, while *warnings* are merely messages which draw your attention to possibility of errors occurring on execution.

**Note:**
You should always take note of the warnings displayed in the error window. A correction will have to be made before the action can be executed free of errors. Warnings cover such errors as incorrect number formats in function calls. Apart from that, eliminating warnings is part and parcel of good programming!

### Eliminating syntax errors

The error line displayed in the *error window* of the C action editor describes the syntax error in detail and specifies the code line. If you 🖱**D** the error line, you automatically position the insertion point to the relevant position in the code. This allows you to analyze and correct the error right at the point where it has been made.

### Test outputs of the C action

You can test the execution of a *C action* with the aid of output instructions, which you must write yourself using the output function of C (*printf*).
These output instructions are output in the diagnostics window when the action is executed. The diagnostics window is configured in the picture using the *Graphics Designer* (*Smart Objects →Application Window →Global Script →GSC_Diagnose*.

## 4.1.5 Importing/exporting functions and actions

Global actions can be exported and imported.
An imported action replaces the action in the active window totally.

*C actions* that have already been formulated can be used again in a number of different ways.

### Storing the C action in the clipboard

If you are going to use the formulated *C action* again immediately in the next action screen (of another property or event), you can export and import the current *C action* or segments of this *C action* by means of the clipboard. You can do this using either the *Copy* and *Paste* buttons in the toolbar or the shortcut key combinations *CTRL C and CTRL V*.

### Exporting C actions

A completely written *C action* can be saved to a file, e.g. *Picturename.act*, at any time - even for copying purposes (copying actions to other properties or events). You do this by selecting the *Export Action* button in the *Global Script* editor and assigning a file name. In this way, you can at any time store basic actions that are used again and again in the picture path of the project (*\<Projectname>\GraCS\*).

### Importing C actions

An existing C action which is made available in the picture path of the project, e.g (*\<Projectname>\GraCS\*), can be inserted into the action screen and modified at any time. Select the *property* you want to dynamize and call the editor for *C actions*. Click the *Import Action* button and select the file name under which the desired action is stored, e.g. *Picturename.act*. In this way, you are able to make use of basiC actions at any time and subsequently adapt them to the dynamics.

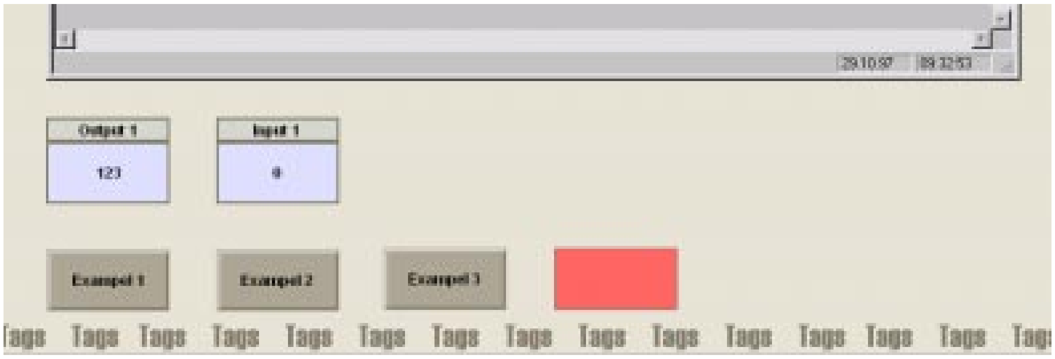
### Storing a configured and linked object as a library object

Objects that have already been preconfigured with (more complex) *C actions* can be stored in the *project library* as a complete graphic object with dynamics, so that it can be used again.
All you have to do is copy the object to the selected *project library* by means of *drag&drop* (selection and copying using the mouse). What's more, the configured actions for the *properties* or *events* are also taken over. An example of objects with *C actions* that have already been stored can be found in the Use Objects folder.

## 4.2 Tags in C

In our project, you can find the examples that relate to the topic of tags by clicking the *Tags* button. By clicking the 🖱**R** , you can display the source code of the example in question.

### Screen for tags


29.10.97
09:32:53
Output 1
123
Input 1
0
Exampel 1
Exampel 2
Exampel 3
Tags Tags Tags Tags Tags Tags Tags Tags Tags Tags Tags Tags Tags Tags Tags

After you have clicked the *Tag* button with the 🖱, you will see the screen depicted above. The buttons in this screen execute the examples described. By clicking the 🖱**R**, you can display the source code of the example in question.

**Before we begin with the examples, let's take a quick theoretical look at the topic of tags.**
Tag consist of a *tag name*, a *tag type*, and the contents. Tags can be likened to a *container*. The tag name we give to the *container* must be unique, since we want to be able to relocate and reuse the *container* and its contents at some later point in time.

Don’t confuse these C tags with WinCC tags. The tags described here are only available within the *C scripts*.

## Tag types in C

C recognizes the following basic types of tag, which correspond to the following value ranges:

| Type | Value range |
|---|---|
| **int** | Integers (numbers with no places after the decimal point), e.g. 2 / -3 / 5 / etc. |
| **char** | A character, e.g. a / k / z , but also the RETURN key |
| **float** | Floating-point numbers (numbers with places after the decimal point), e.g. 3.23 / 4.32 / 3.01 |
| **double** | Floating-point numbers with a range of values twice as large as the float type |

The tag type *double* only differs from the tag type *float* with respect to its range of values. In type double, the numbers can be displayed with greater accuracy (higher resolution).

The tag type *int* can be preceded by one of the keywords *signed* or *unsigned* . If *signed* is positioned before *int*, the program is able to distinguish between signs; if the prefix is *unsigned*, only positive values (+) may be used.
The *int* type can also be preceded by the prefix *long* or *short*.

If nothing else is explicitly specified, all *int* tags are of the type *signed int*.

## Here are the value ranges of the tag types:

| Type | Value range |
|---|---|
| **int** | - 2 147 483 648 through 2 147 483 647 |
| **unsigned int** in WinCC also (**DWORD**) | 0 through 4 294 967 295 |
| **short int** in WinCC also **(SHORT)** | - 32 768 through 32 767 |
| **long int** | - 2 147 483 648 through 2 147 483 647 |
| **unsigned short int** | 0 through 65 535 |
| **unsigned long int** | 0 through 4 294 967 295 |
| **char** | All ASCII characters |
| **unsigned char** | All ASCII characters |
| **float** | -10 ^38 through 10 ^38 |
| **double** | -10 ^308 through 10 ^308 |

In the olden days, there were two types of ASCII characters (displayable characters) to code. The first range of values from -128 through 127, the second range of values from 0 through 255.
To completely avoid this problem of having different types of ASCII value, the *unsigned char* type was introduced. This type only recognized values from 0 through 255. Generally speaking, only *unsigned char* values are used today.

## Parameters for printf

In our example, we output the results in the diagnostics window of the *Global Script* editor by means of *printf*. Here is some information about *printf*.
Structure and example of *printf*:

```
printf("Content of tag: %d\r\n",tag);
```

The *%* is a special character, just as the ** is. The *%* tells the program that the value of a tag is supposed to stand at that point. *%d* displays an *integer tag*.
Lists of tags are separated by commas.

Here are just some of the parameters for *printf:*

| Para-meter | Meaning |
|---|---|
| **%d** | int, short int or char (as a decimal number) |
| **%ld** | Display long int as a number |
| **%c** | Like %d, with char as a character |
| **%x** | Like %d, with char as a hexadecimal character with small a...f |
| **%x** | Like %x, only with large hexadecimal characters A...F |
| **%o** | Like %d, with char as an octal character |
| **%u** | Like %d, nut only unsigned values |
| **%f** | Floating numbers written as floating-point numbers, e.g. 3.43234 |
| **%e** | Float numbers written as exponential numbers, e.g. 23e+432 |
| **%e** | Like %e, but with a large E, e.g. 23E+432 |
| **%g** | Floating numbers written as exponential or floating-point numbers |
| **%s** | char* or char[] |
| **%le** | Display double number |
| **%%** | Is not an instruction, but outputs a % |
| **\n** | Is not an instruction, but outputs a carriage return |
| **\r** | Is not an instruction, but outputs a line feed |
| **\t** | Is not an instruction, but outputs a Tab |

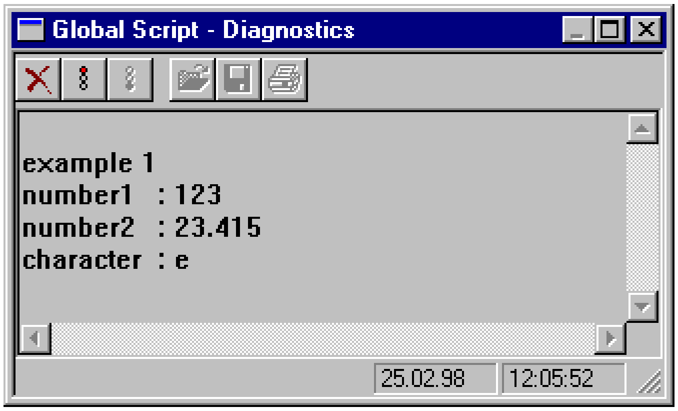
### 4.2.1 Example 1 - Using tag types

In our first example, we will show you how to use the different tag types in a *C action*. The action is configured to the button *example_01 →Events →Mouse →Press left*. Click the button called *Example 1* with the 🖱 and the next script will be edited.

#### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                  char* lpszPropertyName, UINT nFlags, int x, int y)
{
//tag declaration
int number1;
float number2 = 23.415;//set value in declaration
char  character;

//set tag values
number1=123;
character = ’e’;

//output in diagnostics window
printf("\r\nexample 1\r\n");
printf("number1\t : %d\r\n",number1);
printf("number2\t : %.3f\r\n",number2);
printf("character\t : %c\r\n",character);

//set internal tag
SetTagDWord("S32i_course_tag_03",number1);
}
```

Output in the diagnostics window



```
example 1
number1   : 123
number2   : 23.415
character : e
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- In the *Declaration* section, the tags (*number1* , *number2* , and *character* ) are created. This is done by means of the notational form *tag type tag name*, and each line is concluded by *;*.
- In tag *number2*, the value is assigned together with the declaration. This is done by means of the notational form *tag type tag name = value* that is to be assigned.
- The remaining tags have the values assigned to them. We now have the *tags* with a specific variable (*int*, *float*, and *char*).
- The *tags* now have the values (*123*, *23.415*, and *e*). The numbers to the left and right of the decimal point are separated by a point.
- The values are then output in the diagnostics window by means of the *printf* function.

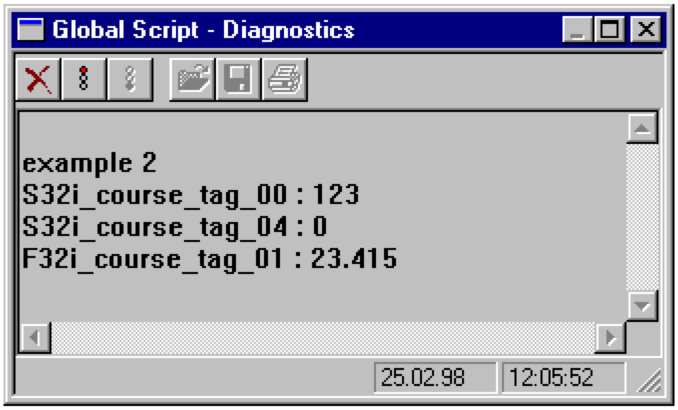
### 4.2.2 Example 2 - C tags in connection with WinCC tags

In our second example, we will show you how to use C tags in connection with WinCC tags, in this case, *internal tags*. The *external tags* (process tags from the tag management) are read and modified in the same way.
This example shows how a tag (internal or external) is set to a defined value *(SetTag)* and how the current value *(GetTag)* of a tag is read.
The action is configured to the button *example_02 →Events →Mouse →Press left*. Click the button called *Example 2* with the 🖱 and the next script will be edited.

#### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                   char* lpszPropertyName, UINT nFlags, int x, int y)
{
//tag declaration
int number1;
int   number2;
float number3;

//set internal tags
SetTagDWord("S32i_course_tag_00",123);
SetTagFloat("F32i_course_tag_01",23.415);

//read internal tag values
number1=GetTagDWord("S32i_course_tag_00");
number2=GetTagDWord("S32i_course_tag_04");
number3=GetTagFloat("F32i_course_tag_01");

//output in diagnostics window
printf("\r\nexample 2\r\n");
printf("S32i_course_tag_00 : %d\r\n",number1);
printf("S32i_course_tag_04 : %d\r\n",number2);
printf("F32i_course_tag_01 : %.3f\r\n",number3);
}
```

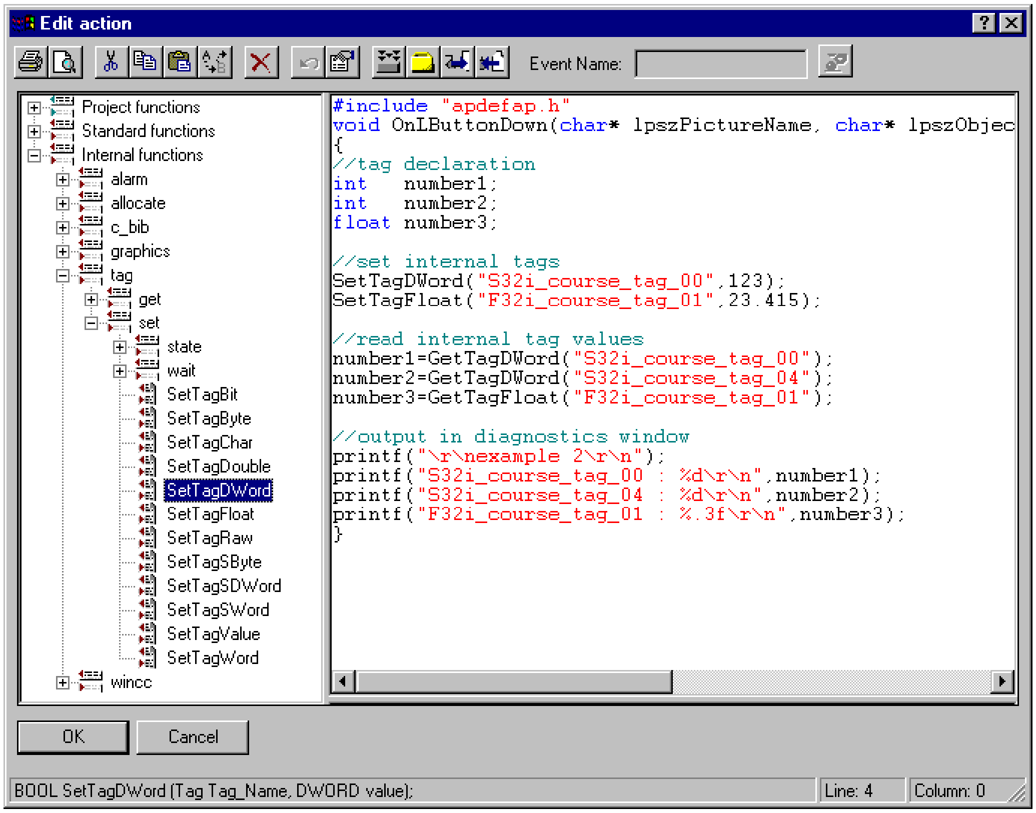
Output in the diagnostics window



```
example 2
S32i_course_tag_00 : 123
S32i_course_tag_04 : 0
F32i_course_tag_01 : 23.415
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags.
- In the *Set internal tags* section, the internal WinCC tags (*S32i_course_tag_00* and *F32i_course_tag_01*) are supplied with values (*123* and *23.415* ) by means of the WinCC functions, *SetTagDWord* and *GetTagFloat*, provided for this purpose.



```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjec
{
//tag declaration
int    number1;
int    number2;
float number3;

//set internal tags
SetTagDWord("S32i_course_tag_00",123);
SetTagFloat("F32i_course_tag_01",23.415);

//read internal tag values
number1=GetTagDWord("S32i_course_tag_00");
number2=GetTagDWord("S32i_course_tag_04");
number3=GetTagFloat("F32i_course_tag_01");

//output in diagnostics window
printf("\r\nexample 2\r\n");
printf("S32i_course_tag_00 : %d\r\n",number1);
printf("S32i_course_tag_04 : %d\r\n",number2);
printf("F32i_course_tag_01 : %.3f\r\n",number3);
}
```

BOOL SetTagDWord (Tag Tag_Name, DWORD value); Line: 4 Column: 0

- In the next section, we use the appropriate WinCC function (*GetTagDWord, GetTagFloat)* to read the value from the internal WinCC tags (*S32i_course_tag_00, S32i_course_tag_01*) and assign these tags to the C tags (*number1, number3*).
- The value of the tag (*S32i_course_tag_04*) is read from Input field1 and assigned to the C tag (*number2)*. This value can be changed. Click *Input field1* with the 🖱, enter the value and complete your input by pressing *Return*.
- The values are then output in the diagnostics window by means of the *printf* function.

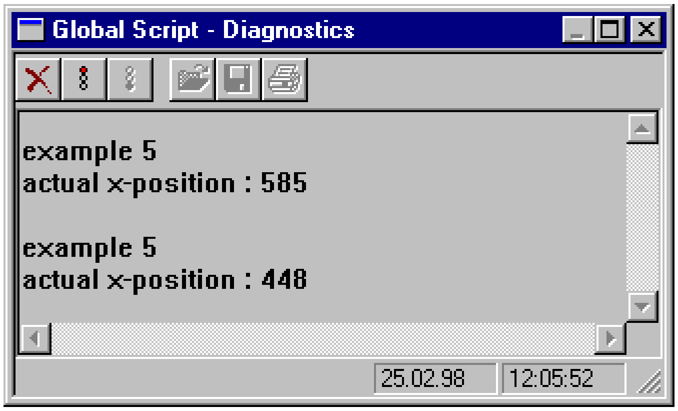
### 4.2.3 Example 3 - Using tags

In our third example, we will show you another way to use a tag. We will change the content of a tag by clicking in it with the mouse. This tag is linked with the X position of an object. When the tag is changed, the object is moved.
The action is configured to the button *example_03* → *Events* → *Mouse* → *Press left*. Click the button called *Example 3* with the 🖱 and the next script will be edited.

#### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
int number;

number=GetTagDWord("S32i_course_tag_02");

//set tags
if (number==448) (number=585);
else (number=448);

SetTagDWord("S32i_course_tag_02",number);

//output in diagnostics window
printf("\r\nexample 5\r\n");
printf("actual x-position : %d\r\n",number);
}
```

Output in the diagnostics window



```
example 5
actual x-position : 585

example 5
actual x-position : 448
```

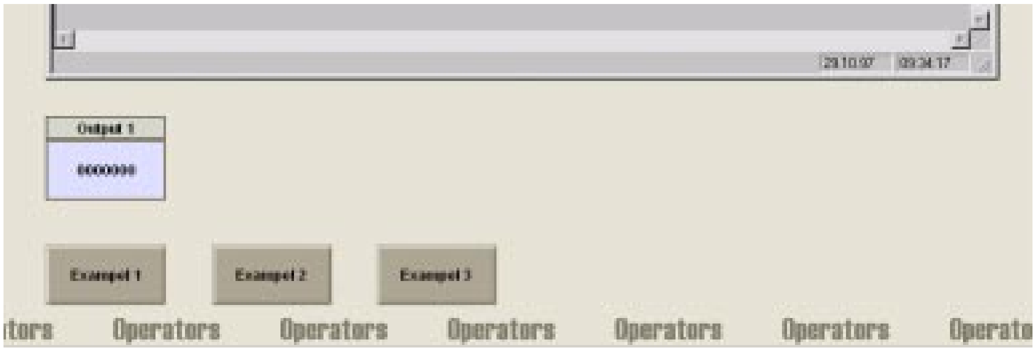
## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- The declaration of the tag.
- In the *set tags* section, the value of the tag is changed in the *if condition*.
- The new value is assigned to the internal WinCC tag (*S32i_course_tag_02*) using the WinCC function, *SetTagDWord*, provided for this purpose.
- In the object *rectangle_01*, this tag is linked to *PositionX*. The position is updated when this tag, *S32i_course_tag_02*, is changed.
- The value is then output in the diagnostics window by means of the *printf* function.

## 4.3 Operators and mathematical functions in C

In our project, you can find the examples that relate to the topic of operators by clicking the *Operators* button. By clicking the 🖱**R**, you can display the source code of the example in question.

### Screen for operators



After you have clicked the *Operators* button with the 🖱, you will see the screen depicted above. The buttons in this screen execute the examples described. By clicking the 🖱**R**, you can display the source code of the example in question.

**Before we begin with the examples, let's take a quick theoretical look at the topic of operators.**
The *operators* control what is to happen to the *tags* and *constants*.
You use *operators* to link *tags* and *constants* to one another, and to derive new contents for *tags* from this.

Operators are divided up into different types. Those described below are the ones that are of most importance to us:

### Arithmetics

| Type | Description |
|---|---|
| + | addition |
| - | subtraction |
| * | multiplication |
| / | division |
| ++ | incrementation |
| -- | decrementation |

The arithmetic operators are, e.g. in the *Dynamic Dialog*, required to set up a dependency between several tags (e.g. Motor Status1 and Motor Status2). The arithmetic operators are also used to formulate formulae within *C actions*.

## Logic, comparison

| Type | Description |
|---|---|
| > | greater than |
| >= | greater than or equal to |
| == | equal to |
| != | not equal to |
| <= | less than or equal to |
| < | less than |
| && | logic AND operation |
| \|\| | logic OR operation |
| ! | logic reversal |

The arithmetic and logic operators are, e.g. in the *Dynamic Dialog*, required to set up a dependency between several tags (e.g. Motor Status and !Error).
You will find the logic operators again in the Abfragen of *C actions*.

## Bit manipulation

| Type | Description |
|---|---|
| & | AND operation of bits |
| \| | OR operation of bits |
| ^ | Exclusive OR (EXOR) operation of bits |
| << | Shifting of bit to the left |
| >> | Shifting of bit to the right |

The bit-by-bit operators are, e.g. in *C actions*, used to query bits or to set individual bits in a data word (e.g. set Motor Status bit to work in Position 1, Motor Status | 0x0002).

## 4.3.1 Boolean algebra

### Logic functions

In mathematics, systems of logic tags that are linked by logic functions are known as **Boolean algebra**.
When *C actions* are being created, logic tags are in many applications linked to new tags by means of logic functions.

A logic function can be viewed in a **function table**. A function table can also be termed a **Truth table**.
If you want to view logic elements, make use of the special symbols, that are standardized. These symbols are also known as **gates**.

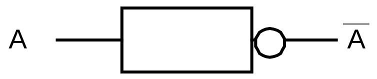
### 4.3.1.1 Basic logic functions

Described below are the most important of the basic logic functions. Logic elements are in many places referred to by their English names.

### NOT gate

The **NOT gate** (or NOT element) negates the input signal. This negation is depicted by means of the circle at the exit of the box.

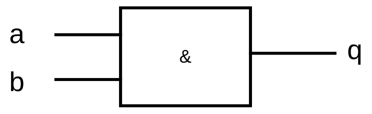
| a | Not |
|---|---|
| 0 | 1 |
| 1 | 0 |



### logic AND operation

The **AND operation** is only 1 if both tags have the value 1.

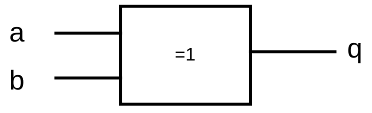
| a | b | And |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



### OR operation

The **OR function** is 1 if the value of at least one tag is 1.

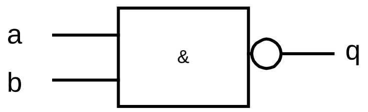
| a | b | Or |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



## NAND operation

An AND operation followed by a negation is known as a **NAND operation**. This function has the value 0 if both tags have the value 1.

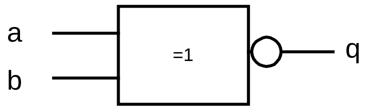
| a | b | Nand |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



## NOR operation

An OR operation followed by a negation is known as a **NOR operation**. This function only has the value 1 if both tags have the value 0.

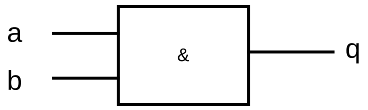
| a | b | Nor |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



## Exclusive OR (EXOR) operation

This function results in the value 1 if one of the two tags (either of them) has the value 1. This function is known as the **EXOR function**.

| a | b | EXOR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



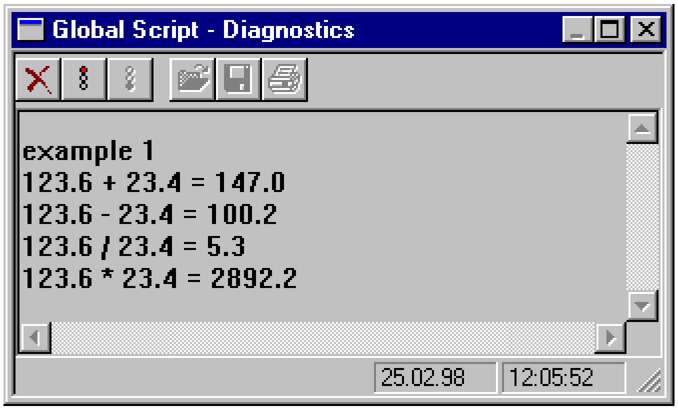
## 4.3.2 Example 1 - Using operators of the basic arithmetic operations

In Example 1, we will show you how to use the different operators of the Basic arithmetic operations in a *C action*.
The action is configured to the button *example_01* → *Events* → *Mouse* → *Press left*. Click the button called *Example 1* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
float a,b;
float result1,result2,result3,result4;

a=123.6;
b=23.4;

result1=a+b;    // add
result2=a-b;    // subtract
result3=a/b;    // divide
result4=a*b;    // multiply

//output in diagnostics window
printf("\r\nexample 1\r\n");
printf("%.1f + %.1f = %.1f\r\n",a,b,result1);
printf("%.1f - %.1f = %.1f\r\n",a,b,result2);
printf("%.1f / %.1f = %.1f\r\n",a,b,result3);
printf("%.1f * %.1f = %.1f\r\n",a,b,result4);
}
```

Output in the diagnostics window



```
example 1
123.6 + 23.4 = 147.0
123.6 - 23.4 = 100.2
123.6 / 23.4 = 5.3
123.6 * 23.4 = 2892.2
```

25.02.98 12:05:52

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags.
- The tags have the values assigned to them.
- Execution of the mathematical operations.
- The values are then output in the diagnostics window by means of the *printf* function.

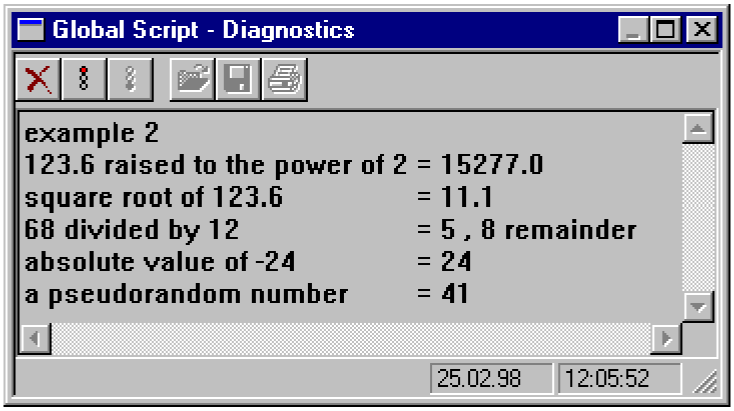
## 4.3.3 Example 2 - Mathematical functions

In Example 2, we will show you how to use mathematical functions in a *C action*.
The action is configured to the button *example_02 →Events →Mouse →Press left*. Click the button called *Example 2* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
float  a;
int    b,c,d;
div_t  result5;
double result1,result2;
int    result3,result4;

a=123.6;
b=-24;
c=68;
d=12;

result1=pow(a,2);
result2=sqrt(a);
result3=abs(b);
result4=rand();
result5=div(c,d);

//output in diagnostics window
printf("\r\nexample 2\r\n");
printf("%.1f raised to the power of 2\t = %.1f\r\n",a,result1);
printf("square root of %.1f\t = %.1f\r\n",a,result2);
printf("%d divided by %d\t\t = %d , %d
remainder\r\n",c,d,result5.quot,result5.rem);
printf("absolute value of %d\t = %d\r\n",b,result3);
printf("a pseudorandom number\t = %d\r\n",result4);
}
```

### Output in the diagnostics window



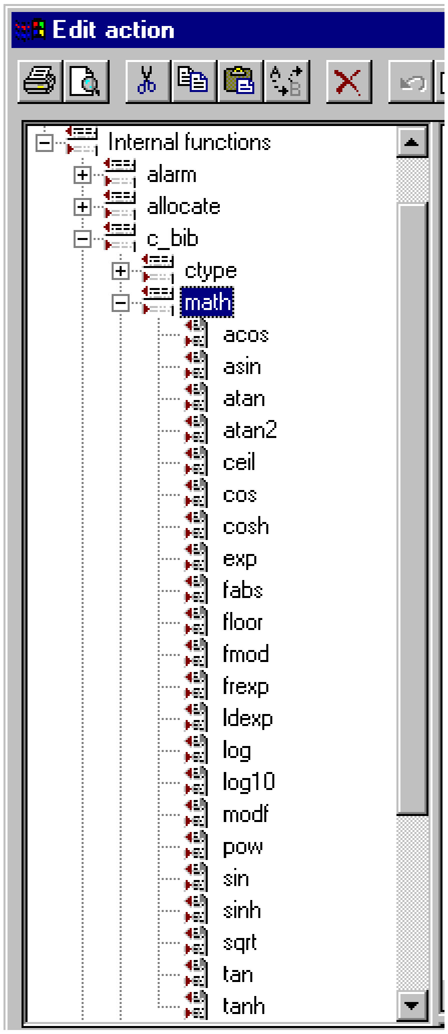
```
example 2
123.6 raised to the power of 2 = 15277.0
square root of 123.6           = 11.1
68 divided by 12               = 5 , 8 remainder
absolute value of -24          = 24
a pseudorandom number          = 41
```

25.02.98 12:05:52

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags.
- The tag has the relevant values assigned to it.
- Execution of the mathematical functions. You can find these functions under → *Internal functions* → *c_bib* → *math*
- The values are then output in the diagnostics window by means of the *printf* function.

Mathematical functions can be used whenever they are available for selection. For example, in the *Dynamic Dialog* when formulating the inquiry or in the *C actions*.



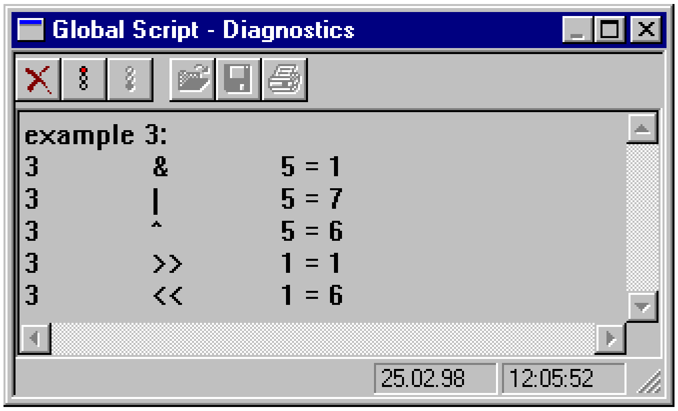
## 4.3.4 Example 3 - Operators for bit operations

In Example 3, we will show you how to use operators for bit operations in a *C action*. The action is configured to the button *example_03 →Events →Mouse →Press left*. Click the button called *Example 3* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
int a,b,c;
int result1,result2,result3,result4,result5;

a=3;  // binary 011
b=5;  // binary 101

result1=a&b; // AND
result2=a|b; // inclusive OR
result3=a^b; // exclusive OR
result4=a>>1;// shift right
result5=a<<1;// shift left

//write result of OR in output_01
SetTagDWord("S32i_course_op_03",result2);

//output in diagnostic window
printf("\r\nexample 3:\r\n");
printf("%d\t&\t%d = %d\r\n",a,b,result1);
printf("%d\t|\t%d = %d\r\n",a,b,result2);
printf("%d\t^\t%d = %d\r\n",a,b,result3);
printf("%d\t>>\t1 = %d\r\n",a,result4);
printf("%d\t<<\t1 = %d\r\n",a,result5);
}
```

Output in the diagnostics window



```
example 3:
3        &        5 = 1
3        |        5 = 7
3        ^        5 = 6
3        >>       1 = 1
3        <<       1 = 6
```

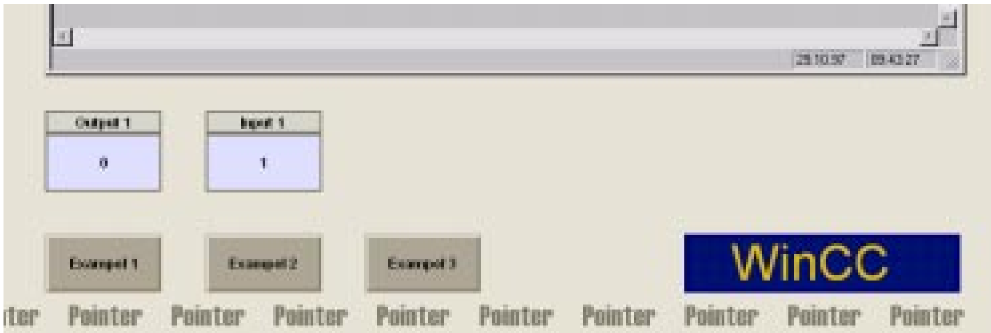
## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags and assignment of the values.
- Execution of the bit operations.
- Output of the results of the *inclusive OR* operation in *Output Field 1*.
- The values are then output in the diagnostics window by means of the *printf* function.

# 4.4 Pointers in C

In our project, you can find the examples that relate to the topic of pointers by clicking the *Pointers* button. By clicking the 🖱**R**, you can display the source code of the example in question.

## Screen for pointers


25.10.97
09:43:27
Output 1
0
Input 1
1
Example 1
Example 2
Example 3
WinCC
ter Pointer Pointer Pointer Pointer Pointer Pointer Pointer Pointer Pointer

After you have clicked the *Pointers* button with the 🖱, you will see the screen depicted above. The buttons in this screen execute the examples described. By clicking the 🖱**R**, you can display the source code of the example in question.

**Before we begin with the examples, let's take a quick theoretical look at the topic of pointers.**
The pointers are an essential component of the programming languages C and C++. Even in German the pointer is frequently referred to as a **Pointer**.

## The definition and notation of pointers

A pointer does not contain the value of a tag; it merely **points to the address** under which the value of the tag is stored.
**The pointer must have the same data type as the tag to which it points!**
In principle, a tag is declared by a **prefixed asterisk** acting as a pointer.
The tag value to which a pointer points is accessed by means of the *** operator**. This is also known as the contents operator.
This, however, must not be confused with the multiplier. These are two separate operators! Depending on the context, the compiler decides on whether it is dealing with a pointer operator or a multiplication.

## Part of a program to explain pointers

```
{
int number1;
int *pointer1;

number1  = 123;
pointer1 = &number1;
}
```

This example is the best way of explaining what a pointer is:

- We create the tag *number1* and the pointer called *pointer1*.
- Tag *number1* is assigned the value *123*.
- The pointer called *pointer1* is assigned **the address** of tag *number1*.
- This is performed by the *address operator &*!
- *pointer1* now contains the address of tag *number1*.

The advantage of this is that the pointers contain only an address and no tags. This makes them extremely flexible and quick.

## String as a pointer

If a string (character string) is generated by a pointer, it is saved randomly at any free point in the memory. The pointer points to the first element of the string.
This does indeed work, but it is a **poor and danger way of going about programming**!

A string of characters is identified in C by the start of the string (start address of the string) and by the end of the string (null character, \0):

| | | | | | |
|---|---|---|---|---|---|
| W | i | n | C | C | \0 |

During WinCC configuration, you will find pointers, for example, in connection with the function parameters:
lpszPictureName is, for example, a pointer which points to the beginning of the name of the picture (e.g. Start):

| | | | | | |
|---|---|---|---|---|---|
| S | t | a | r | t | \0 |

lpszObjectName is a pointer which points to the beginning of the object name (e.g. Circle1)
.

| | | | | | | |
|---|---|---|---|---|---|---|
| C | r | I | i | S | 1 | \0 |

The true benefit of pointers only really becomes clear when they are used in **fields**. These fields are often referred to as **arrays**.
Groups or strings of tags of one type can be referred to as fields.

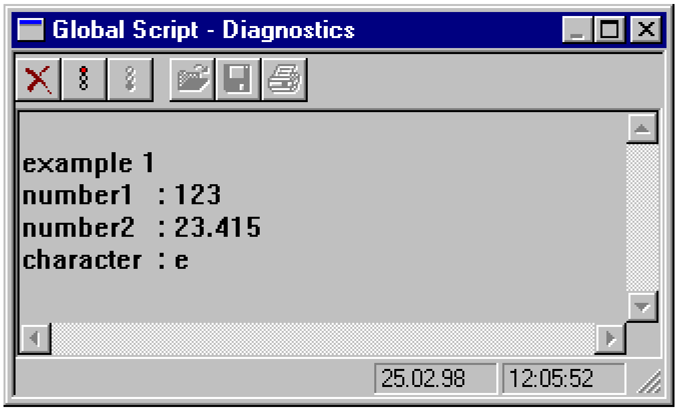
## 4.4.1 Example 1 - Pointers

In Example 1, we will show you how to use pointers in C.
The action is configured to the button *example_01 →Events →Mouse →Press left*. Click the button called *Example 1* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                  char* lpszPropertyName, UINT nFlags, int x, int y)
{
//declaration of tags
int number1;
float number2;
char  character;

//declaration of pointers
int *pointer1;
float *pointer2;
char  *pointerc;

//set tag values
number1=123;
number2  =23.415;
character = ’e’;

//set addresses
pointer1=&number1;
pointer2=&number2;
pointerc=&character;

//set internal tag
SetTagDWord("S32i_course_point_00",*pointer1);

//output in diagnostics window
printf("\r\nexample 1\r\n");
printf("number1 = %d\r\n",GetTagDWord("S32i_course_point_00"));
printf("number2 = %.3f\r\n",*pointer2);
printf("character  = %c\r\n",*pointerc);
}
```

## Output in the diagnostics window



```
example 1
number1   : 123
number2   : 23.415
character : e
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags.
- Declaration of the pointers.
- Assign values to the tags.
- In this section, the *pointers* are assigned the *addresses* of the corresponding tags.
- The values are then output in the diagnostics window by means of the *printf* function.

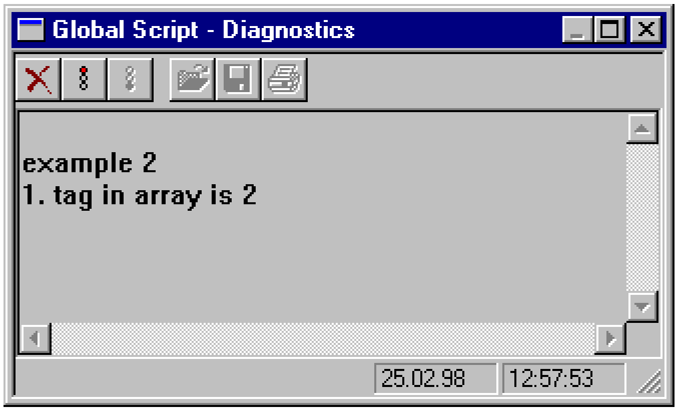
## 4.4.2 Example 2 - Pointers in connection with WinCC tags

In Example 2, we will show you how to use pointers in connection with *internal tags*. The action is configured to the button *example_02 →Events →Mouse →Press left*. Click the button called *Example 2* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
//declaration of array
int array[4];

int index;
int *pointer;

//set tag values
array[0]=2;
array[1]=4;
array[2]=6;
array[3]=8;

//pointer to first tag of array
pointer=&array[0];

//read index from input_01
index=GetTagDWord("S32i_course_point_04");
index--;

//set internal tag
SetTagDWord("S32i_course_point_03",*(pointer+index));

//output in diagnostics window
printf("\r\nexample 2\r\n");
printf("%d. tag in array is %d\r\n",(index+1),*(pointer+index));
}
```

## Output in the diagnostics window



```
example 2
1. tag in array is 2
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of a field (four integers).
- Declaration of a tag.
- Declaration of a pointer.
- Assign values to the fields.
- In this section, the pointer is assigned the address of the first element of the field.
- Read the desired index out of Input Field 1 and adjust to C format (the first element in a field has the index 0).
- The current index is added to the pointer which points to the first position in the field. The pointer that has been calculated in this way is then used to read out the desired element.
- The values are then output in the diagnostics window by means of the *printf* function.

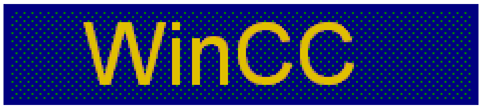
### 4.4.3 Example 3 - Pointers in connection with string processing

In Example 3, we will show you how to use pointers in C. We will alter the way a text is presented by turning it into a toolbar.
The action is configured on the static text with the name *static text_01* → *Properties* → *Font*→ *Text*. By clicking the *Example 3* button with the 🖱, the tag *BINi_varia_point_05* is set to *1* by means of a direct link. The following script is processed every 250 ms. The if condition is, however, only executed when the *Example 3* button is clicked.

#### C action

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char*
                lpszPropertyName)
{
//declaration of static tag
static int i = 10;

char *str = "              WinCC";

if (GetTagBit("BINi_varia_point_05")) {
       if (i>19) (i=0);//check limit
       i++;//inc
       }
//return string
return(str+i);
}
```

The effects are immediately visible in the picture:



WinCC

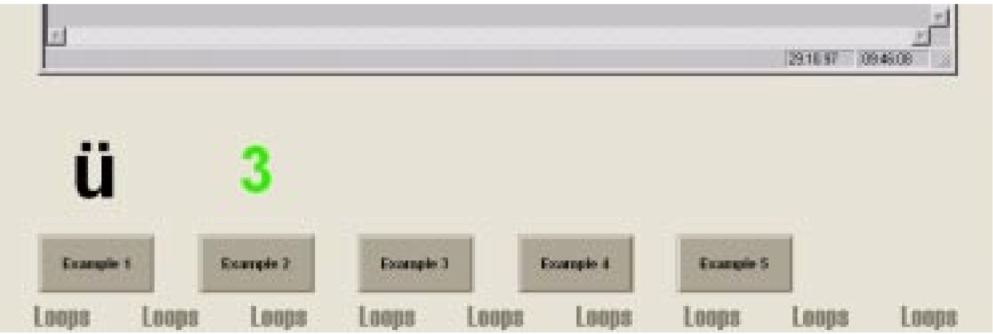
Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the tag of type *static in.* Tags with static prefixed to them retain their value for length of time it takes to open the picture. Initialization only takes place the first time the action is called.
- Declaration of the pointer and value assignment.
- Check whether the *Example 3* button is clicked.
- Check whether the position exceeds the maximum length of characters. If this is the case, the procedure starts from the beginning again.
- Increment the position.
- The position of the pointer which points to the string is incremented and the result is returned on *return*.

## 4.5 Loops and conditional statements in C

In our project, you can find the examples that relate to the topic of loops by clicking the *Loops* button. By clicking the 🖱**R**, you can display the source code of the example in question.

### Screen for loops



After you have clicked the *Loops* button with the 🖱, you will see the screen depicted above. The buttons in this screen execute the examples described. By clicking the 🖱**R**, you can display the source code of the example in question.
*Ü* and *3* are further examples of loops.

### Loops

**Before we begin with the examples, let's take a quick theoretical look at the topic of loops.**
In principle, there are two types of loop: the *pre-checking* and the *post-checking* loop.
The *pre-checking loops* check whether or not a condition is true **before** the body of the loop and the *post-checking loops check* **after** the body of the loop. That is, post-checking loops are run through at least once.

If the body of the loop consists of an instruction, the curly brackets can be left out.

### while

```
while ([condition])
      { body of loop }
```

The body of the loop is repeated for as long as the *condition* is satisfied.

### do - while

```
do
      { body of loop }
while [condition] ;
```

The body of the loop is run through at least once and then repeated until the condition is no longer satisfied.

**Note:**
The while condition statement in *post-checking loops* is **always** followed by a semicolon; in contrast, there must be **no** semicolon in *pre-checking loops*.

for

```
for ( [ Init ] ; [ condition ] ; [ statement ] )
      { body of loop }
```

This all looks very cryptic. So, we rewrite the whole thing a bit differently and end up with the following equivalent expression:

```
Init;
while [condition]
      { body of loop }
      [statement];
      }
```

**Note:**
Please note that loops can also turn into endless loops if the condition is **always** satisfied.
For this reason, test your *C actions* immediately after formulating them, to make sure that the sequence is error-free.

## Conditional statements

In loops, the body of the loop is processed for as long as the condition is true.
In conditional statements, the statement is processed precisely once if the condition is true.
When making comparisons, always pay attention to the tag types!

if

```
if  ( [condition1] )
      { body1 }
else
      { body2 }
```

If the condition is true (not 0), Instruction Body1 is processed.
If the condition is not true (the value is 0), the alternative in Instruction Body2 is processed.
*else* can also be left out, in which case no alternative is executed.

Since *if* simply checks the numeric value of an expression, certain abbreviations are possible. The two statements below are therefore identical.

```
if  (condition)
if  (condition != 0)
```

You can also combine a number of inquiries:

```
if  ( [condition1] )
      { body1 }
else if ( [condition2] ) // second if inquiry (nested)
      { body2 }
else
      { body3 }
```

In this case, the first condition is checked first. If it is not true, the second condition is checked. If this condition is true, Instruction Body2 is executed. If neither of the conditions is true, Instruction Body3 is executed.

**Note:**
If a condition is executed, the other cases are no longer processed.

If you have more than two inquiries, it is better to use switch case.

## switch & case

```
switch ( [tag] ) {
      case [Term1] :
            Action1;
            break;
      case [Term2] :
            Action2;
            break;
      default:
            Action3;
            break;
}
```

In this case, a tag is checked to see whether it matches up. The *switch* statement is used to specify the tag to be checked. First of all, the tag is checked to see whether it corresponds to the *Term1*; if it does, *Action1* is executed. This action can contain as many instructions as you want.
The actions are concluded by a break!

The keyword *default* is not followed by a comparison. These actions are only executed if none of the *case* conditions are satisfied. That is, a default statement is executed as long as no other case arises.
The *break* statement can be left out with *default* (is always the last statement), but this not considered good programming.

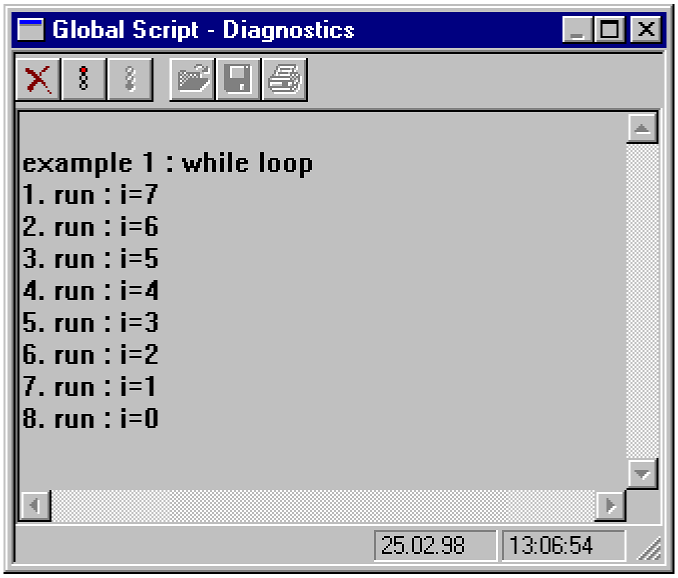
### 4.5.1 Example 1 - while loop

In Example 1, we will show you how to use the *while* loop in a *C action*.
The action is configured to the button *example_01* →*Events* →*Mouse* →*Press left*. Click the button called *Example 1* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                   char* lpszPropertyName, UINT nFlags, int x, int y)
{
//while loop
int i,j;

printf("\r\nexample 1 : while loop\r\n");

j=0;  // execution count
i=8;  // tag used for loop action

while (i>0) {
    j++;
    i--;
    printf("%d. run : i=%d\r\n",j,i);
    }//while
}
```

Output in the diagnostics window



```

example 1 : while loop
1. run : i=7
2. run : i=6
3. run : i=5
4. run : i=4
5. run : i=3
6. run : i=2
7. run : i=1
8. run : i=0
```

25.02.98 13:06:54

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags. *i* is required for the loop function, *j* only for outputting the passes.
- Output in the diagnostics window what comes next.
- Initialize the tags for loop processing.
- The *while* loop is processed for as long as *i>0*.
- Tag *j* is ented; *i* is decremented.
- The current value of the tag is then output in the diagnostics window by means of the *printf* function.

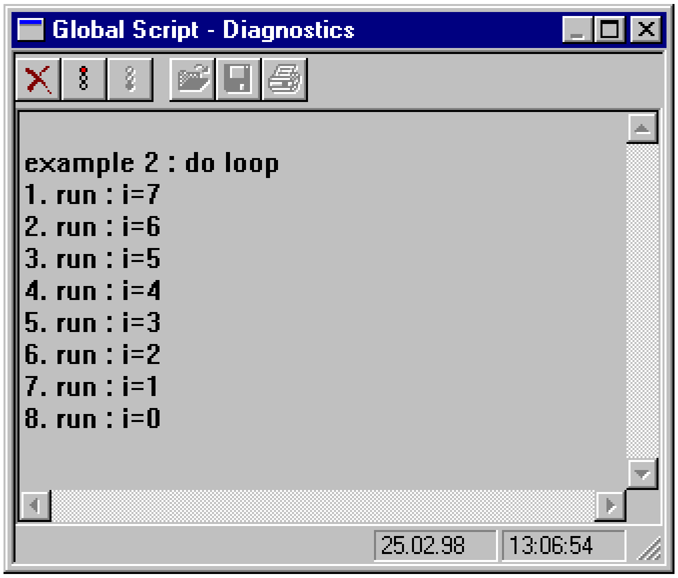
### 4.5.2 Example 2 - do - while loop

In Example 2, we will show you how to use the *do - while* loop in a *C action*.
The action is configured to the button *example_02 →Events →Mouse →Press left*. Click the button called *Example 2* with the 🖱 and the next script will be edited.

#### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                   char* lpszPropertyName, UINT nFlags, int x, int y)
{
//do loop
int i,j;

printf("\r\nexample 2 : do loop\r\n");

j=0;  // execution count
i=8;  //tag used in loop action

do  {
    j++;
    i--;
    printf("%d. run : i=%d\r\n",j,i);
    }//do
while (i>0);
}
```

Output in the diagnostics window



```
example 2 : do loop
1. run : i=7
2. run : i=6
3. run : i=5
4. run : i=4
5. run : i=3
6. run : i=2
7. run : i=1
8. run : i=0
```

25.02.98 13:06:54

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags. *i* is required for the loop function, *j* only for outputting the passes.
- Output in the diagnostics window what comes next.
- Initialize the tags for loop processing.
- The *do - while* loop is processed for as long as *i>0* (at least 1 pass).
- Tag *j* is ented; *i* is decremented.
- The current value of the tag is then output in the diagnostics window by means of the *printf* function.

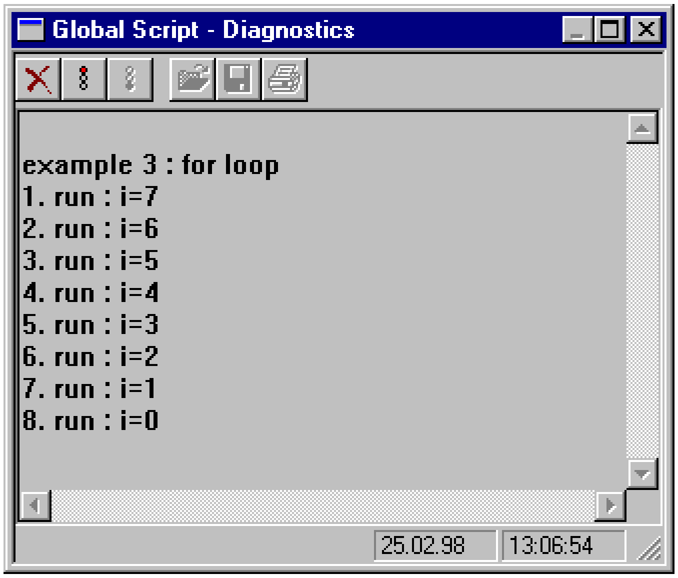
### 4.5.3 Example 3 - for loop

In Example 3, we will show you how to use the *for* loop in a *C action*.
The action is configured to the button *example_03 →Events →Mouse →Press left*. Click the button called *Example 3* with the 🖱 and the next script will be edited.

## C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
//for loop
int i,j;

printf("\r\nexample 3 : for loop\r\n");

j=0;//execution count

for(i=7;i>=0;i--) {
    j++;
    printf("%d. run : i=%d\r\n",j,i);
    }//for
}
```

Output in the diagnostics window



```
example 3 : for loop
1. run : i=7
2. run : i=6
3. run : i=5
4. run : i=4
5. run : i=3
6. run : i=2
7. run : i=1
8. run : i=0
```

25.02.98 13:06:54

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags. *i* is required for the loop function, *j* only for outputting the passes.
- Output in the diagnostics window what comes next.
- The tag for the loop counter is initialized.
- The *for* loop is processed for as long as *i>=0*. The start value of *i* is *7*.
- Tag *j* is incremented during the pass; *i* is decremented after the pass.
- The current value of the tag is then output in the diagnostics window by means of the *printf* function.

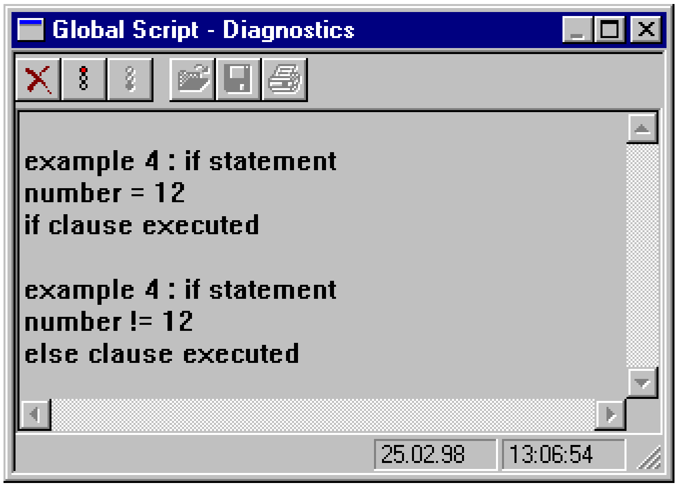
## 4.5.4 Example 4 - Conditional statement with if

In Example 4, we will show you how to use the conditional *if* statement in a *C action*. The action is configured to the button *example_04 →Events →Mouse→ Press left*. Click the button called *Example 4* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                   char* lpszPropertyName, UINT nFlags, int x, int y)
{
//if statement
static int number=12;

printf("\r\nexample 4 : if statement\r\n");

if (number==12) {
    printf("number = 12\r\n");
    printf("if clause executed\r\n");
    number=11;
    }//if
else {
    printf("number != 12\r\n",number);
    printf("else clause executed\r\n");
    number=12;
    }//else
}
```

Output in the diagnostics window



```

example 4 : if statement
number = 12
if clause executed

example 4 : if statement
number != 12
else clause executed
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tag as *static*. The value of this tag is retained until the next change of picture.
- Output in the diagnostics window what comes next.
- Processing of the *inquiry*.
- If the condition *(number==12)* is satisfied, the *if* branch is processed. If the condition is not satisfied, the *else* branch is processed.
- The current result of the inquiry is output in the diagnostics window by means of the *printf* function.

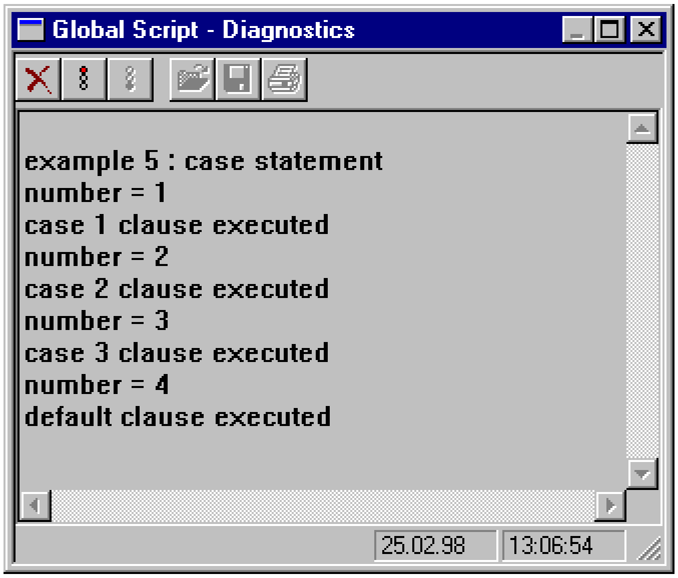
## 4.5.5 Example 5 - Conditional statement with switch and case

In Example 5, we will show you how to use the *switch case* statement in a *C action*.
The action is configured to the button *example_05 →Events →Mouse →Press left*. Click the button called *Example 5* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
//case statement
static int number=1;

if (number==1) printf("\r\nexample 5 : case statement\r\n");

switch (number) {
    case 1:{
    printf("number = %d\r\n",number);
    printf("case 1 clause executed\r\n");
    number++;
    break;
    }//case1
    case 2: {
    printf("number = %d\r\n",number);
    printf("case 2 clause executed\r\n");
    number++;
    break;
    }//case2
    case 3: {
    printf("number = %d\r\n",number);
    printf("case 3 clause executed\r\n");
    number++;
    break;
    }//case3
    default : {
    printf("number = %d\r\n",number);
    printf("default clause executed\r\n");
    number=1;
    break;
    } //default
}//switch
}
```

Output in the diagnostics window:



```
example 5 : case statement
number = 1
case 1 clause executed
number = 2
case 2 clause executed
number = 3
case 3 clause executed
number = 4
default clause executed
```

Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags as *static*. The value of this tag is retained until the next change of picture.
- During the first pass, what comes next is output in the diagnostics window.
- The *switch*statement defines which tag is to be examined *(number)*.
- The *case* statements check whether or not the condition is satisfied. The tag is checked to see whether it matches up. If the condition is not satisfied, the next *case* condition is processed. If this branch is likewise not satisfied, the next *case* condition is checked, etc.
- Each time you click the button with the 🖱, a *case* statement or the *default* statement is processed.
- Output in the diagnostics window is initiated by means of the *printf* function.
- The value of tag *number* is changed.

### 4.5.6 Example 6 - Using static tags with conditional statement and return value

In Example 6, we will show you how to use C tags of the type *static* in connection with a conditional statement. The font size is changed by means of the return value in a *C action*. The action is configured on the property *static_text_01 → Properties → Font → Font size*. The following script is processed every 250 ms when the picture is open.

#### C action

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char*
                  lpszPropertyName)
{
Static int  i=0;
static BOOL a=0;

if ((i<1)||(i>60)) a=!a;

//inc or dec i according to a
if (a) i=i+6;
else i-=6;//short form for i=i-6

//return
return(i+20);
}
```

#### Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags as *static*. The value of these tags is retained until the next change of picture. Both are initialized with *0*.
- The value of tag *a* is determined. Whenever *i* leaves the permissible range, the value of tag *a* is inverted.
- Tag *i* is incremented or decremented depending on tag *a*. If the condition is satisfied, the tag is incremented. A different notation is *if (a!=0)*.
- To calculate the font size, a minimum size of 20 points is added to tag *i* and transferred to the *Font size* property of the object with *return* as the return value.

### 4.5.7 Example 7 - Using static tags with conditional statement and return value

In Example 7, we will show you how to use C tags of the type *static* in connection with a conditional statement. The font color is changed by means of the return value in a *C action*. The action is configured on the property *static_text_02 → Properties → Colors → Font color*. The following script is processed every 250 ms when the picture is open.

#### C action

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char*
                 lpszPropertyName)
{
static long int i=0;
static BOOL a=0;

if ((i<400)||(i>10000000)) a=!a;

//inc or dec i according to a
if (a==1) i+=500; //short form for i=i+500
else i=i-500;

//return color
return(i+100000);
}
```

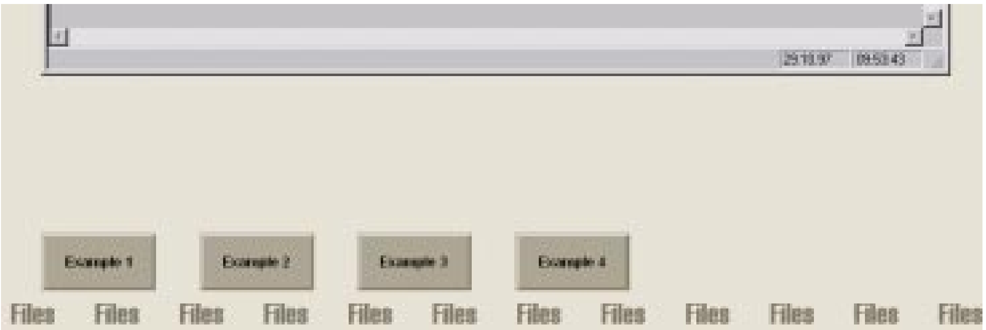
#### Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the C tags as *static*. The value of these tags is retained until the next change of picture.
- Whenever *i* leaves the permissible range, the value of tag *a* is inverted.
- Tag *i* is incremented or decremented depending on tag *a*.
- For returning the color value, the value 100000 is added to tag *i* and transferred to the property of the object. You will find notes on color coding in the appendix, in Section *5.1.6 Color table*.

# 4.6 File operations in C

In our project, you can find the examples that relate to the topic of files by clicking the *Files* button. By clicking the 🖱**R**, you can display the source code of the example in question.

## Screen for files


29.10.97
09:53:43
Example 1
Example 2
Example 3
Example 4
Files Files Files Files Files Files Files Files Files Files Files Files

After you have clicked the *Files* button with the 🖱, you will see the screen depicted above. The buttons in this screen execute the examples described. By clicking the 🖱**R**, you can display the source code of the example in question.

**Before we begin with the examples, let's take a quick theoretical look at the topic of file operations.**
In C, a file is a collection of characters, irrespective of the content.
Before you can edit (read, write) a file in C, you have to *open* it first.

## fopen

A file is opened by means of the *fopen* function. The name of the file to be edited and the editing mode are transferred as the argument.

```
FILE *file;
file = fopen (filename, workmode);
```

The file name can also be transferred with its path details.
If an error arises, no connection to a file description is returned, i.e. in C a *NULL pointer*.
The tag of type *FILE* contains the information which file commands required for editing the file.

In editing mode, define which functions are allowed.

| Mode | Description |
|---|---|
| r | Opening the file for reading ( r = read ).<br>The return value is NULL if the file does not exist or there is no read authorization. |
| W | Opening the file for writing ( w = write ).<br>The return value is NULL if the file is READ-ONLY or it does not exist. |
| a | Opening the file for appending at the end ( a = append ).<br>If the file exists, what has been written is appended at the end. |

| Mode | Description |
|---|---|
| | If the file does not exist, it is created.<br>The return value is NULL if a new file is not allowed to be created or the existing file is not allowed to be written to. |
| r+ | The file is opened for being read and written to alternately.<br>The return value is NULL if the file does not exist or there is no read and write authorization for the file. |
| w+ | Creating a new file for being read and written to alternately.<br>If the file already exists, it is **deleted!**<br>The return value is NULL if the author does not have authorization for these actions. |
| a+ | Opening a file for being read or written to (appended to) at the end.<br>If the file is created if it does not already exist.<br>The return value is NULL if a file is not allowed to be read or written to, or if the file authorization is inadequate. |

## fclose

*fclose* is used to closed files that we have opened beforehand by means of *fopen*.

```
fclose (file);
```

## fscanf

The instruction behaves in exactly the same way as *scanf*, the difference being that *fscanf* specifies the file from which the information comes.

```
fscanf ( file , "%c", character);
```

## fprintf

The instruction behaves in exactly the same way as *printf*, the difference being that *fprintf* specifies the file to which the information is sent.

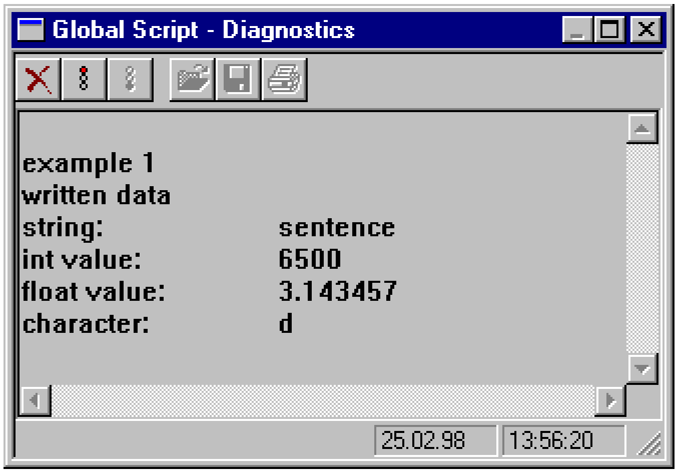
### 4.6.1 Example 1 - Opening, writing to and closing a file

In Example 1, we will show you the basic file operations in C.
The action is configured to the button *example_01 →Events →Mouse →Press left*. Click the button called *Example 1* with the 🖱 and the next script will be edited.

#### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
FILE   *file;
long   l;
float  fp;
char   s[20];
char   c;

//open file to write
datei = fopen( "cours_00.dat", "w+" );

//file open error ??
if( file != NULL ){
       l=6500;
       strcpy(s,"sentence");
       fp=3.1434567;
       c=’d’;

       //write data
       fprintf( file,"%s\r\n%ld\r\n%f\r\n%c\r\n", s, l, fp, c );

       //output in diagnostics window
       printf("\r\nexample 1\r\n");
       printf("written data\r\n");
       printf("string:\t\t%s\r\n",s);
       printf("int value:\t\t%ld\r\n",l);
       printf("float value:\t%f\r\n",fp);
       printf("character:\t%c\r\n",c);

       //close file
       fclose( file );
       }//if
else printf( "file open error\r\n" );
}
```

Output in the diagnostics window:



```
example 1
written data
string:              sentence
int value:           6500
float value:         3.143457
character:           d
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the tag and of the pointer.
- Opening the file to overwrite it. If the file does not yet exist, it is created.
- In this section, opening of the file is checked in a conditional inquiry. If this works, the values are assigned to the tag and written to the file by means of *fprintf*. If the file has not been opened or created successfully, the *else* branch is processed.
- The values are then output in the diagnostics window by means of the *printf* function.
- In the final step, the file is closed.

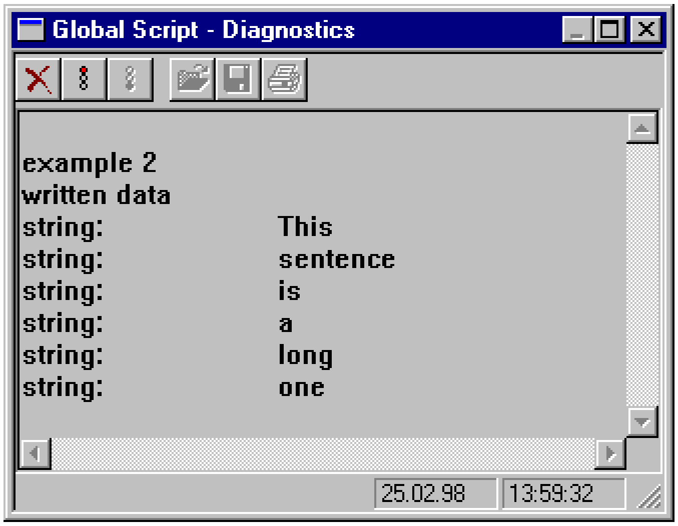
## 4.6.2 Example 2 - Opening, appending to and closing a file

In Example 2, we will show you the basic file operations in C.
The action is configured to the button *example_02 →Events →Mouse →Press left*. Click the button called *Example 2* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
static int   i = 0;
FILE         *file;
char         s[200];

switch (i) {
       case 0 : strcpy(s,"This\r\n");   break;
       case 1 : strcpy(s,"sentence\r\n");      break;
       case 2 : strcpy(s,"is\r\n");     break;
       case 3 : strcpy(s,"a\r\n");             break;
       case 4 : strcpy(s,"long\r\n");   break;
       case 5 : strcpy(s,"one\r\n");    break;
       }

//open file to write and append data
datei = fopen("cours_00.dat","a");

//file open error ??
if( file != NULL ){

       //write data
       fprintf(file,"%s",s);

       //output in diagnostics window
       if (i==0) {
             printf("\r\nexample 2");
             printf("\r\nwritten data\r\n");
             }
       printf("string:\t\t%s",s);

       fclose( datei );//close file
       }//if
else printf("\r\nfile open error\r\n");
i++;
if (i>5) (i=0);
}
```

Output in the diagnostics window:



```
example 2
written data
string:             This
string:             sentence
string:             is
string:             a
string:             long
string:             one
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the tag and of the pointer.
- Generating a text depending on the static tag *i*.
- Opening the file to append the data to the end of the file.
- In this section, opening of the file is checked in a conditional inquiry. If opening works, the current text is written to the file by means of *fprintf*. The data (contents of tag *s*) is appended to the end of the file.
- The text written is then output in the diagnostics window by means of the *printf* function.
- In the final step, the file is closed.

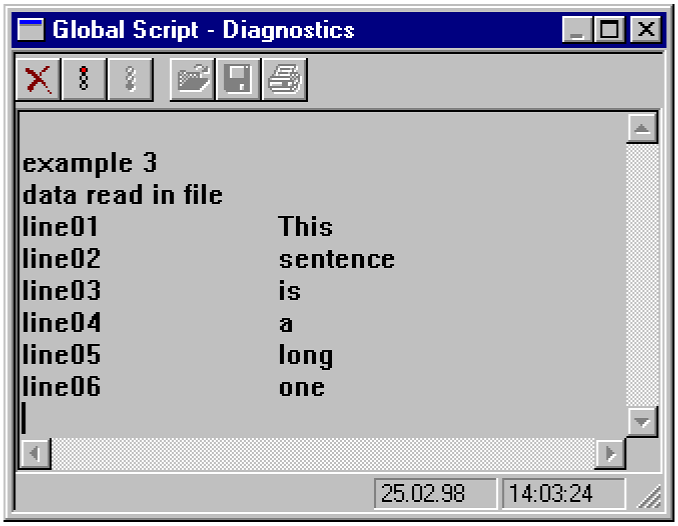
## 4.6.3 Example 3 - Opening, reading and closing a file

In Example 3, we will show you the basic file operations in C.
The action is configured to the button *example_03* →*Events* →*Mouse* →*Press left*. Click the button called *Example 3* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
FILE   *file;
int    i;
char   t[20];
char*  z;
//open file to read
datei = fopen("cours_00.dat","r");

//file open error ??
if( file != NULL ){

printf("\r\nexample 3");
printf("\r\ndata read in file\r\n");

//read data in file
i=1;
while (i) {
       i++;
       z=fgets(t,20,file);
       if (z==NULL) break;//end of file
       //output in diagnostics window
       printf("line%02d\t\t%s",(i-1),t);
       }//while

//close file
       fclose( file );
       }//if
else {
   printf("\r\nfile open error\r\n" );
   }//else
}
```

Output in the diagnostics window:



```
example 3
data read in file
line01          This
line02          sentence
line03          is
line04          a
line05          long
line06          one
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the tag and of the pointer.
- Opening the file.
- In this section, opening of the file is checked in a conditional inquiry. If this works, the data is read out of the file line-by-line by means of *fgets* and output in the diagnostics window by means of the *printf* function. When the end of the file is reached(return value is 0), the *while* loop is exited by means of *break*.
- In the final step, the file is closed.

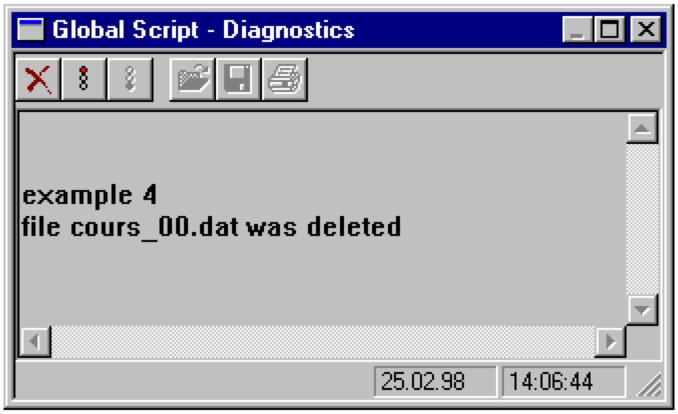
## 4.6.4 Example 4 - Deleting a file

In Example 4, we will show you how to delete the file created beforehand.
The action is configured to the button *example_05* → *Events* → *Mouse* → *Press left*. Click the button called *Example 5* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
//delete file
remove("cours_00.dat");
printf("\r\n\r\nexample 4\r\n");
printf("file cours_00.dat was deleted\r\n");
}
```

### Output in the diagnostics window



```
example 4
file cours_00.dat was deleted
```

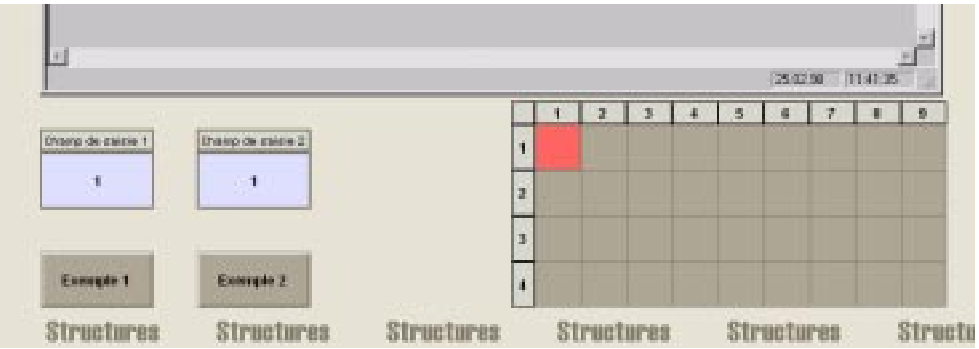
### Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- The file is deleted.
- The diagnostics window then displays what has happened.

## 4.7 Structures in C

In our project, you can find the examples that relate to the topic of structures by clicking the *Structures* button. By clicking the 🖱**R**, you can display the source code of the example in question.

### Screen for structures



After you have clicked the *Structures* button with the 🖱, you will see the screen depicted above. The buttons in this screen execute the examples described. By clicking the 🖱**R**, you can display the source code of the example in question.
In the right-hand field, the values of the *input fields* in *Example 2* are evaluated.

**Before we begin with the examples, let's take a quick theoretical look at the topic of structures.**

### Definition of the structure

In principle, a structure declaration in C looks as follows:

```
struct structurename{
                   [ components ]
                   };
```

In C, components are a collection of defined tags, which are themselves structural elements and a permanent parts of the structure created.
In principle, we can handle the structure in the same way as a tag type, since it is in fact a collection of data types.
With

```
struct whole  {
              int number;
              char character[50];
              };

struct whole  structure[2];
```

The above instruction is used in the declarative section to define the structure layout. Imagine that a data type, *struct whole*, is created and that the tag, *structure*, is of the type *struct whole*.

## The following simplification can be used for the declaration

The declaration of the structure layout and the definition of a tag can be combined into one statement:

```
struct whole  {
              int number;
              char character [50];
              } structure [2];
```

## Accessing the tags in the structure

*structurename.tagname* can be used to access specific pieces of data.
It is essential that there is a decimal point (or dot) between the two elements. It separates the individual elements in the hierarchy.
**The tag name is always the name that is used when creating the structure!**
With

```
structure[0].number = 2;
strcpy(structure[0].character ,"our first structure");
```

The above statement is used to assign the values to the elements. In the case of character strings, it is possible to use a simple assignment (With *strcpy,* a character or a character string is assigned to a character string).
As before, you can also access fields within structures:

```
printf ("%c",structure[0].character[5] );
```

for example, outputs the 6th letter of the character string from the 1st field of *structure [0]*.
**Indices start at 0 !**

## Pointers as structural elements

Any types of tag can be incorporated into a structure. This also includes pointers.
The following structure is, therefore, perfectly conceivable:

```
struct structure{
              int    number;
              char   *character string;
              }
```

**Note:**
If a pointer that points to a character string (e.g. *character string) is used, the memory must also be available or be allocated beforehand (e.g. by means of the internal function sysMalloc (8)).

As an example of this, we will use the structure of the previous example and generate the following pointer:

```
struct struktur     *pointer;
```

If we now want to access an element, we have to write the following:

```
(*pointer).number;  // or more simply  pointer->number;
```

**Note:**
Since the dot takes priority over the * pperator, must first expression must be set in parentheses.

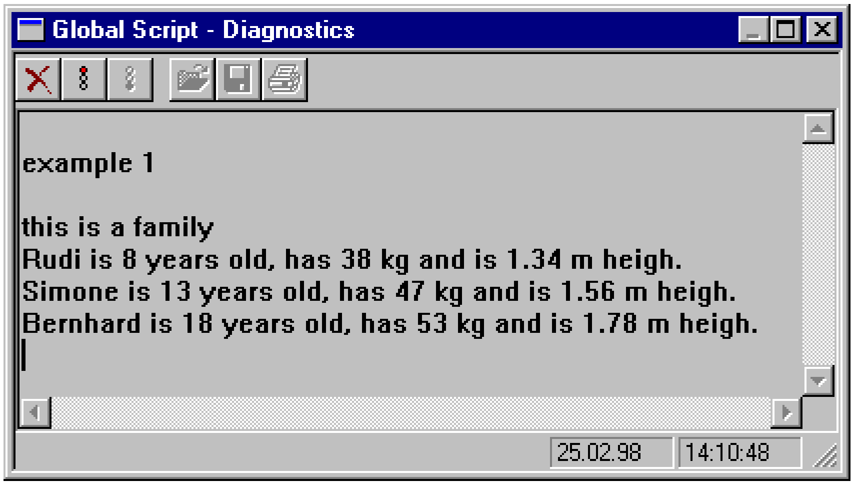
## 4.7.1 Example 1 - Structures in C

In Example 1, we will show you how to use structures in C.
The action is configured to the button *example_01 →Events →Mouse →Press left*. Click the button called *Example 1* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
//declare structure
struct data {
       char name[40];
       int age;
       int weight;
       float height; }  family[3];

int    i;

//set values to structure elements
strcpy(family[0].name,"Rudi");
strcpy(family[1].name,"Simone");
strcpy(family[2].name,"Bernhard");

family[0].age=8;
family[1].age=13;
family[2].age=18;

family[0].weight=38;
family[1].weight=47;
family[2].weight=53;

family[0].weight=1.34;
family[1].weight=1.56;
family[2].weight=1.78;

//output in diagnostics window
printf("\r\nexample 1\r\n");
printf("\r\nthis is a family\r\n");
for (i=0;i<3;i++) {
   printf("%s is %d years old, has %d kg and is %.2f m
heigh.\r\n",family[i].name,family[i].age,family[i].weight,family[i].
height);
   }
}
```

Output in the diagnostics window:



```
example 1

this is a family
Rudi is 8 years old, has 38 kg and is 1.34 m heigh.
Simone is 13 years old, has 47 kg and is 1.56 m heigh.
Bernhard is 18 years old, has 53 kg and is 1.78 m heigh.
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the structure and of the tag.
- The values are assigned to the structural elements.
- The values (pieces of information) are then output in the diagnostics window by means of the *printf* function.

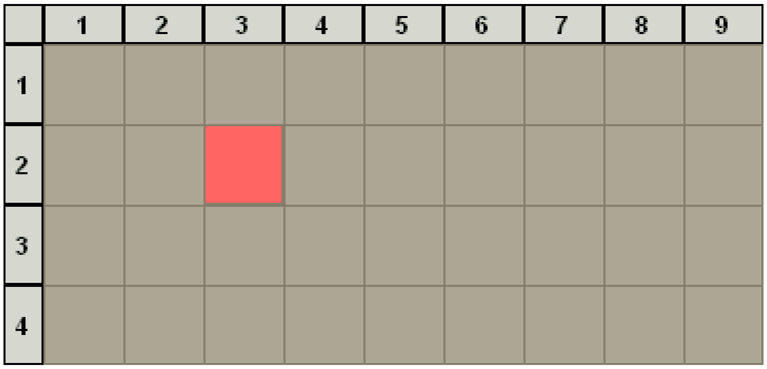
## 4.7.2 Example 2 - C structures in connection with WinCC

In Example 2, we will show you how to use structure tags in WinCC. Before we can use these structure tags in our action, they have to be created in the tag management of the *Control Center*. You will find a detailed explanation in the *Configuration Manual, Volume 2,* in the example *2.8 Using Structure Tags*.
The action is configured to the button *example_02* → *Events* → *Mouse* → *Press left*. Click the button called *Example 2* with the 🖱 and the next script will be edited.

### C action

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
int input1,input2;
int output1,output2;

//read selected position in input_01 and input_02
input1=GetTagDWord("S32i_course_str_01");
input2=GetTagDWord("S32i_course_str_02");

input1=input1-1;
input2=input2-1;

//calculate new rectangle position
output1=500+(input1*40);
output2=470+(input2*40);

//set internal tags which contain rectangle position
SetTagDWord("STUi_course_str_00.xpos",output2);
SetTagDWord("STUi_course_str_00.ypos",output1);
}
```

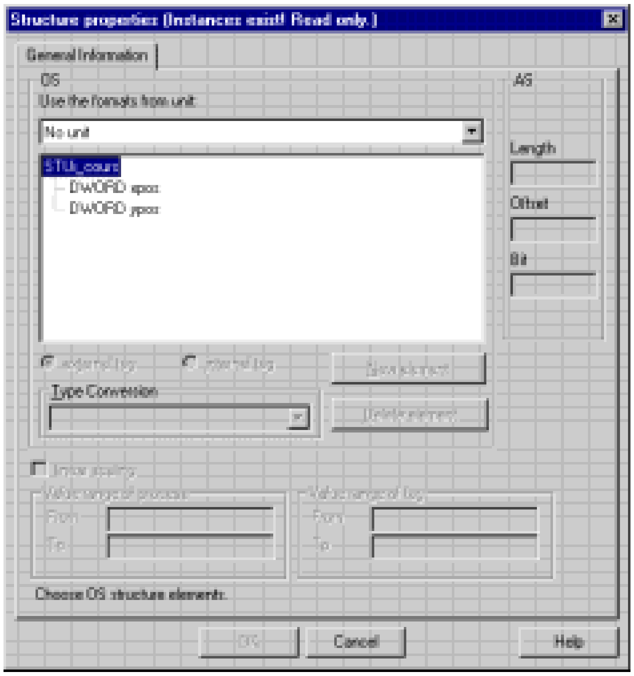
Display on the screen:



## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the tags.
- Reading the values out of the I/O fields. The limits of the input are defined directly in the *I/O fields*.
- Recalculate the positional specifications entered to give positions on the screen.
- Assign the values calculated to the WinCC structure tags. The values are linked to the X and Y positions of the rectangle.
- The structure used must be declared under *Control Center → Data Types → Structure Types*, and has the following layout:

Creating the structure tags in the tag management


Structure properties (Instances exist! Read only.)

Use the formats from unit: No unit

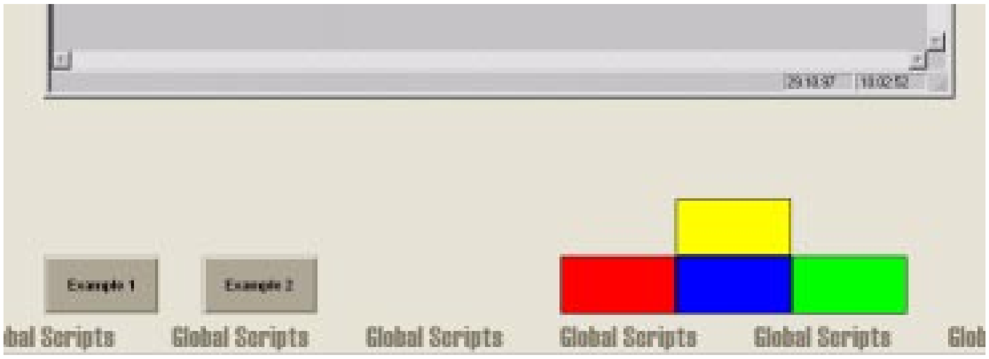
```
STU_cours
  DWORD xpos
  DWORD ypos
```

Choose OS structure elements.

## 4.8 Global Scripts

In our project, you can find the examples that relate to the topic of *Global Scripts* by clicking the *GSC* button. By clicking the 🖱**R**, you can display the source code of the example in question.

### Screen for Global Script



After you have clicked the *GSC* button with the 🖱, you will see the screen depicted above. The buttons in this screen execute the examples described. By clicking the 🖱**R**, you can display the source code of the example in question.

The project functions have been created in the *Global Script* editor and used in the object-oriented actions in the *Graphics Designer*. The functions are selected via the function tree in the editor of the *C action*.

## 4.8.1 Example 1 - Using a project function

In Example 1, we will show you how to use project functions in a C action.
The action is configured to the button *example_01 →Events →Mouse →Press left*. Click the button called *Example 1* with the 🖱 and the next script will be edited.

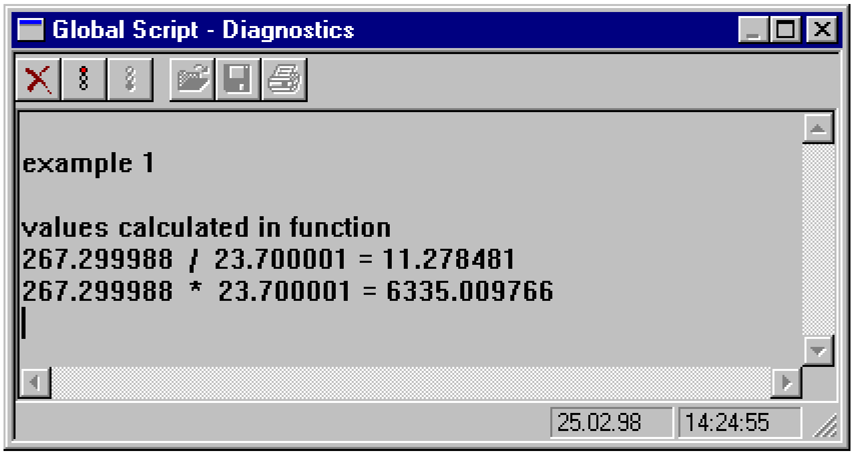
### Program of the project function

```
void calculate(float a,float b,float *pRes1,float *pRes2)
{
//calculate and store value in addresses
*pRes1=a/b;    //divide
*pRes2=a*b;    //multiply
}
```

### Program of the action on the object

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
float number1,number2;
float result1,result2;

number1=267.3;
number2=23.7;

//execute function and set parameters
calculate(number1,number2,&result1,&result2);

//output in diagnostics window
printf("\r\nexample 1\r\n");
printf("\r\nvalues calculated in function\r\n");
printf("%f  /  %f = %f\r\n",number1,number2,result1);
printf("%f  *  %f = %f\r\n",number1,number2,result2);
}
```

## Output in the diagnostics window



```
example 1

values calculated in function
267.299988 / 23.700001 = 11.278481
267.299988 * 23.700001 = 6335.009766
```

## Explanation of the individual program components

- In the program of the action, the tags (*number1, number2*) are declared and the project function *calculate* is transferred.
- In the program of the project function called, the values transferred are calculated. The values calculated are returned by means of the two pointers **pRes1, *pRes2*
- The values in the *C action* are then output in the diagnostics window by means of the *printf* function.

## 4.8.2 Example 2 - Using project functions, further examples

In Example 2, we will show you how to use project functions in a C action.
The action is configured to the button *example_02 →Events →Mouse →Press left*. Click the button called *Example 2* with the 🖱 and the next script will be edited.

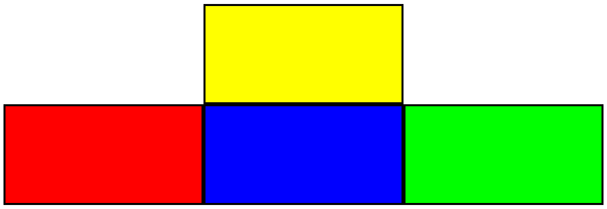
### Program of the project function

```
void colourchange()
{
long int color1,color2,color3,color4;
long int store;

//get rectangle colors
color1=GetBackColor(lpszPictureName,"rectangle_01");
color2=GetBackColor(lpszPictureName,"rectangle_02");
color3=GetBackColor(lpszPictureName,"rectangle_03");
color4=GetBackColor(lpszPictureName,"rectangle_04");

//cyclic color change
store=color1;
color1=color2;
color2=color3;
color3=color4;
color4=store;

//set rectangle colors
SetBackColor(lpszPictureName,"rectangle_01",color1);
SetBackColor(lpszPictureName,"rectangle_02",color2);
SetBackColor(lpszPictureName,"rectangle_03",color3);
SetBackColor(lpszPictureName,"rectangle_04",color4);
}
```

### Program of the action on the object

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName,
                 char* lpszPropertyName, UINT nFlags, int x, int y)
{
//execute function
colourchange(lpszPictureName);
}
```

## Change of colors on mouse click



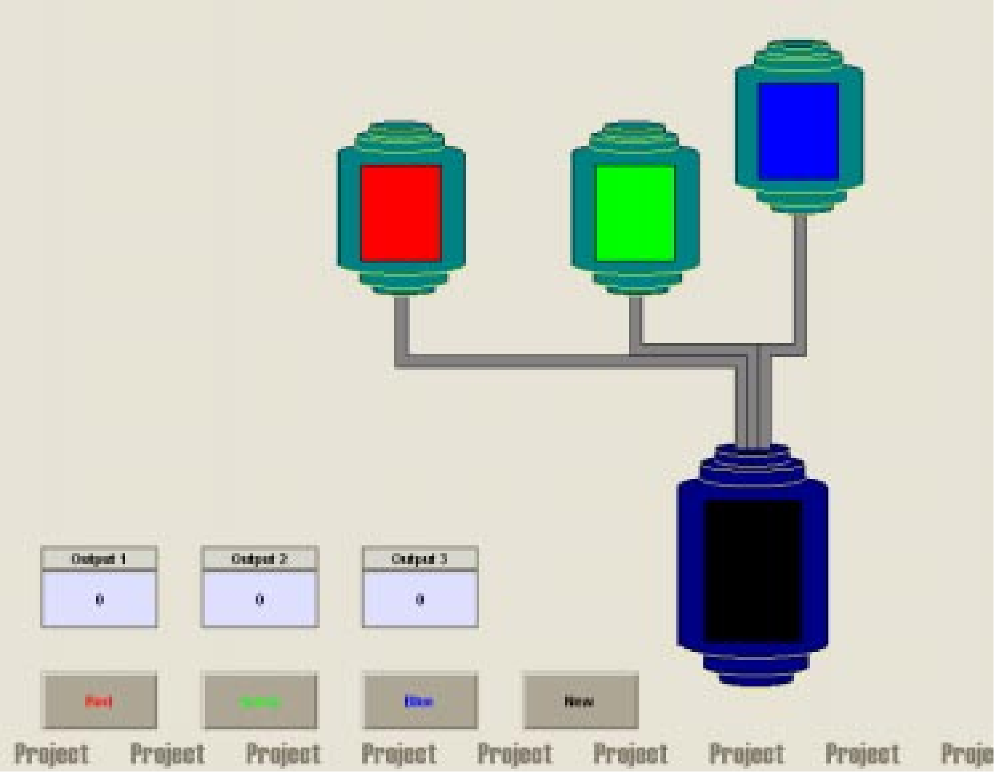
## Explanation of the individual program components

- In the program of the project function, the current colors are read out, shifted one position further and reassigned to the objects.
- The picture name is transferred when the project function is called.

## 4.9 Project as an example

This project is another example of how to use the pointers, loops, statements, file operations, structures, and global scripts described in the previous chapters. The individual program components are not described in any detail here.

### Screen for project


Output 1
0
Output 2
0
Output 3
0
Red
Green
Blue
New
Project Project Project Project Project Project Project Project Proje

When you open the screen containing the project, you see the above picture.
This represents a color mixing system. By *clicking* the *Red*, *Green* and *Blue* buttons with the 🖱, you add the corresponding color. You have to *click* the button again to stop adding the color. You reset the system to its initial state by clicking *Restart*. The output fields indicate the amount of each color already added (max. 50).

# 4.10 Displaying the source code by clicking the right mouse button

In the project with the name *Cours_00*, the sources of the *C action* can be displayed in all examples while runtime is in progress. You display them, in all examples, by clicking the respective button with the 🖱**R**.

```
#include "apdefap.h"
void OnRButtonDown(char* lpszPictureName, char* lpszObjectName,
                  char* lpszPropertyName, UINT nFlags, int x, int y)
{
char name[30],namex[30];
int visx,comp;
int number;
int  ch = ’_’;
char *pdest;

visx=GetVisible("course_0_startpicture_00.PDL","code picture");
strcpy(namex,GetTagChar("T16x_org_picname_02"));

pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameError"));
else {
   number = atoi(strrchr(lpszObjectName,’_’)+1);
   sprintf(name,"course_4_tag_%02d.PDL",number);
   SetTagChar("T16x_org_picname_02",name);
   }
comp=strcmp(name,namex);
if ((comp==0)&&(visx==1)){
       SetVisible("course_0_startpicture_00.PDL","code picture",0);
       SetVisible("course_0_startpicture_00.PDL","code picture",1);
       }
else SetVisible("course_0_startpicture_00.PDL","code picture",1);
}
```

## Explanation of the individual program components

- The first section is the *function header*. It cannot be changed.
- Declaration of the tags.
- Check whether the picture window has already been opened.
- Read out the picture name.
- Define the button number.
- Put together the name of the source code picture.
- Compare to see whether or not the picture name has been changed.
- If the name is the same, close the picture and rebuild it so that it is displayed in the foreground.
- A new picture is the picture that is displayed.

# 5 Appendix

In this Appendix, you will a collection of topics that have not been directly incorporated into the *Configuration Manual*.

## 5.1 Tips and tricks

More examples of configuring with WinCC.

## 5.1.1 Standard input/output at I/O field

Configure the following actions so that an I/O field is displayed in a standard manner and/or the entered value to be transferred to the PLC in a standard manner:

Action on property "Output Value" of an I/O field (important: "float", if you require digits behind the decimal point)

```
float  a;
a=GetTagFloat("DB21_DW1");
return(a/100);
```

Action on event "input value"  of an I/O field (tag "Var1" is an unsigned 16-bit value)

```
float  a;
a=GetInputValueDouble(lpszPictureName,lpszObjectName);
SetTagFloat("Var1",a*100);
```

## 5.1.2 Object-specific actions upon Open Picture

There are cases in which you have to perform actions on the property of one or more objects in a picture only once upon Open Picture. One possibility is to formulate a picture-specific action on the picture object under *Events →Miscellaneous → Open Picture*. This, however, has the disadvantages that the action has to have an effect on objects in the picture and thus the object names have to be firmly specified in the picture. The objects can no longer be handled freely. This solution is not object-oriented.

There is a way in which you can avoid this problem:

- Define an internal tag - for example, *dummy* - which is never updated or deliberately set. Set the trigger for the action on the object to modify that tag. On opening the picture in runtime, the action is activated for the first time and would then only react again thereafter if tag *dummy* were to be modified, which does not happen since this tag is never modified.

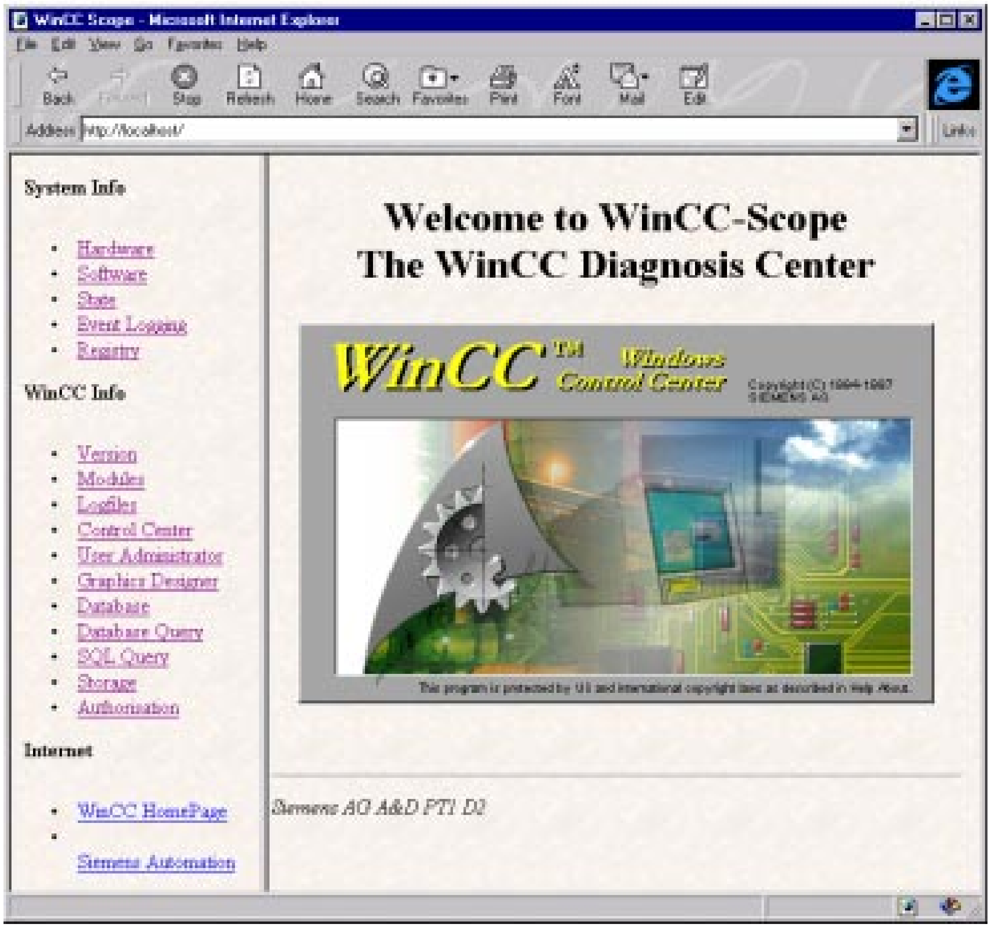
## 5.1.3 WinCC Scope

### General

*WinCC Scope* is a tool that supports you in diagnosing WinCC projects. It makes a great deal of information available about the activated project and also about the computer system concerned. You require a web browser such as Internet Explorer to be able to work with *Scope*. In addition, the TCP/IP network protocol must be installed.

### Starting and running

If you have installed WinCC, *Scope* is also installed by default. Before you can use Scope, you have to start the *WinCCDiagAgent.exe* program. You will find it in the folder called *Siemens\WinCC\WinCCScope\bin*. The program is a simple HTTP server. You can then launch *Scope* from the Start menu. From the start page, you are directed to *How to use the new Diagnostics Interface*, a page containing a general description of how to run *WinCC Scope*. Click *http://localhost* to start *Scope*. You can select and view various kinds of information from a list contained in the left window. The segment called *System Info* contains general information on the different computer systems concerned, whereas you can find information on the WinCC project that has just been activated in the segment called *WinCC Info*.


WinCC Scope - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back
Forward
Stop
Refresh
Home
Search
Favorites
Print
Font
Mail
Edit
Address
Links
WinCC™ Windows Control Center
Copyright (C) 1994-1997 SIEMENS AG
This program is protected by US and international copyright laws as described in Help About.

Address: http://localhost/

System Info

- Hardware
- Software
- State
- Event Logging
- Registry

WinCC Info

- Version
- Modules
- Logfiles
- Control Center
- User Administrator
- Graphics Designer
- Database
- Database Query
- SQL Query
- Storage
- Authorisation

Internet

- WinCC HomePage
- Siemens Automation

Welcome to WinCC-Scope
The WinCC Diagnosis Center

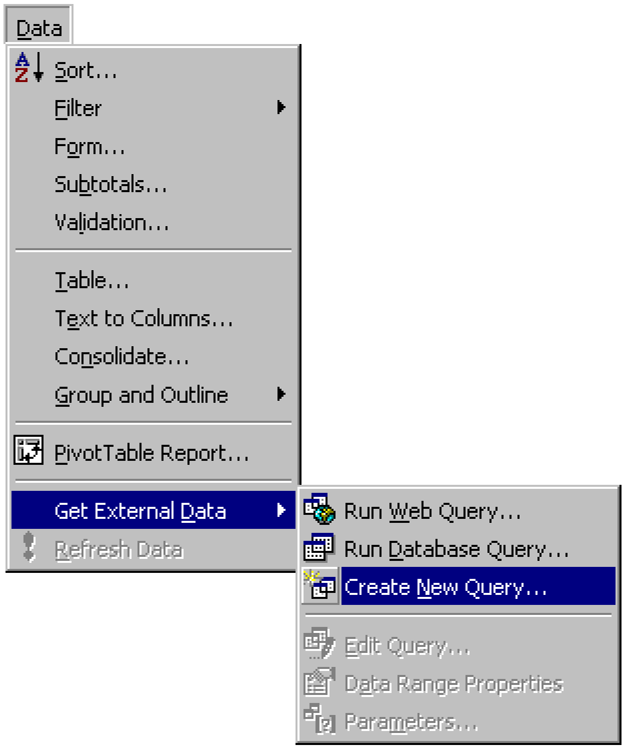
Siemens AG A&D PT1 D2

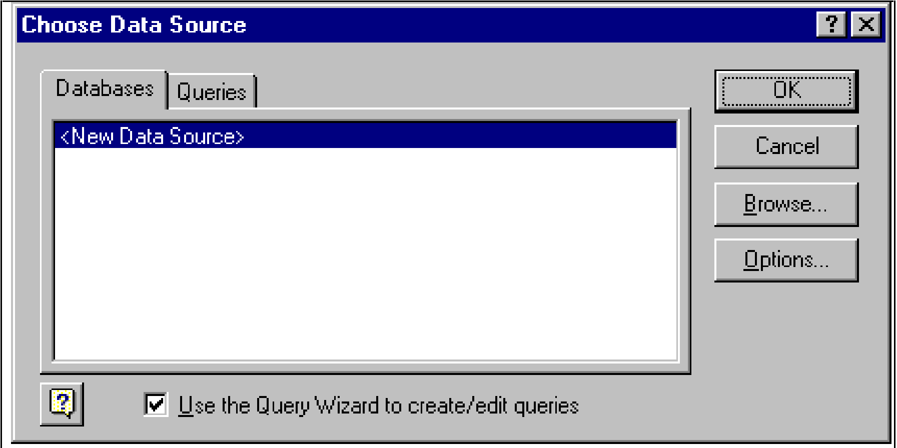
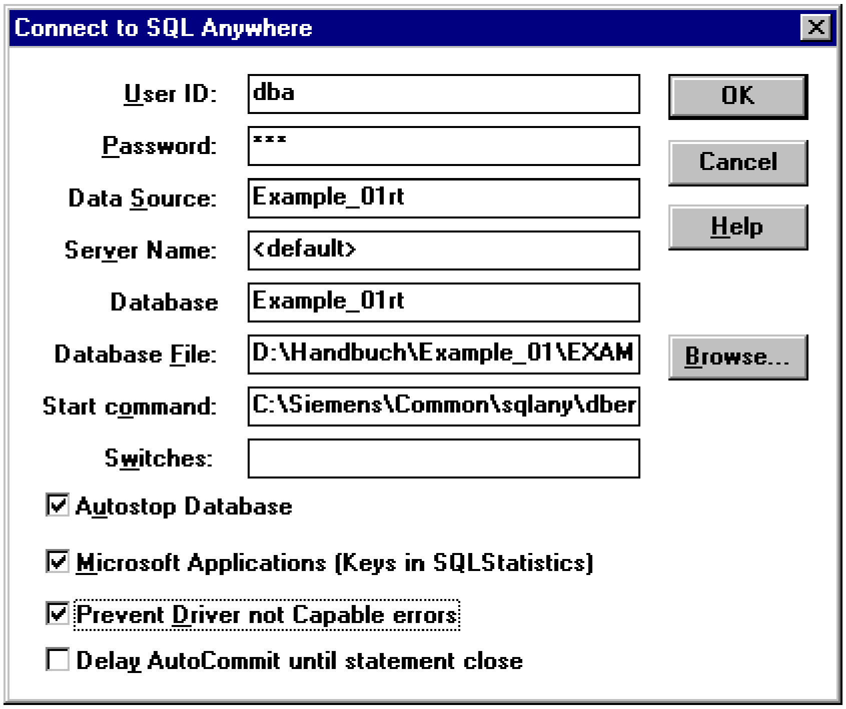
## 5.1.4 Accessing the database

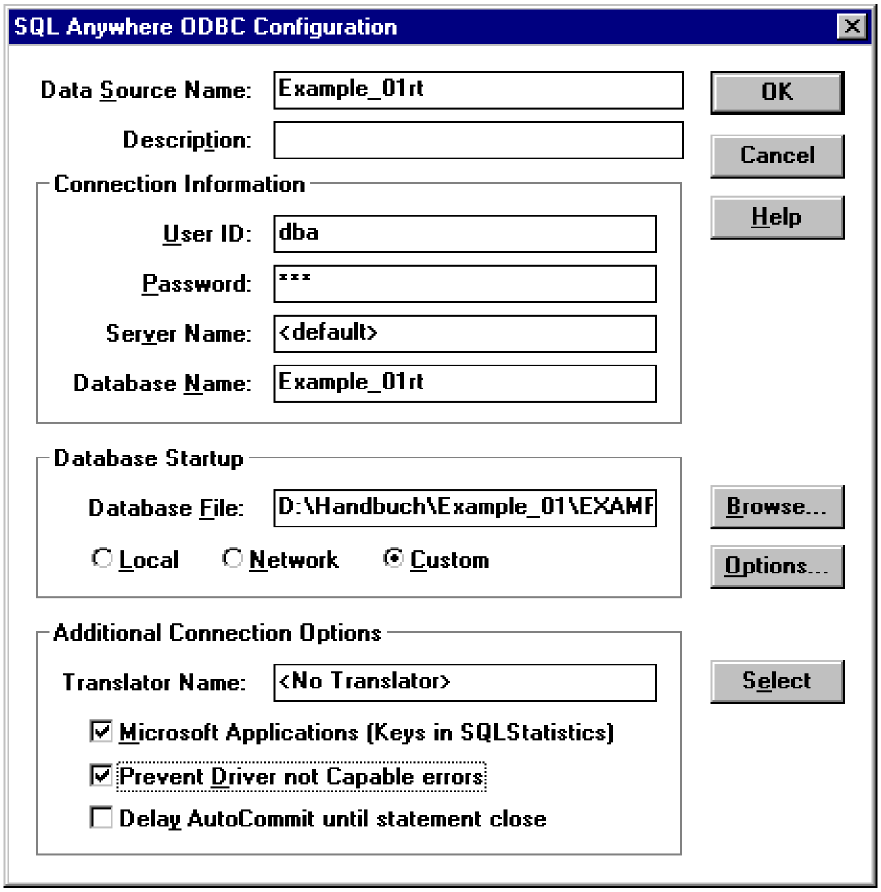
### 5.1.4.1 Accessing the database from Excel/MSQuery

The following description of accessing the WinCC database refers to the use of Microsoft® Excel 97 with SR-1.

### Accessing from Excel/MSQuery

| Step | Procedure: Access from Excel/MSQuery |
|---|---|
| 1 | Open Excel. Choose *Data* →*External Data* →*Create New Query...* from the menu to open the *Select Data Source* in MSQery.<br><br> Data<br>Sort...<br>Filter<br>Form...<br>Subtotals...<br>Validation...<br>Table...<br>Text to Columns...<br>Consolidate...<br>Group and Outline<br>PivotTable Report...<br>Get External Data<br>Refresh Data<br>Run Web Query...<br>Run Database Query...<br>Create New Query...<br>Edit Query...<br>Data Range Properties<br>Parameters...<br>On the *Databases* tab, select the entry *<New Data Source>*. Choose *OK* to create a new data source. |

| Step | Procedure: Access from Excel/MSQuery |
|---|---|
|  | <br>Choose Data Source<br>Databases \| Queries<br>&lt;New Data Source&gt;<br>OK<br>Cancel<br>Browse...<br>Options...<br>☑ Use the Query Wizard to create/edit queries |
| 2 | Specify the name of the new data source in the *Create New Data Source* dialog box. It does not have to be the name of the WinCC database. Select *Sybase SQL Anywhere 5.0* as the driver.<br>Choose the *Connect...* button to open the *Connect to SQL Anywhere* dialog box and enter the information required by the driver. Enter the *User ID* as *dba* and the *Password* as *sql*. Choose the *Browse* button to select the database you want to edit.<br>Choose *OK* to terminate your input.<br><br>Connect to SQL Anywhere<br>User ID: dba<br>Password: ***<br>Data Source: Example_01rt<br>Server Name: &lt;default&gt;<br>Database Example_01rt<br>Database File: D:\Handbuch\Example_01\EXAM<br>Start command: C:\Siemens\Common\sqlany\dber<br>Switches:<br>☑ Autostop Database<br>☑ Microsoft Applications (Keys in SQLStatistics)<br>☑ Prevent Driver not Capable errors<br>☐ Delay AutoCommit until statement close<br>OK<br>Cancel<br>Help<br>Browse... |
| 3 | If a data source has yet to be configured for the database you selected, the message *Name of data source not found and no standard driver specified* appears.<br>Confirm this message and click the *Connect...* button once again. Select the *Machine Data Source* tab in the *Select Data Source* dialog box. The CS and runtime databases of the current WinCC project already appear in the list of data |

| Step | Procedure: Access from Excel/MSQuery |
|---|---|
| | sources. The names of these data sources begin with the character string *CC_*, which is followed by the project name. The name of the data source representing the runtime database ends with an *R* character.<br>If you want to work on a WinCC database, you must first create a data source for it. You do this by choosing the *New* button. In the *Create New Data Source* wizard that appears, select the item called *User Data Source* on the first page and close the page by clicking *Next*. On the next page, select the driver *Sybase SQL Anywhere 5.0* and close the page by clicking *Next*. Close the last page by clicking *Finish*.<br>The *SQL Anywhere ODBC Configuration* dialog box opens, in which you have to enter information required by the driver. The *User ID* you have to enter is again *dba* and the *Password* is *sql*. Choose the *Browse* button to select the database you want to edit.<br>Choose *OK* to close the dialog box.<br><br>SQL Anywhere ODBC Configuration<br>Data Source Name: Example_01rt<br>Description:<br>Connection Information<br>User ID: dba<br>Password: ***<br>Server Name: <default><br>Database Name: Example_01rt<br>Database Startup<br>Database File: D:\Handbuch\Example_01\EXAMF<br>○ Local ○ Network ● Custom<br>Additional Connection Options<br>Translator Name: <No Translator><br>☑ Microsoft Applications (Keys in SQLStatistics)<br>☑ Prevent Driver not Capable errors<br>□ Delay AutoCommit until statement close<br> <br>Select the newly created data source in the *Select Data Source* dialog box and click *OK* to close the dialog box.<br>Confirm the *Connect to SQL Anywhere* dialog box which then appears.<br>You can configure the data source beforehand by means of the Control Panel. Open the *ODBC Data Source Administrator* on the Control Panel. Choose the *Add* button to open the *Create New Data Source* wizard. |

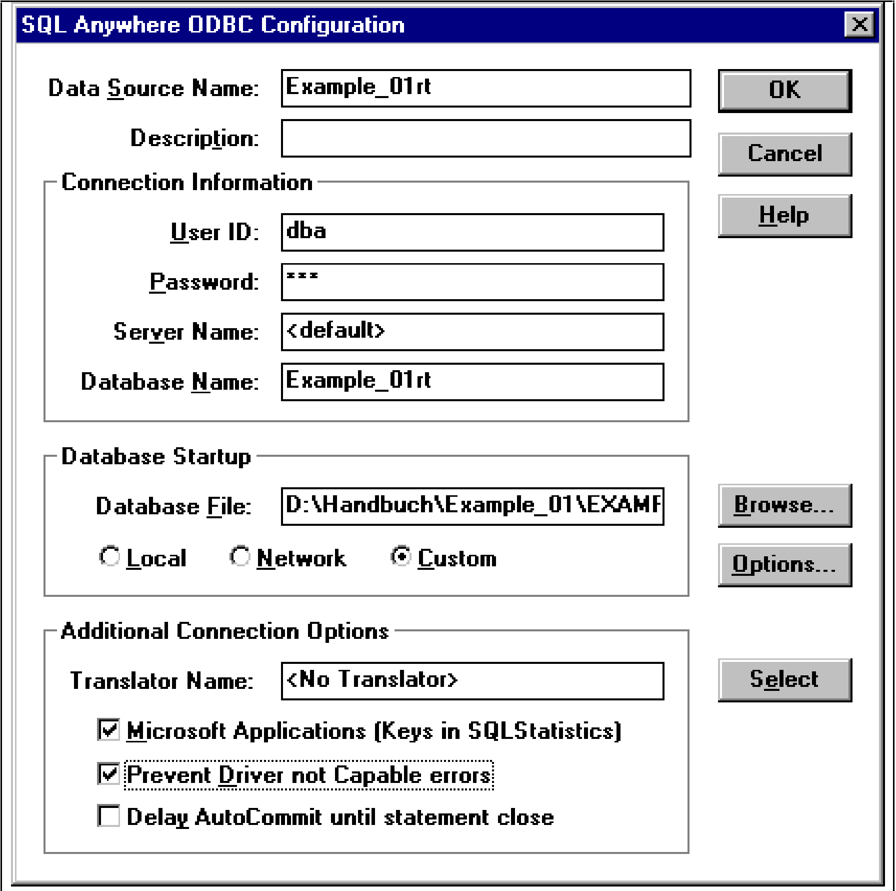
| Step | Procedure: Access from Excel/MSQuery |
|---|---|
| | ODBC |
| 4 | Choose *OK* to close the *Create New Data Source* dialog box.<br>In the *Select Data Source* dialog box, select the new data source you have just created and choose *OK* to close the dialog box.<br>All available tables and columns are displayed on the first page of the *query wizard* that appears. Select the tables and columns you require and close the page by choosing *Next*. You can set filters for the data and their sort order on the next few pages. You determine on the last page whether the data ate to be processed in Excel or MSQuery in future. Choose *Finish* to close the dialog box. |
| 5 | You determine the positioning of the tables you want to insert in the *Return External Data to Excel* dialog box which then appears. You can also set the properties of the external data area. Choose *OK* to close the dialog box. |

### 5.1.4.2 Accessing the database from Access

The following description of accessing the WinCC database refers to the use of Microsoft® Access 97 with SR-1.

#### Accessing with Access

| Step | Procedure: accessing with Access |
|---|---|
| 1 | Open an Access database, or create a new one. Choose *File →Get External Data →Import...* from the menu to open the *Import* dialog box. For the *file type*, select the *ODBC Databases()* list entry.<br>The *Select Data Source* dialog box will be opened automatically. Select a data source on the *Machine Data Source* tab. The CS and runtime databases of the current WinCC project already appear in the list of data sources. The names of these data sources begin with the character string *CC_*, which is followed by the project name. The name of the data source representing the runtime database ends with an *R* character. |
| 2 | If the WinCC database does not appear in the list, create it first as a data source by choosing the *New* button.<br>In the *Create New Data Source* wizard that appears, select the item called *User Data Source* on the first page and close the page by clicking *Next*. On the next page, select the driver *Sybase SQL Anywhere 5.0* and close the page by clicking *Next*. Close the last page by clicking *Finish*.<br>The *SQL Anywhere ODBC Configuration* dialog box opens, in which you have to enter information required by the driver. Enter the *User ID* as *dba* and the *Password* as *sql*. Choose the *Browse* button to select the database you want to edit.<br>Choose *OK* to close the dialog box. |

| Step | Procedure: accessing with Access |
|---|---|
| | <br> SQL Anywhere ODBC Configuration<br>OK<br>Cancel<br>Help<br>Browse...<br>Options...<br>Select<br>Data Source Name: Example_01rt<br>Description:<br>Connection Information<br>User ID: dba<br>Password: ***<br>Server Name: <default><br>Database Name: Example_01rt<br>Database Startup<br>Database File: D:\Handbuch\Example_01\EXAMF<br>○ Local ○ Network ● Custom<br>Additional Connection Options<br>Translator Name: <No Translator><br>☑ Microsoft Applications (Keys in SQLStatistics)<br>☑ Prevent Driver not Capable errors<br>☐ Delay AutoCommit until statement close<br>Select the newly created data source in the *Select Data Source* dialog box and click *OK* to close the dialog box. |
| 3 | You can select the database tables you require in the *Import Objects* dialog box that appears. Choose *OK* to insert them in the Access database. |

### 5.1.4.3 Accessing the database from ISQL

Direct access to the WinCC database is possible using ISQL. This, however, is entirely your personal responsibility since the configuration data may become inconsistent as a result of editing or deleting tables.

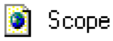
### Accessing with ISQL

| Step | Procedure: accessing with ISQL |
|---|---|
| 1 | Start ISQL.EXE in the folder called Siemens\Common\sqlany.<br>The *Interactive SQL Logon* dialog box appears. Enter the *User ID* as *dba* and the *Password as sql*. If you acknowledge by choosing OK, the program is automatically connected to the current WinCC database - in other words, to the CS database. If, however, you wish to access another database - for example, the |

| Step | Procedure: accessing with ISQL |
|---|---|
| | runtime database - choose *Command* → *Connect* from the menu. In the dialog box that appears, enter the same details for the *User ID* and *Password*. For *Database File*, enter the database you require with its full path. |
| 2 | You can now enter SQL statements in the Command window and execute them by choosing the *Execute* button.<br>Here are some examples of SQL statements:<br>• select* from systable: displays all table names<br>• select * from <*table_name*> : shows the contents of the table called <*table_name*><br>• unload tabele <*table_name*> to <*filename*> : exports the table called <*table_name*> to the file called <*filename*><br>• drop table <*table_name*> : deletes the table called <*table_name*> |

### 5.1.4.4 Accessing the database from WinCC Scope

#### Accessing with WinCC Scope

| Step | Procedure: accessing with WinCC Scope |
|---|---|
| 1 | Before starting WinCC Scope from the Start menu, start the application called WinCCDiagAgent.exe in the *Siemens\WinCC\WinCCScope\bin* folder.<br> |
| 2 | You can obtain a general description of how to run WinCC Scope from the first page by selecting *How to use the new Diagnostics Interface*.<br>Click *http://localhost* to start Scope. |
| 3 | You can select different functions from a list in the left segment.<br>• Select *Database* to obtain general information about the WinCC database.<br>• You can view different tables in a database by selecting *Database Query*. The default *data source* is the CS database of the current WinCC project. The names of these data sources begin with the character string *CC_*, which is followed by the project name. The name of the data source representing the runtime database ends with an *R* character. However, other data sources can also be displayed.<br>• SQL statements can be applied to a data source you select by means of *SQL query*. However, it is advisable to edit the WinCC database only if your system knowledge of SQL statements is very good. Examples of SQL statements are listed in the preceding section called *Accessing the database from ISQL*. |

### 5.1.4.5 Exporting from a database using C actions

Data export can also be activated from a WinCC runtime picture. To do this, it is possible to start an interactive SQL with a command line by means of ProgramExcecute. The action which has to be executed is stored in a command file (archiv.sql in our example).

C action - on button, for example

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char*
                 lpszPropertyName)
{

char* path =   "C:\\SIEMENS\\Common\\SQLANY\\ISQL -q -b -c";
char* parameters =
               "UID=DBA;PWD=SQL;DBN=CC_Project_97-10-21_09:53:27R";
char* action = "read D:\\WinCC\\Project\\archiv.sql";

char ExportArchive[200];

sprintf(ExportArchive,"%s %s %s",path,parameters,action);

ProgramExecute(ExportArchive);
}
```

- The *path* tag contains the path to the ISQL.exe program with its call parameters.
- The *parameters* tag contains the input that has to be performed in the *Interactive SQL Logon* dialog box for connecting to the database. The input consists of:
  - UID (User ID) : DBA
  - PWD (Password) : SQL
  - DBN (Data Base Name) : Name of the ODBC data source. The names of these data sources begin with the character string *CC_*, which is followed by the project name together with the date and time of creation of the project. The name of the data source representing the runtime database ends with an *R* character. You can set this name, when the project has been activated, by choosing *Control Panel* → *ODBC* → *User DSN* tag.
- The *action* tag specifies that the SQL statements listed in the *archiv.sql* file have to be executed.
- The statements are combined in *ExportArchive* and executed with the *ProgramExecute()* function.

**Note:**
If you want to export from a database other than one of the two project databases, specify the DBF parameter, the database file with specification of the path to the database, instead of the DBN parameter. This method dies not work for the currently activated project database, however.

### Contents of the file: archiv.sql

```
select * from PDE#HD#ProcessValueArchive#Analog;
output to D:\WinCC\Projekt\archiv.txt format ascii
```

- In the open database, the measured value archive *pde#hd#ProcessValueArchive#Analog* is selected and then exported by means of the output instruction to the ASCII file called *archiv.txt*.

## 5.1.4.6 Database selections

The *select* instruction in the command file that was described earlier selects tables. Subsets of these tables can be selected with additional parameters and are then exported with the *output* instruction. A few examples of this subject are shown below.

### Selecting a time range

```
select * from PDE#HD#ProcessValueArchive#Analog where T between
                 ’1996-5-1 10:10:0.00’ and ’1996-6-1 10:10:0.00’
```

### Selection from a time stamp

```
select * from PDE#HD#ProcessValueArchive#Analog where T >
                 ’1996-5-1 10:10:0.00’
```

### Selecting a process value with and without sorting

```
select * from PDE#HD#ProcessValueArchive#Analog where V > 100
select * from PDE#HD#ProcessValueArchive#Analog where V > 100
                 order by T
```

### Selecting with selection of columns t (time) and v (value) to process value

```
select T,V from PDE#HD#ProcessValueArchive#Analog where V > 100
                 order by T
```

### 5.1.5 Serial connection

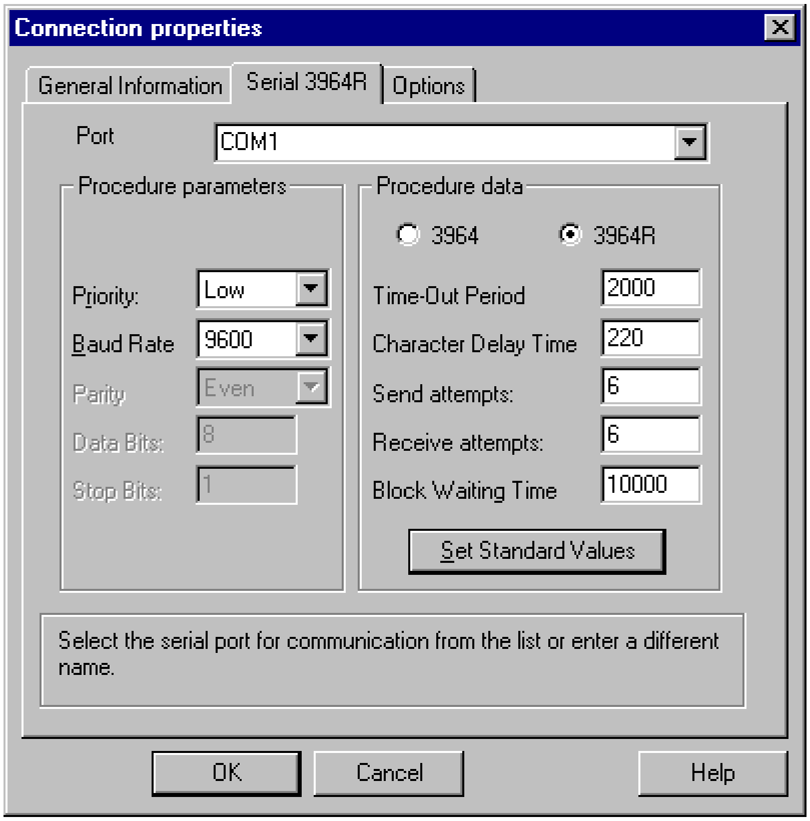
To establish a serial connection, perform the following settings:

#### CP525 settings:

| | | |
|---|---|---|
| Message: | Parameter CP525 Name: | P3964R |
| Procedure: | COMPONENT: RK | Version : 01 |
| Baud rate:: | 9600 | Signal element length: 8 |
| Number of stop bits: 1 | Priority: LOW | |
| Parity: | EVEN | |

On the PLC, you require SYNCHRONOUS in the startup circuit for the CP525 and SEND/RECEIVE ALL in the cyclic program.

#### WinCC settings:



Connection properties

Port: COM1

Procedure parameters

Priority: Low

Baud Rate: 9600

Parity: Even

Data Bits: 8

Stop Bits: 1

Procedure data

○ 3964 ● 3964R

Time-Out Period: 2000

Character Delay Time: 220

Send attempts: 6

Receive attempts: 6

Block Waiting Time: 10000

Select the serial port for communication from the list or enter a different name.

For optimization purposes, one of the peers - preferably WinCC - should have the priority *high*.

## 5.1.6 Color table

The color values are composed from a large palette.
The sixteen basic colors are:

| Color | Color Value (hex) | Symbolic Constant |
|---|---|---|
| Red | 0x000000FF | CO_RED |
| Dark red | 0x00000080 | CO_DKRED |
| Green | 0x0000FF00 | CO_GREEN |
| Dark green | 0x00008000 | CO_DKGREEN |
| Blue | 0x00FF0000 | CO_BLUE |
| Dark blue | 0x00800000 | CO_DKBLUE |
| Cyan | 0x00FFFF00 | CO_CYAN |
| Dark cyan | 0x00808000 | CO_DKCYAN |
| Yellow | 0x0000FFFF | CO_YELLOW |
| Dark yellow | 0x00008080 | CO_DKYELLOW |
| Magenta | 0x00FF00FF | CO_MAGENTA |
| Dark magenta | 0x00800080 | CO_DKMAGENTA |
| Light gray | 0x00C0C0C0 | CO_LTGRAY |
| Gray | 0x00808080 | CO_DKGRAY |
| Black | 0x00000000 | CO_BLACK |
| White | 0x00FFFFFF | CO_WHITE |

*symbolic constant predefined externally by #define.*
Mixed colors are produced by means of intermediate values on the palette.

If color changes are created with the help of the Dynamic dialog box and the configured data are subsequently processed with the C actions, the color values can similarly be read out, though they are in decimal format.

## 5.2 Documentation of S5 Alarm Logging

### Task and function of S5 alarm logging

The present document describes the functions and properties of the SIMATIC S5 software:

**S5 alarm logging**
The software is used to ensure binary messages are acquired in the correct chronological order, to process them and to store them temporarily. The program package provides the requisite software functionality within the SIMATIC S5 to implement chronologically correct message acquisition of the WinCC system.

The basic method of operation of the software can be described as follows. The software monitors the binary signal condition of the messages which the user makes available to the S5 alarm system at a message interface. If a signal condition changes, the message is identified by means of its message number and date/time stamped. To these data are added a 32-bit process variable and an alphanumeric job/batch identifier - if configured by the user. The message block configured in this manner is stored temporarily, if required, in a FIFO buffer. Temporary storage of message data is necessary whenever more message are issued per unit of time than can be transferred over an existing BUS connection to the WinCC system. This functionality makes it possible to achieve decoupling with respect to time between chronological message acquisition in the SIMATIC S5 and higher-level WinCC alarm logging and enables message processing with real-time compatibility.

The message blocks generated by S5 alarm logging are made available to the S5 user program at a data block interface. By using S5 communication software, which has to be implemented by the user, these data are transferred over a BUS connection - for example, SINEC H1 - to higher-level WinCC alarm logging. A wide range of processing functions is available for the messages at that level - for example, visualization, archiving, reporting, etc.

The configuration of S5 alarm logging is performed by the user by means of a data block interface (system DB 80). At this point the user determines the system envelope within which alarm logging works. Definitions of the memory areas used by S5 alarm logging, the type and scope of the messages that have to be processed and allocation of the assigned address areas are specified at this point.

The present section describes the use and handling of S5 alarm logging in a SIMATIC S5 environment. The user is provided with an overview of the function and data blocks used by the software, and the storage space required. This is followed by an in-depth interface description of all the existing data interfaces between S5 alarm logging and the S5 user program. To help you, an example project is presented to make S5 alarm logging easier for the user when starting.

### 5.2.1 List of software blocks

The SIMATIC S5 software can be found on the CD together with the examples relating to this manual is stored under the file name *'WINCC1ST.S5D'*.
The file contains the following function and data blocks for S5 alarm logging:

| FB | Name | Size | Function |
|---|---|---|---|
| FB 80 | SYSTEMFB | 1114 | Sequenced Reporting |
| FB 81 | ANLAUFFB | 135 | Startup and initialization of Sequenced Reporting |
| FB82 | PCHECK | 574 | Called by FB 81 |
| FB 83 | MBLOCK | 699 | Called by FB 80 |
| FB 84 | SCHREIB | 94 | Called by FB 80 |
| FB 87 | VOLL | 87 | Called by FB 80 |
| DB 80 | System DB | 512 | Assign Parameters to Alarm Logging |
| All | | 2703 | |

Table 1

The minimum storage requirement depends on the configuration of S5 alarm logging. The following data blocks are also required.

| | | |
|---|---|---|
| *FIFO buffer (min.)* | *2 DB* | *= 1024 bytes* |
| *Transfer mailbox for WinCC* | *1 DB* | *= 512 bytes* |

512 bytes have to be allowed additionally for every offset and parameter data block.

Calculation of the exact size of the offset data blocks will follow in Section 5.2.4.1 Structure of offset data block, and that of the size of the parameter data block concerned in Section 5.2.4.10 Structure of the parameter data block.

## 5.2.2 Hardware requirements

The function blocks specified in Table 1 for S5 alarm logging require the following hardware for them to be executed correctly:

| PLC | CPU |
|---|---|
| AG 115U | CPU 944 * , CPU 945 |
| AG 135U | CPU 928B |
| AG 155U | CPU 946/ 947, CPU 948 |

Table 2

* Only the CPU 944 featuring two PU interfaces has a system clock

These CPUs have an internal clock and are this in a position to provide the current date and time for construction of the message blocks.
For every WinCC channel created, an up-to-date date/time message frame is read periodically to the SIMATIC S5 CPU. The internal clock of the SIMATIC S5 is synchronized with the WinCC system clock by means of function block **FB 86 : MELD:UHR**.

### 5.2.3 Embedding S5 alarm logging in the SIMATIC-S5 user program

To embed the SIMATIC S5 software for alarm logging in the SIMATIC S5 user program, perform the following steps:

Transfer all the blocks specified in the Table 1 from the file called **WinCCST.S5D** to the corresponding PLC.
If not already implemented by default or not yet available on the PLC, transfer the data handling blocks for the PLC concerned.

| Step | Action: Embed alarm logging |
|---|---|
| 1 | Transfer all the blocks specified in the Table 1 from the file called **WinCCST.S5D** to the corresponding PLC. |
| 2 | If not already implemented by default or not yet available on the PLC, transfer the data handling blocks for the PLC concerned. |
| 3 | Assign parameters to data block DB 80 as described in Section 5.2.6 |
| 4 | Create data blocks for the send mailbox, the FIFO buffer, the message offset and, if necessary, the message parameters (refer to Section 5.2.4). |
| 5 | Initialize offset data blocks for the different message categories (refer to Section 5.2.4.1- idle condition, basic message number, etc.). |
| 6 | Specify process variables, job and batch identifier for the different messages in the user program (refer to Section 5.2.4.10). |
| 7 | Call the following blocks in the startup OBs (OB 20, OB 21, OB 22):<br>• **SPA HTB : SYNCHRON** (the data handling block of the CPU concerned)<br>SPA FB 81 : ANLAUFFB |
| 8 | Call the following blocks in OB 1:<br>• for periodic processing of messages **SPA FB 80 : SYSTEMFB**<br>• a user-created function block for transferring message blocks to the higher-level WinCC system (refer to Section 5.2.5.4) |
| 9 | Insert additional functions as described in the following sections:<br>• Synchronization of date and time by means of FB 86 : MELD:UHR (refer to Section 5.2.8). |

Table 3

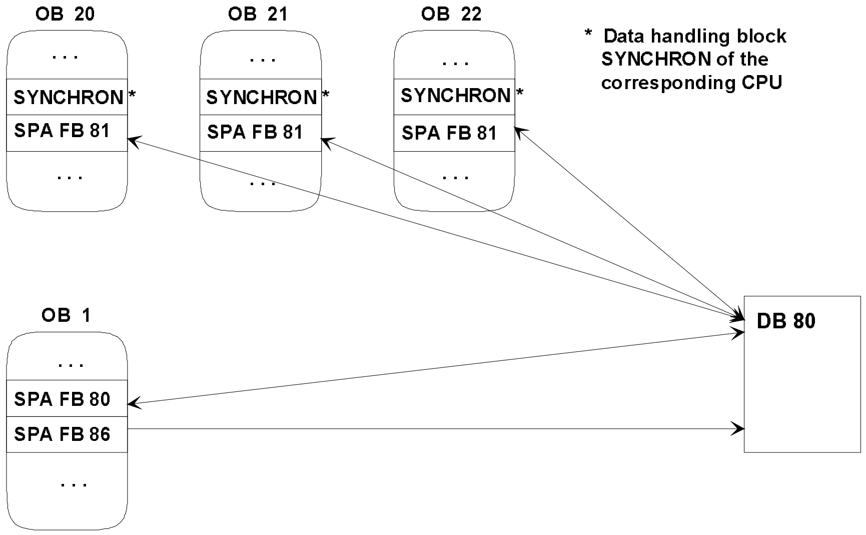

OB 20
. . .
SYNCHRON *
SPA FB 81
. . .
OB 21
. . .
SYNCHRON *
SPA FB 81
. . .
OB 22
. . .
SYNCHRON *
SPA FB 81
. . .
* Data handling block SYNCHRON of the corresponding CPU
OB 1
. . .
SPA FB 80
SPA FB 86
. . .
DB 80

Figure 1

## 5.2.4 General description of S5 alarm loggings

A description is presented for the following components of S5 alarm logging:

- offset data block
- parameter data block
- message block
- FIFO buffer
- send mailbox
- system data block

Relationship between the different components:

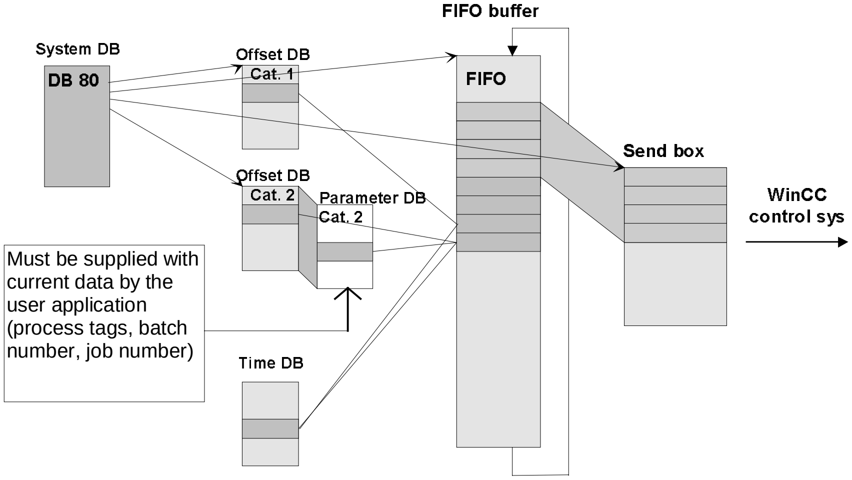


Figure 2

Before the message acquisition system can monitor and acquire messages, they have to be configured in the corresponding data blocks. We distinguish between four categories of message:

| Catego ry | Definition: Message Categories |
|---|---|
| 1 | Message without parameters |
| 2 | Message with process variables (2 DW) |
| 3 | Message with process variables (2 DWs) and job/batch identifier (3 DWs) |
| 4 | Message with process variables (2 DWs) and job/batch identifier (3 DWs) and reserve (3 DWs) |

Table 4

For alarm logging, you can globally specify a date/time stamp for the construction of message blocks. If the date/time stamp is missing, the corresponding information is added by the WinCC system to the message blocks (refer to Section 5.2.4.11).

### 5.2.4.1 Structure of offset data block

The offset data block has an identical structure in all message categories. The corresponding data block address is specified for every message category required (refer to 5.2.4.1) in system data block DB 80.

Offset data block for corresponding message category:

| DW | Contents | Assignment |
|---|---|---|
| DW 0 | Not assigned | Head |
| DW 1 | Basic message number | |
| DW 2 | Address of last signal condition block | |
| DW 3 | Not assigned | |
| | | |
| DW 4 | Signal conditions of messages - bit No.<br>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | Signal condition block 1 |
| DW 5 | Idle condition bits | |
| DW 6 | Acknowledgement bits | |
| DW 7 | Edge trigger flags | |
| | | |
| DW 8 | Signal conditions of messages - bit No.<br>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | Signal condition block 2 |
| DW 9 | Idle condition bits | |
| DW 10 | Acknowledgement bits | |
| DW 11 | Edge trigger flags | |
| | | |
| DW 12 | Signal conditions of messages - bit No.<br>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | Signal condition block 3 |
| DW 13 | Idle condition bits | |

Table 5

The following items are described below:

- Basic message number
- offset message number
- signal condition block
- Address of last signal condition block
- signal conditions
- Idle condition bits
- Acknowledgement bits
- Edge trigger flags

### 5.2.4.2 Basic message number

Every message is assigned a certain message number by means of which you can distinguish between the messages that are issued. The message number consists of the basic message number and an offset message number.
A different basic message number has to be specified for every message category you use.
Messages of same basic message number are numbered serially by means of the offset message number.
The basic message number for the corresponding message category is specified in DW 1 of the offset data block concerned (refer to 5.2.4.10).

### Special case

When message category 1 is being used, two offset data blocks are possible. To achieve serial message numbering for this message category, you have to enter the basic message number of the second offset data block on the basic message number of the first offset data block plus its message capacity (1008 messages).

**Calculating the message number:**
*Message number = basic message number + offset message number*

### Example:

| Calculation | Description: |
|---|---|
| Given: | Message category 1, serial message numbering starting at: message number 10000 |
| Required: | Basic message number of the two offset data blocks |
| 10000 | Basic message number of the first offset data block |
| 10000 + 1008 = 11008 | Basic message number of the second offset data block |

### 5.2.4.3 Offset message number/signal conditions of messages

The signal conditions of messages are contained in the offset data blocks of the corresponding message category at the bit position concerned of the offset message number.
The offset message number of the corresponding message results starting at the 16 bits (bits 0-15) of DW 4. Serial numbering is performed in increments of four (DW8, DW12, etc.).

| Signal condition block | Signal Condition Block Starts at Data Word | Bit Number 0 - 15 Corresponds to Offset Message Number |
|---|---|---|
| 1 | 4 | 0 - 15 |
| 2 | 8 | 16 - 31 |
| 3 | 12 | 32 - 47 |
| 4 | 16 | 48 - 63 |
| ... | | ... |
| 62 | 248 | 976 - 991 |
| 63 | 252 | 992 - 1007 |

Table 6

**Calculating the offset message number:**

*Offset message number = message number - basic message number*<br>
*offset message number = (data word / 4 -1) * 16 + bit No. (0-15)*<br>
*offset message number = (signal condition block - 1) * 16 + bit No. (0-15)*

**Calculating DB, DW, bit No. from the offset message number:**

*Data block = offset data block*<br>
*Data word = (offset message number / 16+ 1) * 4*<br>
*Bit No. = offset message number % 16*

**In the case of message category 1, the length of a data word may be greater than 252, in which case the following applies:**

*Data block = offset data block+ 1*<br>
*Data word = data word - 252*<br>
*Bit No. = Bit No.*

## Example 1:

*Given: DW 248, bit 7, basic message number = 10000*<br>
*Required: Message number*

*signal condition block = 248 / 4*<br>
*= 62*<br>
*offset message number = (signal condition block - 1) * 16 + bit number*<br>
*= (62 - 1) * 16 + 7 = 983*<br>
*Message number = basic message number + offset message number*<br>
*= 10000 + 983 = 10983*

**The required message number is 10983.**

## Example 2:

*Given: Message category 1 with two offset data blocks,*<br>
*message number = 12000, basic message number = 10000*<br>
*Required: DB, DW, bit No.*

*offset message number = message number - basic message number*<br>
*= 12000 - 10000 = 2000*<br>
*Bit No. = offset message number % 16 = 0*<br>
*Data word = (offset message number / 16+ 1) * 4*<br>
*= (2000 / 16 + 1) * 4 = 504*

**The data word is greater than 252.**

*Data block = offset data block+ 1*<br>
*Data word = 504 - 252 = 252*<br>
*Bit No. = 0*

**Message number 12000 will be found in the second offset data block of message category 1, data word 252, bit No. 0.**

### 5.2.4.4 Signal condition block

The first signal condition block starts at data word address 4, the next signal condition blocks follow at intervals of 4 data words (DW 8, DW 12, etc.).
Refer also to Table 5 and Table 6.

63 signal condition blocks are possible for every offset data block (signal condition blocks 1 to 63).

A signal condition block contains 16 signal conditions. There are thus 63 * 16 = 1008 messages possible in an offset data block.

### Structure of signal condition block:

| DW | Bit number | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | signal conditions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | Idle conditions | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | Acknowledgement bits | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | Edge trigger flags | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 7

Additional information on these four bit states is provided in this chapter.

**calculating the signal condition block concerned:**
*Signal Condition Block = (offset message number / 16) + 1*
*signal condition block = data word / 4*

**Calculating the data word which starts at the signal condition block concerned:**
*First data word of the signal condition block = signal condition block * 4*

### 5.2.4.5 Address of the last signal condition block

Specifying the DW address, the last signal condition block to be assigned messages specifies the number of possible messages of the corresponding message category.

**Calculating the last signal condition block:**

*Last signal condition block = required messages in this message category / 16*

```
// Incompletely filled (16 messages) signal condition block
if ((required messages of this message category % 16) != 0)
{
      ++ last signal message block;
}
```

**With message category 1, a message volume of more than 1008 messages may occur, in which case the following applies:**

### 1st offset DB:

*Last signal condition block = 63*
*Address of last signal condition block = 63 * 4 = 252*

### 2nd offset DB:

*Last signal condition block = (required messages in this message category - 1008) / 16*

```
// Incompletely filled (16 messages) signal condition block
if (((required messages in this message category - 1008) % 16) != 0)
{
      ++ last signal message block;
}
```

**Calculating the DW address of the last signal condition block:**
*DW address of last*
*signal condition block = last signal condition block * 4*

### Example:

*Given: 1030 messages of message category 1*

### 1st offset DB:

*Address of the last signal condition block = 63 * 4 = 252*

### 2nd offset DB:

*Required messages - 1008 = 1030 - 1008 = 22*
*(Required messages - 1008) / 16 = 22 / 16 = 1*
*(Required messages - 1008) % 16 = 22 % 16 = 6*
*Last signal condition block = 2*
*Address of last signal condition block = 2 * 4 = 8*

### 5.2.4.6 Signal conditions

Position: 1st data word of the signal condition block (refer to Table 5).

The user must make sure that the signal conditions of the corresponding messages are entered in the data words provided for them in the offset data blocks of the corresponding message category. This can be performed by the PLC by means of continual signal updating during the process.

### 5.2.4.7 Idle conditions

Position: 2nd data word of the signal condition block (refer to Table 5).

By the idle condition of a signal we mean the signal level in the passive operating state. This defines whether a signal (message) is active 'low' or 'high'. This information is required to find out whether a message is 'arriving' or 'departing'.
If a change of event has the negated state with regard to the idle condition, it is a message which is 'arriving'. In the case of a 'departing' message, the state of the change of event is identical to that of the associated idle condition.

The idle conditions of messages has to be specified by the user at the corresponding positions.

### 5.2.4.8 Acknowledgement bits

Position: 3rd data word of the signal condition block (refer to Table 5).

Acknowledgement bits are not configured but evaluated within the current program. In this instance, the messages are acknowledged directly by the higher-level PC in accordance with the acknowledgement philosophy concerned. These message-related acknowledgements are sent by the PC to the PLC concerned with the associated configured messages of the integrated message acquisition system.

The corresponding acknowledgement bit is set one PLC cycle long by S5 alarm logging. The application program has to evaluate this information accordingly.

### 5.2.4.9 Edge-triggered flags

Position: 4th data word of the signal condition block (refer to Table 5).

The edge-triggered flags are used to determine any change of events which may have occurred (change of message). They are not configured but evaluated within S5 alarm logging.

### 5.2.4.10 Structure of the parameter data block

For message categories 2 to 4, it is necessary to configure parameter data blocks for additional data of the message concerned, in addition to an offset data block. The signal condition of a message is stored in the offset data block. The addresses of the parameter data blocks are stored in consecutive data blocks and are directly contiguous with the corresponding offset data blocks.

**Relationship between the offset and parameter data blocks:**

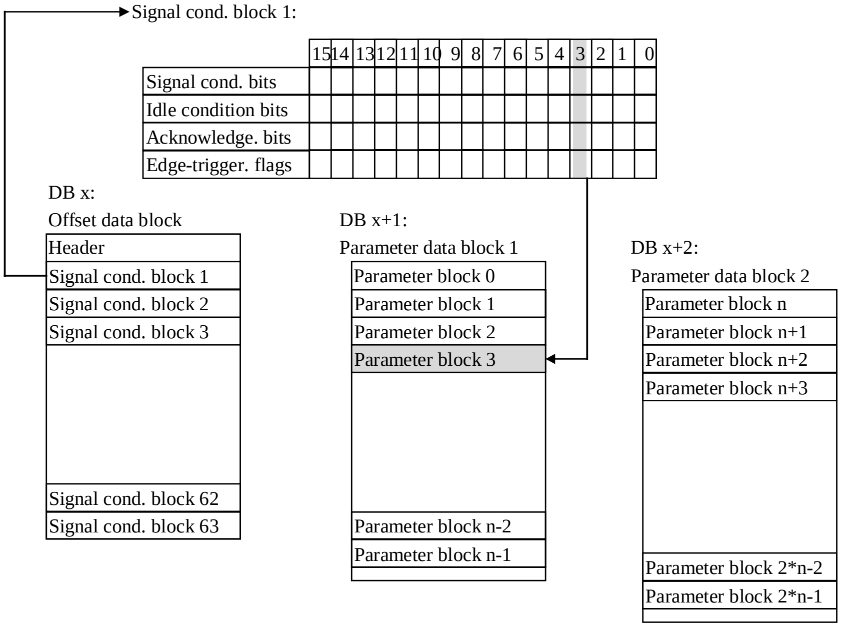


Figure 3

| Category | Max. Number | Size of Parameter Block | No. of Blocks per Parameter DB | Max. No. of Parameter DBs |
|---|---|---|---|---|
| 1 | 1008 / 2016 | - | - | - |
| 2 | 1008 | 2 DW | 128 | 8 |
| 3 | 1008 | 5 DW | 51 | 20 |
| 4 | 1008 | 7 DW | 36 | 28 |

Table 8

**Calculating the number of parameter data blocks:**

$$\text{Number of Parameters Data Block} = \frac{\text{used Messages}}{\text{Number of parameter blocks per parameter data block}}$$

When configuring, care must be taken to ensure that addresses do not overlap with data blocks of another message category and that the number of parameter data blocks can cope with any future upgrade.
A parameter data block contains parameter blocks which are assigned to the different messages. The parameter blocks are stored consecutively, starting with the parameter block for the first message of that message category, in the parameter data block. The parameter blocks are incremented consecutively beyond the borders of the parameter DB. On reaching the end of the parameter DB, the parameter block with the next number is continued from DW 0 onwards of the next parameter DB. Only whole parameter blocks are stored in the parameter data block.

**Calculating the start address of a parameter block:**

| | | |
|---|---|---|
| *Offset message number* | = | *message number - basic message number* |
| *Parameter DB* | = | *offset DB + 1 + (offset message number / parameter blocks per parameter DB)* |
| *Start address of parameter DB* | = | *(offset message number % parameter blocks per parameter DB) * size of parameter block* |

The user must ensure that the corresponding data (process variables, job number, batch identifier) are available at the appropriate address.

### 5.2.4.11 Message block structure

A message block that is sent to the higher-level **WinCC** system consists of several, consecutive data words. The data words contain message-specific information. The sum of the data words result in one message block. The size of the message blocks differs from message category to message category (refer to Table 10).

Irrespective of it message category, a message block always consists of at least two data words. These are the message number and the message status. Depending on whether the messages are date/time stamped (three data words) and have been provided with the appropriate parameters, a message block may be as many as 12 data words long.

| DW | Description: |
|---|---|
| 1st DW | Message number |
| 2nd DW | Message status |
| 3rd DW | Time |
| 4th DW | Time |
| 5th DW | Date |
| 6th DW | Process tag |
| 7th DW | Process tag |
| 8th DW | Job number |
| 9th DW | Job number |
| 10th DW | Batch identifier |
| 11th DW | Reserve |
| 12th DW | Reserve |

Table 9

If messages are not date/time stamped, the three data words required for them at the end of the envisaged third to fifth positions of the block are not needed. The parameter data words are then attached without a gap to the status data word. The size of message blocks (number of DWs) varies from message category to message category and the desired date/time stamp and can be taken from Table 10.

Determining the message block length as a function of message category:

| Category | Message Block Length in DWs without Date and Time | Message Block Length in DWs with Date and Time |
|---|---|---|
| 1 | 2 | 5 |
| 2 | 4 | 7 |
| 3 | 7 | 10 |
| 4 | 9 | 12 |

Table 10

### 5.2.4.12 Message number

Every message is assigned a certain message number by which it can be clearly identified.

### 5.2.4.13 Message status

The message status is structured as follows:

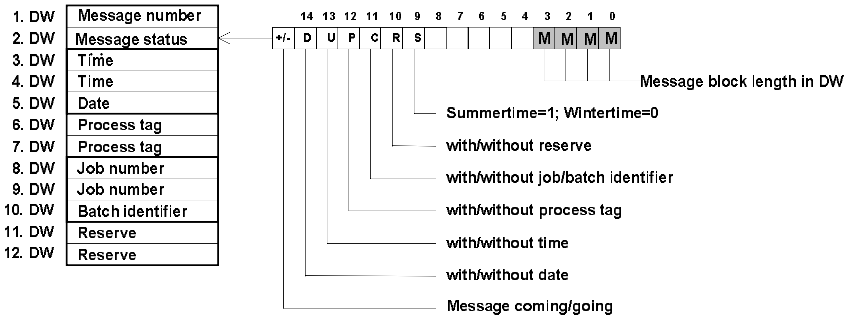


Table 11

### 5.2.4.14 Date/time stamp

The date and time are made available by function block **FB 86 : MELD:UHR** in binary code. Refer to Section 5.2.10 Purpose and function of S5-time synchronization.

### 5.2.4.15 Process variable

Two data words by means of which process variables can be detained upon the arrival of a message and passed to the process system.

### 5.2.4.16 Job number/batch identifier

Depending on the configuration, the first two data words have to be interpreted as a signed 32-bit binary number or as a total of four ASCII characters. The third data word has to be interpreted as two ASCII characters.
The current job number and the batch identifier can be passed to the **WinCC** system by means of these three data words upon the arrival of a message.

### 5.2.4.17 Reserve

The two reserve data words of message category 4 are intended for future add-ins but not yet implemented in the **WinCC** system.

### 5.2.4.18 Construction of a message block

After a message block has been detected, the corresponding message number is determined by the currently checked bit position and stored as the first data word of the message block in the FIFO buffer. Depending on the arriving or departing message, category and requirement for a date/time stamp, the corresponding status mask is selected and stored as the second data word of the message block in the FIFO buffer. If the corresponding bit in the system data block has been assigned parameters for a date/time stamp, the three words now follow which are available in system data block 80 from address DW 190 and thereafter in the required PC format. Depending on the message category, the associated parameter block is read, if necessary, from the corresponding data archive (parameter data block) and added to the last input to finish creating the message block in the FIFO buffer. The next status bit of the message that follows is then examined. This is performed until all parameterized messages have been processes.

### 5.2.4.19 The internal FIFO buffer

A FIFO buffer is a memory at the end of which its beginning follows once again - in other words, the memory area is literally connected like a ring. This results in the memory being limited in size, on the one hand, and not being finite owing to its restarting at the beginning, on the other.
This results in the message acquisition system, upon reaching the virtual end without extraction of the previous data (buffer is full), in the oldest data being overwritten by the most recent data, thus being lost as information.

The FIFO buffer in RAM acts, as its name suggests, as a buffer for the acquired messages before they are forwarded to the PC. In RAM, the FIFO buffer consists of a memory area of at least two data blocks and can, depending on parameter assignment, be made any size required as is allowed by the maximum permissible data blocks of a programmable controller and the remaining available DBs of the user program. The user passes to alarm logging the number of data blocks available to it for archiving.
With more than one data block, data blocks having consecutive DB numbers are imperative. Thus the user specifies the initial DB number and the number of the final DB of the buffer as parameters in the system DB. All data blocks whose values lie between the initial data block and the final data block (including the two data blocks) belong to the buffer as storage space.

### 5.2.4.20 The send mailbox - data transfer to the higher-level WinCC system

As a matter of principle, all message entries of a current cycle are written initially to the internal FIFO buffer of S5 alarm logging.
The message entries - up to a maximum of the contents of one data block - are transferred to the message interface (send mailbox) upon completion of acquisition, providing the send mailbox is ready. The message interface, in the form of a data block, acts as a data source for the transfer function blocks (STEP 5 - data handling blocks). The data handling blocks form an interface to the \corresponding communication processor for the process bus being used - for example, for the SINEC-H1 bus).

#### Send mailbox structure:

| DW | Contents |
|---|---|
| DW 0 | Length of data block |
| DW 1 | KY = [ PLC No. ] , [ CPU No. ] |
| DW 2 | KY = [ 0 ] , [ number of messages ] |
| DW 3 | Useful data start (message blocks) |

Table 12

#### DW 0

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Triggering Edge Length of the Data Block

Table 13

From DW 0 of the send mailbox, it is first necessary to extract the triggering flank of a desired job by means of bit number 14 and, secondly, to determine the source data length from bits Nos. 0-8.

Since the data block which is required to be transferred must not be longer than 256 data words but only one number up to 255 can be represented with one byte, separate polling of the bytes using the commands DL and DR is not possible. We therefore recommend that DW 0 be transferred in an auxiliary flag. This also has the advantage that the enable bit can be evaluated separately and directly. This operation cannot be utilized when data words are being used.

If the condition is fulfilled, the bit which is used as an edge for one-shot triggering of a send job is reset. The remaining set bits then correspond to the transferred source data length and can be written to the data area of indirect parameter assignment as QLAE.
Following a successfully completed WRITE request (SINEC-H1) to the **WinCC**system ('free of faults' (FOF)), DW 0 of the send mailbox has to be overwritten with the value '0'. This re-enables the send mailbox and further message blocks, to the extent present, can be transferred from the internal FIFO buffer to the send mailbox.

The WRITE request (SINEC-H1) has to be implemented by means of the SEND direct function and details will be found in the manual for the PLC concerned.

## 5.2.5 Interface description

Descriptions are provided for the following interfaces and blocks:

- system data block DB 80:
  For assigning parameters to S5 alarm logging.
- offset data block for the corresponding message category:
  Binary interface of the message signals to S5 alarm logging with specification of message characteristics.
- parameter data block for corresponding message category:
  For specifying additional message data of categories 2 through 4.
- Send mailbox:
  transfer interface to the **WinCC** system.

### 5.2.5.1 System data block 80

By using the system data block DB 80, it is possible to configure independent data areas for four message categories, a FIFO memory and a send mailbox. Data words 0 through 20 in DB 80 are earmarked for configuration.
A detailed description of data words 0 through 20 is presented in Section 5.2.6.

### 5.2.5.2 Offset data block

S5 alarm logging evaluates the signal conditions of the corresponding messages and constructs the corresponding message blocks, if necessary.

The user has to make sure that

- the idle conditions of the individual messages are specified during configuration.
- the message states during the runtime of the S5 user program are written to the corresponding signal condition bits.
- the corresponding acknowledgement bits are read out and evaluated, as and when required.

### 5.2.5.3 Parameter data block

With message categories 2 through 4, additional information about the current state of the system can be transferred by means of the message block.

The user has to make sure that

- upon the arrival of a message, the valid process variables (process value, job and batch numbers) are located in the corresponding parameter blocks.

### 5.2.5.4 Send mailbox/transfer mailbox

As soon as it contains message blocks, the send mailbox is transferred with a WRITE request (SINEC-H1) directly to the **WinCC** system (refer to Section 5.2.4.20).

The user has to make sure that

- the corresponding data handling blocks of the CPU concerned are available.
- during configuration of the **WinCC** system, appropriate communication channels are specified for process bus connection.
- a WRITE request, as described in Section 5.2.4.20, is initiated.

### 5.2.6 Assigning parameters to S5 alarm logging/system DB 80

**Description of configurable data words of system data block DB 80:**

| DW | Description |
|---|---|
| 0 | DB address: internal FIFO start |
| 1 | DB address: internal FIFO end |
| 2 | 0: without date and time 1: with date and time |
| 3 | DB offset for category 1 messages |
| 4 | 1: a DB offset of category 1  2: two DB offsets of category 1 |
| 5 | DB offset for category 2 messages |
| 6 | DB offset for category 3 messages |
| 7 | DB offset for category 4 messages |
| 8 | Reserve |
| 9 | Reserve |
| 10 | DB address: send mailbox CPU -> PC |
| 11 | 1: Acquisition optimized (ACOP) |
| 12 | ACOP from n messages onwards |
| 13 | PLC type (115 / 135 / 155) |
| 14 | Reserve (must be 1) |
| 15 | PLC No: 1 to 255; CPU No: 1 to 4 |
| 16 | Reserve |
| 17 | Reserve |
| 18 | Reserve |
| 19 | Reserve |
| 20 | Parity error of the plausibility check |

Table 14

**DW 0, DW 1 : DB memory area of the internal FIFO buffer**

The internal FIFO buffer area for messages is defined by means of the two data words. The storage space must have a size of at least two data blocks and care must taken to parameterize the FIFO end larger than the FIFO start.

The memory area of the buffer memory results from the data block area limited by the FIFO start and FIFO end, including the two specified data blocks.

**Selecting FIFO buffer size:**

When the storage capacity of the FIFO buffer is reached, the oldest messages are overwritten. The number of DBs must be selected large enough so that, in the event of a torrent of messages occurring, no messages are overwritten before they could be paged out. To make sure that this does not occur, the following rule of thumb applies.

**Determining the number of DBs in the FIFO buffer:**
*Messages per DB = (255 DW / DB) / message block length*
*Refer to* Table 10

$$\text{Number of DBs} = \frac{\text{Number of Messages if a Message Overload occurs}}{\text{Messages per DB}}$$

For *'ACOP mode'* of the message acquisition system, it is advisable to allow one or two data blocks more.

**DW 2 : date and time identifier**
The option of date/time stamping messages refers to all messages assigned parameters. Either all the messages which have to be acquired are date/time stamped (DW2 = 1) or none (DW2 = 0). If DW2 = 0 is set, the **WinCC** system adds a date/time stamp to arriving message blocks.

**DW 3, DW 4 : offset DB of message category 1**
If category 1 messages (messages without parameters and batch identifier) have to be to configured, the address of the offset data block has to be specified in data word 3. The signal conditions of these messages have to be written consecutively by the PLC program in the data blocks specified at this point.

If more than 1008 1 messages (not more than 2016 messages) are planned, a further data block is enabled for category 1 messages by entering the figure ’2’ in data word 4. Referred to the address in DW 3, the second data block automatically has the next higher address. With a maximum of 1008 category 1 messages, a ’1’ is entered in DW 4.

**DW 5, DW 6, DW 7 : offset DB of message categories 2, 3, 4**
Similar to DW 3, data words 5-7 contain the data block addresses concerned at which the signals of messages are stored.

DW 5 contains the address of the data block for message category 2, whereas DWs 6 and 7 contain the addresses for message categories 3 and 4, respectively.

If a message category is not used, a ’0’ has to be included in the DW concerned.

The addresses specified in DWs 5-7 are ’offset DBs’. Depending on the message category and the number of messages per category, they are assigned a corresponding number of ’secondary DBs’. The secondary DBs contain the parameters of the messages. For this reason, it is necessary to make sure, when assigning the offset DB addresses, that there is sufficient space (data blocks) for the parameter DBs between the previous offset DBs and the one to be specified.

Up to 1008 messages can be configured for message categories 2 through 4. When fully used, there is consequently a different number of ’secondary DBs’ (parameter DBs) from offset DBs for the different categories (refer to Table 8).

**DW 10 : message interface to higher-level WinCC system**
Parameters must always be assigned to data word 10, though the mode in which the message acquisition system has to operate is of no consequence. The DB address of the transfer mailbox is assigned in DW 10. The transfer mailbox acts as an interface between the SIMATIC S5 and the higher-level **WinCC** system.

**DW 11 , DW 12 : mode selection for optimum acquisition and corresponding number of messages**

Two operating modes are provided:

- ’0’ in DW 11 -> **'Normal Mode'** of the message acquisition system
- ’1’ in DW 11 -> **'Optimum Acquisition Mode'** of the message acquisition system

## Normal Mode:

As many acquired messages are paged out to be sent in the course of a cycle from the internal buffer as the message interface can handle, provided that it is prepared to accept data.

Very large numbers of messages within any cycle or successive cycles will result in this process having a relatively high cycle time. The larger the message blocks of the message categories involved, the longer the cycle time. In this instance, acquisition of message blocks involves more effort and takes longer.

## Optimized acquisition:

With the messages that are issued, chronological acquisition has priority over sending to the PC. Attention is focused on the relative time between the messages issued by the system. The fact that messages might arrive at the PC a few milliseconds later is of secondary importance. The inertia of the human eye and the receptivity of an observer in the control room are decisive for that.

In order to reduce the cycle time of the message acquisition system in such time-critical instances, the option of running this system at ’optimized acquisition operation’ has been introduced. The minimum number of messages issued within an OB1 cycle has to be specified in DW 12. If the number of messages exceeds this minimum number during the current OB1 cycle, the messages are only acquired and buffered. They are not paged out or subsequently sent to a communication peer in this OB1 cycle.

**DW 15 : PLC/CPU number**

This data word is required for constructing the message frame head and needs specification of the project-related PLC and CPU numbers of that programmable controller. The CPU number is particularly important when several CPUs are operating in a single PLC. Only in conjunction with the data word containing the ID for the messages can the higher-level WinCC system interpret sent data as a message which assign message-specific message texts and evaluate them accordingly.

The only data word to have S5 data format ’KY’ during configuration is DW 15. This means that two bytes can be presented separately (separated by a comma). The left byte contains the PLC number, which may be between 1 and 255. The CPU number, which may be between 1 and 4, is specified in the right byte.

## Example:

| | | |
|---|---|---|
| *KY* | = | *10,2* |
| *PLC number* | = | *10* |
| *CPU number* | = | *2* |

**DW 20 : parameter assignment errors**

A plausibility check is performed on all the data words assigned parameters in the system DB upon start-up of S5 alarm logging. During the check, differentiation is made between exceeding possible ranges of values, overlappings and multiple assignment of data blocks to which parameters have been assigned and missing details.

An output parameter in data word format, this function block has a PAFE word (Parameter Error word); it is similar to the system-specific data handling blocks. The status of the PAFE word can be taken from system DB 80 (DW 20). The PAFE word can be examine for errors which may have occurred following the program return from FB 81. Following that, suitable action can be taken.

It is practical to have the programmable controller 'jump' to Stop status in the event of a PAFE word other than zero. If the PAFE word is ignored, the program cannot be guaranteed to run without errors.

**Evaluating the PAFE word**

If the program or PLC is brought to its Stop status, in accordance with the recommendation, following the occurrence of an error (PAFE word <> 0), the error can be specifically analyzed and rectified by means of the error number. The table below provides information on the type of error caused during parameter assignment.

## Format of PAFE word:

*KY* = *error number, group error ID*

## Example:

KY = 9,1

The parameter assignment error having the number 9 corresponds to:

Offset DB addr. of category 1 is larger than the maximum permissible DB address.

| Error No. | Meaning |
|---|---|
| 1 | Start DB of internal buffer not defined |
| 2 | Start DB of internal buffer has the same address as the system DB ('80') |
| 3 | Start DB address of internal buffer larger than maximum permissible DB address |
| 4 | End DB of internal buffer has the same address as the system DB ('80') |
| 5 | End DB address is smaller than start DB address of internal buffer |
| 6 | End DB address of internal buffer larger than maximum permissible DB address |
| 7 | Offset DB of category 1 has the same address as the system DB ('80') |

| Error No. | Meaning |
|---|---|
| 8 | Offset DB addr. of category 1 is within the internal buffer area |
| 9 | Offset DB addr. of category 1 is larger than the maximum permissible DB address. |
| 10 | Offset DB of category 2 has the same address as the system DB ('80') |
| 11 | Offset DB of category 2 has the same address as that of category 1 |
| 12 | Offset DB of category 2 has the same address as the 2nd offset DB of category 1 |
| 13 | Offset DB addr. of category 2 is within the internal buffer area |
| 14 | Offset DB addr. of category 2 is larger than the maximum permissible DB address. |
| 15 | Offset DB of category 3 has the same address as the system DB ('80') |
| 16 | Offset DB of category 3 has the same address as that of category 1 |
| 17 | Offset DB of category 3 has the same address as the 2nd offset DB of category 1 |
| 18 | Offset DB of category 3 has the same address as that of category 2 |
| 19 | Offset DB addr. of category 3 is within the internal buffer area |
| 20 | Offset DB addr. of category 3 is larger than the maximum permissible DB address. |
| 21 | Offset DB of category 4 has the same address as the system DB ('80') |
| 22 | Offset DB of category 4 has the same address as that of category 1 |
| 23 | Offset DB of category 4 has the same address as the 2nd offset DB of category 1 |
| 24 | Offset DB addr. of category 4 is within the internal buffer area |
| 25 | Offset DB addr. of category 4 is larger than the maximum permissible DB address. |
| 26 | Offset DB of category 4 has the same address as that of category 2 |
| 27 | Offset DB of category 4 has the same address as that of category 3 |
| 28 | The PC send mailbox has the same address as the system DB ('80') |
| 29 | The PC send mailbox is not defined ('0') |
| 30 | PC send mailbox addr. within internal buffer area |
| 31 | PC send mailbox addr. larger than maximum permissible DB address |
| 32 | The PC send mailbox has the same address as the offset DB of category 1 |
| 33 | The PC send mailbox has the same address as the offset DB of category 2 |
| 34 | The PC send mailbox has the same address as the offset DB of category 3 |
| 35 | The PC send mailbox has the same address as the offset DB of category 4 |
| 36 | The PC send mailbox has the same address as the 2nd offset DB of category 1 |
| 37 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 38 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 39 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 40 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 41 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |

| Error No. | Meaning |
|---|---|
| 42 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 43 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 44 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 45 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 46 | Reserve DW 9 and/or reserve DW 10 not equal to 0 |
| 47 | Number of messages for the minimum limit of the selected operating mode with optimum acquisition is missing |
| 48 | PLC type is not defined |
| 49 | Reserve DW 14 not equal to 1 |
| 50 | PLC No. for message frame head is not defined |
| 51 | CPU No. for message frame head is not defined |
| 52 | CPU No. is larger than allowed (1 to 4) |

Table 15

## 5.2.7 S5 alarm logging configuration example

### Description

S5 alarm logging is required to be configured for the following message categories:

| Category | Definition: Message Categories |
|---|---|
| 1 | 1200 messages (from message number 10000 through 11199)<br>Messages 11000 through 11199 are ’low active’ |
| 2 | No messages planned |
| 3 | 11 messages (from message number 30000 through 30010) |
| 4 | No messages planned |

All messages are to be date/time stamped.
A 135U, PLC No. 1, CPU No. 1 are in use.

### 5.2.7.1 Parameter assignment DB 80

| Category | Max. Number | Size of Parameter Block | No. of Blocks per Parameter DB | Max. No. of Parameter DBs |
|---|---|---|---|---|
| 1 | 1008 / 2016 | - | - | - |
| 2 | 1008 | 2 DW | 128 | 8 |
| 3 | 1008 | 5 DW | 51 | 20 |
| 4 | 1008 | 7 DW | 36 | 28 |

DB 81 is used as the PC send mailbox.

With a uniform occurrence of the existing messages, the average message block length (with date and time) is:
*(1200 * 5 + 11 * 10) / (1200 + 11) = 5.04*

**Assumption:**
S5 alarm logging is required to accommodate a messages overload of 100 messages in one PLC cycle and to operate at 30 messages or more in ’Optimum Acquisition mode’.

*5 DW/mess. * 100 mess. = 500 DWs*
*(500 DW) / (256 DW/DB) = 1.95 DBs*

There therefore results four data blocks for the FIFO buffer, since one or two additional data blocks have to be allowed for ’Optimum Acquisition mode’.
The FIFO buffer starts at data block address 82 and thus the end address of the FIFO buffer is DB 85.

To provide a reserve for any future upgrade of the FIFO buffer, the offset data block of category 1 is in DB 88 and DB 89 (more than 1008 category 1 messages).

DB 90 becomes the offset data block of message category 3. A parameter DB of message category 3 can accommodate 51 parameter blocks; if the 11 blocks used are subtracted, this results in an upgradability 40 category 3 messages with only one parameter data block (DB 91).

| DW | Description | Value |
|---|---|---|
| 0 | DB address: internal FIFO start | 82 |
| 1 | DB address: internal FIFO end | 85 |
| 2 | 0: without date and time<br>1: with date and time | 1 |
| 3 | DB offset for category 1 messages | 88 |
| 4 | 1: one DB offset of category 1<br>2: two DB offsets of category 1 | 2 |
| 5 | DB offset for category 2 messages | 0 |
| 6 | DB offset for category 3 messages | 90 |
| 7 | DB offset for category 4 messages | 0 |
| 8 | Reserve | 0 |
| 9 | Reserve | 0 |
| 10 | DB address: send mailbox CPU -> PC | 81 |
| 11 | 1: Acquisition optimized (ACOP) | 1 |
| 12 | ACOP from n messages onwards | 30 |
| 13 | PLC type (115 / 135 / 155) | 135 |
| 14 | Reserve | 1 |
| 15 | PLC No: 1 to 255; CPU No: 1 to 4 | 1, 1 |
| 16 | Reserve | 0 |
| 17 | Reserve | 0 |
| 18 | Reserve | 0 |
| 19 | Reserve | 0 |
| 20 | Parity error of the plausibility check | 0 |

Data block 100 is used by DW 10 through DW 20 to synchronize the time.
Data block 101 is used by DW 0 through DW 255 for receiving commands.

### 5.2.7.2 Creating data blocks

Create data blocks DB 81 - DB 85, DB 88 - DB 91 and DB 101 of DW 0 - DW 255.
Create data block DB 100 of DW 0 - DW 20.

### 5.2.7.3 Initializing offset data blocks

**Message category 1**
DB 88 and DB 89 are provided for message category 1. DB 88 contains the messages numbered 10000 through 11007, and DB 89 the messages numbered 11008 through 11199. A total of 1200 category 1 messages have to be configured.

See Chapter 5.2 Address of the last signal condition block .
Offset message number = message number - basic message number = 0 through 1199

#### 1st offset DB:

address of last signal condition block: DW 252

#### 2nd offset DB:

address of last signal condition block: DW 252

*1200 - 1008 = 192*

*192 / 16 = 12*
*192 % 16 = 0*

*Address of the last*
*signal condition block in offset data block 2 = 12 * 4 = 48*

#### DB 88:

| DW | Description | Value |
|---|---|---|
| DW 0 | Not assigned | |
| DW 1 | Basic message number | 10000 |
| DW 2 | Address of last DW | 252 |
| DW 3 | Not assigned | |

#### DB 89:

| DW | Description | Value |
|---|---|---|
| DW 0 | Not assigned | |
| DW 1 | Basic message number | 11018 |
| DW 2 | Address of last DW | 48 |
| DW 3 | Not assigned | |

See Chapter 5.2.4.3 Offset message number/signal conditions of messages.

Messages 11000 through 11199 are ’low active’.

**Position of the idle condition bit of message number 11000:**

| | |
|---|---|
| *Offset message number:* | *11000 - 10000 = 1000* |
| *Start of signal condition block* | *(offset message number / 16 + 1) * 4 =*<br>*= (62 + 1) * 4 * DW 252* |
| *Data word of the idle condition bit:* | *DW 253* |
| *Data bit:* | *offset message number % 16 = 8* |
| *Data block:* | *offset data block = DB 88* |

**Position of the idle condition bit of message number 11000:**

| | |
|---|---|
| *Offset message number:* | *11199 - 10000 = 1199* |
| *Start of signal condition block:* | *(offset message number / 16+ 1) * 4 =*<br>*= (74 +1) * 4 = 300*<br>*300 - 252 = 48* |
| *Data word of the idle condition bit:* | *DW 49* |
| *Data bit:* | *offset message number % 16 = 15* |
| *Data block:* | *offset data block + 1 = DB 89* |

**The following idle condition bits have to be modified:**

## DB 88:

DW 253: set data bits 8 through 15 to ’1’

## DB 89:

DW 5, DW 9, DW 13, through DW 49 : set data bits 0 through 15 to ’1’

**Message category 3**

For message category 3, DB 90 is provided as the offset data block with messages 30000 through 30010, and DB 91 as the parameter data block.
A total of 11 category 3 messages have to be configured.
See Chapter 5.2.4.10 Structure of the parameter data block.

*OffsetMessageNo. = MessageNo. - BasicMessageNo. = 0 through 10*

## Offset DB:

| | | |
|---|---|---|
| **Address of the last signal condition block** | | *11 / 16 = 0*<br>*11 % 16 = 11* |
| *Address of last signal condition block* | = | *(0+1) * 4 = 4* |

## DB 89:

| DW | Description | Value |
|---|---|---|
| DW 0 | Not assigned | |
| DW 1 | Basic message number | 30000 |
| DW 2 | Address of last DW | 4 |
| DW 3 | Not assigned | |

All idle condition bits are '0'.
See Chapter 5.2.4.10 Structure of the parameter data block

Parameter DB = *offset DB + 1 + (offset message number / parameter blocks per parameter DB)*

**Message number 30000:**
*Parameter DB = 90 + 1 + 0 / 51 = 91*

**Message number 30010:**
*Parameter DB = 90 + 1 + 10 / 51 = 91*

*Start address of the parameter block concerned* = *(offset message number % parameter blocks per parameter DB) * size of parameter block*

**Message number 30000:**
*Start address of the parameter block concerned = (0 % 51) * 5 = DW*

**Message number 30010:**
*Start address of the parameter block concerned = (10 % 51) * 5 = DW 50*

**DB 91: parameter data block 91 for offset data block 90**

| Message number | Process Values | Job number | Batch identifier |
|---|---|---|---|
| 30000 | DW 0, 1 | DW 2, 3 | DW 4 |
| 30001 | DW 5, 6 | DW 7, 8 | DW 9 |
| 30002 | DW 10, 11 | DW 12, 13 | DW 14 |
| 30003 | DW 15, 16 | DW 17, 18 | DW 19 |
| 30004 | DW 20, 21 | DW 22, 23 | DW 24 |
| 30005 | DW 25, 26 | DW 27, 28 | DW 29 |
| 30006 | DW 30, 31 | DW 32, 33 | DW 34 |
| 30007 | DW 35, 36 | DW 37, 38 | DW 39 |
| 30008 | DW 40, 41 | DW 42, 43 | DW 44 |
| 30009 | DW 45, 46 | DW 47, 48 | DW 49 |
| 30010 | DW 50, 51 | DW 52, 53 | DW 54 |

## 5.2.8 Documentation of the SIMATIC S5 command blocks

### Purpose and function of S5 command blocks

The software is used to 'process' bits, bytes, words and double words in the SIMATIC S5 over a process bus - for example, SINEC H1. Only byte and word values in the SIMATIC S5 can be addressed over the process bus.
By default, the following operations can be executed:

- Data blocks (DB and DX), timers and counters have to be changed only as words.
- Bit memories, inputs, outputs, I/O (P and Q) have to be changed only as bytes.

The program package provides the requisite software functionality within the SIMATIC S5 to implement the following operations on the WinCC system over the given process bus.

- Set an initializing pulse for one OB1 cycle
- Set, reset, invert the bit in DB/DX
- Set, reset, invert the bit in bit memory
- Write left/right bytes to DB/DX
- Write word/double word to DB/DX
- Write byte/word to bit memory
- Write byte/word to I/O
- Write byte/word to upgraded I/O

The required changes in the SIMATIC S5 are made available as a raw data tag by the WinCC Control Center across a data interface. The instructions have to be sent to the S5 by means of a raw data tag. These instructions are evaluated directly in the S5 by instruction interpreter **FB 87 : EXECUTE** and executed.

The present manual describes the use and handling of S5 command blocks in a SIMATIC S5 environment. The user is provided with an overview of the function and data blocks used by the software, and the storage space required. There follows a detailed interface description of the existing data interface. An example configuration is presented to help you.

### 5.2.8.1 List of software blocks

The *'S5 command blocks'* SIMATIC S5 software will be found on the WinCC CD in the file called **WINCC1ST.S5D**.
The file contains the following function blocks for the S5 command blocks:

| FB | Name | Size in Bytes | Function |
|---|---|---|---|
| FB 87 | Execute | 152 | Enables bit, byte, word and double word manipulation over the process bus |
| FB 88 | OPCODE | 399 | Called by FB 87 |
| All | | 551 | |

Table 16

In addition, a 512-byte command data block is also required.

### 5.2.8.2 Hardware requirements

The function blocks specified in Table 16 require the following hardware for them to be executed correctly:

| PLC | CPU |
|---|---|
| AG 115U | CPU 943, CPU 944, CPU 945 |
| AG 135U | CPU 928A, CPU 928B |
| AG 155U | CPU 946/ 947, CPU 948 |

### 5.2.8.3 Call parameters of FB 87: EXECUTE

A description of the call parameters of function block **FB 87 : EXECUTE** is presented in the following.

| Name | Execute | Parameters |
|---|---|---|
| ID: | DBNR | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF |
| ID: | DBDX | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF |
| ID: | RIMP | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY |

**DBNR:** Data block number of the command transfer interface

**DBDX:** Type of data source for the command transfer interface

DB.......Data source for a data block (DB).
DX.......Data source for an extended data block (DX).

**RIMP:** Bit position for the initializing pulse

RIMP.......Bit memory number, bit number

### 5.2.9 Interface description

Descriptions are provided for the following interfaces and blocks:

- command function block FB 87
- command data block: instruction transfer interface to the SIMATIC S5.

In the SIMATIC S5, the instruction interpreter (**FB 87 : EXECUTE**) is called periodically in OB 1. The type and address of the command DBs are transferred as parameters. When a command is queued, the Op code and four parameters are forwarded to **FB 88 : OPCODE** and executed directly. After an instruction has been executed, the command counter (DW 1) is decremented by one. The process of instruction transfer and decrementing of the command counter is repeated until all queued instructions have been processed.

Details of the type and address of the data block have to agree in both the WinCC Control Center and S5 program, and the data block has to be present in the S5. A DB data block and a DX data block and their addresses - for instance, DX 234 - are available to chose from. The data block has to be opened by the user up to data word 255, since data words 0 - 255 can be addressed in the data block specified.

**The following syntax has been defined for the commands stored in the command data block:**

| DW | Description |
|---|---|
| 0 | Not used |
| 1 | Number of commands to be executed |
| 2 | Op code of the first command |
| 3 | Parameter 1 (Op code 1) |
| 4 | Parameter 2 (Op code 1) |
| 5 | Parameter 3 (Op code 1) |
| 6 | Parameter 4 (Op code 1) |
| 7 | Op code of the second command |
| 8 | Parameter 1 (Op code 2) |
| 9 | Parameter 1 (Op code 2) |
| 10 | Parameter 2 (Op code 2) |
| 11 | Parameter 3 (Op code 2) |
| 12 | Parameter 4 (Op code 2) |
| 13 | Op code of the third command |
| 14 | Parameter 1 (Op code 3) |
| etc. | |

The syntax of the implemented commands are described in the following:

**Transfer of Op code and parameters to the command DB**

| Instruction | Op code | Parameter | Parameter 2 | Parameter 3 | Parameter 4 |
|---|---|---|---|---|---|
| Set bit in DB | 10 | DB | DW | Bit | - |
| Reset bit in DB | 11 | DB | DW | Bit | - |
| Invert bit in DB | 12 | DB | DW | Bit | - |
| Set right byte in DB | 15 | DB | DW | Value | - |
| Set left byte in DB | 16 | DB | DW | Value | - |
| Write data word to DB | 17 | DB | DW | Value | - |
| Write double word to DB | 18 | DB | DW | Value | Value |
| Set bit in DX | 20 | DX | DW | Bit | - |
| Reset bit in DX | 21 | DX | DW | Bit | - |
| Invert bit in DX | 22 | DX | DW | Bit | - |
| Set right byte in DX | 25 | DX | DW | Value | - |
| Set left byte in DX | 26 | DX | DW | Value | - |
| Write data word to DX | 27 | DX | DW | Value | - |
| Write double word to DX | 28 | DX | DW | Value | Value |
| Set memory bit | 30 | MB | Bit | - | - |
| Reset memory bit | 31 | MB | Bit | - | - |
| Invert memory bit | 32 | MB | Bit | - | - |
| Write memory byte | 35 | MB | Value | - | - |
| Write memory word | 36 | MW | Value |  | - |
| Write I/O byte | 45 | PB | Value | - | - |
| Write I/O word | 46 | PW | Value | - | - |
| Write extended I/O byte | 55 | QB | Value | - | - |
| Write extended I/O word | 56 | QW | Value | - | - |
| Set initializing pulse | 60 | - | - | - | - |

### 5.2.9.1 Example configuration for the S5 command blocks

The S5 command blocks should be set up.
The initializing pulse is made available in memory word 56, bit 4. The command data block is to be DX 237. Ensure that in the PLC data block DX 237 is open from DWs 0 through 255.
In the WinCC Control Center, enter the required data block upon specifying the channel parameters - for example, SINEC H1.

### Dump from OB 1:

```
......
: SPA  FB 87
Name    EXECUTE
DBNR   : KF +237
DBDX   : KC DX
RIMP   : KY 56, 4
etc.
```

## 5.2.10 Purpose and function of S5-time synchronization

The present document describes the functions and properties of the SIMATIC S5 software:

### S5 time synchronization

The software is used to synchronize the SIMATIC S5 system clock. Further, it supplies a suitable date/time data format for creating the message blocks to ’chronological message acquisition’ of S5 alarm logging.

Function block **FB 86 : MELD:UHR** also makes available the current time of day in a format required by ’chronological message acquisition’. The data are made available in system data block 80 from DW 190 onwards.

If a change to a message signal condition occurs, the message is identified by means of its message number by function block **FB 80 : SYSTEMFB** and stamped with the current date and time from system data block 80.

The present manual describes the use and handling of S5 time synchronization in a SIMATIC S5 environment. The user is provided with an overview of the function and data blocks used by the software, and the storage space required. An example configuration is presented to help you.

### 5.2.10.1 List of software blocks

**The SIMATIC S5 software (S5 time synchronization) will be found on the WinCC CD in the file called WINCC1ST.S5D.**

The file contains the following function and data blocks:

| FB | Name | Size in Bytes | Function |
|---|---|---|---|
| FB 86 | MELD:UHR | 1135 | Synchronization of time |
| All | | 1135 | |

Table 17

| | | | |
|---|---|---|---|
| *Clock data area 115U:* | *27 DWs* | *=* | *54 bytes* |
| *Clock data area 135U/155U:* | *12 DWs* | *=* | *24 bytes* |
| *Data area for S5 alarm logging:* | *3 DWs* | *=* | *6 bytes* |

### 5.2.10.2 Hardware requirements

The function blocks specified for S5 alarm logging require the following hardware for them to be executed correctly:

| PLC | CPU |
|---|---|
| AG 115U | CPU 944 * , CPU 945 |
| AG 135U | CPU 928B |
| AG 155U | CPU 946/ 947, CPU 948 |

* Only the CPU 944 featuring two PU interfaces has a system clock

## 5.2.11 Call parameters of FB 86 : MELD:UHR

A description of the call parameters of function block **FB 86 : MELD:UHR** is presented in the following.

A description of the call parameters of function block **FB 87 : EXECUTE** is presented in the following.

| Name | MELD:UHR | Parameters |
|---|---|---|
| ID: | CPUT | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF |
| ID: | DCF7 | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF |
| ID: | QTYP | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF |
| ID: | QSYN | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY |
| ID: | UDAT | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY |
| ID: | ZINT | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF |
| ID: | ZUHR | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KY |
| ID: | ZSYN | E/A/D/B/T/Z: D KM/KH/KY/KC/KF/KT/KZ/KG: KF |

### CPUT:

| No. of CPU | Type |
|---|---|
| 1 | CPU 943 / CPU 944 |
| 2 | CPU 945 |
| 3 | CPU 928B |
| 4 | CPU 946 / 947 |
| 5 | CPU 948 |

### DCF7:

**Operating mode**
0 = operation with S5 system clock
1 = operation with DCF77 radio clock

### QTYP:

**Type of data source for the time synchronization message frame.**
0 = data source is a data block (DB)
1 = data source is an extended data block (DX)

## QSYN:

**Data source of time data**

| | | |
|---|---|---|
| DCF7 = 0: | QSYN = | DB number, DW number of the received time synchronization message frame |
| DCF7 = 1: | QSYN = | DB number, DW number of the DCF77 - Time |

## UDAT:

**Address of clock data area**
UDAT = DB number, DW number

## ZINT:

Time interval in minutes for sending the synchronization message frame (DCF7 = 1)

## ZUHR:

**Destination data area for time data in alarm logging format.**
ZUHR = DB number, DW number

## ZSYN:

Destination data area for time synchronization message frame (DCF7 = 1)

If chronological message acquisition functionality is to be used with S5 alarm logging, a special time data format will be expected in DB 80 from DW 190 onwards.
This time data format will be derived from the S5 system time and written to the corresponding data area ZUHR (DB 80, DWs 190 - 192).

**Relationship between 'chronological reporting' and FB 86 : MELD:UHR :**

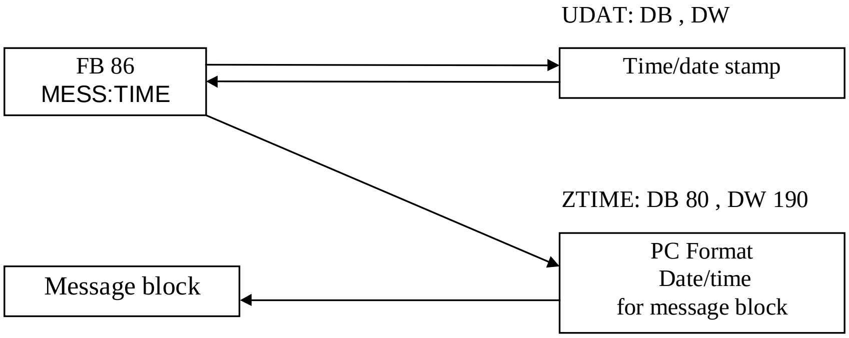


Figure 4

## 5.2.12 Data formats for date and time

**Time synchronization message frame from a system (WinCC does not currently support the time message frame)**

The first data word of the time synchronization message frame contains a source ID, which is sent by the system together with data and time data.
Function block **FB 86 : MELD:UHR** extracts the queued message frame only as soon as the source ID 'FFFF' is positioned here. Receipt of the message frame is acknowledged with a '0' in this data word. Not until a new message frame arrives (DW 1 = 'FFFF') is it read out again and evaluated.

| Meaning | Data word | Contents | Validity | Comment |
|---|---|---|---|---|
| Source ID/time message frame | 1 | FFFF | | |
| Message frame ID | 2 | FFFF | | Not used |
| Seconds | 3 | 00xx | xx: 0 to 59 | |
| Minutes | 4 | 00xx | xx: 0 to 59 | |
| Hours | 5 | 00xx | xx: 0 to 23 | |
| Day | 6 | 00xx | xx: 1 to 31 | |
| Month | 7 | 00xx | xx: 1 to 12 | |
| Year | 8 | 00xx | xx: 0 to 127 (1990-2117) | Year + 1990 |
| Day of week | 9 | 00xx | xx: 0 to 6 | Sunday = 0 |
| Day of year | 10 | 00xx | xx:1 to 365 | |
| Summer time, winter time<br>Leap year | 11 | yyxx | xx: winter time = 00<br>summer time = 01<br>yy: leap year<br>current year = 00<br>last year = 01<br>two year ago = 02<br>three years ago = 03 | |

Table 18

### 5.2.12.1 Time data area CPU 944, CPU 945

The data word numbers are relative details. The actual position of the area is determined by the call parameters: UDAT = DB No., DW No. of **FB 86 : MELD:UHR.**

| DW | Location |
|---|---|
| 0 | internal tags |
| 1 | |
| 2 | |
| 3 | |
| 4 | current time |
| 5 | |
| 6 | |
| 7 | |
| 8 | time operating range |
| 9 | |
| 10 | |
| 11 | |
| 12 | reserve - alarm time |
| 13 | |
| 14 | |
| 15 | |
| 16 | reserve - operating hours |
| 17 | |
| 18 | |
| 19 | |
| 20 | |
| 21 | |
| 22 | time / date<br>after RUN / STOP |
| 23 | |
| 24 | |
| 25 | |
| 26 | |

Table 19

**Current time in time data area:**

| DW | Word/Left | Word/Right |
|---|---|---|
| 4 | --- | Day of week |
| 6 | Day | Month |
| 7 | Year | AM/PM (bit, No. 7), hour |
| 8 | Minute | Second |

Figure 5

**Setting area in time data area:**

| DW | Word/Left | Word/Right |
|---|---|---|
| 9 | Leap year | Day of week |
| 10 | Day | Month |
| 11 | Year | AM/PM (bit, No. 7), hour |
| 12 | Minute | Second |

Figure 6

### 5.2.12.2 Time data area CPU 928B, CPU 948

The data word numbers are relative details. The actual position of the area is determined by the call parameters: UDAT = DB No., DW No. of **FB 86 : MELD:UHR**.

| DW | Location |
|---|---|
| 0 | internal tags |
| 1 | |
| 2 | |
| 3 | |
| 4 | current time |
| 5 | |
| 6 | |
| 7 | |
| 8 | time operating range |
| 9 | |
| 10 | |
| 11 | |

Figure 7

**Current time in time data area:**

| **DW** | **Word/Left** | | | | | | | | **Word/Right** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 4 | Seconds | | | | | | | | 0 | | | | | | | |
| 5 | Format | | Hours | | | | | | Minutes | | | | | | | |
| 6 | Day of month | | | | | | | | Day of week | | | | 0 | | | |
| 7 | Year | | | | | | | | Second | | | | | | | |

Figure 8

**Current time in setting area:**

| **DW** | **Word/Left** | | | | | | | | **Word/Right** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 8 | Seconds | | | | | | | | 0 | | | | | | | |
| 9 | Format | | Hours | | | | | | Minutes | | | | | | | |
| 10 | Day of month | | | | | | | | Day of week | | | | 0 | | | |
| 11 | Year | | | | | | | | Second | | | | | | | |

Figure 9

### 5.2.12.3 Time data area CPU 946, CPU 947

The data word numbers are relative details. The actual position of the area is determined by the call parameters: UDAT = DB No., DW No. of **FB 86 : MELD:UHR**.

| DW | Location |
|---|---|
| 0 | internal tags |
| 1 | |
| 2 | |
| 3 | |
| 4 | current time |
| 5 | |
| 6 | |
| 7 | |
| 8 | time operating range |
| 9 | |
| 10 | |
| 11 | |

Figure 10

Current time in time data area:

| DW | Word/Left | | Word/Right | |
|---|---|---|---|---|
| 4 | 10 sec. | 1 sec. | 1/10 sec. | 1/100 sec. |
| 6 | 10 hrs. | 1 hr. | 10 min. | 1 min. |
| 7 | 10 days | 1 day | Day of week | 0 |
| 8 | 10 years | 1 year | 10 months | 1 month |

Figure 11

Current time in setting area:

| DW | Word/Left | | Word/Right | |
|---|---|---|---|---|
| 9 | 10 sec. | 1 sec. | 1/10 sec. | 1/100 sec. |
| 10 | 10 hrs. | 1 hr. | 10 min. | 1 min. |
| 11 | 10 days | 1 day | Day of week | 0 |
| 12 | 10 years | 1 year | 10 months | 1 month |

Figure 12

### 5.2.12.4 Clock data format for message blocks

The data word numbers are relative details. The actual position of the area is determined by the call parameters: ZUHR = DB No., DW No. of **FB 86 : MELD:UHR**.
If chronological message acquisition functionality is to be used with S5 alarm logging, the data DB 80, DW 190 have to be entered in parameter ZUHR.
The date and time are made available by function block **FB 86 : MELD:UHR** for message processing in binary code:

**Current time in setting area:**

| Meaning | Data word | Bit | Validity | Comment |
|---|---|---|---|---|
| 1/100 second | 1 | 0 - 6 | 0 to 99 (0 - 990 msec) | In one 10 msec reference |
| Seconds | 1 | 7 - 12 | 0 to 59 | |
| Minutes | 0 | 0 - 5 | 0 to 59 | |
| Hours | 0 | 6 - 10 | 0 to 23 | |
| Day | 2 | 0 - 4 | 1 to 31 | |
| Month | 2 | 5 - 8 | 1 to 12 | |
| Year | 2 | 9 - 15 | 0 to 127 (1990-2117) | Year + 1990 |

Figure 13

**DW3: time**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | Hours | | | | | Minutes | | | | | |

**DW4: time**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | Seconds | | | | | | Milliseconds | | | | | | |

**DW4: date**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Year | | | | | | | Month | | | | Day | | | | |

Figure 14

## 5.2.13 Interface description

To use the S5 time synchronization software, the user has to:

- fill out the call parameters of **FB 86 : MELD:UHR** as described in section 5.2.11 Call parameters of FB 86 : MELD:UHR
- open the data areas in the PLC.

### Example configuration

Let us assume a CPU 944 with two PU interfaces. The S5 time synchronization is required to be set up for S5 alarm logging without the DCF77 clock on this CPU.

**Data areas:**

| | |
|---|---|
| Time synchronization message frame: | DB 100, DW 20 - DW 30 |
| Clock data area of the S5 system clock: | DB 100, DW 31 - DW 47 |
| Message block data*: | DB 80, DW 190 -DW 192 |

* Use of this data area is imperative when using SIMATIC S5 alarm logging.

On specifying the channel parameters - for example, SINEC H1, the required data block (DB 100, DW20 - DW 30) has to be entered in the system when specifying ‘time synchronization:’.

You have to make sure that DB 80 of DW 0 through DW 255 and DB 100 of DW 0 of DW 47 are opened.

### Dump from OB 1:

```
......
: SPA  FB 86
Name   : MELD: UHR
CPUT   : KF +1
DCF7   : KF +0
QTYP   : KF +0
QSYN   : KY 100, 20
UDAT   : KY 100, 31
ZINT   : KF +0
ZUHR   : KY 80, 190
ZSYN   : KF +0
etc.
```

## 5.2.14 Interaction with WinCC alarm logging

**The following must be taken into account with regard to the interaction of WinCC alarm logging with the S5 message block:**

256 has to be specified in the S5's send block as the number of data words to be transferred.

A new connection has to be set up in the Control Center in the S5Trsp channel. Specify the type Fetch-passive at the item Connection/Read function.

Two raw data tags have to be created for each PLC for data exchange with alarm logging.
The first is responsible for receiving messages.
Their addressing has to be set as follows: data area : DB, DB No. : xx, addressing word, DW : 0, raw data type : event
The second is required for sending acknowledgement information
Their addressing has to be set as follows: data area : DB, DB No. : 80, addressing word, DW : 90, raw data type : event

Connect the event tag in alarm logging to the receive raw data tag (the bit information is meaningless in this instance).
Connect the acknowledgement tag to the send raw data tag (the bit information is meaningless in this instance).
Enter the file S5STD.NLL as the Format DLL.
Tip: Using the interconnection wizard, you can connect all the messages concerned in one operation.

**Only positive fixed-point numbers are allowed for process values. Floating point values are not supported.**

| S5 | | WinCC |
|---|---|---|
| prozess value | ⇨ | prozess value 1 |
| job description | ⇨ | prozess value 2 |
| batch description | ⇨ | prozess value 3 |
| reserve | ⇨ | prozess value 4 |

## 5.3 Format DLL interface to AlarmLogging and TagLogging

### Objective

The AlarmLogging and TagLogging applications acquire the process data via the WinCC data manager. Depending on the communication type for the process,

- different channel DLLs are involved in data transfer,
- the process data are stored in message frames (raw data tags) of different structures.

AlarmLogging and TagLogging are required to process the process data, however, irrespective of the type of communication concerned, in exactly the same way. For this reason, a separate Format DLL is used for every type of communication which knows the precise structure of the message frames concerned and derives from it a generally valid "process data form" for AlarmLogging and TagLogging.

A Format DLL basically belongs to a channel DLL; like it, it should be possible to insert and remove it simply into and from the overall system. Still, it does not have a direct interface to the associated channel DLL.

This document describes the integration and interface of every Format DLL to the WinCC applications AlarmLogging and TagLogging. It originated during drafting of the S7PMC Format DLL, and for this reason the expression "S7PMC Format DLL" is for the most part synonymous with the expression "Format DLL".

### Basic process

The S7PMC Format DLL is a passive group of programs which has interfaces only to the applications AlarmLogging and TagLogging. The S7PMC Format DLL processes S7PMC-specific functions for AlarmLogging and TagLogging.

AlarmLogging and TagLogging log in with a start call at the Format DLL. On doing so, certain parameters are transferred in a start structure to the Format DLL and their characteristics are received by means of IDs.

**Two data transfer directions are required to process the S7PMC functions RUNTIME mode:**

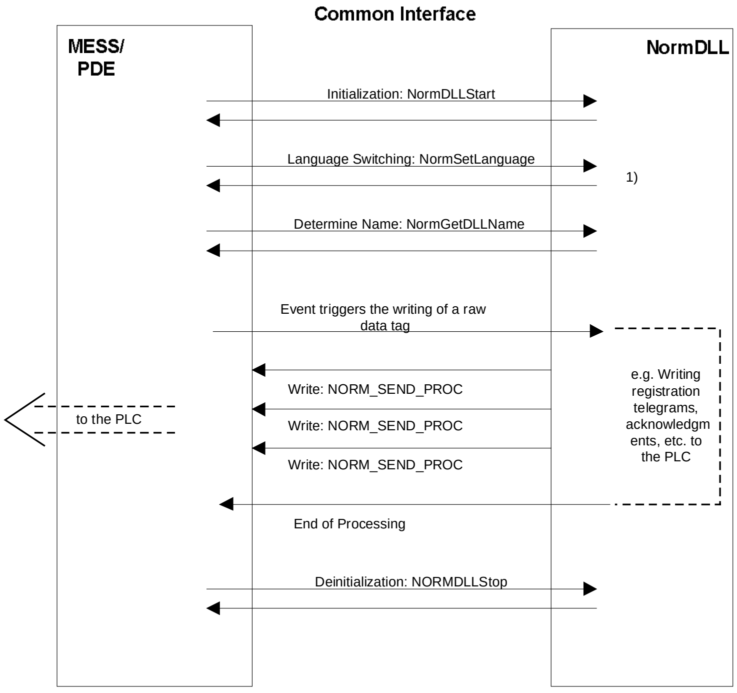
- OS to PLC: (sending of logon/logoff requests, acknowledgements)
- PLC to OS: (reception of messages and archive data)

With an initialization call, TagLogging/AlarmLogging announces the configured archive variable names and message numbers to S7PMC DLL. The Format DLL (WinCC) has to log on at the PLC for these objects. The initialization call can be processed at any point in time,

The Format DLL is called by AlarmLogging/TagLogging to uninitialize in order to return resources etc.

## 5.3.1 Shared interface to AlarmLogging and TagLogging

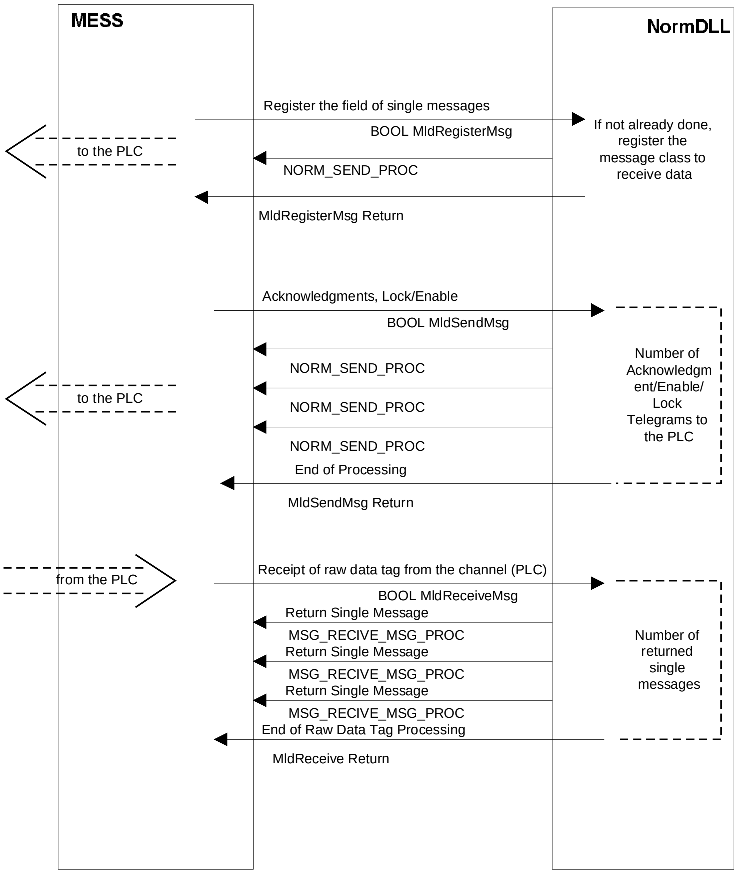
The general functions of the Format DLL, which are identical for AlarmLogging and TagLogging, are grouped in a shared interface. The function names all begin with ‘NORM...’.
(prefix for AlarmLogging-specific functions: ‘Mld...’, prefix for TagLogging-specific functions: ‘Pde...’.)


Common Interface
MESS/ PDE
NormDLL
Initialization: NormDLLStart
Language Switching: NormSetLanguage
1)
Determine Name: NormGetDLLName
Event triggers the writing of a raw data tag
Write: NORM_SEND_PROC
Write: NORM_SEND_PROC
Write: NORM_SEND_PROC
to the PLC
e.g. Writing registration telegrams, acknowledgm ents, etc. to the PLC
End of Processing
Deinitialization: NORMDLLStop

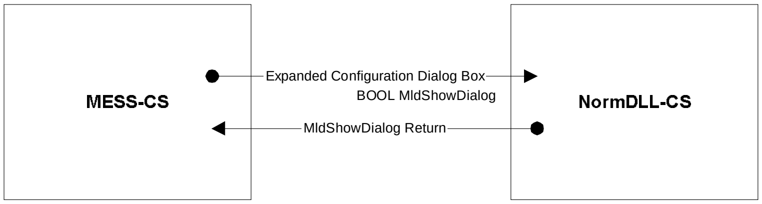
1) The language switch is only required for format DLLs that contain a dialog box

MELD = AlarmLogging
PDE = TagLogging

AlarmLogging-specific extras
**Runtime**

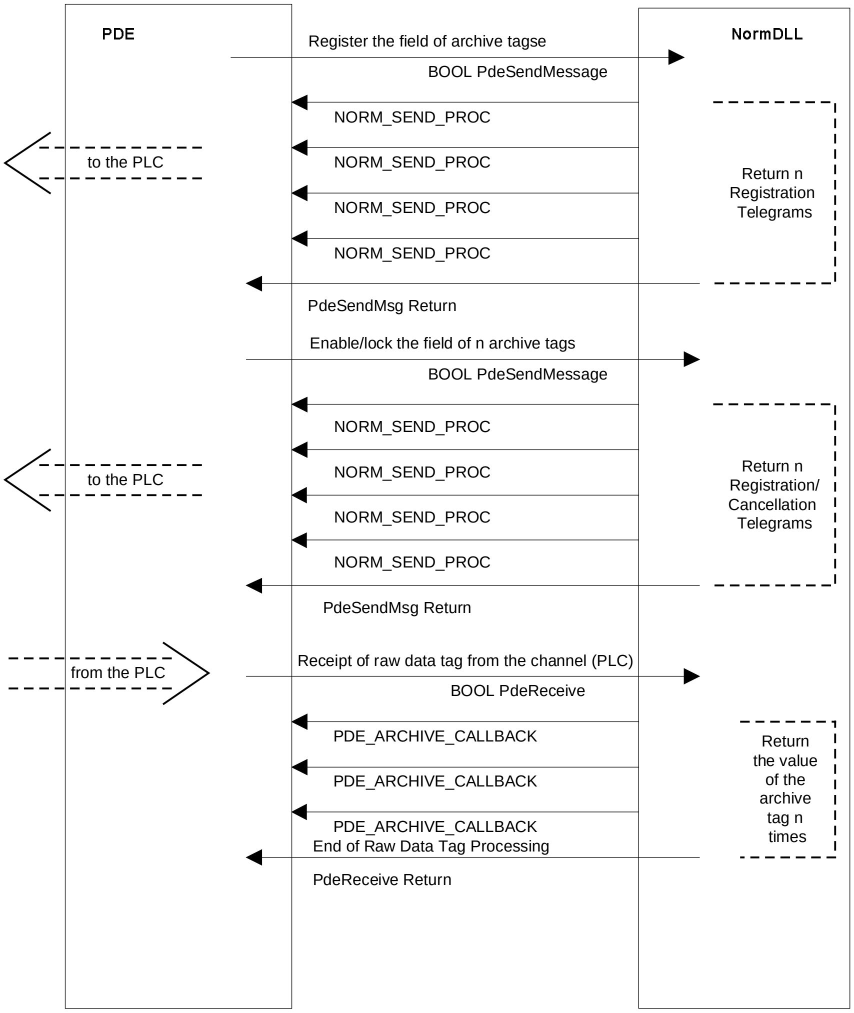


Extended configuration dialog

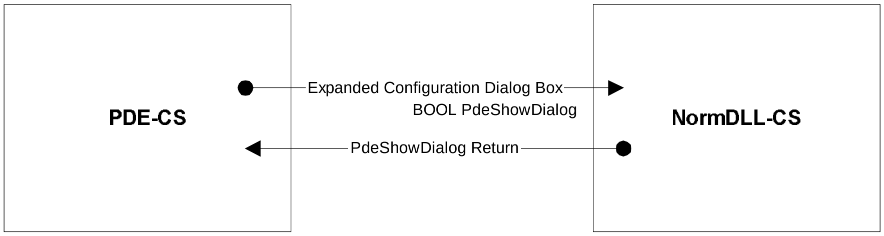


## 5.3.2 TagLogging-specific extras

### Runtime



### Extended configuration dialog



## 5.3.3 API functions of a WinCC Format DLL

**The Format DLL is divided into the following sub-areas:**

- Initialization of the Format DLL
    - Initialization by the operating system while the Format DLL is being loaded (LibMain)
    - Polling the properties of a Format DLL
    - Polling the name of the Format DLL
- Shut down of the Format DLL
    - Shut down by TagLogging and AlarmLogging
    - Unloading by the operating system
- Add ins to the configuration
    - Dialog extension during message configuration
    - Dialog extension during archive tag configuration
- Online services
    - Registration of all Format DLL-specific objects (messages, archive tags)
    - Language switch
- Standardization
    - Standardization of messages
    - Standardization of archive tags

### 5.3.3.1 Initialization of the Format DLL

#### Initialization during the load operation

The applications AlarmLogging and/or TagLogging load a WinCC Format DLL with the help of the *LoadLibrary* system call. Following that, the Format DLL is loaded by the operating system and initialized by its standard mechanisms. All the entry addresses of the Format DLL are defined.

### 5.3.3.2 Polling the properties of a Format DLL

AlarmLogging and TagLogging log in at the relevant Format DLL by means of the NormDLLStart call. It is intended for the exchange of information between the Format DLL and the application.

**NormDLLStart**

```
include <winccnrm.h>

BOOL NormDLLStart(
        LPVOID lpUser,
        BOOL bModeRuntime,
        PNORM_STARTSTRUCT pcis,
        PCMN_ERROR lpError);
```

| Parameter | Description |
|---|---|
| lpUser | Pointer to application data, forward unchanged to Callback |
| bModeRuntime | TRUE when the Format DLL is started in RUNTIME mode, FALSE in Configuration Mode; is not currently evaluated by the Format DLL |
| pcis | Pointer to start structure. |
| lpError | Pointer to default WinCC error structure |

| Return | Description |
|---|---|
| TRUE | No error |
| FALSE | Error in API function, description of error cause via the pointer lpError |

NORM_STARTSTRUCT

| Component | Description | I/O |
|---|---|---|
| dwSize | Size of structure in bytes | A |
| lpstrProjectPath | path of the currently selected project | I |
| NORM_SEND_PROC<br>pfnWriteRwData | Ptr to Callback function of the application through which the Format DLL sends a raw data tag via the DM to the PLC. | I |
| dwAppID | Application ID:<br>1 = AlarmLogging<br>2 = TagLogging<br>3 = USER (reserved for additional appls. , not currently used) | I |
| dwLocaleID | Current language setting at time of call | I |
| dwNormCap | Properties of the Format DLL in accordance with the table below | A |

The Callback function for sending raw data tags to the WinCC Data Manager is supplied as follows:

```
typedef BOOL(*NORM_SEND_PROC)(
        LPDM_VAR_UPDATE_STRUCT    lpDmVarUpdate,
        DWORD                     dwWait,
        LPVOID                    LpUser,
        LPCMN_ERROR               lpError );
```

| Parameter | Description |
|---|---|
| lpDmVarUpdate | Pointer to raw data tag |
| dwWait | Identification of whether the application should wait until completion of the write call or not:<br>WAIT_ID_NO with SET_VALUE<br>WAIT_ID_YES with SET_VALUE_WAIT |
| lpUser | Pointer to application data, noted upon call NormDLLStart |
| *lpError* | Pointer to default WinCC error structure |

| Return | Description |
|---|---|
| TRUE | No error |
| FALSE | Error in API function, description of error cause via the pointer lpError |

**A bit is assigned to every property in accordance with the table below.**

| DEFINE | Bit Mask | | Meaning |
|---|---|---|---|
| NORMCAP_DIALOG | 0x00000001 | Set | Format DLL features special dialog box |
| | | Deleted | Format DLL does not feature a dialog box |
| NORMCAP_REENTRANT | 0x00000002 | Set | Format DLL is re-enterable |
| | | Deleted | Format DLL is not re-enterable |
| NORMCAP_MSG_FREE_LOCK | 0x00000004 | Set | Login/logoff is possible for messages. |
| | | Deleted | Logon/logoff is not possible for messages. |
| NORMCAP_ARC_FREE_LOCK | 0x00000008 | Set | Logon/logoff is possible for archive tags. |
| | | Deleted | Logon/logoff is not possible for archive tags. |
| NORMCAP_MSG_GENERIC | 0x00000010 | Set | Messages can be generated generically. |
| | | Deleted | Messages cannot be generated generically. |
| NORMCAP_ARC_GENERIC | 0x00000020 | Set | Archive tags can be created generically. |
| | | Deleted | Archive tags cannot be created generically. |

### 5.3.3.3 Polling the name of the Format DLL

**NormGetDLLName**

```
include <winccnrm.h>

LPTSTR NormGetDLLName( void );
```

| Return | Description |
|---|---|
| LPTSTR | Pointer to a string which contains the name of the Format DLL in plain language; the name depends on the current language setting. |

## 5.3.4 Shutting down the Format DLL

### Shut down by TagLogging and AlarmLogging

TagLogging and AlarmLogging inform the Format DLL when the applications are being closed. The resources are then returned properly in the Format DLL.

**NormDLLStop**

```
include <winccnrm.h>

BOOL NormDLLStop (void);
```

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function |

### Unloading by the operating system

No special precautions are necessary.

### 5.3.4.1 Configuration add-ins

Specific details are necessary for S7PMC objects. These details are first requested in a dialog box using standard means (without MFC) and enter directly into either the WinCC message number or the name of the archive tag. This means that the Format DLL does not have to store and manage these details itself. To guarantee the uniqueness of a message number throughout a project, an assignment between the message number or archive tag and the associated raw data tag is necessary. This assignment information is an integral part of the message number or archive tag name.

### 5.3.4.2 Dialog box extension when configuring S7PMC messages

The Format DLL has an API function for defining the S7PMC-specific message number. This function is called by CS alarm logging upon assigning parameters to single messages belonging to a S7PMC Format DLL. The message number assigned by the S7PMC Format DLL is the number which consists of two parts:

### Part 1:

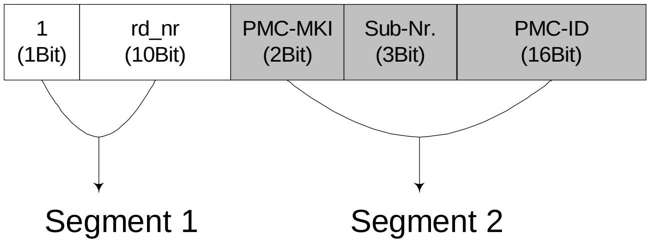
The number which uniquely identifies a PLC CPU throughout a project (raw data tag number)

### Part 2:

The number belonging to the message from a PLC point of view and uniquely identifying it within a PLC CPU (Format DLL-specific)

The following selection has to be made in the configuration dialog box to compile the message number:

**Structure of an S7PMC message number (32 bit)**



## Regarding Part 1

Every message belongs to a raw data tag which identifies a PLC CPU. In order to be able to perform the assignment raw data tag – message number, the following definition was established.
The name of the raw data tag for S7PMC - and all connection types with Format DLL - has the following permanent structure:

**@rd_alarm#rd_nr**

@rd_alarm# Permanent integral part of the name of a raw data tag for Format DLLs
rd_nr Decimal number between 0 and 1023 for identification of a raw data tag (without leading zeros)

The most significant bit of the message number is set in the case of message numbers which are assigned by Format DLLs (externally). These messages may be processes only by the associated Format DLLs, meaning that the message number cannot be modified via the configuration dialog box of AlarmLogging.

## Regarding Part 2

This part of the message number can be assigned by the Format DLL concerned. For the S7PMC, it has the following meaning:

MKl Message class; one of the classes has to be selected:
SCAN (1)
ALARM/NOTIFY (2)
ALARM_8P/ALARM_8 (2)
LTM (3)
Sub-Nr Submessage number applicable only to ALARM_8 and ALARM_8P:
1 to 8
PMC-ID PMC message number (block input parameter EV-ID):
1 to 16386
for message classes SCAN and ALARM/NOTIFY and
ALARM_8P/ALARM_8
1 to 7
for message class LTM

### MldShowDialog

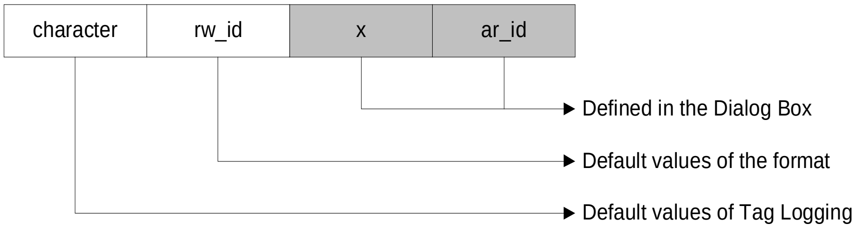
```
include <winccnrm.h>

BOOL WINAPI MldShowDialog(
                HWND                  hwnd,
                LPMSG_CSDATA_GENERIC  lpmCS,
                LPDM_PROJECT_INFO     lpDMProjectInfo,
                LPCMN_ERROR           lpError );
```

| Parameter | Description |
|---|---|
| hwnd | Window handle |
| lpmCS | Pointer to single message data |
| lpDMProjectInfo | Pointer to project information structure |
| lpError | Pointer to default WinCC error structure |

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

## 5.3.4.3 Dialog box extension when configuring archive tags

The Format DLL has an API function for defining the S7PMC-specific archive tag name. This function is called by CS TagLogging CS upon assigning parameters to archive tags belonging to an S7PMC connection. The archive tag name assigned by the AS7PMC Format DLL is a name consisting of several components and contains, among other things, the number belonging to the archive on the PLC. With this algorithm, the S7PMC tag numbers are uniquely contained in the WinCC archive tag description, resulting in Runtime mode in the quickest possible assignment.
TagLogging guaranties that the archive tag names are all unique.
**Structure of an S7PMC archive tag name** (not loner than 18 bytes)


character
rw_id
x
ar_id
Defined in the Dialog Box
Default values of the format
Default values of Tag Logging

| Identifier | Length in Bytes | Assigned By | Meaning |
|---|---|---|---|
| character | 9 | TagLogging | Fixed character string, which is assigned by TagLogging, consists of the name of the Format DLL and # as the data delimiter - for example, for S7PMC: NRMS7PMC, does not appear on the user interface |
| rw_id | 8 | TagLogging/ | Raw data ID in hexadecimal characters (including leading zeros), resulting in unique |

| Identifier | Length in Bytes | Assigned By | Meaning |
|---|---|---|---|
| | | Format DLL | assignment to the raw data tag (connection) to which the archive number belongs. The named portion is constructed by the Format DLL using the TagLogging input parameters. |
| x | 1 | Format DLL<br>CS portion | S7PMC-specific ID for differentiating between BSEND and AR_SEND:<br>‘A’ = AR_SEND<br>‘B’ = BSEND |
| ar_id | 4 | Format DLL<br>CS portion | ID as hex characters (including leading zeros)<br>Depending on ID x:<br>S7PMC-specific archive number AR_ID or<br>S7-specific R_ID upon BSEND |

Example of an archive tag name for an S7PMC: #00000001#A#0014

**PdeShowDialog**

```
include <winccnrm.h>

BOOL WINAPI PdeShowDialog(
              LPVOID        hwnd,
              LPTSTR        lpszArcVarName,
              DWORD         dwArcVarNameLength,
              LPDM_VARKEY   lpVarKey,
              LPCMN_ERROR   lpError
              );
```

| Parameter | Description |
|---|---|
| hwnd | Window handle |
| lpszArcVarName | Pointer to string field foe storing Format DLL-specific archive tag name portion |
| dwArcVarNameLength | Maximum length of Format DLL-specific name portion |
| lpVarKey | Pointer to Varkey of raw data tags |
| lpError | Pointer to WinCC error structure |

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

## 5.3.4.4 Online services

### Register all messages

This function is necessary because the Format DLL does not have configuration information about the relevant messages. But messages are not sent by the PLC until the application (WinCC) has logged on for receiving messages. AlarmLogging currently calls the function MldRegisterMsg for every relevant message and thus transfers configuration information for the single message to the Format DLL. Apart from a message description, the Format DLL receives a pointer to the raw data tag (connection) that is assigned to this message. This means that the Format DLL can create a table in main memory at runtime, with which the S7PMC-specific logon message frames can be structured.

**MldRegisterMsg**

```
include <winccnrm.h>

BOOL WINAPI MldRegisterMsg(
              LPDM_VARKEY   lpDMVarKey,
              LPDWORD       lpMsgNumber,
              DWORD         DwNumMsgNumber,
              LPCMN_ERROR   lpError );
```

| **Parameter** | **Description** |
|---|---|
| lpDMVarKey | Pointer to Varkey of raw data tags |
| lpMsgNumber | Pointer to field with single message numbers |
| dwNumMsgNumber | Number of single message numbers |
| lpError | Pointer to WinCC error structure |

| **Return** | **Description** |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

### 5.3.4.5 Register all archive tags

This function is necessary because the Format DLL does not have configuration information about the relevant archive tags. The function PdeSendMsg is therefore called for a certain number of relevant archive tags and thus the configuration information and TagLogging additional information made known for the archive tags.
More than archive tag of a connection can be registered with a single call.
TagLogging transfers one double word per archive tag as additional information to the Format DLL, which is retained in Format DLL memory. This additional information is required by TagLogging as soon as archive tags have to be processed (in the Callback function TagLogging_ARCHIVE_CALLBACK).
This means that the Format DLL can create a table in main memory at runtime, with which the S7PMC-specific logon message frames can be structured for the archives concerned. The logon message frames are required to announce to the PLC readiness to receive for the archive number concerned. Not until after logon has been successful does the PLC send archive data to the application (WinCC).

**PdeSendMsg**

```
include <winccnrm.h>

BOOL WINAPI PdeSendMsg(
              NORM_SEND_PROC      lpfnCallBack,
              DWORD               dwFunctionId,
              LPSZ_ARC_VAR_NAME   lpszArcVarName,
              LPDWORD             lpdwData
              DWORD               dwNumArchVarName,
              LPDM_VARKEY         lpVarKey,
              LPVOID              lpUser,
              LPCMN_ERROR         lpError
              );
```

| Parameter | Description |
|---|---|
| lpfnCallBack | Pointer to Callback routine with which the raw data tag structured by the Format DLL has to be transferred to the DM.<br>If zero, the Callback routine has to be called from the Ini structure.<br>The function address from the Ini structure is not identical with this parameter. |
| dwFunctionId | Function ID FUNC_ID_REGISTER (refer to table below), the same function applies to all tags listed |
| lpszArcVarName | Pointer to a pointer field whose elements refer to the names of the tag variables |
| lpdwData | Pointer to a field whose elements contain additional data for the archive tags, can also be zero.<br>The additional value belonging to an archive tag is applied without being modified to internal lists of the Format DLL with the function FUNC_ID_REGISTER (logon archive tag) and forwarded when necessary to TagLogging_ARCHIVE_CALLBACK.<br>Without meaning with the other function IDs. |
| dwNumArchVarName | Number of archive tag names that have to be processed |
| lpVarKey | Pointer to Varkey of raw data tags |

| Parameter | Description |
|---|---|
| lpUser | Pointer to user data, transferred unmodified to Callback |
| lpError | Pointer to WinCC error structure |

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

Possible functions of the PdeSendMsg procedure (values from dwFunctionId):

| DEFINE | Bit Mask | Meaning |
|---|---|---|
| FUNC_ID_ LOCK | 0x00000001 | Disable archive tag |
| FUNC_ID_ FREE | 0x00000002 | Enable archive tag |
| FUNC_ID_ REGISTER | 0x00000004 | Log on archive tag |
| FUNC_ID_UN REGISTER | 0x00000008 | Log off archive tag (currently not required) |

### 5.3.4.6 Language change

The configuration dialog must be language-dependent; this means that Format DLL must know the currently set language. The language setting is also specified in the start structure during start-up. Dynamic language switch has to be forwarded to the Format DLL as well by AlarmLogging. The call for this is

**NormSetLanguage**

```
include <winccnrm.h>

BOOL NormSetLanguage(
                DWORD              dwLocaleID,
                LPCMN_ERROR        lpError
                );
```

| Parameter | Description |
|---|---|
| dwLocaleID | Current language setting at time of call |
| lpError | Pointer to default WinCC error structure |

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

## 5.3.5 Formatting

If an application has logged on at the PLC to receive messages or archive data, it receives these data by means of the raw data tag concerned. The Format DLL logs on during registration. Data message frames can be sent by the PLC from that time on. Data message frames are packed in raw data tags and are forwarded to the Format DLL via the channel DLL, the data manager and the application concerned (in this instance, TagLogging or AlarmLogging); the Format DLL is responsible for the type of raw data tag. The Format DLL interprets the incoming data and constructs messages and/or archive data from them.

### 5.3.5.1 Derivation of single messages

The contents of a raw data tag (of a message frame) may store *n* single messages. The Format DLL has to interpret this S7PMC-specific message frame and forward the resultant single messages to AlarmLogging.

The message number (EV_ID) of S7PMC is part of the WinCC message number.
Up to ten process values can be delivered by S7PMC in a single message. The "string" type is allowed as a process value. This process value type is not supported by AlarmLogging; supplementary values of that kind have to be rejected by the Format DLL.

The MldReceiveMsg function is invoked by AlarmLogging every time the status of the raw data tag has changed; this means when the Faulty status following OK or the other way round is determined by DM. The change of status of the raw data tag (corresponding to a connection) is of importance only for the S7PMC Format DLL. Further details are provided in the section entitled "Processing upon change of status".

**MldReceiveMsg**

```
include <winccnrm.h>

BOOL WINAPI MldReceiveMsg(
              MSG_RECEIVE_MSG_PROC    lpfnMsgReceive,
              LPDM_VAR_UPDATE_STRUCT  lpDMVar,
              LPVOID                  lpUser,
              LPCMN_ERROR             lpError );
```

| **Parameter** | **Description** |
|---|---|
| lpfnMsgReceive | Pointer to Callback routine with which the single message structured by the Format DLL has to be transferred to AlarmLogging. |
| lpDMVar | Pointer to raw data tag |
| lpUser | Pointer to user data, transferred unmodified to Callback |
| lpError | Pointer to WinCC error structure |

| **Return** | **Description** |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

The Callback function for sending single messages to AlarmLogging is supplied as follows:

```
typedef BOOL(*MSG_RECEIVE_MSG_PROC)(
              LPMSG_RTCREATE_STRUCT   lpMsgCreate,
              DWORD                   dwNumMsg,
              LPVOID                  lpUser,
              LPCMN_ERROR             lpError );
```

| Parameter | Description |
|---|---|
| lpMsgCreate | Pointer to a WinCC message |
| dwNumMsg | Number of single messages |
| lpUser | Pointer to application data |
| lpError | Pointer to default WinCC error structure |

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

### 5.3.5.2 Acknowledging, disabling/to enabling messages

The message and alarm concept of WinCC AlarmLogging and S7PMC makes provision for messages being acknowledged irrespective of their configuration. The acknowledgement information is known to AlarmLogging but also has to be managed in the message acknowledgement memory of the PLC. To achieve this, AlarmLogging sends acknowledgement message frames to the PLC via the Format DLL corresponding to the connection.

On the basis of these input data, the S7PMC Format DLL constructs the corresponding S7PMC message frames, which are forwarded by means of the AlarmLogging Callback function NORM_SEND_PROC to the DM.

The same procedure applies when a single message is disabled/enabled again by AlarmLogging; this means that its generation at the source on the PLC has to be inhibited/released again.

**MldSendMsg**

```
include <winccnrm.h>

BOOL WINAPI MldSendMsg(
              NORM_SEND_PROC          lpfnMsgSend,
              LPMSG_SEND_DATA_STRUCT  lpSendData,
              DWORD                   dwNumData,
              LPVOID                  lpUser,
              LPCMN_ERROR             lpError );
```

| Parameter | Description |
|---|---|
| lpfnMsgSend | Pointer to AlarmLogging Callback routine with which the raw data tag constructed by the Format DLL has to be transferred for writing to the PLC. The parameters are described in section entitled "Polling the properties of a Format DLL". |
| lpSendData | Pointer to send data; its structure is described further below |

| Parameter | Description |
|---|---|
| dwNumData | Number of single requests for processing |
| lpUser | Pointer to user data, transferred unmodified to Callback |
| lpError | Pointer to WinCC error structure |

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

**Structure of AlarmLogging send data (single request)**

| Tag | Meaning |
|---|---|
| DWORD dwVarID | Raw data tag ID of DM |
| DWORD dwNotify | Notify possible values<br>MSG_STATE_QUIT Acknowledge message<br>MSG_STATE_LOCK Disable message<br>MSG_STATE_UNLOCK Enable message<br>MSG_STATE_QUIT_EMERGENCY Acknowledge all messages |
| DWORD dwData | With QUIT, LOCK, UNLOCK --> message number<br>With EMERGENCY-ACK --> Unused |

### 5.3.5.3 Processing upon change of status

The change of status of a connection (raw data tags) has to be notified to the Format DLL. This is performed by the **MldReceiveMsg** function.

| Status Change From - To | Processing in the S7PMC Format DLL |
|---|---|
| Faulty - OK | Pass to the PLC logon message frames for all S7PMC messages categories for which at least one message has been configured. Logon is performed for specific S7PMC message categories.<br>The Format DLL already knows all the configured messages on account of their registration. |
| OK - Faulty | The Format DLL has to reject active jobs which have already been sent to the PLC but could not be fully processed any more owing to a change of status (acknowledgements missing). |

### 5.3.5.4 Message update of the S7PMC Format DLL

In the case of message update, the S7PMC Format DLL reads the message status of all the messages notified to it by means of registration and sends it as a single message to AlarmLogging. This means that it is possible to build on a consistent message picture upon system start-up.
A message update is necessary when

- a change of status from Faulty to OK has been detected (that is the implicit case upon system start-up as well)
- the PLC sends a "message update message frame" to the PLC. This message frame is sent to every logged on node when, for example, message overflow is determined when messages are acknowledged or released by other nodes.

During a message update, the PLC sends the message acknowledgement states and the disable IDs. The supplementary values and the time are not sent. The Format DLL supplies the time of the single message in this case with the current system time or writes the ID MSG_STATE_UPDATE to the message status.

### 5.3.5.5 Formatting archive tags

The Format DLL makes two functions available for TagLogging:

- derivation of individual archive tag values from the contents of a raw data tag
- disabling/to enabling archive tags

### 5.3.5.6 Derivation of individual archive tag values

The contents of a raw data tag (of a message frame) may store archive tag values. The Format DLL has to interpret this S7PMC-specific message frame and forward the resultant archive tag values to TagLogging.

Process value converters can also be sent for an archive tag. The S7PMC Format DLL then performs the requisite conversion from process value to archive tag value. This process involves scaling functions which already exist in WinCC. The precise procedure has still to be determined.

The PdeReceive function is invoked by TagLogging every time the status of the raw data tag has changed; this means when the Faulty status following OK is or the other way round is determined by DM. The change of status of the raw data tag (corresponding to a connection) is of importance only for the S7PMC Format DLL. Further details are provided in the section entitled "Processing upon change of status".

**PdeReceive**

```
include <winccnrm.h>

BOOL PdeReceive (
              LPDM_VAR_UPDATE_STRUCT        lpDmVarUpdate,
              TagLogging_ARCHIVE_CALLBACK   lpfnCallBack,
              LPVOID                        lpUser,
              LPCMN_ERROR                   lpError
              );
```

| Parameter | Description |
|---|---|
| lpDmVarUpdate | Pointer to raw data tag |
| lpfnCallBack | Pointer to Callback routine with which the Format DLL transfers the individual archive tag values to TagLogging. |
| lpUser | Pointer to user data, transferred unmodified to Callback |
| lpError | Pointer to WinCC error structure |

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

The Callback function for sending individual archive tag values to TagLogging is supplied as follows:

```
BOOL (*PDE_ARCHIVE_CALLBACK) (
              LPTSTR         lpszArcVarName,
              double         doValue,
              SYSTEMTIME*    lpstTime
              DWORD          dwFlags
              DWORD          dwData,
              LPVOID         lpUser,
              LPCMN_ERROR    lpError
              );
```

| Parameter | Description |
|---|---|
| lpszArcVarName | Archive tag value as from raw data ID |
| doValue | Archive tag value |
| lpstTime | Pointer to time stamp derived from the useful data of the raw data tag |
| dwFlags | IDs whose precise meaning still has to be defined. |
| dwData | Additional date that was also provided upon registration, transferred without modification |
| lpUser | Pointer to user data, applied without modification by function call |
| lpError | Pointer to WinCC error structure |

| Return | Description |
|---|---|
| TRUE | Function successful |
| FALSE | Error in API function, description of error cause via the pointer `lpError` |

### 5.3.5.7 Disabling/enabling archive tags

With this function, TagLogging avails itself of the possibility in S7PMC of controlling the reception of archive tag values. The S7PMC Format DLL then creates a call to log off the archive concerned or to log it on again and forwards this call via NORM_SEND_PROC to the DM.

From the point of view of the S7PMC Format DLL, disabling/enabling of archive tags is virtually identical to the functions which are required for registering an archive tag. For both functions, the same function is called in the Format DLL (PdeSendMsg).

A distinction is made between registration and disabling/enabling by means of the function ID dwFunctionId: with disabling/enabling, the supplementary data are meaningless for each archive tag lpdwData. Refer to the section entitled "Register all archive tags".

### 5.3.5.8 Processing upon change of status

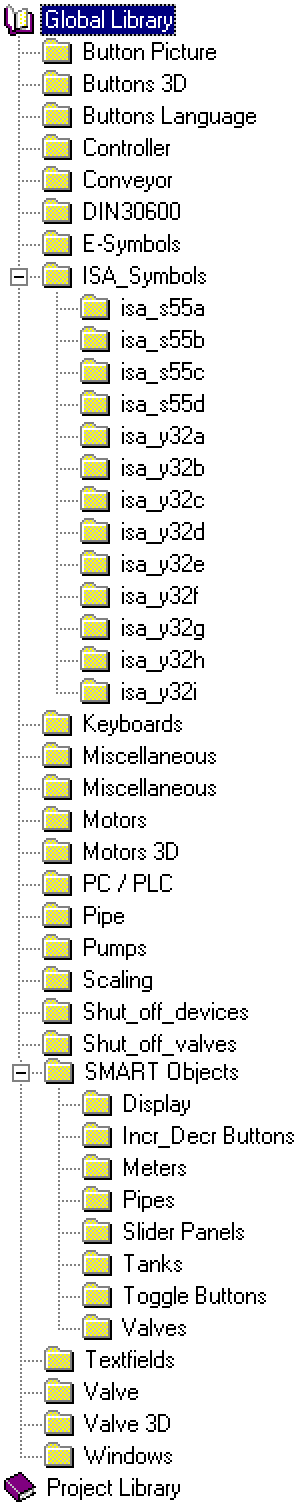
The change of status of a connection (raw data tag) must be notified to the Format DLL; this is done by using the function **PdeReceive**.

| Status Change<br>From - To | Processing in the S7PMC Format DLL |
|---|---|
| Faulty - OK | Logon message frames for all archive tags of all connections<br>The Format DLL already knows all the configured archive tags on account of their registration. |
| OK - Faulty | The Format DLL has to reject active jobs which have already been sent to the PLC but could not be fully processed any more owing to a change of status (acknowledgements missing). |

## 5.4 Creating picture modules for WinCC

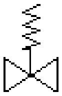
This chapter does not exist any more. It is replaced by the documentation for the WinCC Option: *IndustrialX.*

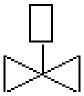
## 5.5 Global Library

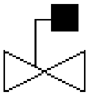


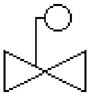
- Global Library
  - Button Picture
  - Buttons 3D
  - Buttons Language
  - Controller
  - Conveyor
  - DIN30600
  - E-Symbols
  - ISA_Symbols
    - isa_s55a
    - isa_s55b
    - isa_s55c
    - isa_s55d
    - isa_y32a
    - isa_y32b
    - isa_y32c
    - isa_y32d
    - isa_y32e
    - isa_y32f
    - isa_y32g
    - isa_y32h
    - isa_y32i
  - Keyboards
  - Miscellaneous
  - Miscellaneous
  - Motors
  - Motors 3D
  - PC / PLC
  - Pipe
  - Pumps
  - Scaling
  - Shut_off_devices
  - Shut_off_valves
  - SMART Objects
    - Display
    - Incr_Decr Buttons
    - Meters
    - Pipes
    - Slider Panels
    - Tanks
    - Toggle Buttons
    - Valves
  - Textfields
  - Valve
  - Valve 3D
  - Windows
- Project Library

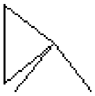
## 5.5.1 Shut-off devices

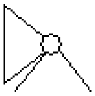

01

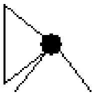

02

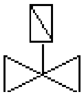

03

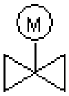

04


05

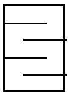

06

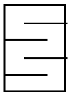

07

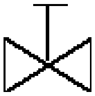

08

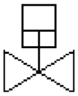


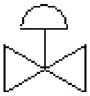
09

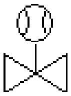

10


11

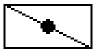

12

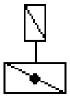

13

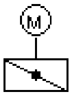

14

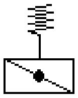

15

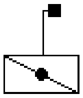
### 5.5.2 Shut-off valves

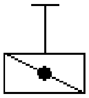

01

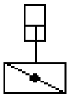

02

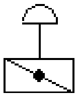

03

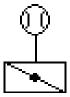

04

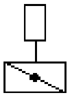

05

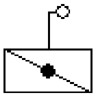

06

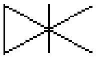

07

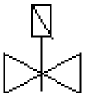

08

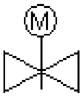

09

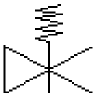

10

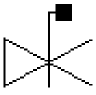

11

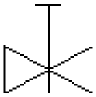

12

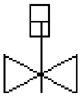

13

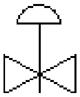

14

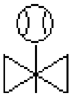

15

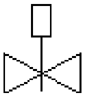

16

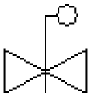

17


18


19


20


21

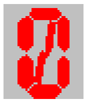

22

### 5.5.3 Smart objects

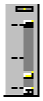
#### 5.5.3.1 Displays

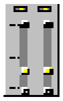
8-Bit Display

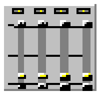
8-Bit Display + I/O Field


Digital output

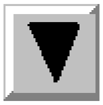
#### 5.5.3.2 Slider panels

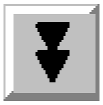

1_Slider

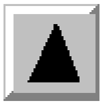

2_Slider

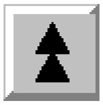

4_Slider

#### 5.5.3.3 Incr_Decr buttons


Decrement_-1


Decrement_-Step

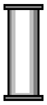

Increment_+1


Increment_+Step

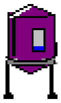
#### 5.5.3.4 Pipes

3D Pipe angle

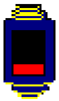
3D Pipe horizontal


3D Pipe vertical

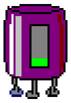
### 5.5.3.5 Tanks



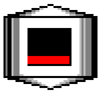
Tank1



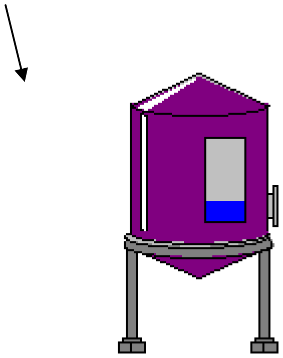
Tank2



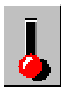
Tank3

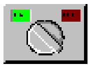


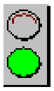
Tank4

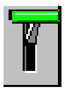


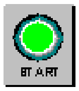
### 5.5.3.6 Toggle buttons

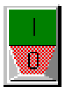

On_Off_1


On_Off_2


On_Off_3


On_Off_4

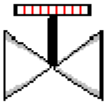

On_Off_5

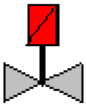

On_Off_6

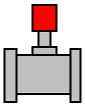
On_Off_7

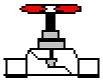
On_Off_8

### 5.5.3.7 Valves

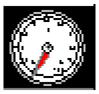

Valve1

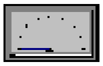

Valve2

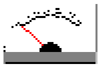

Valve3

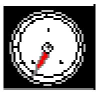

Valve4

### 5.5.3.8 Meters

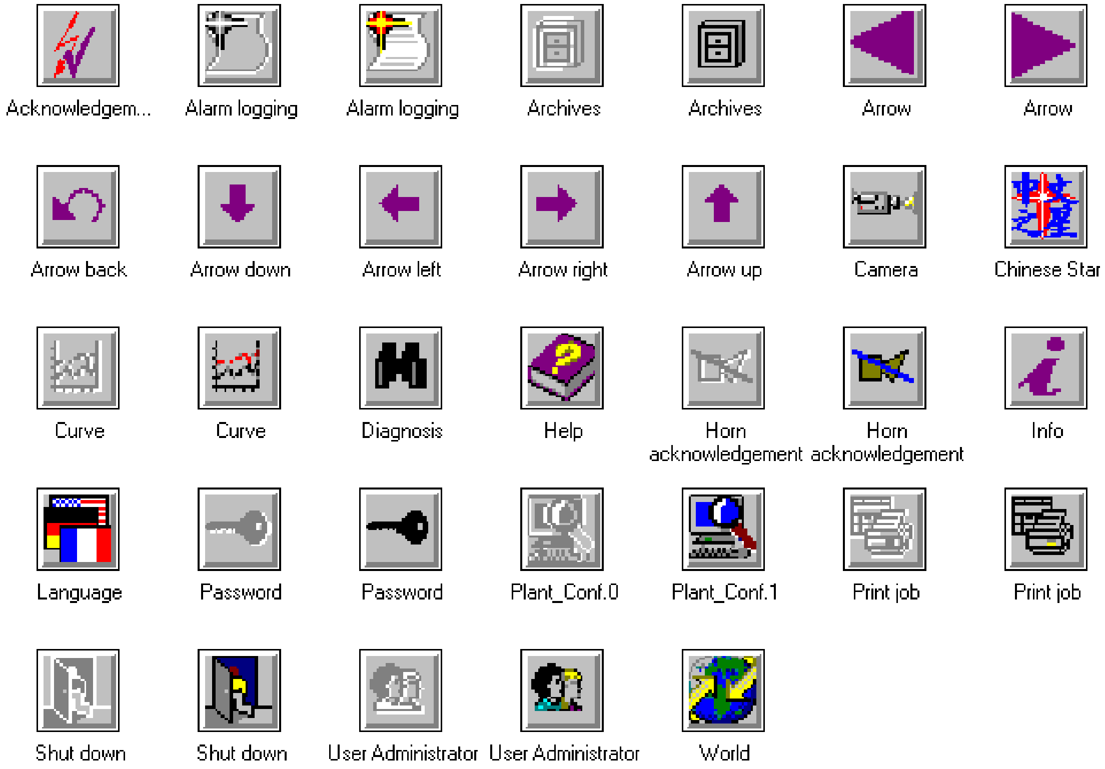

Meter1_0-100


Meter1_Min-Max

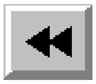

Meter2_Min-Max


Meter3_Min-Max

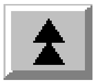
### 5.5.4 Button pictures



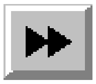
### 5.5.5 Buttons 3D



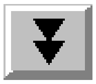
01



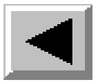
02



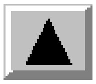
03



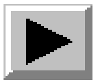
04



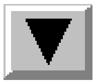
05



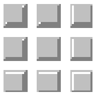
06



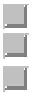
07



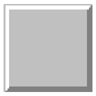
08



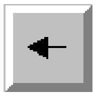
09



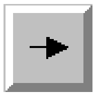
10



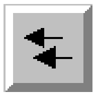
11



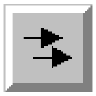
12



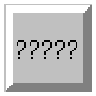
13



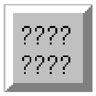
14



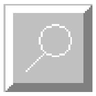
15



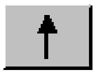
16



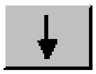
17



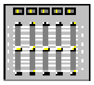
18



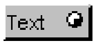
19



20



21



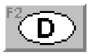
22

### 5.5.6 Buttons language

Bouton F

Button D

Button GB

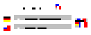


D

F

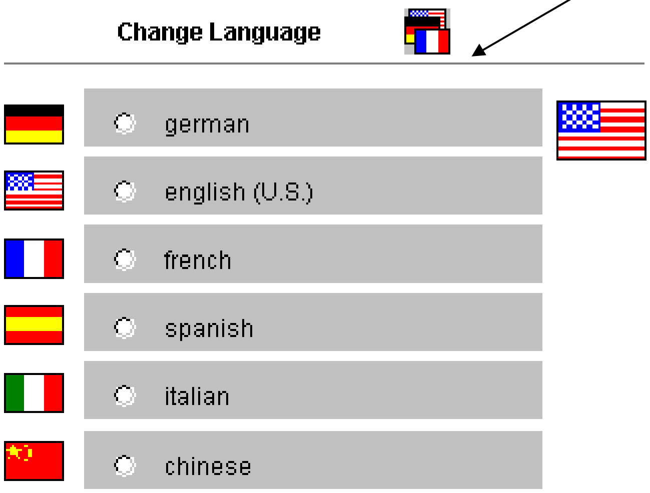
GB

Language D-F



Language D-USA

Language all



Change Language

○ german

○ english (U.S.)

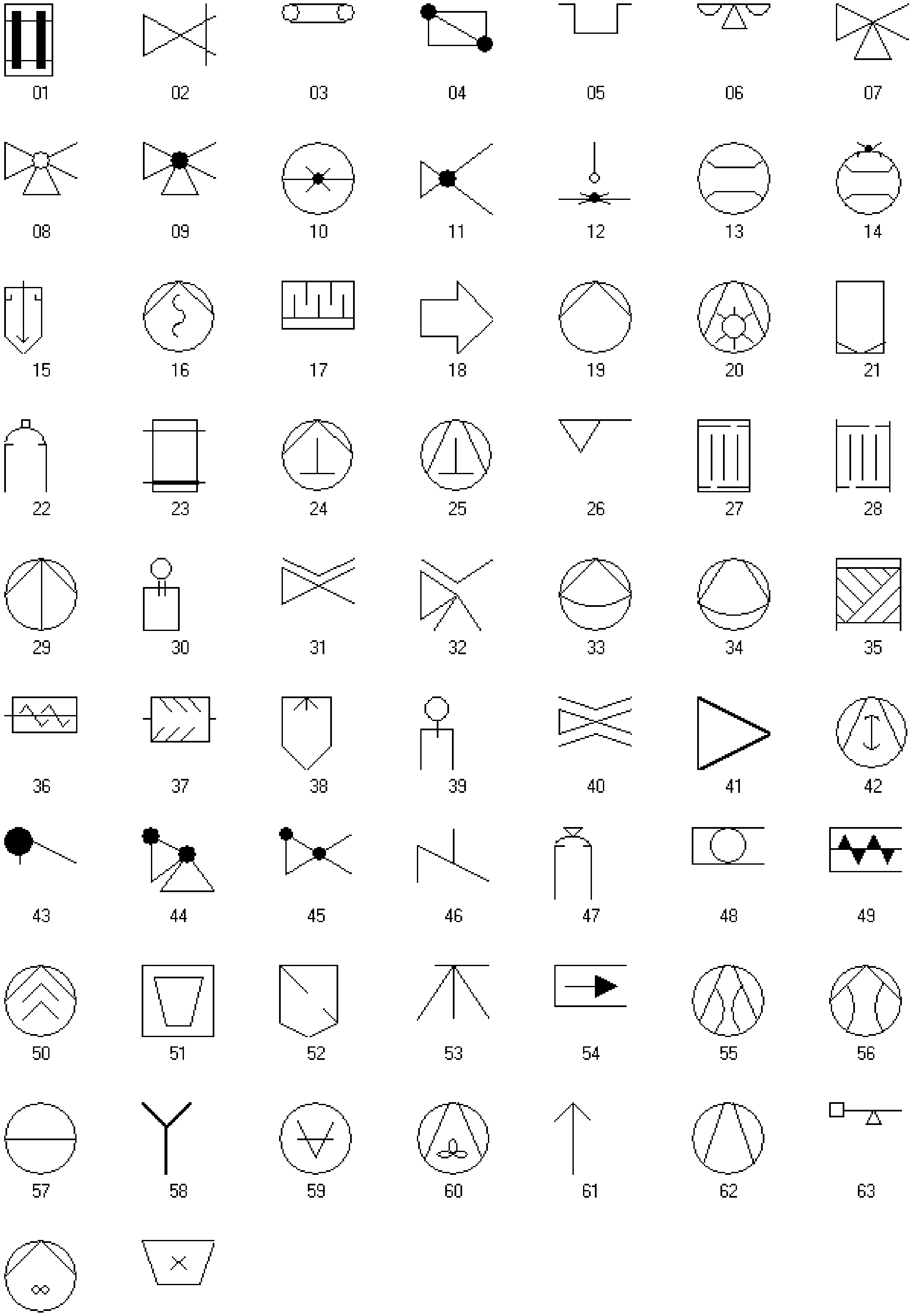
○ french

○ spanish

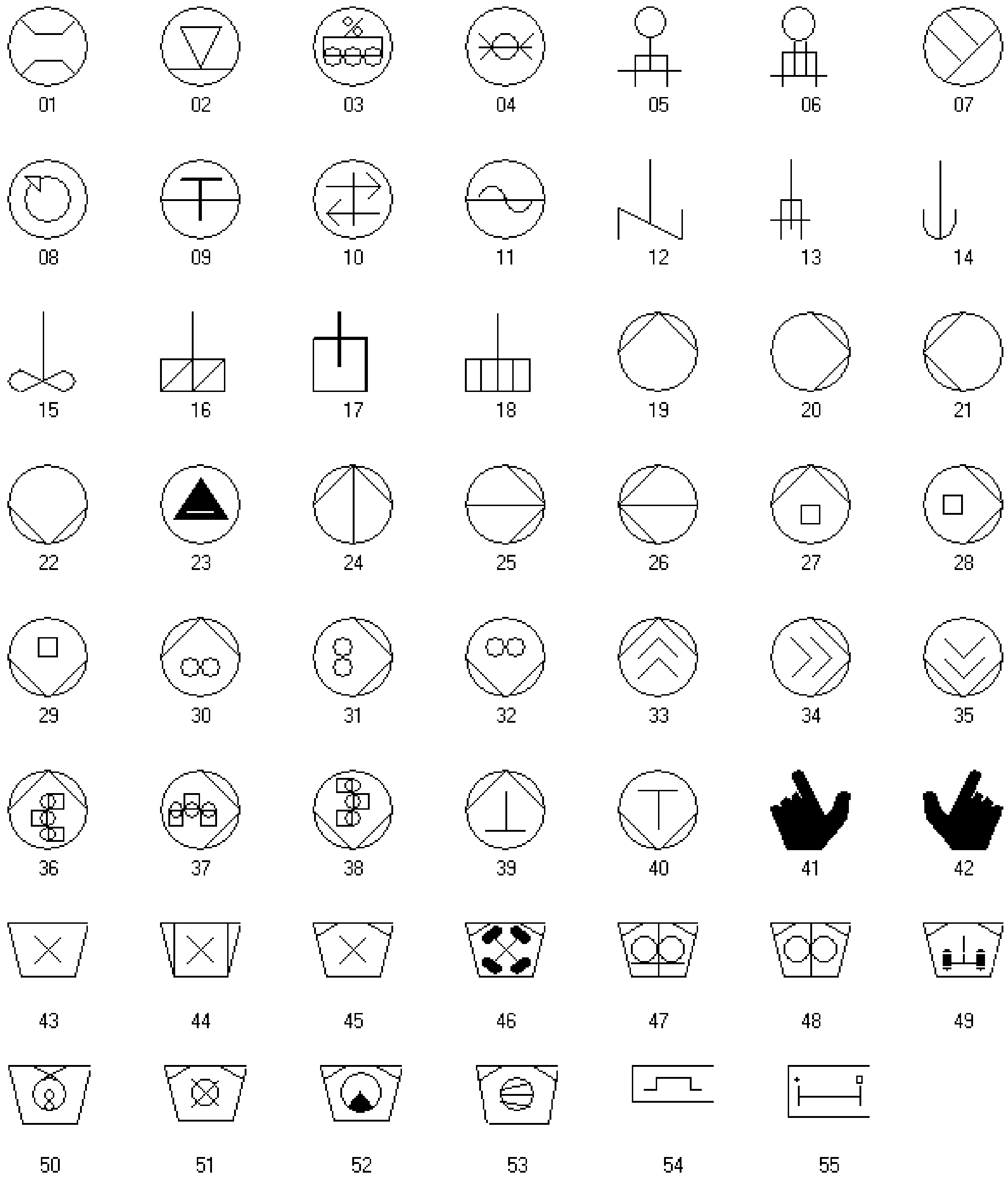
○ italian

○ chinese

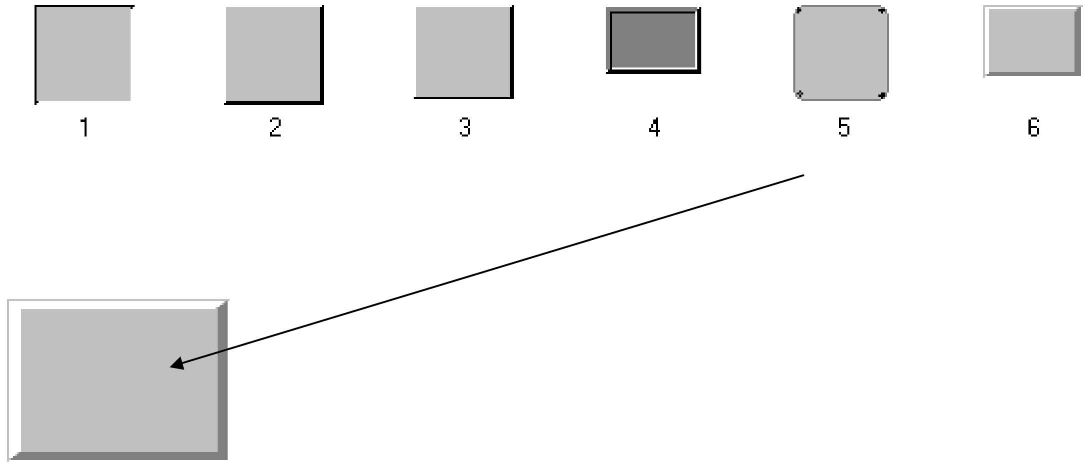
### 5.5.7 DIN30600



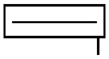
### 5.5.8 E symbols

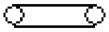


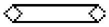
### 5.5.9 Windows

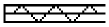


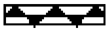
### 5.5.10 Conveyors

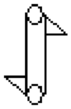

1


2


3

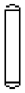

4

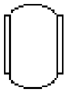

5

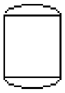

6

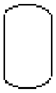
## 5.5.11 ISA symbols

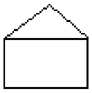
### 5.5.11.1 isa_s55a

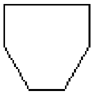

01

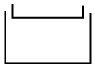

02

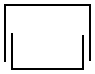

03

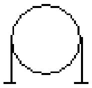

04

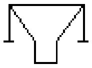

05


06

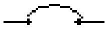

07

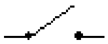

08

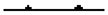

09

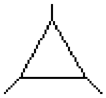

10

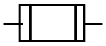
### 5.5.11.2 isa_s55b

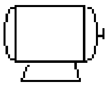

1


2

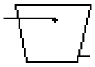

3

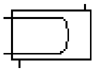

4

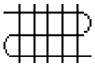

5

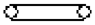

6

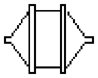
### 5.5.11.3 isa_s55c

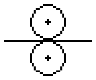

01

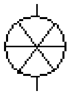

02

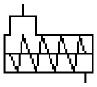

03

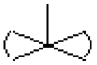

04

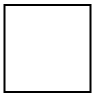

05


06

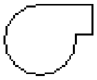

07

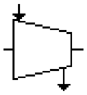

08

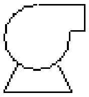

09

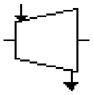

10

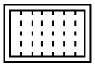
### 5.5.11.4 isa_s55d

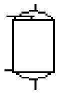

01

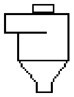

02

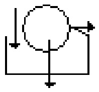

03

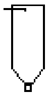

04


05

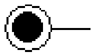

06

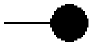

07

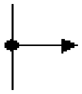

08

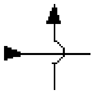

09

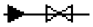
### 5.5.11.5 isa_y32a

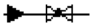

01

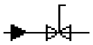

02

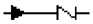

03


04

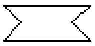

05

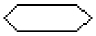

06

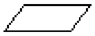

07

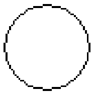

08

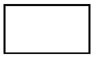
### 5.5.11.6 isa_y32b

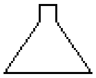

01

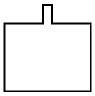

02

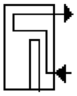

03

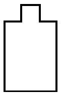

04

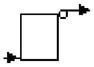

05


06

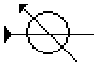

07

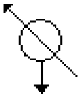

08

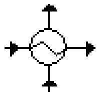

09

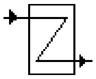

10

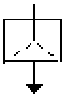
### 5.5.11.7 isa_y32c

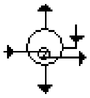

01

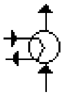

02

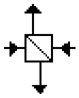

03

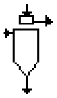

04


05

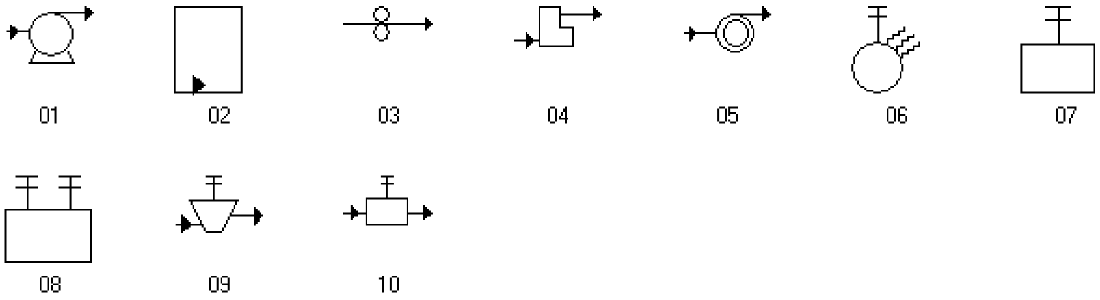

06


07

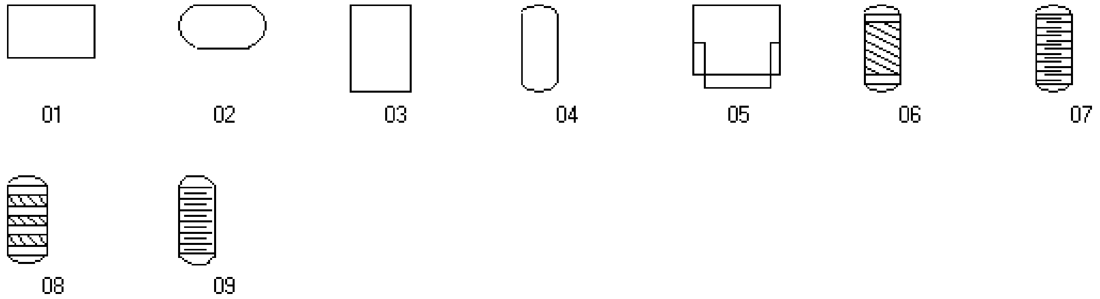

08


09

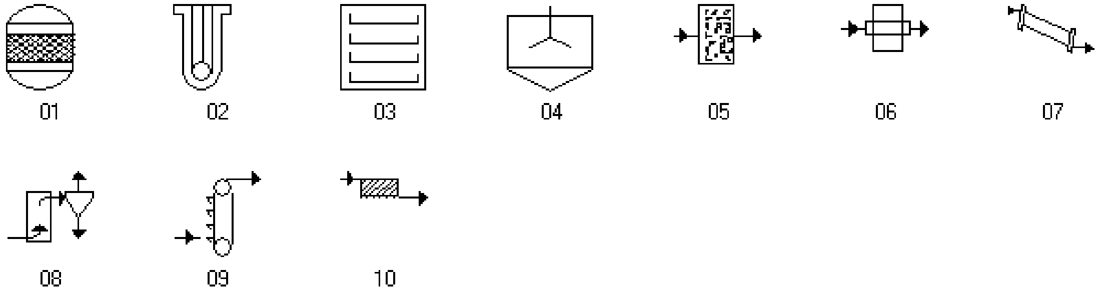
### 5.5.11.8 isa_y32d



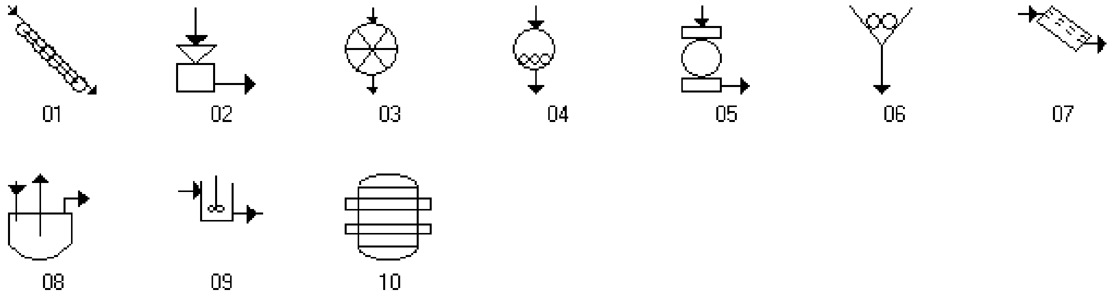
### 5.5.11.9 isa_y32e



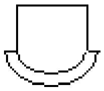
### 5.5.11.10 isa_y32f

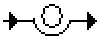


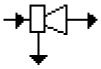
### 5.5.11.11 isa_y32g

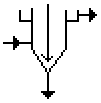


### 5.5.11.12 isa_y32h

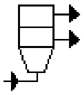

01

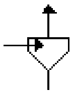

02

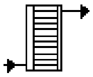

03

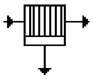

04

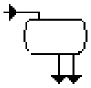
### 5.5.11.13 isa_y32i

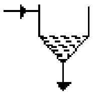

01

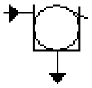

02


03

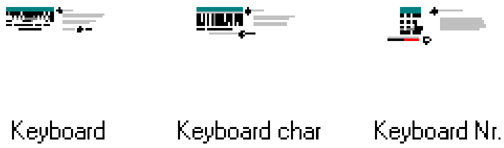

04

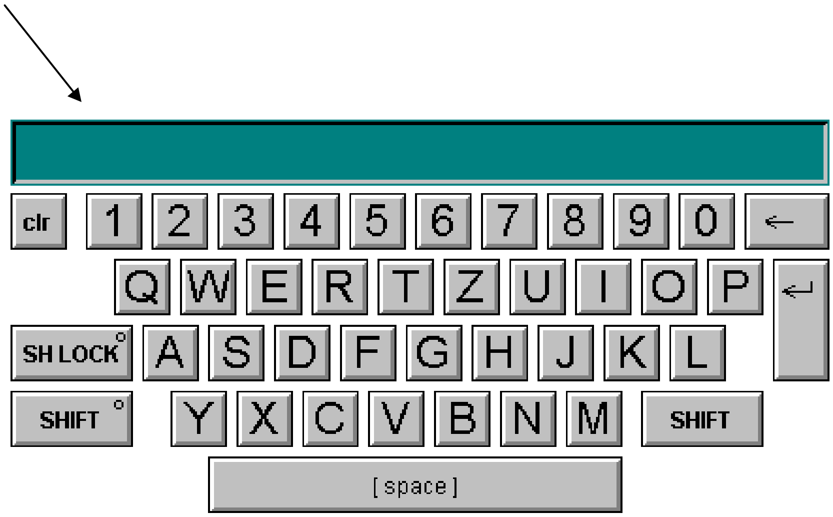

05


06

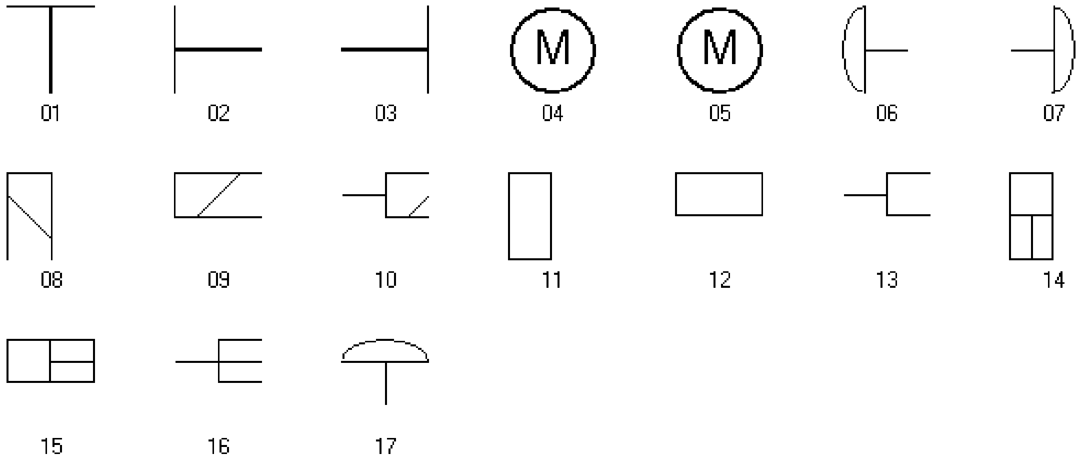

07

### 5.5.12 Keyboards



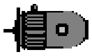


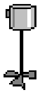
### 5.5.13 Motors

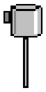


### 5.5.14 Motors 3D

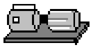
B3

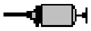

B5

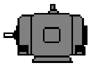

Motor001

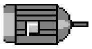

Motor002

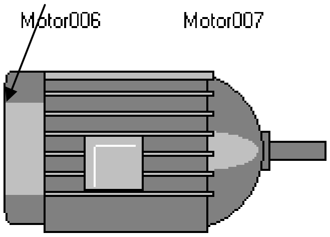
Motor003


Motor004

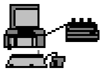

Motor005


Motor006

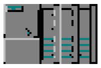

Motor007



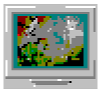
### 5.5.15 PC / PLC



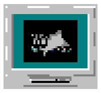
PC



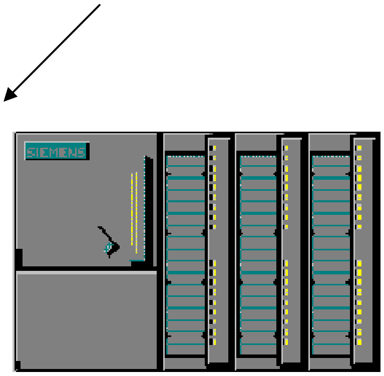
PLC S7 400



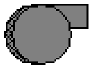
Screen WinCC1

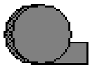


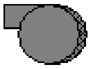
Screen WinCC2

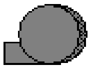


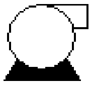
### 5.5.16 Pumps

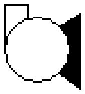

Pumpe001

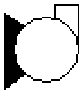

Pumpe002

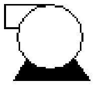

Pumpe003

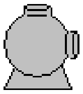

Pumpe004

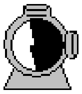

Pumpe005

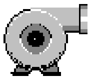

Pumpe006

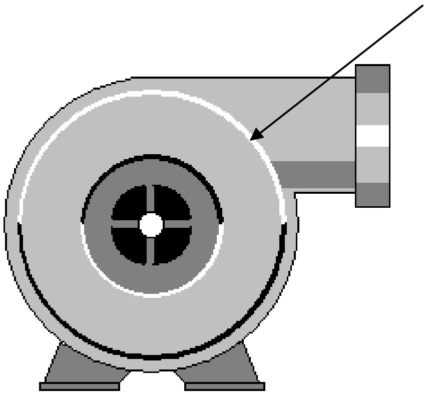

Pumpe007


Pumpe008

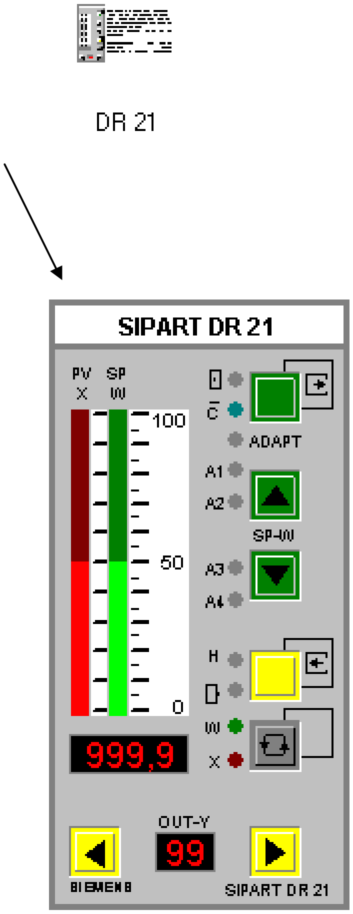

Pumpe009


Pumpe010

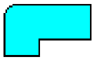

Pumpe011

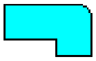


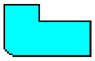
### 5.5.17 Controller

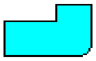


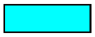
### 5.5.18 Pipes

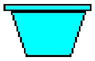

01

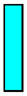

02

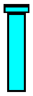

03

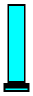

04

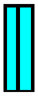

05

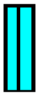

06

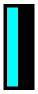

07

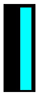

08

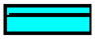

09

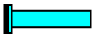

10

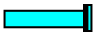

11

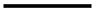

12

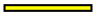

13

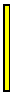

14

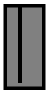

15

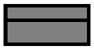

16

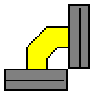

17

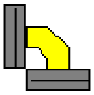

18

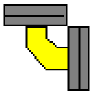

19

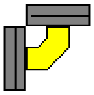

20


21


22

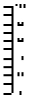

23


24


25

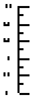
### 5.5.19 Scaling

01



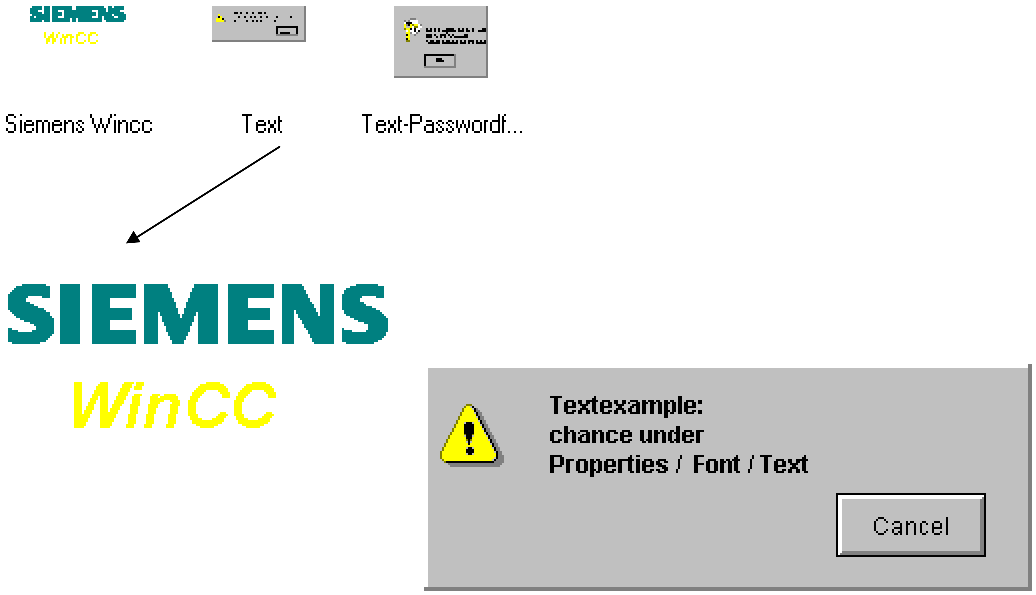
02

03



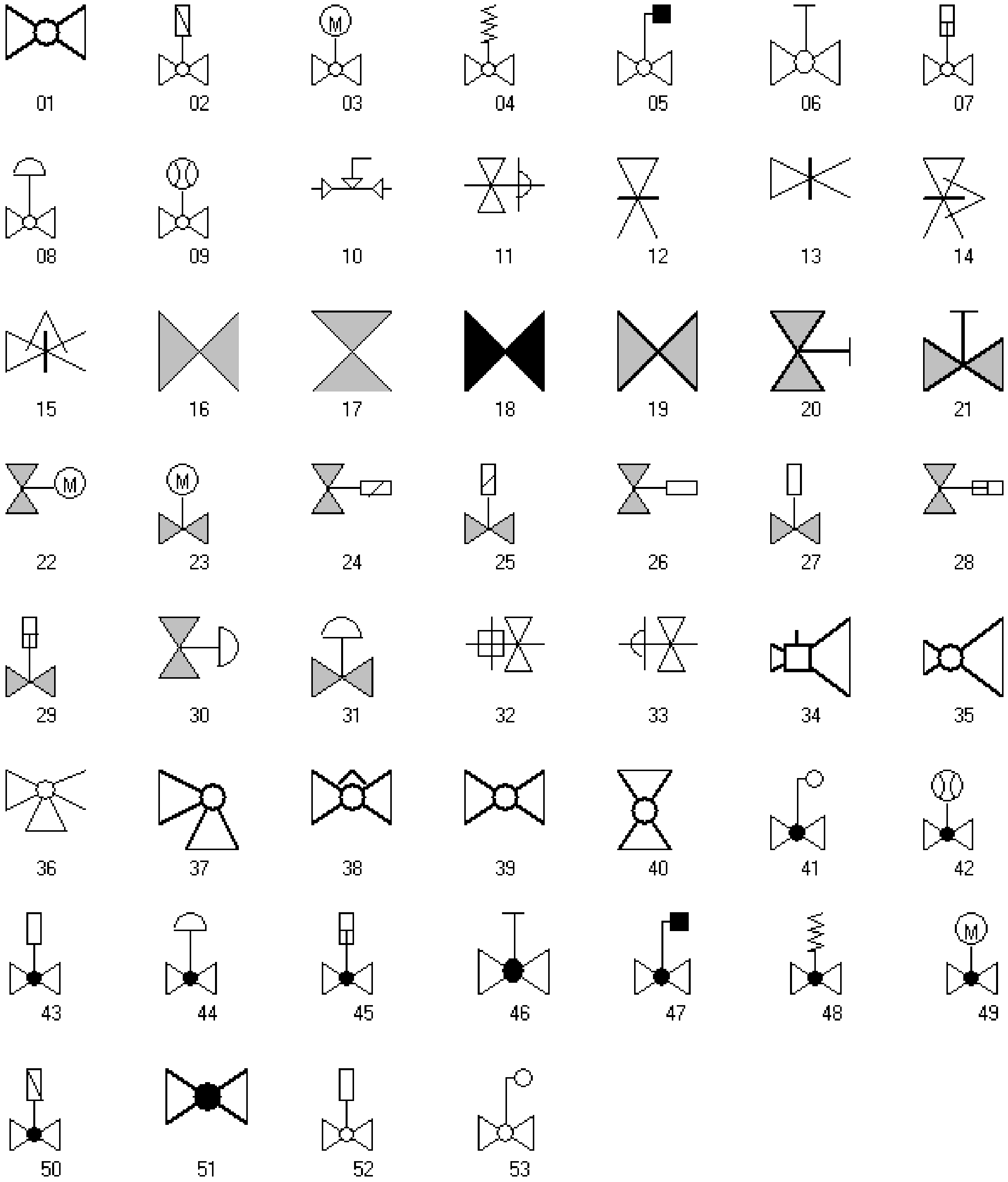
04

## 5.5.20 Text fields

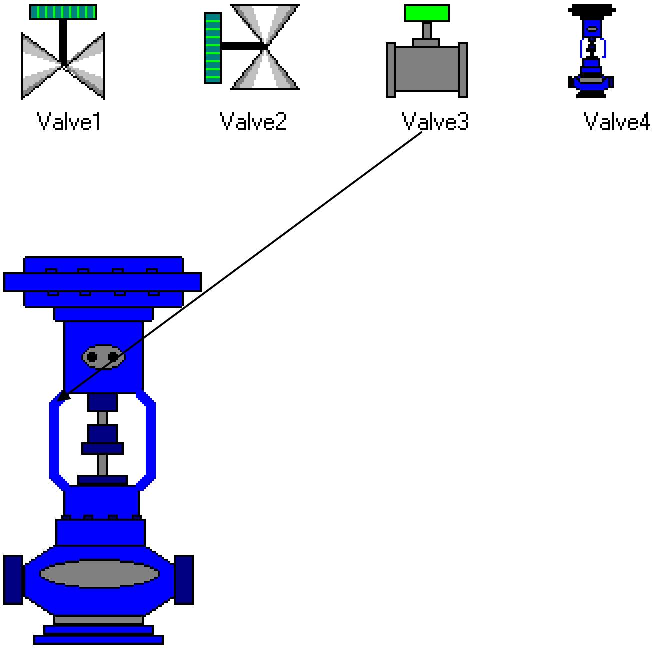


Textexample:
chance under
Properties / Font / Text

### 5.5.21 Valves


01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

### 5.5.22 Valves 3D



# Index